Version 2.0



DEPARTMENT OF THE AIR FORCE
Software Technology Support Center

GUIDELINES for SUCCESSFUL ACQUISITION and MANAGEMENT of SOFTWARE-INTENSIVE SYSTEMS:

Weapon Systems
Command and Control Systems
Management Information Systems

JUNE 1996

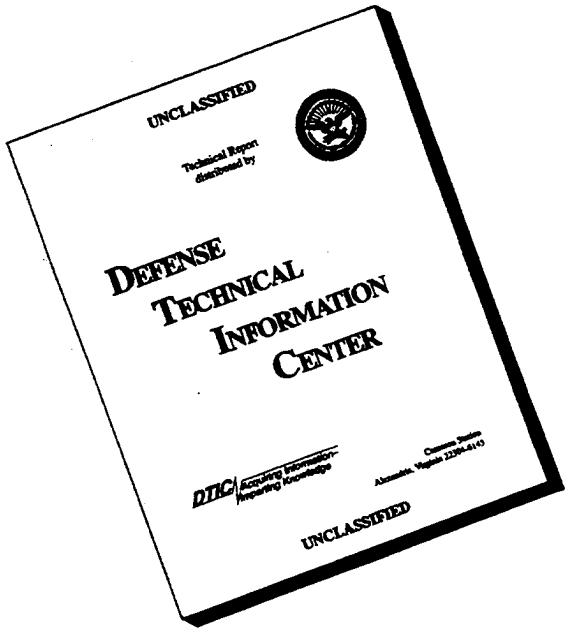
DTIC QUALITY INSPECTED 4

Volume 2
APPENDICES

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited.

19961004 059

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Version 2.0



DEPARTMENT OF THE AIR FORCE
Software Technology Support Center

GUIDELINES for SUCCESSFUL ACQUISITION and MANAGEMENT of SOFTWARE-INTENSIVE SYSTEMS:

Weapon Systems
Command and Control Systems
Management Information Systems

JUNE 1996

Volume 2
APPENDICES

Blank page.

Version 2.0

Volume 2

APPENDICES

Blank page.

TABLE OF CONTENTS

VOLUME 2
Appendices

PART I
Points of Contact and Web Directories

APPENDIX A **Government and Industry Points of Contact**

APPENDIX B **Web Directory of Software Acquisition and Engineering Policy, Information, Education, and Services**

PART II
Policy and Information-Related Appendices

APPENDIX C **Policy Memoranda**

APPENDIX D **Software-Related Government and Industry Documents**

APPENDIX E **Selected Technical References**

TAB 1 Alphabetized Listing and Synopsis of Selected Technical Reports

TAB 2 Comprehensive Approach to Reusable Defense Software (CARDS)

TAB 3 GSA Information Resource Management Publications

APPENDIX F **Selected Reading and Reference Material**

PART III
Engineering-Related Appendices

APPENDIX G Software Architecture
TAB 1 The Importance of Architecture in DoD Software
TAB 2 A New Process for Acquiring Software Architecture

APPENDIX H How Should Military Software Be Documented?

APPENDIX I A Detailed Comparison of ISO 9001 and the Capability Maturity Model (CMM)

APPENDIX J Counting Rules for Function Points and Feature Points

APPENDIX K Software Support
TAB 1 How Post-Deployment Software Support Starts in the Design Phase
TAB 2 LSA for Software: A Practical Approach
TAB 3 Computer Resource Integrated Support Document (CRISD) Foundation to Post-Deployment Software Support (PDSS)
TAB 4 Software Quality Assurance Impact on Post-Deployment Software Support Cost
TAB 5 The Generic Fighter: F-22's Historical Foundation
TAB 6 How More is Less: The F-22 Streamlined Block Change Cycle

APPENDIX L STARS and I-CASE Tools and Services
TAB 1 Software Technology for Adaptable, Reliable Systems (STARS)
TAB 2 Integrated-Computer-aided Software Engineering (I-CASE)

Table of Contents

PART IV
Management-Related Appendices

APPENDIX M Software Source Selection
- TAB 1 Source Selection Under Acquisition Reform
- TAB 2 Software Capability Evaluation (SCE)
- TAB 3 Sample RFP Preparation Checklists
- TAB 4 Sample Paragraphs for RFP Inclusion
- TAB 5 Source Selection for Software Supportability

APPENDIX N Automated Information Systems (AIS) Operational Requirements Documents (ORDs) Recommendations

PART V
Additional Addenda

APPENDIX O Additional Volume 1 Addenda

Table of Contents

Blank page.

Version 2.0

PART I

Points of Contact and Web Directories

Blank page.

APPENDIX

A

Government and Industry Points of Contact

Blank page.

APPENDIX A

Government and Industry Points of Contact

NOTE: Additional information and points of contact can be identified through the World-Wide Web. See Appendix B for a list of software acquisition and engineering policy, information, education and services Web addresses.

CONTENT **PAGE**

GOVERNMENT POINTS OF CONTACT A-2
- Software Acquisition Policy and Information A-2
- Software Engineering and Technology Information A-2
- Software Education and Classes A-5
- Software Publications and Services A-5

INDUSTRY POINTS OF CONTACT A-6
- Software Products, Policy, and Information A-6
- Software Education and Classes A-8

SOFTWARE ESTIMATING TOOLS AND MODELS POINTS OF CONTACT A-9
- Software Size Estimation A-9
- Software Effort, Cost, and Schedule Estimation A-9
- COCOMO Models A-9
- Historical Software Development Database Sources A-10

APPENDIX A Points of Contact

GOVERNMENT POINTS OF CONTACT

Software Acquisition Policy and Information

Air Force Software Acquisition Policy
Deputy Assistant Secretary of the Air Force (Science, Technology, and Engineering)
Systems Engineering Division
SAF/AQRE
1919 S. Eads Street, Suite 100
Phone: (703) 602-9200
DSN 332-9200
Fax: (703) 602-9199

Air Force Acquisition Model (AFAM) Information
ASC/CYM
Building 17
2060 Monahan Way
Wright-Patterson AFB, OH 45433-6503
Phone: (513) 255-0423
DSN 785-0423

Automated Information Systems (AIS) Operational Requirements Document (ORD) Information
HQ AFOTEC/TKT
8500 Gibson Blvd S.E.
Kirtland AFB, NM 87117-5558
Phone: (505) 846-8130
DSN 246-8130

Software Engineering and Technology Information

Ada Information Clearinghouse
P.O. Box 1866
Falls Church, VA 22041
Phone: (800) AdaIC-11
(800 232-4211)
or: (703) 681-2466
Fax: (703) 681-2869
E-mail: *adainfo@sw-eng.falls-church.va.us*

Air Force Cost Analysis Agency (AFCAA)
1111 Jefferson-Davis Highway, Suite 403
Arlington, VA 22202-4306
Phone: (703) 604-0488
Fax: (703) 604-6646

Air Force Software Technology Support Center (STSC)
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Phone: (801) 777-8045
DSN 458-8068
Fax: (801) 777-8069
DSN 458-8069
E-mail: *custserv@software.hill.af.mil*

Comprehensive Approach to Reusable Defense Software (CARDS)
100 University Drive
Fairmont, West Virginia, 26554
CARDS hotline:
Phone: (800) 828-8161
Fax: (304) 367-8211
E-mail: *hotline@cards.com*

APPENDIX A Points of Contact

Defense Information Systems Agency (DISA)
Public Affairs Office
701 South Courthouse Road
Code PAO
Arlington, VA 22204-2199
Phone: (703) 607-6900

Evaluating C3 Systems
ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731
Phone: (617) 377-8488
DSN 478-8488

Evaluating Embedded/Avionics Systems
ASC/ENC Bldg 126
2664 Skyline Drive
Wright-Patterson AFB, OH 45433-7800
Phone: (513) 255-7126 x-285
DSN 785-7126 x-285

Evaluating MIS Systems
SSG/EN
200 E. Moore Drive
Maxwell AFB-Gunter Annex, Alabama 36114-3004
Phone: (334) 416-4377
DSN 596-4377

General Services Administration (GSA) Virtual Library
Partnership with National Technical Information Service (NTIS)
[To get a free copy of the list faxed to your office (24hrs a day):]
Phone: (703) 487-4099
[NOTE: When prompted, enter "#" for the Fax Direct Code, and enter "420" for the document number. To receive the free GSA information technology newsletter, fax your name, organization, mailing address, phone number, and fax number to:]
Fax: (202) 208-1261

I-CASE Pilot Programs Information and Lessons Learned
Software Engineering Environments Department
DISA Center for Operational Support
5600 Columbia Pike
Falls Churc, VA 22041
Phone: (703) 681-2335
DSN 761-2335

Integrated Computer Aided Manufacturing Definition Language (IDEF)
SAF/AQK
1060 Air Force Pentagon
Washington, DC 20330-1060
Phone: (703) 697-3313

National Software Data & Information Repository (NSDIR)
Program Office
ESC/ENS
5 Eglin Street
Bldg 1704
Hanscom AFB, MA 01731-2116
Phone: (617) 377-9365

Practical Software Measurement: A Guide to Objective Program Insight
Joint Logistics Commanders
Joint Policy Coordinating Group on Computer Resources Management
Naval Undersea Warfare Center, Code 2252
1176 Howell Street
Newport, RI 02841-5047
Phone: (401) 841-4581
DSN 948-4581
Fax: (401) 841-2130
E-mail: *psm@ada.npt.nuwc.navy.mil*

Program Management Support System (PMSS)
HQ MSG/SZ
4225 Logistics Ave, Suite 20
Wright-Patterson AFB, OH 45433-5759
Phone: (513) 257-8405

APPENDIX A Points of Contact

Reuse Information Clearinghouse
P.O. Box 1068
Falls Church, Virginia 22041
Phone: (800) 738-7379
(703) 681-2471
E-mail: *reuseic@sw-eng.falls-church.va.us*

Rome Laboratory
RL/C3CB
525 Brooks Road
Rome, New York 13441-4505
Phone: (315) 330-4063
Fax: (315) 330-7989

Software Development Integrity Program
ASC/ENCR
Wright-Patterson AFB, OH 45433-6503
Phone: (513) 255-3656
DSN 785-3656

Software Measurement Information and Lessons Learned
Rome Lab Software Quality Consortium
RL/ C3C
525 Brooks Road
Rome, NY 13441-4505
Phone: (315) 330-4063

Software Operational Test and Evaluation:
HQ AFOTEC/SAS
8500 Gibson Blvd S.E.
Kirtland AFB, NM 87117
Phone: (505) 846-5310
DSN 246-5310

Software Process Improvement and Software Maturity Assessments (Air Force In-House Software Development Organizations Only)
AFC4A/XP
203 W Losey Street, Suite 1020
Scott AFB, IL 62225-5219
Phone: (618) 256-8915
DSN 576-8915

Space and Missile Systems Center (SMC)
SMC/SDEC
160 Skynett Street, Suite 2315
Los Angeles AFB, CA 90245-4683
Phone: (310) 363-2436
DSN 833-2436

Standard Systems Group
SSG/EN
200 E. Moore Drive
Maxwell AFB-Gunter Annex, Alabama
36114-3004
Phone: (334) 416-4377
DSN 596-4377

Systems Security Engineering (SSE) Capability Maturity Model (CMM[SM])
National Security Agency
Attn: V21
9800 Savage Road
Ft Meade, MD 20755-6000
Phone: (410) 859-6091

Systems Engineering Process (SEP) Handbook 700-10
SSG/ENSP
201 E. Moore Drive
Maxwell AFB-Gunter Annex, Alabama
36114-3005
Phone: (334) 416-4021
DSN 596-4021

APPENDIX A Points of Contact

Software Education and Classes

Air Force Education and Training Command (AFETC)
333 TRS/TTCSA
600 First Street, Suite 101
Keesler AFB, MS 39531
Phone: (601) 377-3602

Air Force Institute of Technology (AFIT) Software Engineering (Graduate)
AFIT/ENG
Wright-Patterson AFB, OH 45433-7765
Phone: (513) 255-2024
DSN 785-2024
Fax: (513) 476-4055
DSN 986-4055

AFIT Software Management (Graduate)
AFIT/LAS
Wright-Patterson AFB, OH 45433-7765
Phone: (513) 255-4845
DSN 785-4845

AFIT Software Management (Short Courses)
AFIT/LSS
Wright-Patterson AFB, OH 45433-7765
Phone: (513) 476-4500
DSN 986-4500

AFMC Systems Acquisition School
6575 SCHS/DPASC
Brooks AFB, TX
Phone: (512) 536-2623
DSN 240-2623
Fax: DSN 240-2522

Defense Acquisition University (DAU)
DAU Catalog
Director, National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
[NOTE: The catalog is accessible electronically on the World-Wide Web. See Appendix B for the Web address and access instructions.]

Software Publications and Services

Air Force Software Technology Support Center (STSC)
Customer Service
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056
Phone: (801) 777-8045
DSN 777-8045
Fax: (801) 777-8069
DSN 458-8069
E-mail: *custserv@software.hill.af.mil*

Software Program Managers Network (SPMN)
P.O. Box 2523
Arlington, VA 22202
Phone: (703) 521-5231
Fax: (703) 521-2603
E-mail: *nbrown@nadc.navy.mil*

INDUSTRY POINTS OF CONTACT

NOTE: The various companies whose products and services are mentioned throughout these Guidelines, as well as others who are not mentioned (either for lack of space or because they were not known to us), will usually provide initial consultation without cost. However, always beware of "*vaporware*" and slick-sounding promises. Remember, there are no Silver Bullets. If you want to improve your process and develop better quality products, be prepared to make a career-long commitment. Ask to see vendors' products demonstrated, and for references from user organizations who have benefited from their consultation and/or products.

Software Products, Policy, and Information

Ada-ASSURED
GrammaTech, Inc.
One Hopkins Place
Ithaca, NY 14850
Phone: (607) 273-7340
Fax: (607) 273-8752
E-mail: *jeff@grammatech.com*

Ada-based Design Approach for Real-Time Systems (ADARTS) Information
Software Productivity Consortium Clearinghouse
Phone: (800) 827-4SPC
or contact
The ADARTS Support Line
Phone: (703) 742-7130

AdaMAT
Rational Software Corporation
2800 San Tomas Expressway
Santa Clara, California 95051-0951
Phone: (408) 496-3600

AdaTEST
Quality Check Software
P.O. Box 6656
Beaverton, OR 97007-0656
Phone: (503) 645-5610
Fax: (503) 645-5075
E-mail: *qcs@teleport.com*

Amadeus Software Research, Inc.
10 Young Court
Irvine, CA 92715
Phone: (714) 725-6400
Fax: (714) 725-6411
E-mail: *amadeus-info@amadeus.com*

Analysis Complexity Tool
McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045
Phone: (800) 638-6316
Fax: (410) 995-1528

Asset Source for Software Engineering Technology (ASSET) Account and Product Information
ASSET
1350 Earl L. Core Road
PO Box 3305
Morgantown, West Virginia 26505
Phone: (800) 547-8306
(304) 284-9009
Fax: (304) 284-9001
E-mail: *info@source.asset.com*

Association for Computing Machinery (ACM)
ACM Order Department
P.O. Box 12114
Church Street Station
New York, NY 10257
Phone: (800) 342-6626
(212) 626-0500
E-mail: *acmhelp@acm.org*

Battlemap Analysis Tool
McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045
Phone: (800) 638-6316
Fax: (410) 995-1528

COHESION™ Team/SEE™
Digital Equipment Corporation
Attn: Cohesion Group
Mailstop MR1-1/P5
200 Forest Street
Marlboro, MA 01752
Phone: (508) 467-3130
Fax: (508) 467-1137

COHESIONworX™
Digital Equipment Corporation
Attn: Cohesion Group
Mailstop MR1-1/P5
200 Forest Street
Marlboro, MA 01752
Phone: (508) 467-3130
Fax: (508) 467-1137

APPENDIX A Points of Contact

Comprehensive Approach to Reusable Defense Software (CARDS)
CARDS System Architect
Loral Defense Systems-East
100 University Drive
Fairmont, West Virginia, 26554
Phone: (304) 367-8245
E-mail: *weisman@cards.com*
or contact
CARDS hotline:
Phone: (800) 828-8161
Fax: (304) 367-8211
E-mail: *hotline@cards.com*

Global Engineering Documents (to request International Electrotechnical Committee (IEC), Electronics Industry Association (EIA), and other documents)
7730 Carondelet Ave, Suite #407
St Louis, MO 63105
Phone: (800) 854-7179

IEEE Service Center (to request IEEE Standards and Publications)
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Phone: (908) 981-1393
Fax: (908) 981-9667

INFORMIX On-Line Secure Product Information
INFORMIX Federal
Phone: (703) 524-3600

Logicon - I-CASE Product Information
2100 South Washington Boulevard
Arlington, VA 22204-5704
Phone: (703) 486-3500

McCabe Design Complexity Tool
McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045
Phone: (800) 638-6316
Fax: (410) 995-1528

McCabe Slice Tool
McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045
Phone: (800) 638-6316
Fax: (410) 995-1528

National Software Data & Information Repository (NSDIR) User Support
WVU/NASA IV&V Facility
100 University Drive
Fairmont, WV 26554
Phone: (304) 367-8305
Fax: (304) 367-8211

Object-Oriented Tool
McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045
Phone: (800) 638-6316
Fax: (410) 995-1528

Oregon Graduate Institute (OGI)
Department of Computer Science
P.O. Box 9100
Portland, OR 97291-1000
Phone: (503) 690-1169

Process Weaver® Information
Loral Space Information Systems
3700 Bay Area Blvd.
Houston, TX 77058
Phone: (713) 282-8418
E-mail:
earllee@houvmscc.vnet.ibm.com

Program Design Language (PDL/81)
Steve Caine
Caine, Farber, & Gordon, Inc.
1010 East Union Street
Pasadena, CA 91106
Phone: (800) 424-3070
E-mail: *info@cfg.com*

Program Management Support System (PMSS)
Robbins-Gioia, Inc.
Phone: (513) 426-8081
Web url: *http://www.rgalex.com*

Quantitative Software Management Inc.
Lawrence Putnam, President
Phone: (800) 424-6755

Rational Software Corporation
2800 San Tomas Expressway
Santa Clara, California 95051-0951
Phone: (408) 496-3600

Software Engineering Institute (SEI)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-7700

Software Productivity Consortium (SPC)
SPC Building, 2214 Rock Hill Road
Herndon, VA 22070
Phone: (800) 827-4772
Fax: (703) 742-7200
E-mail: *ask-spc@software.org*

Software Productivity Research
T. Capers Jones, Chairman
Phone: (617) 273-0140
E-mail: *capers@spr.com*

APPENDIX A Points of Contact

Software Quality Engineer Certification Information
American Society for Quality Control (ASQC)
Phone: (800) 248-1946

SOftware Reliability Modeling & Analysis Toolset (SORTS)
SAIC
3800 Watt Avenue, Suite 210
Sacramento, CA 95821
Phone: (916) 974-8800

Type Commander Readiness Management System (TRMS)
Naval Computer and Telecommunications Area Master Station LANT
9456 Fourth Ave., Suite 20O
Norfolk, VA 23511-2199
Phone: (804) 444-3873
Fax: (804) 444-2835

Universal Network Architecture Service (UNAS) Product Information
Rational Software Corporation
3320 Scott Boulevard
Santa Clara, California 95054-3197
Phone: (408) 496-3600

Software Education and Classes

Fastrak Training Inc.
9175 Guilford Road, Suite 300
Columbia, MD 21046-1844
Phone: (800) 488-2321
Fax: (301) 924-3049
E-mail: *contact@fastrak.com*
Web url: *http://www.fastrak.com*

Learning Tree International
Phone: (800) THE TREE
(800 843-8733)
Fax: (800) 709-6405
E-mail: *uscourses@learningtree.com*
Web url: *http://www.learningtree.com*

Peer Inspection Training
Don O'Neill
Independent Consultant
9305 Kobe Way
Gaithersburg, MD 20879
Phone: (301) 990-0377

Software Engineering Institute (SEI)
Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-7700

Software Productivity Consortium (SPC)
SPC Building, 2214 Rock Hill Road
Herndon, VA 22070
Phone: (800) 827-4772
Fax: (703) 742-7200
E-mail: *ask-spc@software.org*

APPENDIX A Points of Contact

SOFTWARE ESTIMATING TOOLS AND MODELS POINTS OF CONTACT

NOTE: This section contains an alphabetical list of software estimating tools and models to aid your program planning and management process. It does not represent an endorsement or approval of any specific software estimating model or tool. It is also not all-inclusive.

Software Size Estimation

ASSET-R
Resource Calculations, Inc.
7853 East Arapahoe Court, Suite 2500
Englewood, CO 80112-1361
Phone: (303) 267-0379
Fax: (303) 220-5620

Checkpoint
Software Productivity Research
77 South Bedford Street
Burlington, MA 01803-5154
Phone: (617) 273-0140
Fax: (617) 273-5176

PRICE-S
Martin Marietta
PRICE Systems
300 Route 38
Moorestown, NJ 08057-3270
Phone: (800) 437-7423

SASET
Air Force Cost Analysis Agency (AFCAA/ FMP)
1111 Jefferson Davis Highway, Suite 403
Arlington, VA 22202
Phone: (703) 692-0006
Fax: (703) 746-5984

SEER-SSM
Galorath Associates, Inc
SEER Technologies Division
P.O. Box 90579
Los Angeles, CA 90009
Phone: (310) 670-3404

SLIM Planner
Quantitative Software Management, Inc.
1057 Waverly Way
McLean, VA 22101
Phone: (703) 790-0055

Software Effort, Cost, and Schedule Estimation

Checkpoint
Software Productivity Research
77 South Bedford Street
Burlington, MA 01803-5154
Phone: (617) 273-0140
Fax: (617) 273-5176

COCOMO Models

NASA COST MODELER
Lyndon B. Johnson Space Center
Mail Code: P14/Roush
Houston, TX 77058-3696

RECKON
Monterege Design, Inc.
P.O. Box 1222
Dartmouth, NS, Canada
B2Y 4B9
Phone: (902) 466-0246
Fax: (902) 464-1134

APPENDIX A Points of Contact

REVIC
Air Force Cost Analysis Agency
(AFCAA/FMP)
1111 Jefferson Davis Highway, Suite 403
Arlington, VA 22202
Phone: (703) 692-0006
Fax: (703) 692-0001

SECOMO
IIT Research Institute
201 Mill Street
Rome, NY 13440
Phone: (315) 336-2359

SOFTSTAR
Softstar Systems
28 Ponemah Road
Amherst, NH 03031
Phone: (603) 672-0987

SWAN
IIT Research Institute
201 Mill Street
Rome, NY 13440
Phone: (315) 336-2359

COCOMOID
AFMC/FMCI
Wright-Patterson AFB, OH 45433
Phone: (513) 257-3927

PRICE S
Martin Marietta
PRICE Systems
300 Route 38
Moorestown, NJ 08057-3270
Phone: (800) 437-7423

SASET
Air Force Cost Analysis Agency
(AFCAA/FMP)
1111 Jefferson Davis Highway, Suite 403
Arlington, VA 22202
Phone: (703) 692-0006
Fax: (703) 692-0001

SEER-SEM
Galorath Associates, Inc
SEER Technologies Division
P.O. Box 90579
Los Angeles, CA 90009
Phone: (310) 670-3404

SLIM
QSM, Inc.
1057 Waverly Way
McLean, VA 22101
Phone: (703) 790-0055

Softcost-Ada
Resource Calculations, Inc.
7853 East Arapahoe Court, Suite 2500
Englewood, CO 80112-1361
Phone: (303) 267-0379
Fax: (303) 220-5620

Softcost-R
Resource Calculations, Inc.
7853 East Arapahoe Court, Suite 2500
Englewood, CO 80112-1361
Phone: (303) 267-0379
Fax: (303) 220-5620

Historical Software Development Database Sources

QSM
2000 Corporate Ridge, Suite 900
McLean, VA 22102
(703) 790-0055

Software Productivity Research
1 New England Executive Park
Burlington, MA 01803
(617) 273-0140

Resource Calculations, Inc.
7853 E. Arapahoe Court, Suite 2500
Englewood, CO 80112-1361
(303) 267-0379

APPENDIX

B

Web Directory of Software Acquisition and Engineering Policy, Information, Education, and Services

Blank page.

APPENDIX B

Web Directory of Software Acquisition and Engineering Policy, Information, Education, and Services

NOTE: **The following addresses will allow you to find current DoD and industry information on software acquisition and engineering. This is not an exhaustive list of references or directions, and is intended only as a starting point. Connections such as "*other sites*" or "*interesting links*" (available on some home pages) may provide additional topic-related sites.**

CONTENT **PAGE**

DoD Software Acquisition Policy and Information B-1
DoD Software Engineering and Technology Information and Services B-3
DoD Software Education and Training B-5
DoD Software Publications B-5
Commercial Software Acquisition and Engineering Policy, Information, Education, and Services B-6

DoD SOFTWARE ACQUISITION POLICY AND INFORMATION

To access Air Force Acquisition policy and information:
start at **http://www.safaq.hq.af.mil**
click on **Policy**
select the information you wish to access

To access the Air Force Acquisition Model (AFAM):
start at **http://afamsun.wpafb.af.mil**
select the information you wish to access

APPENDIX B Web Directory

To access Army Acquisition policy and information:
start at **http://www.army.mil**
click on the letter **A** under "*By Subject*"
click on **Assistant Secretary of the Army for Research, Development and Acquisition**
select the information you wish to access

To access Defense Information Systems Agency (DISA) policy, standards, and information:
start at **http://www.itsi.disa.mil**
select the information you wish to access

To access DoD Directives and Instructions online:
start at **http://www.dtic.dla.mil/adm/**
select the information you wish to access

To access the DoD Index of Specifications and Standards (DoDISS) on-line:
start at **http://www.dtic.mil/dps-phila/dodiss/**
select the information you wish to access

To access the Federal Acquisition Jumpstation:
start at **http://msfcinfo.msfc.nasa.gov/fedproc/home.html**
select the server you wish to access

To access the Federal Acquisition Regulations:
start at **http://www.safaq.hq.af.mil**
click on **Contracting**
click on **FAR Etc**
select the information you wish to access

To access current Legislation Status, including the most recent versions of important Information Technology bills:
start at **http://www.fcw.com**
click on **Reference Desk**
click on **Legislation**
select the information you wish to access

To access Navy Acquisition reform information:
start at **http://www.navy.mil**
click on **Naval Web Sites (alphabetical)**
click on **Navy Acquisition Reform (ARO)**
select the information you wish to access

To access Office of the Secretary of Defense, Acquisition and Technology, Military Specifications and Standards Reform information:
start at **http://www.acq.osd.mil/es/std/**
select the information you wish to access

APPENDIX B Web Directory

To access Office of the Secretary of Defense, C3I policy and information:
start at **http://www.dtic.dla.mil/defenselink/**
click on **Secretary of Defense**
click on **Assistant Secretary of Defense (Command, Control, Communications and Intelligence)**
click on **Policy/Guidance**
select the policy you wish to access

DoD SOFTWARE ENGINEERING AND TECHNOLOGY INFORMATION AND SERVICES

To access the Ada Information Clearinghouse:
start at **http://sw-eng.falls-church.va.us**
click on **AdaIC**
select the information you wish to access

To access Air Force software engineering and technology information:
start at **http://www.af.mil**
click on **Air Force Sites**
select the site of the information you wish to access
select the information you wish to access

To access Air Force software process improvement and assessments, and software reuse information:
start at **http://infosphere.safb.af.mil**
select the information you wish to access

To access the Army Reuse Center:
start at **http://arc_www.belvoir.army.mil/**
select the information you wish to access

To access Army software engineering and technology information:
start at **http://www.army.mil**
either
click on the letter **S** under "*By Subject*"
click on **Information Systems Software Center** (or another topic/organization under a "***Software***" heading)
select the information you wish to access
or
click on the letter **A** under "*By Subject*"
click on **Software Development Center - Washington** (under the "*Ada*" heading)
select the information you wish to access

To access the Comprehensive Approach to Reusable Defense Software (CARDS) program:
start at **http://www.cards.com/**
select the information you wish to access

APPENDIX B Web Directory

To access the General Accounting Agency (GAO):
start at **http://www.gao.gov**
select the information you wish to access

To access the National Aeronautics and Space Administration (NASA):
start at **http://www.nasa.gov/nasa_homepage.html**
select the information you wish to access

To access the National Software Data & Information Repository (NSDIR):
start at **http://www.cards.com/nsdir**
select the information you wish to access

To access Navy software engineering and technology information:
start at **http://sepo.nosc.mil**
select the information you wish to access
[NOTE: This home page has links to many major DoD and industry software policy and engineering home pages, and makes a great starting point for exploring what's out there.]

To access Rome Laboratory software engineering and technology information:
start at **http://www.rl.af.mil:8001**
click on **C3**
select the information you wish to access

To access the Reuse Information Clearinghouse:
start at **http://sw-eng.falls-church.va.us**
click on **ReuseIC**
select the information you wish to access

To access Software Technology for Adaptable, Reliable Systems (STARS) program information:
start at **http://www.stars.reston.unisysgsg.com/stars/lmtds**
select the information you wish to access

To access the Software Technology Support Center (STSC):
start at **http://www.stsc.hill.af.mil**
select the information you wish to access

To access Standards Systems Group (SSG) software engineering and technology information:
start at **http://www.ssc.af.mil/HQSSGhome.html**
select the information you wish to access

APPENDIX B Web Directory

DoD SOFTWARE EDUCATION AND TRAINING

To see Air Force Institute of Technology (AFIT) graduate software and management classes:
start at **http://www.au.af.mil**
click on **AFIT Catalog**
select the information you wish to access

To see Air Force Institute of Technology (AFIT) professional continuing education (PCE) software and management classes:
start at **http://www.afit.af.mil**
click on **School of Systems and Logistics** (under Professional Continuing Education)
select the information you wish to access

To see Air Force Software Technology Training classes (includes Ada training):
start at **http://bova.kee.aetc.af.mil/synopsis.htm**
click on **S Flight**
select the information you wish to access

To see Defense Acquisition University, Software Acquisition Management (SAM) classes:
start at **http://www.ndu.edu**
click on **Information Resources Management College**
click on **Defense Acquisition University**
either
click on **List of DAU Courses** (under "What Courses are Available?")
click on **SAM**
select the information you wish to access
or
click on **DAU Catalog, Schedules, and Related Documents**
download the DAU catalog for viewing

To see Naval Post Graduate School classes:
start at **http://www.navy.mil**
click on **Naval web sites (alphabetical)**
click on **Naval Postgraduate School**
click on **Academic Departments**
select the information you wish to access

DoD SOFTWARE PUBLICATIONS

To access *CrossTalk* publication online:
start at **http://www.stsc.hill.af.mil/www/crostalk.html**
click on View **CrossTalk** Online
select the information you wish to access
*[NOTE: Click on **Subscription Form** to request a subscription to CrossTalk.]*

To access *CHIPS* publication online:
start at **http://www.chips.navy.mil/chips**
select the information you wish to access

COMMERCIAL SOFTWARE ACQUISITION AND ENGINEERING POLICY, INFORMATION, EDUCATION, AND SERVICES

To access the American National Standards Institute (ANSI):
start at **http://www.ansi.org**
select the information you wish to access

To access the American National Standards Institute (ANSI) standards catalog:
start at **http://www.ansi.org**
click on **The ANSI Catalog**
click on **Searching the ANSI Catalog**
click on **Search**
enter **software** for the search
select the information you wish to access

To access Asset Source for Software Engineering Technology (ASSET) account and product information:
start at **http://source.asset.com**
select the information you wish to access

To access the Association for Computing Machinery (ACM):
start at **http://www.acm.org**
select the information you wish to access

To access the Fed Center:
start at **http://www.fedcenter.com**
select the information you wish to access

To access *Federal Computer Week (FCW)*:
start at **http://www.fcw.com**
click on **Publications**
select the information you wish to access
[NOTE: to see the "mega list of federal web sites:"
*click on **Reference Desk***
*click on **Megalist of Federal Sites** (under Gateway Government Hotlists)*
select the information you wish to access]

To access *Government Computer News (GCN)*:
start at **http://www.cahners.com/~gcn/gcnhome.htm**
select the information you wish to access

To access the Institute of Electrical and Electronics Engineers (IEEE):
start at **http://www.ieee.org**
select the information you wish to access

APPENDIX B Web Directory

To access the Institute of Electrical and Electronics Engineers (IEEE) standards catalog:
start at **http://www.ieee.org**
click on **standards**
click on **Products and Publications**
click on **IEEE Standards Products Catalog**
click on **Catalog Listing (searchable)**
either
click on **Search the catalog**
enter **software** for the search
select the information you wish to access
or
click on **Software Engineering**
select the information you wish to access

To access the International Standards Organization (ISO):
start at **http://www.iso.ch/welcome.html**
select the information you wish to access

To access the International Standards Organization (ISO) standards catalog:
start at **http://www.iso.ch/welcome.html**
click on **ISO Catalogue**
click on **International Standards**
enter **software** for the search
select the information you wish to access

To access the National Software Council (NSC):
start at **http://www.nscusa.org**
select the information you wish to access

To access the Software Engineering Institute (SEI):
start at **http://www.sei.cmu.edu**
click on **Products**
select the information you wish to access

To access the Software Productivity Consortium (SPC):
start at **http://software.software.org/**
select the information you wish to access

To access the Software Program Manager's Network (SPMN):
start at **http://spmn.com**
select the information you wish to access

To access Systems Security Engineering (SSE) Capability Maturity Model (CMM℠) information:
start at **http://www.ssecmm.ashton.csc.com**
select the information you wish to access

Blank page.

PART II

Policy and Information-Related Appendices

Blank page.

APPENDIX

C

Policy Memoranda

Blank page.

APPENDIX C

Policy Memoranda

NOTE: The following selection of policy memoranda are current as of the time of this printing. Some of what was previously captured in policy memoranda is now being considered for inclusion in the *Defense Acquisition Deskbook*. With the acquisition environment undergoing change, the best way to find out latest policy memoranda is through the World-Wide Web. See Appendix B for a list of OSD and service component Web addresses.

DATE	DOCUMENT TITLE	PAGE
30 MAR 95	Technical Architecture Framework for Information Management (TAFIM), Version 2.0 [ASD(C3I)]	C-2
31 MAY 94	Software Maturity Criteria for Dedicated Operational Test and Evaluation of Software-Intensive Systems [DOT&E]	C-3
23 JUN 94	Technical Architecture Framework for Information Management (TAFIM) [ASD(C3I)]	C-5
29 JUN 94	Specifications & Standards — A New Way of Doing Business [SECDEF]	C-6
08 JUL 94	Software Acquisition Best Practices Initiative [OUSD(A&T)]	C-10
08 SEP 94	Data Element Standardization in Automated Information Systems Development [DCS/C4, SAF/AQK]	C-11
04 JAN 93	Preparation for Implementing Army Software Test and Evaluation Panel (STEP) Metrics Recommendations, [SAIS-ADW]	C-13
23 JUL 93	Air Force Software Release Policy, [DSC/C4]	C-15
23 OCT 93	C4I Systems Policy and Standards [SAF/SC]	C-17

APPENDIX C Policy Memoranda

ASSISTANT SECRETARY OF DEFENSE (Command, Control, Communications & Intelligence)

MEMORANDUM FOR UNDER SECRETARIES OF DEFENSE
ASSISTANT SECRETARY OF THE ARMY (RD&A)
ASSISTANT SECRETARY OF THE NAVY (RD&A)
ASSISTANT SECRETARY OF THE AIR FORCE (ACQUISITION).(SAF/AQ)
DIRECTORS OF THE DEFENSE AGENCIES
DIRECTOR, JOINT STAFF

SUBJECT: **Technical Architecture Framework for Information Management (TAFIM), Version 2.0**
DATE: March 30, 1995

My memorandum dated June 23, 1994 established the TAFIM as the single framework to promote the integration of Department of Defense (DoD) information systems, expanding the opportunities for interoperability and enhancing our capability to manage information resources across the Department. The latest version of the TAFIM, Version 2.0, is complete and fully coordinated. Version 2.0 consists of seven volumes as shown in the attachment. The TAFIM will continue to guide and enhance the evolution of the Department's information systems technical architectures.

I want to reiterate two important points that I made in my June 1994 memorandum. First, the Department remains committed a long range goal of an open systems environment where interoperability and cross functional integration of our system and portability/reusability of our software are key benefits. The TAFIM establishes that direction. Second, the further selection and evaluation of migration systems should take into account this long range goal by striving for conformance to the TAFIM to the extent possible.

Effective immediately, new DoD information systems development and modernization programs will conform to the TAFIM. Evolutionary changes to migration systems will be governed by conformance to the TAFIM

The TAFIM is maintained the Defense Information Systems Agency (DISA) and is available electronically via the DISA On-Line Standards Library. Hardcopy is available through the Defense Technical Information Center. The TAFIM is an evolving set of documents and comments for improving may be provided to DISA at any time. The DISA Action officer is Mr. Bobby Zoll, (703) 735-3552. The OSD action officer is Mr. Terry Hagle, (703) 604-1486.

Emmett Paige, Jr.

APPENDIX C Policy Memoranda

ATTACHMENT

Technical Architecture Framework for Information Management Version 2.0

DoD DATA ADMINISTRATION: TECHNIQUES, METHODS, & PRACTICES COURSE	
Introduction	Course Overview
Module 1	DoD Data Administration (role of CIM)
Module 2	Introduction to Activity & Data Modeling
Module 3	Activity
Module 4	Entity Relationship Diagrams
Module 5	Fundamentals of Data Modeling
Module 6	Applied Data Modeling
Module 7	Fully-Attributed Data Models
Module 8	Completing Standardization
Module 9	Model Integration
Module 10	Facilitated Coordination and Approval
Module 11	Database Implementation

MEMORANDUM FOR
SECRETARIES OF THE MILITARY DEPARTMENTS
ATTENTION: SERVICE ACQUISITION EXECUTIVES
ASSISTANT SECRETARY OF DEFENSE (COMMAND, CONTROL, COMMUNICATIONS, & INTELLIGENCE)
DIRECTOR, DEFENSE INFORMATION SYSTEMS AGENCY
DIRECTOR, DEFENSE LOGISTICS AGENCY
DIRECTOR FOR FORCE STRUCTURE, RESOURCES & ASSESSMENT, JOINT STAFF (J-8)
DIRECTOR, TEST AND EVALUATION, OUSD(A&T)
DEPUTY UNDER SECRETARY OF THE ARMY (OPERATIONS RESEARCH)
DIRECTOR, NAVY TEST & EVALUATION & TECHNOLOGY REQUIREMENTS
DIRECTOR, AIR FORCE TEST & EVALUATION

SUBJECT: **Software Maturity Criteria for Dedicated Operational Test and Evaluation of Software-Intensive Systems**

REFERENCE: GAO/NSIAD-93-198, "*Test and Evaluation: DoD Has Been Slow in Improving Testing of Software-Intensive Systems*," dated September 29, 1993

DATE: May 31, 1994

As a part of the Department's initiative to address the General Accounting Office's (GAO) recommendations on the Department's test and evaluation policy of software-intensive systems, I am issuing the following guidance to establish the software maturity criteria for the dedicated OT&E (in support of full rate production decisions or deployment decisions) of software-intensive systems. It is my intent to include this guidance in the revisions to the DoD

5000 and 8120 policy documents.

To improve the success rate of OT&E for software-intensive systems, and to prevent immature software-intensive systems from entering OT&E, software maturity must be demonstrated prior to the start of the dedicated OT&E. The following conditions must be satisfied and the results presented at the operational test readiness review that precedes the OT&E:

a. The system must not possess any know Priority I or II problems (as defined by the DoD-STD-2167A) that impact the OT&E so as to constitute a deficiency relative to a critical operational issue. Priority III problems must be documented with appropriate impact analyses completed. These impact analyses must focus on the problems' potential impact to the system's mission capability and the ability to resolve the affected critical operational issues. After the problems and their associated impact analyses are reviewed by the functional proponent, operational test agency, and other participating organizations, recommendations on whether to proceed, delay, or cancel the OT&E can be made to the designated Service or Agency operational test certification official.

b. System functionality to be operationally tested and evaluated must be available prior to the start of OT&E and must have been developmentally tested. In particular, the system features that are required to support specific requirements and the system interfaces that are required to interoperate with external systems must be certified to be functional, preferably in an operationally realistic environment (real users, data, procedures, etc.) against operational requirements.

c. The program management office in conjunction with the Service's or Agency's independent evaluator must identify all the unmet critical technical parameters and open deficiencies that have been noted during the developmental test and evaluation. During certification of readiness for dedicated OT&E, the acquisition executive must certify and the operational test agency must agree that the software requirements and design are stable, that software and interface testing of sufficient depth and breadth has been performed, and that required functionality has been successfully demonstrated at the system level in developmental testing. Impact analyses, on the shortfalls' potential impact to the system's mission capability and the ability to resolve the affected critical operational issues, must be completed.

d. A deficiency identification, tracking, and reporting system must be in place to support the monitoring of deficiency reports by the operational test agency. Further, a software configuration management system with the associated control procedures must be in place prior to the start of OT&E. Software-intensive systems to be operationally tested must be baselined in the configuration management system. During the operational test phase, the operational test agency must have complete access to the configuration management system.

e. Software or firmware changes, if any, must be completed prior to the start of OT&E and must not be implemented during the OT&E unless specifically acknowledged and concurred by the responsible operational test agency. The expected impact of these changes on the OT&E data stream and the evaluation of the critical operational issues must be addressed by the responsible operational test agency to assist in the decision to allow the change(s) during OT&E.

L.H. Frame for
Director

APPENDIX C Policy Memoranda

OFFICE OF THE ASSISTANT SECRETARY OF DEFENSE (C3I)
3300 DEFENSE PENTAGON
WASHINGTON, DC 20301-3300

MEMORANDUM FOR UNDER SECRETARIES OF DEFENSE
COMPTROLLER OF THE DEPARTMENT OF DEFENSE
ASSISTANT SECRETARY OF THE ARMY (RD&A)
ASSISTANT SECRETARY OF THE NAVY (RD&A)
ASSISTANT SECRETARY OF THE AIR FORCE ((ACQUISITION)(SAF/AQ))
DIRECTORS OF THE DEFENSE AGENCIES
DIRECTOR, JOINT STAFF

SUBJECT: **Technical Architecture Framework for Information Management (TAFIM)**
DATE: June 23, 1994

This memorandum affirms Department of Defense (DoD) commitment to the Technical Architecture Framework for Information Management (TAFIM). Since January 1993, the TAFIM has served as the single framework to promote the integration of DoD information systems, thus expanding the opportunities for interpretability and enhancing our capability to manage information resources across the Department. The TAFIM will guide the evolution of the Department's information system technical architectures.

As a long-range goal, the Department is fully committed to an open systems environment, enabling information systems to be developed, operated, and maintained independent of proprietary technical solutions. The TAFIM establishes the direction for an open systems environment that focuses on a standards-based architecture critical to achieving interoperability and cross-functional integration.

New DoD information systems development and modernization programs will conform to the TAFIM, Volume 1 - Implementation Concept, Volume 2 -Architecture Guidance and Design Concepts, Volume 3 -Reference Model and Standards Profile and subsequent forthcoming volumes. The selection and evaluation of migration systems should take into account our long-range goal by striving for conformance to the TAFIM to the extent possible. As stated in my November 12, 1993 memorandum, "Selection of Migration Systems," conformance to the TAFIM is a key technical factor to be considered in the selection of migration systems. In addition, the evolutionary changes to migration systems will be governed by conformance to the TAFIM. Requests for exceptions, with supporting rationale, should be forwarded to this office for consideration.

The TAFIM is maintained by Defense Information systems Agency's (DISA's) Center for Architecture and is available through the National Technical Information Service (NTIS) and the Defense Technical Information Center (DTIC). Attached are the DTIC accession numbers for each document. The TAFIM is an evolving set of documents and comments for improving may be provided to DISA at any time. The action officer for this matter is Mr. Terry Hagle, (703) 604-1486.

Emmett Paige, Jr.

ATTACHMENT

APPENDIX C Policy Memoranda

DTIC ACCESSION NUMBERS FOR TAFIM

DOCUMENT NUMBER	ACCESSION
Implementation Concept, Vol 1	AD-A261 911
Architecture Guidance and Design Concepts, Vol 2	AD-A261 912
Reference Model and Standards Profile, Vol 3	AD-A261 913

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
CHAIRMAN OF THE JOINT CHIEFS OF STAFF
UNDER SECRETARIES OF DEFENSE
COMPTROLLER
ASSISTANT SECRETARY OF DEFENSE (COMMAND, CONTROL, COMMUNICATIONS, AND INTELLIGENCE)
GENERAL COUNSEL
INSPECTOR GENERAL
DIRECTOR OF OPERATIONAL TEST AND EVALUATION
DIRECTORS OF THE DEFENSE AGENCIES
COMMANDER-IN-CHIEF, U.S. SPECIAL OPERATIONS COMMAND

SUBJECT: **Specifications & Standards — A New Way of Doing Business**
DATE: 29 June 1994

To meet future needs, the Department of Defense must increase access to commercial state-of-the-art technology and must facilitate the adoption by its suppliers of business processes characteristic of world class suppliers. In addition, integration of commercial and military development and manufacturing facilitates the development of dual-use processes and products and contributes to an expanded industrial base that is capable of meeting defense needs at lower costs.

I have repeatedly stated that moving to greater use of performance and commercial specifications and standards is one of the most important actions that DoD must take to ensure we are able to meet our military, economic, and policy objectives in the future. Moreover, the Vice President's National Performance Review recommends that agencies avoid government-unique requirements and rely more on the commercial marketplace.

To accomplish this objective, the Deputy Under Secretary of Defense (Acquisition Reform) chartered a Process Action Team to develop a strategy and a specific plan of action to decrease reliance, to the maximum extent practicable, on military specifications and standards. The Process Action Team report, "Blueprint for Change," identifies the tasks necessary to achieve this objective. I wholeheartedly accept the Team's report and approve the report's primary recommendation to use performance and commercial specifications and standards in lieu of military specifications and standards, unless no practical alternative exists to meet the user's needs. I also accept the report of the Industry Review Panel on Specifications and Standards and direct the Under Secretary of Defense (Acquisition and Technology) to appropriately implement the Panel's recommendations.

I direct the addressees to take immediate action to implement the Team's recommendations and assign the Under Secretary of Defense (Acquisition and Technology) overall implementation responsibility. I direct the Under Secretary of Defense (Acquisition and Technology) to immediately arrange for reprogramming the funds needed to FY94 and FY95 to efficiently implement the recommendations. I direct the Secretaries of the Military Departments and the Directors of the Defense Agencies to program funding for FY96 and beyond in accordance with the Defense Planning Guidance.

APPENDIX C Policy Memoranda

Policy Changes

Listed below are a number of the most critical changes to current policy that are needed to implement the Process Action Team's recommendations. These changes are effective immediately. However, it is not my intent to disrupt on-going solicitations or contract negotiations. Therefore, the Component Acquisition Executive (as defined in Part 15 of DoD Instruction 5000.2), or a designee, may waive the implementation of these changes for on-going solicitations or contracts during the next 180 days following the data of this memorandum. The Under Secretary of Defense (Acquisition and Technology) shall implement these policy changes in DoD Instruction 5000.2, the Defense Federal Acquisition Regulation Supplement (DFARS), and any other instructions, manuals, regulations, or policy documents, as appropriate.

Military Specifications and Standards: Performance specifications shall be used when purchasing new systems, major modifications, upgrades to current systems, and nondevelopmental and commercial items, for programs in any acquisition category. If it is not practicable to use a performance specification, a non-government standard shall be used. Since there will be cases when military specifications are needed to define an exact design solution because there is no acceptable non-government standard or because the use of a performance specification or non-government standard is not cost effective, the use of military specifications and standards is authorized as a last resort, with an appropriate waiver.

Waivers for the use of military specifications and standards must be approved by the Milestone Decision Authority (as defined in Part 2 of DoD Instruction 5000.2). In the case of acquisition category 1D programs, waivers may be granted by the Component Acquisition Executive, or a designee. The Director, Naval Nuclear Propulsion shall determine the specifications and standards to be used for naval nuclear propulsion plants in accordance with Pub. L. 98-525 (42 U.S.C. §7158 note). Waivers for reprocurement of items already in the inventory are not required. Waivers may be made on a "class" or item basis for a period of time not to exceed two years.

Innovative Contract Management: The Under Secretary of Defense (Acquisition and Technology) shall develop, within 60 days of the date of this memorandum, Defense Federal Acquisition Regulation Supplement (DFARS) language to encourage contractors to propose non-government standards and industry-wide practices that meet the intent of the military specifications and standards. The Under Secretary will make this language effective 180 days after the date of this memorandum. This language will be developed for inclusion in both requests for proposal and in on-going contracts. These standards and practices shall be considered as alternatives to those military specifications and standards cited in all new contracts expected to have a value of $100,000 or more, and in existing contracts of $500,000 or more having substantial contract effort remaining to be performed.

Pending completion of the language, I encourage the Secretaries of the Military Departments and the Directors of the Defense Agencies to exercise their authority to use solicitation and contract clause language such as the language proposed in the Process Action Team's report. Government contracting officers shall expedite the processing of proposed alternatives to military specifications and standards and are encouraged to use the Value Engineering no-cost settlement method (permitted by FAR 48.104-3) in existing contracts.

Program Use of Specifications and Standards: Use of specifications and standards listed in DoD Instruction 5000.2 is not mandatory for Program Managers. These specifications and standards are tools available to the Program Manager, who shall view them as guidance, as stated in Section 6-Q of DoD Instruction 5000.2

Tiering of Specifications and Standards: During production, those system specifications, subsystem specifications and equipment/product specifications (through and including the first-tier references in the equipment/product specifications) cited in the contract shall be mandatory for use. Lower tier references will be for guidance only, and will not be contractually binding

unless they are directly cited in the contract. Specifications and standards listed on engineering drawings are to be considered as first-tier references. Approval of exceptions to this policy may only be made by the Head of the Department or Agency Standards Improvement Office and the Director, Naval Nuclear Propulsion for specifications and drawings used in nuclear propulsion plants in accordance with Pub. L. 98-525 (42 U.S.C. §7158 Note).

New Directions

Management and Manufacturing Specifications and Standards: Program Managers shall use management and manufacturing specifications and standards for guidance only. The Under Secretary of Defense (Acquisition and Technology) shall develop a plan for canceling these specifications and standards, inactivating them for new designs, transferring the specifications and standards to non-government standards, converting them to performance-based specifications, or justifying their retention as military specifications and standards. The plan shall begin with the ten management and manufacturing standards identified in the Report of the Industry Review Panel on Specifications and Standards and shall require completion of the appropriate action, to the maximum extent practicable, within two years.

Configuration Control: To the extent practicable, the Government should maintain configuration control of the functional and performance requirements only, giving contractors responsibility for the detailed design.

Obsolete Specifications: The "Department of Defense Index of Specifications and Standards" and the "Acquisition Management System and Data Requirements Control List" contain outdated military specifications and standards and data requirements that should not be used for new development efforts. The Under Secretary of Defense (Acquisition and Technology) shall develop a procedure for identifying and removing these obsolete requirements.

Use of Non-Government Standards: I encourage the Under Secretary of Defense (Acquisition and Technology) to form partnerships with industry associations to develop non-government standards for replacement of military standards where practicable. The Under Secretary shall adopt and list in the Department of Defense Index of Specifications and Standards" (DoDISS) non-government standards currently being used by DoD. The Under Secretary shall also establish teams to review the federal supply classes and standardization areas to identify candidates for conversion or replacement.

Reducing Oversight: I direct the Secretaries of the Military Departments and the Directors for the Defense Agencies to reduce direct Government oversight by substituting process controls and non-government standards in place of development and/or production testing and inspection and military-unique quality assurance systems.

Cultural Changes

Challenge Acquisition Requirements: Program Managers and acquisition decision makers at all levels shall challenge requirements because the problem of unique military systems does not begin with the standards. The problem is rooted in the requirements determination phase of the acquisition cycle.

Enhance Pollution Controls: The Secretaries of the Military Departments and the Directors of the Defense Agencies shall establish and execute an aggressive program to identify and reduce or eliminate toxic pollutants procured or generated through the use of specifications and standards.

Education and Training: The Under Secretary of Defense (Acquisition and Technology) shall ensure that training and education programs throughout the Department are revised to incorporate specifications and standards reform.

APPENDIX C Policy Memoranda

Program Reviews: Milestone Decision Authority (MDA) review programs at all levels shall include consideration of the extent streamlining, both in the contract and in the oversight process, is being pursued. The MDA (i.e., the Component Acquisition Executive or his/her designee, for all but ACAT 1D programs) will be responsible for ensuring that progress is being made with respect to programs under his/her cognizance.

Standards Improvement Executives: The Under Secretaries of the Military Departments, and the Director of the Defense Logistics Agency shall appoint Standards Improvement Executives within 30 days. The Standards Improvement Executives shall assume the responsibilities of the current Standardization Executives, support those carrying out acquisition reform, direct implementation of the military specifications and standards reform program, and participate on the Defense Standards Improvement Council. The Defense Standards Improvement Council shall be the primary coordinating body for the specification and standards program within the Department of Defense and shall report directly to the Assistant Secretary of Defense (Economic Security). The Council shall coordinate with the Deputy Under Secretary of Defense (Acquisition Reform) regarding specification and standards reform matters, and shall provide periodic progress reports to the Acquisition Reform Senior Steering Group, who will monitor overall implementation progress.

Management Commitment

This Process Action Team tackled one of the most difficult issues we will face in reforming the acquisition process. I would like to commend the team, composed of representatives from all of the Military Departments and appropriate Defense Agencies, and its leader, Mr. Darold Griffin, for a job well done. In addition, I would like to thank the Army, and in particular, Army Materiel Command, for its administrative support of the team.

The Process Action Team's report and the policies contained in this memorandum are not a total solution to the problems inherent in the use of military specifications and standards; however, they are a solid beginning that will increase the use of performance and commercial specifications and standards. Your leadership and good judgment will be critical to successful implementation of this reform. I encourage you and your leadership teams to be active participants in establishing the environment essential for implementing this cultural change.

This memorandum is intended only to improve the internal management of the Department of Defense and does not create any right or benefit, substantive or procedural, enforceable at law or equity by a party against the Department of Defense or its officers and employees.

William J. Perry

APPENDIX C Policy Memoranda

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
ATTN: SERVICE ACQUISITION EXECUTIVES
VICE CHAIRMAN, JOINT CHIEFS OF STAFF

SUBJECT: **Software Acquisition Best Practices Initiative**
DATE: July 8, 1994

Current DoD software acquisition practices have not proven to provide an effective framework for managing the acquisition of large-scale software development and maintenance programs that are an essential part of our increasingly complex weapons systems. Although many excellent practices for effectively managing such programs exist in both industry and government, their understanding and use within our software acquisition programs is not widespread. The April 6, 1994, memorandum from the Director, Acquisition Program Integration, OUSD(A&T), alerted Service Acquisition Executives to the importance of integrating these practices into their software acquisition processes.

We share these concerns, and together have established the *Software Acquisition Best Practices Initiative* to improve and restructure our software acquisition management process. The purposes of this initiative are to:

- Focus the Defense acquisition community on employing effective, high-leverage software acquisition management practices;
- Enable Program Managers to focus their software management efforts on producing quality software, rather than on activities directed towards satisfying regulations that have grown excessively complex over time;
- Enable Program Managers to exercise flexibility in implementing Best Practices within disparate corporate and program cultures; and,
- Provide Program Managers and staff with the training and tools necessary to effectively use and achieve the benefits of these practices.

This Initiative will identify practices used by successful software projects from both government and industry, and will expand and support the efforts now underway by the Software Program Managers Network to identify and convey these practices. These practices are to be defined as criteria-based practices. The Defense Acquisition Management process, managed under DoD Instructions 5000.2 and 8000, and the associated program review processes will be modified to effectively implement them. Successfully accomplishing this initiative is fundamental to achieving urgently needed improvements to our software acquisition practices.

The Director, Test and Evaluation, OUSD(A&T) and the Deputy Assistant Secretary (C3I Acquisition) are directed to jointly define, implement, and manage this initiative. Please designate a representative having substantial experience in directly managing large-scale Mission-Critical or C3I software development projects to participate in formulating and implementing this initiative. Provide your designee's name, phone, and resume by July 29, 1994, to the Director, Test and Evaluation, OUSD(A&T). Please direct questions to the Network's Coordinator, Norm Brown, at (703) 521-5231 (e-mail: nbrown@nadc.navy.mil).

Noel Longuemare
Under Secretary of Defense
(Acquisition and Technology) (Acting)
Emmett Paige, Jr.
Assistant Secretary of Defense
(Command, Control, Communications, and Intelligence)

APPENDIX C Policy Memoranda

MEMORANDUM FOR
SEE DISTRIBUTION

SUBJECT: **Data Element Standardization in Automated Information Systems Development**
DATE: September 8, 1994

The DoD program to establish standard data elements is central to the goal of improving information system interoperability across the department. It is an essential element in the Joint Staff goals within the C4I for the Warrior initiative and the Air Force "Horizon" strategy. In working toward these goals, the support of the Air Force development community is key to the program's success. The attached document provides policy and direction to the Air Force automated information systems development community, clarifying steps program managers will take to use approved DoD standard data elements in their applications development and acquisition.

Additionally, the document highlights the importance of organizing data according to the classes of data identified in DoD Manual 83201-M-1 as an evolutionary step to creating standard data elements where none yet exist. This policy is effective immediately for all Air Force organizations and will be incorporated in the next revision of AFI 33-110.

Questions with respect to this policy can be directed to HQ USAF/SCTA, Maj Roger VanEpps, DSN 225-1704.

CARL G. O'BERRY, Lt Gen, USAF
DCS/Command, Control Communications, and Computers
Headquarters United States Air Force

LLOYD K MOSEMANN, II
Deputy Assistant Secretary
of the Air Force (Communications,
Computers, and Support Systems)

ATTACHMENT

IMPLEMENTATION OF DATA STANDARDS IN AIS DEVELOPMENT

1. INTRODUCTION

The DoD program to establish standard data elements across the department is essential in achieving interoperability to meet the needs of the warfighter. As a supplement to the DoD 8320.1-M guidance, the Air Force has issued AFI 33-110. This document reiterates key aspects of the Air Force policy to the automated information system (AIS) development community and gives specific guidance on several data elements which are applicable in development and modification efforts already underway.

2. SCOPE

AFI 33-110 establishes roles and responsibilities throughout the Air Force to facilitate development and implementation of the DoD standardized data elements. Although the current pool of DoD-approved standard data elements contains less than 1000 elements, the rate of growth is expected to increase dramatically in the months ahead, and development organizations must begin incorporating these standard data elements into their applications. Program managers will look in the Defense Data Repository System (DDRS) for approved and candidate data elements which will meet their requirements. In cases where no appropriate data element is found in DDRS, program managers will comply with DoD Manual 8320.1-M-1 to use industry standards where available and seek support from AFC4A to develop standard data elements if

necessary. Additionally, the near-term guidance defined below establishes some specific instances of standard data formats to be implemented as an initial effort in standardization. This policy is mandatory for all new AIS development projects and for development or modifications which affect 30 percent or more of the system.

3. IMPLEMENTATION

In order to use standard data elements in new systems development and major systems modifications, developers will first need to understand the overall concepts of the DoD data administration program. To support this, training opportunities are available as outlined below. Developers will also need to have access to the DDRS to determine what data elements are already standardized and to investigate which ones might apply to their applications (see Access information below). In addition to DDRS access, AFC4A/XPSD can provide assistance in performing queries and interpreting the contents of the DDRS.

3.1 Access

Access to the pool of DoD standardization data elements is available via Defense Digital Network (DDN) in the DDRS and is controlled by the DISA/JIEO Data Administration Program Management Office (DAPMO). Requests for access are processed by Ms Patty Rowland, (703) 681-2166.

3.2 Training

The current DoD Data Administration training curriculum is outlined below:

DoD DATA ADMINISTRATION: TECHNIQUES, METHODS, & PRACTICES COURSE	
Introduction	Course Overview
Module 1	DoD Data Administration (role of CIM)
Module 2	Introduction to Activity & Data Modeling
Module 3	Activity
Module 4	Entity Relationship Diagrams
Module 5	Fundamentals of Data Modeling
Module 6	Applied Data Modeling
Module 7	Fully-Attributed Data Models
Module 8	Completing Standardization
Module 9	Model Integration
Module 10	Facilitated Coordination and Approval
Module 11	Database Implementation

Training authorizations are available through the Air Force C4 Agency (contact Ms Ruth Johnston, AFC4A/XPSD, DSN 576-5700).

3.3 Near-Term Guidance

In the next 12 to 18 months, while initiatives are underway to develop the required data models and data element submission packages, program managers will, as a minimum, apply the following format standards as an initial step:

- In specific instances where data elements are being defined for date values, the DoD standard yyyymmdd (yyyy = year, mm = month, dd = day) digital format will be applied. This means that all date values will be stored in a standard format. Applications may use alternate formats to display this data, but all storage and transmission of date data will be as defined here. This will facilitate full standardization at a later time when the standard data element name is selected.

- In instances where data elements are being defined to store latitude and longitude data, the format for latitude will be floating point expressed with a low range of minus 90 (stored as -90.00000000) and a high range of plus 90 (stored as 90.00000000). The format for longitudes will be expressed similarly (stored as -180.00000000 to a high range of 180.0000000).
- In instances where time of day is being expressed in hours, minutes, seconds, and the level of accuracy is to the second, the standard digital format will be hhmmss (hh = hours, mm = minutes, ss = seconds), and values will extend from 000001 to 240000.

In other cases, data should be organized according to DoD 8320.1-M-1 defined classwords. By organizing data bases such that the data types are aligned with these classes of data, developers will be one step closer to compliance with the DoD standard and will be able to more easily relate their data to DoD standardized elements as they are approved.

4. REFERENCES

4.1 DoD Directive 8320.1, "DoD Data Administration," 26 Sep 91.
4.2 DoD Manual 8320.1-M, "Data Administration Procedures Manual," Mar 94.
4.3 DoD Manual 8320.1-M-1, "Data Element Standardization Procedures," Jan 93.
4.4 Air Force Instruction 33-110, "Air Force Data Administration Program," 25 May 94.
4.5 FIPS Pub 184, "Integrated Definition for Information Modeling (IDEF1X)," 21 Dec 93.

SAIS-ADW
MEMORANDUM FOR DISTRIBUTION

SUBJECT: **Preparation for Implementing Army Software Test and Evaluation Panel (STEP) Metrics Recommendations**
DATE: January 4, 1993

1. This memorandum announces the Army's intent to implement the use of STEP metrics. Much time and effort went into STEP with its primary thrust being to improve the readiness of Army software using testing and evaluation (T&E) methods and processes.
2. Testing of all Army software, to include metrics, will be governed by AR 73-1, Test and Evaluation Policy. The Deputy Under Secretary of the Army (Operations Research) supported by the Army's Test and Evaluation Management Agency are the proponents for this new policy. Procedures for implementing T&E policy will be contained in draft DA Pam.73-1, Test and Evaluation Guidelines. Draft DA Pam 73-1 is implemented (20 Oct 92) as an interim operating instruction concurrent with Army-wide staffing. Publication is projected for end of 1st QTR FY 93.
3. The basis for establishing the Army's new metrics policy is contained in DoDD 5000.1 and DoDI 5000.2, both dated 23 Feb 91. These DoD policies stipulate that thorough T&E be conducted to determine system maturity and to identify areas of technical risk. Metrics will be used to effect necessary discipline of the software development process. Currently, weapons systems (identified as materiel systems computer resources (MSCR)) follow procedures outlined in DoDD 5000.1. Those systems identified as automated information systems (AIS) will be acquired IAW DoDD 7920.1 which is currently under revision.
4. Implementation of the new Army software metrics guidelines will apply to all Automated Information Systems (AIS) Classes 1 through 4 for which software is being developed or modified. Implementation of these same requirements will apply to (Acquisition Categories) ACATS I through III materiel or weapons systems for which software is being developed or modified.

5. All programs which have not reached Milestone II approval by the date of this memorandum must comply with the new metrics policy. "*Grandfathering*" from the STEP metrics will include the following considerations:

a. All programs containing software which have obtained Milestone II approval. However, they must input whatever metrics are being used in their already existing format.

b. The milestone decision authority (ASARC/MAISRC) may require the application of a subset of the STEP metrics for systems beyond Milestone II.

c. Waivers from the metrics policy will be submitted to this office and approved by the ASARC/MAISRC decision authority.

6. Inquiries should be forwarded to the following points-of-contact:

a. Software T&E Policy: TEMA — Dr. John Foulkes, coml 695-8995, DSN 225-8995.

b. Software Metrics/Metrics Training:

OPTEC — Mr. Raymond Paul, coml 703-756-1817, DSN 289-1817.

7. ODISC4 point-of-contact is Ms. Adele McCullough-Graham, coml 703-614-2422, or DSN 224-2422.

Walter W. Hollis
Deputy Under Secretary
of the Army (Operations Research)

Peter A. King
Lieutenant General, GS
Director of Information Systems for Command, Control, Communications and Computers

EDITORS NOTE: Following list are the metrics prescribed by the preceding memorandum.

	METRICS	OBJECTIVE
1	SCHEDULE	Track progress vs. schedules
2	COST	Track software expenditures
3	COMPUTER RESOURSE UTILIZATION	Track planned vs. actual size
4	SOFTWARE ENGINEERING ENVIRONMENT	Rate contractor environment
5	DESIGN STABILITY	Track design changes and effects
6	REQUIREMENTS TRACEABILITY	Track requirements to code
7	REQUIREMENTS STABILITY	Track changes to requirements
8	FAULT PROFILES	Track open vs. closed anomalies
9	COMPLEXITY	Assess code quality
10	BREADTH OF TESTING	Track testing of requirements
11	DEPTH OF TESTING	Track testing of code
12	RELIABILITY	Monitor potential downtime due to software

APPENDIX C Policy Memoranda

SUBJECT: **Air Force Software Release Policy**
DATE: July 23, 1993

It is DoD acquisition policy to reuse software to maximum extent possible. To facilitate the reuse process, policy on releasing Air Force software is needed. Currently software is being released by software libraries and software developing agencies. The attached policy and memorandum of agreement provides policy and guidance to releaser and releasee to help further the reuse of software while informing both parties of Air Force policy and hopefully avoiding incidents of inappropriate release. The attached policy is effective immediately.

FOR THE CHIEF OF STAFF
CARL G. O'BERRY, Lt Gen, USAF
DSC/Command, Control,
Communications, and Computers

ATTACHMENTS:
1. Distribution List
2. Software Release Policy

POLICY STATEMENT AND MEMORANDUM OF AGREEMENT FOR THE RELEASE OR DISCLOSURE OF US AIR FORCE-OWNED OR DEVELOPED COMPUTER SOFTWARE

BACKGROUND

The decision to release or disclose software that is owned or has been developed exclusively with Government funds by the United States Air Force (USAF) is under the jurisdiction of the office of primary responsibility (OPR) for the software. The approval authority may be at a higher level depending upon the recipient (e.g., approval authority for foreign release is OSAF/IADD). When not for foreign release, but the OPR is in doubt regarding the release of software, he or she may forward the request to HQ USAF/SCX for resolution.

After deciding to release the software, the OPR shall require the requester to sign this memorandum of agreement. (The only exception is Cooperative Research and Development Agreements, which shall have their own release terms and conditions.) Note: The Government is entitled to royalty-free use of any Air Force-owned software that it releases.

POLICY STATEMENT

1. It is USAF policy to release software that it has developed exclusively with Government funds or otherwise owns. The preferred method is through a software reuse library like the Central Archive for Reusable Defense Software (CARDS) or the Defense Software Repository System (DSRS). USAF-owned software, or that which is developed exclusively with Government funds, can also be provided under the Government Furnished Property provisions of the contract. Note that the Defense Federal Acquisition Regulation Supplement (DFARS) states that when the Government has unlimited rights in computer software in the possession of a contractor, the Government will not pay for the use of such software in performance of Government contracts or for the later delivery to the Government of such computer software, **provided,** however, that the contractor shall be entitled to compensation for converting the software into the prescribed form for reproduction and delivery to the Government. In addition to adhering to the contract provisions, the contractor also must sign this memorandum of agreement.
2. There are other situations when the USAF releases software to an organization with which it does not have a contractual arrangement. In such situations, the recipient must sign the memorandum of agreement.

3. The USAF must ensure that it is not held liable for any failure of the software or maintenance thereof. This policy also applies to USAF software deposited in all software reuse libraries. As such, the OPR must ensure that reuse libraries use this memorandum of agreement, or equivalent, when releasing software donated to the library by the USAF.
4. The decision to release or disclose software shall be based on review of all significant factors including, but not limited to, security, Military Critical Technology, royalty arrangements, potential for Cooperative Research and Development, Depot Business Strategy, or pre-existing license agreements.

MEMORANDUM OF AGREEMENT

1. I/we the undersigned, on behalf of the Requesting Organization listed below (hereafter referred to as the "Requester"), request release of USAF software and understand and agree to the following:

a. NON-DISCLOSURE AGREEMENT. (Applies to commercial components with limited or restricted rights accessed through Government reuse libraries). The Requester requests some or all of the following from _____ (insert the name of the specific Government Reuse Library): data, technical data, computer software, computer software documentation, computer programs, source code, firmware, and other information of like kind, type or quality, either commercial or non-commercial, all of which may be subject to limited rights, restricted rights, Government purpose license rights, patents, copy rights, trade secret rights, or other confidential or proprietary constraints (collectively, the "Data"). In consideration therefore, the Requester agrees:

1) that the Data shall be used only for Government, non-commercial or non-profit purposes;

2) to strictly abide by and adhere to any and all restrictive markings placed on the Data, and the Requester shall not knowingly disclose or release the Data to third parties who are not engaged in work related to Government, non-commercial, or non-profit purposes;

3) that any restrictive markings on the Data shall be included on all copies, modifications, and derivative works, or any parts or portions thereof, in any form, manner or substance, which are produced by the Requester including but not limited to incorporation of the Data into any other data, technical data, computer software, computer software documentation, computer programs, source code, or firmware, or other information of like kind, type or quality. In all such events, Requester shall clearly denote where such Data initiates and concludes by use of annotations or other standard markings.

b. WAIVER OF WARRANTIES AND LIMITATIONS OF DAMAGES AGREEMENT. The requester and the Approving Authority agree that:

1) no guaranties, representations, or warranties either expressed or implied shall be construed to exist in any language, provision, or term contained in these materials or in any other documentation provided herewith (all such items are collectively referred to as the "Agreement"), and furthermore, the releasing organization disclaims and the requester waives and excludes any and all warranties of merchantability and any and all warranties of fitness for any particular purpose;

2) the Requester shall obtain from the releasing organization all of the "Data" (Defined in the Non-Disclosure Agreement above), or any other products or services contemplated by the Agreement, in an "as is" condition;

c. The Requester's use of the Data shall not prevent the Government from releasing the Data at any point in the future.

d. The Requester shall not offer the released Data or any modified version thereof for resale to the Government, in whole or as part or subpart of a Government deliverable, without explicitly stating that he is doing so by providing certification documentation (e.g., Section K of the Government Solicitation) to the contracting officer before contract award.

e. The Requester may use the released Data in a contract with the Government, but understands that the Government shall not pay the Requester for rights of use of such Data in performance of Government contracts or for the later delivery to the Government of such

Data. The Requester may be entitled to compensation for converting, modifying, or enhancing the Data into another form for reproduction and delivery to the Government, if authorized under a contract with the Government.

f. The Requester is not entitled to any released Data that are subject to national defense security classification or the proprietary rights of others. The Requester shall report promptly the discovery of any such restricted Data to the USAF release approving authority below, and will follow all instructions concerning the use, safeguarding, or return of such Data. The Requester shall not copy, or make further study or use of, any released Data later found to be subject to such restrictions.

g. As required, the Requester shall be responsible for compliance with any proscriptions on foreign disclosure of the released Data (contained, for example, in the Department of State International Traffic in Arms Regulations or the Department of Commerce Export Administration Regulations).

h. There may be a fee to cover the copying and shipping of the Data and any documentation.

i. The Requester and the Approving Authority intend that all agreements under this Memorandum of Agreement shall be governed by the laws of the United States of America.

NAME OF REQUESTOR	NAME/TITLE OF USAF APPROVING AUTHORITY
Requesting Organization/Address	Air Force Organization/Address
City, State, Zip Code	City, State, Zip Code
Signature of Requestor and Date	Signature of USAF Approving Authority and Date

23 OCT 93

CSAF WASHINGTON DC//CC//
ALMAJCOM-FOA//CC//
INFO ALMAJCOM-FOA/SC

UNCLAS
SUBJ: C4I SYSTEMS POLICY AND STANDARDS

THE USAF/SC IS THE AIR FORCE C4I SYSTEMS FOCAL POINT WITH RESPONSIBILITY FOR C4I SYSTEMS POLICY, STANDARDS, AND ARCHITECTURE. USAF/SC IS ALSO IMPLEMENTING A STRATEGIC PLANNING PROCESS (SPP) TO GUIDE C4I SYSTEM INTEGRATION AND ENSURE FUTURE DEVELOPMENT AND ACQUISITION EFFORTS ARE CONSISTENT WITH DOD AND JOINT POLICY. THE SPP WILL PROVIDE ARCHITECTURAL VERIFICATION, VALIDATION, AND ACCREDITATION THROUGH A PERMIT SYSTEM SIMILAR TO THE BUILDING CODE/PERMIT/ INSPECTION PROCESS USED BY BUILDING ARCHITECTS. NEW AIR FORCE C4I SYSTEMS POLICY AND STANDARDS DIRECTIVES ARE IN STAFFING. CURRENT AIR FORCE GUIDANCE REMAINS IN EFFECT UNTIL THE DRAFT DIRECTIVES ARE APPROVED.

CC(1) CVAE(1) AFCOS(1) HQ AFFSA(1) PE(3) FM(1) IG(3) SE(1) IN(1)
RE(7) XO(1) LG(1) CVA(1) HQ AFFREQMGTA(3) SC(1) AFISA(1) CE (1) AQ(2)
TE(2) 7CG(2) AFPTC(1) AFSAA(1) AFAA(1) AFPEO(2) AFRBA(1) AFCAA(1)

LTCOL RYAN
SCTA, 51704
CRC: UNCLASSIFIED
041723ZOCT93

Blank page.

APPENDIX

D

Software-Related Government and Industry Documents

Blank page.

APPENDIX D

Software-Related Government and Industry Documents

NOTE: The acquisition environment has undergone several changes in the past year. This appendix is a snapshot in time, and captures those publications current at the time of publication. Some of the documents listed here are in revision, are expected to be superseded by new policy, or will be replaced by commercial guidance. Additionally, some of the commercial standards listed as "draft" are expected to be finalized after the printing of this document. To make sure you know about and have access to the latest standards and policy, either contact the appropriate office listed in Appendix A or look-up the information on-line (e.g., DoDISS, ISO, etc.) at the Web sites listed in Appendix B.

CONTENTS **PAGE**

Table D-1 DoD Directives, Instructions, and Manuals D-2
Table D-2 Military Handbooks D-5
Table D-3 DoD and Military Standards D-6
Table D-4 Air Force Documents D-9
Table D-5 Army Documents D-17
Table D-6 Federal Standards D-17
Table D-7 American National Standards Institute (ANSI) Standards D-19
Table D-8 Institute of Electrical and Electronics Engineers (IEEE) Standards ... D-20
Table D-9 Institute of Electrical and Electronics Engineers (IEEE) Standards in Development D-21
Table D-10 International Standards Organization (ISO) Standards D-22

APPENDIX D Software-Related Documents

DoD Directive (DoDD) 3405.1 ***Computer Programming Language Policy*** (under revision)	• Establishes policy on programming languages in developing and supporting all DoD software • Requires language selection be based on life cycle cost analysis in order of preference: off-the-shelf application packages, Ada, and other approved HOLs • Mandates Ada for: intelligence, command and control, weapon systems, and all other applications unless approved HOL has lower life cycle costs
DoDD 5000.1 ***Defense Acquisition*** ***March 15, 1996***	• States policies and principles for all DoD acquisition programs • Identifies the DoD's key acquisition officials and forums • Establishes a disciplined, yet flexible management approach for acquiring quality products that satisfy the operational user's needs • Replaces DoDD 5000.1, dated 23 February 1991 • Cancels DoDI 5000.2, dated 23 February 1991
DoD 5000.2-R ***Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information Systems (MAIS) Acquisition Programs*** ***March 15, 1996***	• Establishes a simplified and flexible management framework for translating mission needs into stable, affordable, and well-managed MDAPs and MAIS Acquisition Programs • Sets forth mandatory procedures for MDAPs and MAISs, and for other than these (where specifically stated) • Serves as a general model for other than MDAPs and MAISs • Implements DoDD 5000.1 • Addresses the following: - Acquisition management process - Life cycle resource and cost estimates - Integrated Product and Process Development - Systems engineering - Logistics - Test and Evaluation - Defense Acquisition Board (DAB) - Periodic reporting

Table D-1 DoD Directives, Instructions, and Manuals

APPENDIX D Software-Related Documents

DoD 5000.4-M, *Department of Defense Manual Cost Analysis Guidance and Procedures*	• Provides guidance on the scope of the cost analysis, the analytical methods to be used in preparing cost estimates, and the procedures and presentation of the estimates to the cost analysis improvement group • Provides definitions for seven cost terms and provides an understanding as to how they relate to life-cycle cost categories, work breakdown structure elements, and appropriations • Describes the requirements, objectives, uses, and administration of the "*Visibility And Management Of Operating And Support Costs (Vamosc) Program.*"
DoD 5010.12-L *Acquisition Management System and Data Requirements Control List*	• AMSDL contains all source documents and related DIDs cleared for use in defense contracts by OMB
DoD Regulation 5200.1-R *Information Security Program Regulation*	• Establishes DoD policies, standards, criteria, and procedures for the security classification, downgrading, declassification, and safeguarding of information under control of DoD • Applies to the protection of classified information processed, stored, communicated, displayed or disseminated by automated systems
DoDD 5200.28, *Security Requirements for Automated Information Systems*	• Policy for AIS security • Organizational responsibilities • Definition of minimum security requirements • Procedures to determine minimum computer-based AIS security requirements
DoDD 5205.2 *DoD Operations Security Program*	• Establishes the DoD Operations Security (OPSEC) Program • Provides policy and assigns responsibilities • OPSEC program applies to DoD contractors participating in the DoD Industrial Security Program when the DoD component concerned has determined that such measures are essential for the adequate protection of classified information with respect to a specific contract
DoDD 7740.1 *DoD Information Resources Management Program*	• Established the DoD Information Management Program which coordinates and integrates information management functions • Provides policy, procedures, and responsibilities for effective economic acquisition and use of DoD information

Table D-1 DoD Directives, Instructions, and Manuals (cont.)

APPENDIX D Software-Related Documents

DoDD 7740.2 ***Automated Information System Strategic Planning***	• Provides policy and responsibilities for AIS strategic planning • Requires that AIS strategic planning programs must interface with the existing Planning, Programming, and Budgeting System (PPBS)
DoDD 7041.3 ***Economic Analysis and Program Evaluation for Resource Management***	• Policy guidance and framework for consistent application of economic analysis on proposed DoD programs and projects
DoDD 8000.1 ***Defense Information Management (IM) Program***	• Does not apply to computer resources and services associated with a weapon system or basic R&D activities • Does apply to IM resources and services used for routine administrative and business applications and C3I (unless exempted) • Organizational responsibilities
DoDD 8120.1 ***Life-Cycle Management of Automated Information Systems***	• Life cycle management policy • Organizational responsibilities • Procedures for life cycle management
DoDD 8320.1, ***DoD Data Administration***	• Establishes policy for DoD data administration • Provides definitions, concept, responsibilities, and procedures to plan, manage and regulate data within DoD • Authorizes the development of DoD Information Resource Dictionary System (DoDIRDS)
DoDD 8320.1-M-1 ***Data Element Standardization Procedures***	• Describes data administration procedures including: general information on data administration, data element concepts (design, definition, and naming), data element development, the standardization approval process, and data element maintenance

Table D-1 DoD Directives, Instructions, and Manuals (cont.)

APPENDIX D Software-Related Documents

MIL-HDBK-245C ***Preparation of Statement of Work***	• Covers 5 types of SOWs relating to acquisition life cycle phases identified in DoDI 5000.2
MIL-HDBK-347 ***Mission Critical Computer Resources Software Support***	• For use with DoD-STD-2167A • Definitions • Software support concepts - Software support agency (SSA) role and charter • Pre-deployment software support • Post-deployment software support
MIL-HDBK-782 ***Software Support Environment Acquisition***	• Provides common interpretation of requirements, use of DoD-STD-1467, and information needed to use it effectively
MIL-HDBK-805 (OM) ***Microcomputer Software and Hardware Guidelines***	• Updates and includes information about trends, capabilities, and features of 16-bit microcomputer technology. Similar information about 8-bit technology is also retained in the OM version

Table D-2 Military Handbooks

APPENDIX D Software-Related Documents

MIL-STD-100E ***Engineering Drawing Practices***	• Contains practices for preparation of engineering drawings, format, and media delivery • Lists requirements for drawings derived from CAD • Provides definitions and examples of drawing types to be prepared for DoD • Contains numbering, coding, and identification procedures for drawings, associated lists, and documents referenced on the drawings
MIL-STD-498 ***Software Development and Documentation***	• Merges DoD-STD-2167A and DoD-STD-7935A to define activities and documentation suitable for development of both weapon systems and Automated Information Systems • Includes all activities pertaining to software development • Can be applied in any phase of the system life cycle, and can be applied to contractors, subcontractors, or Government in-house agencies performing software development • Supersedes DoD-STD-2167A, DoD-STD-7935A, and DoD-STD-1703(NS)
MIL-STD-499B (draft) ***System Engineering***	• Primary systems engineering functions • Planning and implementation tasks • Role of systems engineering in the development cycle • Test and evaluation requirements • Risk analysis and management • Systems engineering tools and models
MIL-STD-881B ***Work Breakdown Structures for Defense Materiel Items***	• Establishes criteria governing preparation and employment of WBSs for use during the acquisition process • Definitions and requirements • Summary of WBSs and definitions for: - Aircraft systems - Electronics systems - Missile systems - Ordinance systems - Space systems
MIL-STD-882C ***System Safety Program Requirements***	• Provides guidance for developing a formal safety program to be used for all DoD system acquisitions • Purpose is to reduce mishap risks throughout the life cycle of each system • Focuses on in-house and contractor efforts

Table D-3 Military Standards

APPENDIX D Software-Related Documents

MIL-STD-973 ***Configuration Management***	• Consolidates and defines configuration management requirements • Defines CSCI and HWCI requirements • Reviews configuration management administration • Reviews configuration identification, control, status accounting, and audits
DoD-STD-1467 (Army) ***Software Support Environment***	• Establishes requirements for the contractor to define a Developmental Software Support Environment (DSSE) compatible with the existing contracting activity's Life Cycle Software Support Environment (LCSSE) • Provides guidance on the rights in documentation and software to be developed • Defines DSSE content, implementation, and quality assessment requirements and how they relate to the LCSSE • Contract data requirements and applicable DIDs are listed
MIL-STD-1472D ***Human Engineering Design Criteria for Military Systems, Equipment and Facilities***	• Establishes general human engineering criteria for the design and development of military systems, equipment, and facilities
MIL-STD-1750A ***Sixteen-Bit Computer Instruction Set Architecture***	• Defines the instruction set architecture for airborne computers • Purpose is to establish a uniform instruction set architecture for airborne computers which will be used in Air Force avionics weapon systems, to facilitate use and re-use of available support software such as compilers and instruction level simulators
MIL-STD-1801 ***User/Computer Interface***	• Human factors information on computer input/output devices • Defines lexical elements • Describes declarations and types
MIL-STD-1840B ***Automated Interchange of Technical Information Computer Acquisition and Logistics Support***	• Standardizes formats for the exchange of digital information between organizations or systems necessary for the development and logistical support of DoD systems throughout their life cycle
MIL-STD-2002 ***Document Interchange Format***	• Provides a standard set of codes needed by vendors who are installing office automation equipment for DoD activities • Defines the Document Interchange Format (DIF) for use within communications protocols • DIF transmission of data between devices via an appropriate communications methodology permitting transfer of a 7-bit ASCII file in a transparent mode

Table D-3 Military Standards (cont.)

DoD-STD-5200.28 ***DoD Trusted Computer Evaluation Criteria***	• Provides evaluation methodology and criteria for automated systems security
MIL-C-87232 ***Computational Systems, Airborne***	• Guidance on digital computers ruggedized for airborne use
MIL-Q-9858A ***Quality Program Requirements***	• Requires contractors to have a quality assurance program for all areas of contract performance including: design, development, fabrication, processing, assembly, inspections, testing, maintenance, packaging, shipping, storage, and site installation
NCSC-TG 005 ***Trusted Network Interpretation***	• Provides DoD Trusted Computer System Evaluation Criteria (DoD 5200.28-STD) for trusted computer/communications network systems • Describes security services that are used with networks and includes: communications integrity, denial of service, and transmission security

Table D-3 Military Standards (cont.)

APPENDIX D Software-Related Documents

AFPD 10-6 ***Mission Needs and Operational Requirements***	• Provides policies, procedures, and responsibilities for identifying and approving operational requirements which result in research, development, T&E, or procurement appropriations • Establishes policy that Air Force organizations use a formal, coherent process to identify, evaluate, and prioritize the mission needs and operational requirements that compete for limited resources through the Department of Defense (DoD) acquisition system
AFI 10-601 ***Mission Needs and Operational Requirements Guidance and Procedures***	• Implements AFPD 10-6 • Provides guidance and procedures for developing and processing Air Force mission needs and operational requirements that may result in funding with Air Force appropriations • Describes how to prepare, validate, and approve Mission Need Statements (MNS), Operational Requirements Documents (including the Requirements Correlation Matrix) and Cost and Operational Effectiveness Analyses (COEA). • Supersedes AFP 57-2
AFI 10-602 ***Determining Logistics Support and Readiness Requirements***	• Implements AFPD 10-6 • Provides a framework for defining readiness and logistics support requirements throughout the system acquisition or modification process • Attachment 9 Addresses Software Design and Supportability Measures
AFPD 10-11 ***Operations Security***	• Sets Air Force policy • Assigns responsibilities and implements Air Force operations security
AFI 10-1101 ***Operations Security (OPSEC) Instructions***	• Implements AFPD 10-11, DoDD 5205.2, CJCS Instruction 3213.01, *Joint Operations Security*, and all operations security requirements for DoD Instruction 5000.2 • In that OPSEC is one of the critical pillars in C2W strategy, it also directly supports AFPD 10-7, *Command and Control Warfare (C2W)* • Provides the necessary instructions for all Air Force personnel and supporting contractors as they implement the OPSEC concept and maintain OPSEC programs • Describes the OPSEC process, and, for the first time, explores and directs the integration of the OPSEC concept into Air Force plans, operations and support activities

Table D-4 Air Force Documents

APPENDIX D Software-Related Documents

AFI 16-110 ***US Air Force Participation In International Cooperative Research, Development, and Acquisition (ICRD&A)***	• Applies to all Air Force personnel who prepare, manage, review or work with International Cooperative Research, Development, and Acquisition • Chapter 5, *Defense Data Exchange Program*, Paragraph 5.5, addresses software vs. object code and source code
AFPD 33-1 ***Command, Control, Communications, and Computer (C4) Systems***	• Establishes policy for ensuring that C4 systems are acquired, operated, and maintained in accordance with Air Force objectives
AFI 33-101 ***Command, Control, Communications, and Computer Systems Management Guidance and Responsibilities***	• Implements AFPD 33-1 • Provides management procedures for commanders to ensure availability, interoperability, and maintainability of C4 systems • Provides guidance for life cycle management of C4 systems • Refers programs costing $5M or more, involving development, or selected by HQ USAF to AFI 10-601, DoDI 5000.2/AF Sup 1, and the Air Force 63-series publications • Includes guidance superseding that formerly found in AFR 700-9, Vol I (*Information Systems Standardization Program*) and Vol II (*Information Systems Data Element Standardization and Management Program*)
AFI 33-102 ***Command, Control, Communications, Computers, and Intelligence (C4I) Capabilities Planning Process***	• Implements AFPD 33-1 • Establishes management process for C4I capability planning efforts • Provides guidance in applying policy, standards, and resources to processes used to develop and maintain capability planning for C4I systems • Describes objective and process for C4I capability planning • Implements DoDD 8000.1 and DoDD 7740.2 • Includes guidance superseding that formerly found in AFR 700-2 (*Communications-Computer Systems Planning and architecture*)
AFI 33-104 ***Base-Level Planning and Implementation***	• Implements AFPD 33-1 • Provides direction to C4I systems developers • Provides guidance to activities requiring, implementing, and supporting C4 systems • Defines management responsibilities when program acquisition will cost less than $5M • Includes guidance superseding that formerly found in AFR 700-4, Vol II (*Communications-Computer Systems Acquisition and Major Automated Information Systems Review Requirements*)

Table D-4 Air Force Documents (cont.)

APPENDIX D Software-Related Documents

AFI 33-110 ***Air Force Data Administration Program***	• Implements AFPD 33-1 • Provides guidance and procedures to administer the Air Force Data Administration Program • Applies to all Air Force organizations that plan, design, model, synchronize, standardize, and control Air Force data at all echelons • Includes guidance superseding that formerly found in AFR 700-20, Vol I (*Air Force Data Dictionary (On-line)*) and Vol II (S) (*Classified Data elements and Codes (U)*)
AFI 33-114 ***Software Management***	• Implements AFPD 33-1 • Incorporates procedures outlined in Department of Defense (DoD) Directive 5200.28, Security Requirements for Automated Information Systems (AISs), March 21, 1988; and DoD Instruction 5000.2, *Defense Acquisition Management Policies and Procedures*, February 23, 1991 • Identifies responsibilities for managing, developing, maintaining, and implementing Air Force computer software
AFPD 63-1, *Acquisition System*	• Establishes policies for the Air Force acquisition system and ensures acquisition of systems in accordance with public law, appropriate instructions, and international agreements
AFI 63-101 ***Acquisition System***	• Implements AFPD 63-1, DoDD 5000.1, and DoDD 8120.1 • Sets guidelines for implementing the Air Force Acquisition System • Incorporates functional descriptions, information, and procedures formerly in AFR 800-1
AFMCPAM 63-103 ***Software Development Capability Evaluation (SDCE)***	• Provides guidance for planning and conducting an SDCE • Evaluation covers entire software development process: - Systems and software engineering - Management - Quality - Product control - Organizational support - Tools - Facilities - Personnel experience and qualifications • Volume I: provides detailed description of SDCE methodology, including step-by-step explanation of how to perform an SDCE • Volume II: contains support material (e.g., examples, templates, forms, checklists, briefing charts)

Table D-4 Air Force Documents (cont.)

AFM 63-119 ***Certification Of System Readiness For Dedicated Operational Test and Evaluation***	• Provides a structured mechanism for identifying and reducing risks associated with transitioning from developmental test and evaluation (DT&E) to dedicated operational test and evaluation (OT&E) • Attachment 21 addresses software
AFP 63-503 ***Quality Assurance Of Training Systems Contracts***	• Implements AFPD 63-5, *Quality Assurance* • Gives general information and specific guidance concerning Government contract quality assurance (QA) and certain other contract administration functions applicable to training systems contracts • Chapter 4, Section D, addresses software development and support
AFI 91-103 ***Air Force Nuclear Safety Certification Program***	• Defines the process for certifying hardware, software, and procedures used with nuclear weapon systems • Section C, Paragraph 7 addresses software
AFM 91-118 ***Safety Design And Evaluation Criteria For Nuclear Weapon Systems***	• Outlines criteria to evaluate systems, equipment, and software for nuclear safety certification
AFI 99-101 ***Developmental Test And Evaluation***	• Implements AFPD 99-1, *Test and Evaluation Process* • Provides guidance and procedures for the developmental test and evaluation (DT&E) of Air Force systems • Chapter 5, Section E, Paragraph 5.25 addresses software testing
AFI 99-102 ***Operational Test And Evaluation***	• Implements the OT&E policies outlined in Department of Defense (DoD) Directive 5000.1, 23 February 1991; DoD Instruction 5000.2, 23 February 1991; DoD 5000.2-M, 23 February 1991; and AFPD 99-1 • Provides guidance and procedures for operational test and evaluation (OT&E) in the Air Force • Addresses software and its support; includes a typical supplement for software evaluation

Table D-4 Air Force Documents (cont.)

APPENDIX D Software-Related Documents

AFI 99-103 ***Test And Evaluation Process***	• Implements AFPD 99-1, *Test and Evaluation Process* • Directs and describes the Air Force Test and Evaluation Process and its relationship to the systems acquisition process within the policies outlined in Department of Defense (DoD) Directive 5000.1, *Defense Acquisition*, February 23, 1991; DoD Instruction 5000.2, *Defense Acquisition Management Policies and Procedures*, February 23, 1991, with Change 1; and Air Force Supplement 1, *Acquisition Management Policies and Procedures*, August 31, 1993, with Change 1; and DoD 5000.2-M, *Defense* Acquisition Management Documentation and Reports, February 1991, with Change 1
AFM 99-104 ***Armament/Munitions Test Process—Direction And Methodology***	• Implements AFI 99-103, *Air Force Test and Evaluation Process, for Armament/Munitions Test and Evaluation* • Chapter 3, Paragraph 19 addresses software testing
AFP 172-4 ***The Air Force Budget Process***	• Intended to serve as ready reference to budget process • Discusses federal budget system and provides overview of the budget process and major players • Covers laws and rules, constitutional basis for budget process, and forms of Congressional action • Discusses phases of the PPBS cycle including: timetable, process, roles played by JCS and Air Force; budget enactment and execution; Air Force appropriations and funds • 1987 publication data — document is under revision and will be reissued soon as AFPAM 65-606
AFP 700-50, Vol. I ***Air Force Communications Computer Systems Architecture Overview***	• Contains a general overview of goals, attributes, and key concepts for Air Force communications-computer systems architecture • Prescribed by AFR 700-2
AFP 700-50, Vol. II ***Deployable Communications Computer Systems***	• Describes technical architecture for deployable communications-computer systems • Provides standards, interfaces, protocols and planning guidance for developing communications-computer systems planning documents

Table D-4 Air Force Documents (cont.)

AFP 700-50, Vol. IV ***Local Information Transfer***	• Introduces the Integrated Services Digital Network concept of which the Local Information Transfer is one structural block • Current technology and trends for moving large volumes of digital data
AFP 700-50, Vol. V ***Long Haul Information Transfer***	• Describes the Air Force Communications-Computer Systems Long Haul Information
AFP 700-50 Vol. VI ***Integrated Systems Control***	• Transfer architecture designed to support command and control missions through all levels of conflict
AFP 700-50, Vol. VII ***Air Force Communications-Computer Systems Architecture/Software Architecture***	• Provides a framework for software life cycle support, guidelines for using standards for automated support tools for all aspects of software management in systems, data base, and applications software • User interface design and software maintenance methodologies
AFSSI 5010 ***Computer Security in the Acquisition Life Cycle***	• Provides guidance for ensuring computer security (COMPUSEC) is considered and included throughout the life cycle of computer systems • Applies to computer systems developers, purchasers, or program managers who deliver systems to customers
AFSSM 5014 ***Multi-user Security Guide***	• Provides information on security of multi-user computer systems • Provides general guidance and procedures to security managers and users for using the computer system • Applies to all Air Force military and civilian personnel including the US Air Force Reserve, Air National Guard, and Air Force contractors who use, operate, or manage Air Force computer systems and facilities • Implements DoDD 5200.28, DoD Manual 5200.28, DoD Standard 5200.28, DoD Instruction 5215.2, Office of Management and Budget (OMB) Circular A-130, Public Law 100-235, AFR 56-1, and AFSSI 5100
AFSSI 5016 ***Embedded Computer Systems Security***	• Provides general information on embedded computer system security • Gives guidance and direction for individuals and organizations who procure and develop embedded computer systems

Table D-4 Air Force Documents (cont.)

APPENDIX D Software-Related Documents

AFSSI 5017 ***Deployable Computer Systems Security***	• Provides Air Force security policy for deployable and mobile computer systems • Implements AFR 56-1, *Command, Control, Communications, and Computer (C4) Systems Security Policy*, and AFSSI 5100
AFSSM 5018 ***Risk Analysis***	• Summarizes Air Force guidance and outlines procedures for conducting a risk analysis for operational Air Force systems • Guide providing advice and information useful for performing a risk analysis • Implements AFR 56-1, and AFSSI 5100
AFSSM 5022 ***Network Risk Analysis Guide***	• Summarizes Air Force guidance and outlines procedures for conducting a network risk analysis • Complements Air Force System Security Memorandum, (AFSSM) 5018 which should be used when nodes in the network need to be accredited • Applies to all Air Force military and civilian personnel including the US Air Force Reserve, Air National Guard when published in NGR(AF) 0-2, and Air Force contractors who use, operate, or manage Air Force computer systems and facilities • Implements DoDD 5200.28, DoD Manual 5200.28, DoD Standard 5200.28, DoDI 5215.2, Office of Management and Budget (OMB) Circular A-130, Public Law 100-235, AFR 56-1, and AFSSI 5100
AFSSI 5100 ***The Air Force Computer Security (COMPUSEC) Program***	• Prescribes the requirements for the Air Force COMPUSEC program • Implements DoD Directive 5200.28, DoD Instruction 5215.2, Office of Budget and Management (OMB) Circular A-130; Public Law 100-235; and AFR 56-1 • Directs reader to various Air Force Systems Security Instructions (AFSSI) and Air Force Systems Security Memorandums (AFSSM) for detailed procedures of the various computer security areas • Supersedes AFR 205-16
AFSSI 5101 ***Computer Security in the Air Force Acquisition System***	• Outlines computer security (COMPUSEC) policy and requirements for Air Force computer systems under development or in the acquisition process • Applies to computer system developers, acquiring activities, and program managers who deliver systems to customers

Table D-4 Air Force Documents (cont.)

AFSSI 5102 ***Computer Security (COMPUSEC) for Operational Systems***	• Provides COMPUSEC policy and requirements for all automated information systems (AIS) in the operations and support phase of their life cycle • Implements Air Force Regulation (AFR) 56-1 and AFSSI 5100
Planning, Programming, and Budgeting system (PPBS) Primer [HQ USAF/PEI]	• Describes the current DoD and Air Force Planning, Programming, and Budgeting System (PPBS) as well as the organization, responsibilities, and general procedures by which they function, with the objective of generating a better understanding of the overall PPBS process • Quick reference tool supplementing various OSD and USAF directives and instructions covering the system • Explains how the Air Force conducts business using the Resource Allocation Process, supports the field commanders, and interfaces with OSD*SAF/AQ Program Element Monitor/Action Officer Handbook,* developed by ANSER for use by SAF/AQ personnel • Describes PEM/Action Officer responsibilities • Provides useful background information on organizational relationships [DoD, JCS, SAF, Air Staff, PEOs, Congress (SASC, HASC, SAC, and HAC)] • Provides overview of acquisition processes including AFSARC, AFCAIG, and DAB • Discusses Air Force Budget process [including Air Force & Financial Plan (F&FP); POM; BES; enactment and apportionment; money management; reprogramming; and IWSM[• Provides guidance for developing selected documentation

Table D-4 Air Force Documents (cont.)

APPENDIX D Software-Related Documents

AMCP 70-13 ***Software Management Indicators***	OPR: HQ CECOM Attn: AMSEL-RD-SE-AST-SE Fort Monmouth, NJ 07703 DSN 992-2117
AMCP 70-14 ***Software Quality Indicators***	OPR: HQ CECOM Attn: AMSEL-RD-SE-AST-SE Fort Monmouth, NJ 07703 DSN 992-2117

Table D-5 Army Documents

FIPS 46-2 ***Data Encryption Standard (DES)***	• The DES specifies an algorithm to be implemented in electronic hardware devices and used for the cryptographic protection of sensitive but unclassified computer data
FIPS 48 ***Evaluation of Techniques for Automated Personal Identification***	• Describes methods for verifying the identity of users seeking to gain access to computer systems or networks via terminals
FIPS 57 ***Measurement of Interactive Computer Service Response Time and Turnaround Time***	• Defines measures and methodologies for measuring interactive computer network service
FIPS 72 ***Measurement of Remote Batch Computer Service***	• Defines measures and describes methodologies for measuring remote batch computer network service
FIPS 73 ***Security of Computer Applications***	• Describes methods and techniques to reduce the hazards associated with computer applications • Describes different security objectives for a computer application • Explains control measures • Identifies critical decisions at each stage in the life cycle of a sensitive computer application
FIPS 76 ***Planning and Using a Data Dictionary System***	• Capabilities of a data dictionary system • Selection considerations • Guidance on pre-implementation planning, implementation, and operational use
FIPS 81 ***DES Mode of Operation***	• Supplements FIPS 46 in defining 4 modes of operation specifying how data will be encrypted and decrypted • Guidance on: - Electronic code book mode - Cipher block chaining mode - Cipher feedback mode - Output feedback mode
FIPS 83 ***User Authentication Techniques for Computer Network Access Control***	• Guidance on selection and implementation of techniques for authenticating the users of remote terminals to safeguard against unauthorized access to computers and computer networks

Table D-6 Federal Standards

FIPS 101 ***Life-Cycle Validation, Verification, and Testing of Computer Software***	• Presents an integrated approach to validation, verification, and testing for use throughout the software life cycle
FIPS 102 ***Computer Security Certification and Accreditation***	• Describes how to establish and carry out a certification (i.e., the technical evaluation of a sensitive application to determine if it meets security requirements) and accreditation (i.e., the official management authorization for operation of the application) program for computer security
FIPS 146-1 ***Government Open Systems Interconnection Profile (GOSIP)***	• Adopts GOSIP as the common set of data communication protocols which enable systems developed by different vendors to interoperate and enable users of different applications on these systems to exchange information

Table D-6 Federal Standards (cont.)

APPENDIX D Software-Related Documents

NOTE: The following table contains ANSI standards taken from a search of ANSI's home page.

ANSI/IEEE 1044-1994	Classification for Software Anomalies
ANSI/NISO Z39.67-1993	Computer Software Description
ANSI/IEEE 1074-1992	Developing Software Life Cycle Processes
ANSI/ANS 10.3-1995	Documentation of Computer Software
ANSI/AIAA G-009-1991	Guide for Implementing Software Development Files Conforming to DoD-Std-2167A
ANSI/IEEE 982.2-1988	Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software
ANSI/IEEE 1042-1987	Guide to Software Configuration Management
ANSI/IEEE 1016.1-1993	Guide to Software Design Descriptions
ANSI/IEEE 1062-1994	Recommended Practice for Software Acquisition
ANSI/IEEE 830-1993	Recommended Practice for Software Requirements Specifications
ANSI/IEEE 828-1990	Software Configuration Management Plans
ANSI/IEEE 1016-1987	Software Design Descriptions, Recommended Practice for
ANSI/IEEE 1002-1987 (R1993)	Software Engineering Standards, Standard Taxonomy for
ANSI/IEEE 1219-1993	Software Maintenance
ANSI/IEEE 1045-1993	Software Productivity Metrics
ANSI/IEEE 1058.1-1987	Software Project Management Plans
ANSI/IEEE 983-1986	Software Quality Assurance Planning, Guide for
ANSI/IEEE 730.1-1989	Software Quality Assurance Plans
ANSI/IEEE 1061-1993	Software Quality Metrics Methodology
ANSI/AIAA R-013-1992	Software Reliability
ANSI/IEEE 1028-1988	Software Reviews and Audits
ANSI/IEEE 829-1983 (R1991)	Software Test Documentation
ANSI/IEEE 1008-1987	Software Unit Testing
ANSI/IEEE 1063-1989	Software User Documentation
ANSI/IEEE 1012-1987 (R1993)	Software Verification and Validation Plans
ANSI/IEEE 982.1-1988	Standard Dictionary of Measures to Produce Reliable Software
ANSI/IEEE 1228-1994	Standard for Software Safety Plans
ANSI/IEEE 610.12-1990	Standard Glossary of Software Engineering Terminology (Revision and Redesignation of ANSI/IEEE 729-1983)

Table D-7 American National Standards Institute (ANSI) Standards

APPENDIX D Software-Related Documents

NOTE: The following two tables contain IEEE standards taken from a search of IEEE's home page on the Web.

730-1989	IEEE Standard for Software Quality Assurance Plans
828-1990	IEEE Standard for Software Configuration Management Plans (1-55937-064-5)
829-1983	IEEE Standard for Software Test Documentation (Reaff 1991)
830-1993	IEEE Recommended Practice for Software Requirements Specifications (ISBN 1-55937-395-4)
982.1-1988	IEEE Standard Dictionary of Measures to Produce Reliable Software
982.2-1988	IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software . This standard is the companion document to IEEE Std 982.1-1988.
990-1987	IEEE Recommended Practice for Ada as a Program Design Language (Reaff 1992)
1002-1987	IEEE Standard Taxonomy for Software Engineering Standards (Reaff 1992)
1008-1987	IEEE Standard for Software Unit Testing (Reaff 1993)
1012-1986	IEEE Standard for Software Verification and Validation Plans (Reaff 1992)
1016-1987	IEEE Recommended Practice for Software Design Descriptions (Reaff 1993)
1016.1-1993	IEEE Guide to Software Design Descriptions (1-55937-297-4)
1028-1988	IEEE Standard for Software Reviews and Audits
1042-1987	IEEE Guide to Software Configuration Management (Reaff 1993)
1044-1993	IEEE Standard Classification for Software Anomalies (1-55937-383-0)
1045-1992	IEEE Standard for Software Productivity Metrics (1-55937-258-3)
1058.1-1987	IEEE Standard for Software Project Management Plans (Reaff 1993)
1059-1993	IEEE Guide for Software Verification and Validation Plans (1-55937-384-9)
1061-1992	IEEE Standard for a Software Quality Metrics Methodology (1-55937-277-X)
1062-1993	IEEE Recommended Practice for Software Acquisition (1-55937-385-7)
1063-1987	IEEE Standard for Software User Documentation (Reaff 1993)
1074-1991	IEEE Standard for Developing Software Life Cycle Processes (1-55937-170-6)
1209-1992	IEEE Recommended Practice for the Evaluation and Selection of CASE Tools (1-55937-278-8)
1219-1992	IEEE Standard for Software Maintenance (1-55937-279-6)
1228-1994	IEEE Standard for Software Safety Plans (1-55937-425-X)
1220-1994	IEEE Trial-Use Standard for Application and Management of the Systems Engineering Process (1-55937-496-9)
1298-1992	(AS 3563.1-1991) Software Quality Management System, Part 1: Requirements (1-55937-221-4)
610-1990	IEEE Standard Computer Dictionary, Compilation of IEEE Standard Computer Glossaries (1-55937-079-3)

Table D-8 Institute of Electrical and Electronics Engineers (IEEE) Standards

610.2-1987	IEEE Standard Glossary of Computer Applications Terminology
610.3-1989	IEEE Standard Glossary of Modeling and Simulation Terminology
610.4-1990	IEEE Standard Glossary of Image Processing and Pattern Recognition Terminology
610.5-1990	IEEE Standard Glossary of Data Management Terminology (1-55937-046-7)
610.6-1991	IEEE Standard Glossary of Computer Graphics Terminology (1-55937-186-2)
610.7-1995	IEEE Standard Glossary of Computer Networking Terminology (1-55937-498-5)
610.12-1990	IEEE Standard Glossary of Software Engineering Terminology (1-55937-067-X)
610.12A-1990	IEEE Standard Glossary of Software Engineering Terminology-ASCII Version, 3.5" diskette with softbound text (1-55937-084-X)
610.12B-1990	IEEE Standard Glossary of Software Engineering Terminology-ASCII Version, two 5.25" diskettes with softbound text (1-55937-085-8)
610.12H-1990	IEEE Standard Glossary of Software Engineering Terminology-HyperCard(TM) Stack, diskette only (HyperCard v. 1.2 and up) (1-55937-083-1)
610.13-1993	IEEE Standard Glossary of Computer Languages (1-55937-296-6)
754-1985	IEEE Standard for Binary Floating-Point Arithmetic (Reaff 1990)
1175-1991	IEEE Trial-Use Standard Reference Model for Computing System Tool Interconnections (1-55937-197-8)
1278-1993	IEEE Standard for Information Technology-Protocols for Distributed Interactive Simulation Applications, Entity Information and Interaction (1-55937-305-9)
1295-1993	IEEE Standard for Information Technology-XWindow System-Modular Toolkit Environment (1-55937-387-3)

Table D-8 Institute of Electrical and Electronics Engineers (IEEE) Standards (cont.)

730.2	Apr-95, D5 Guide for Software Quality Assurance Planning
1044.1	Sept-95, D3 Classification for Software Anomalies
1074	D1.0 Developing Software Life Cycle Processes
1074.1	July-95, D2.2 Guide for Developing Software Life Cycle Processes
1226.3	Dec-95, D6.0 Broad Based Environment for Test (ABBET) - Software Interface for Resource Management
1226.4	Dec-95, D2.0 Broad Based Environment for Test (ABBET) - Software Interface for Instrument Drivers
1387.2	Apr-95, D14a POSIX-System Administrator Part 2-System Software Administration
1420.2	1995, D2.1 Software Reuse - Data Model for Reuse Library Interoperability (BIDM)
1498	July-95, D3 Information Technology Software Life Cycle Processes

Table D-9 Institute of Electrical and Electronics Engineers (IEEE) Standards in Development

APPENDIX D Software-Related Documents

NOTE: The following table contains selected International Standards Organization (ISO) standards taken from a search of ISO's home page on the Web.

ISO/IEC TR 10034:1990	Guidelines for the preparation of conformity clauses in programming language standards
ISO 9127:1988	Information Processing Systems — User Documentation and Cover Information for Consumer Software Packages
ISO/IEC DIS 14834	Information Technology — Distributed Transaction Processing — The XA Specification
ISO/IEC DTR 11735	Information Technology — Extensions for Real-time Ada
ISO/IEC 14102:1995	Information Technology — Guideline for the Evaluation and Selection of CASE Tools
ISO/IEC TR 9294:1990	Information Technology — Guidelines for the Management of Software Documentation
ISO/IEC DTR 14399	Information Technology — Mapping of Relevant Software Engineering Standards — Standards Relevant to ISO/IEC JTC 1/SC 7 — Software Engineering
ISO/IEC DIS 10164-18	Information Technology — Open Systems Interconnection — Systems Management: Software Management Function
ISO/IEC 13719-1:1995	Information Technology — Portable Common Tool Environment (PCTE) — Part 1: Abstract Specification
ISO/IEC 13719-2:1995	Information Technology — Portable Common Tool Environment (PCTE) — Part 2: C programming language binding
ISO/IEC 13719-3:1995	Information Technology — Portable Common Tool Environment (PCTE) — Part 3: Ada Programming Language Binding
ISO/IEC 9945-1:1990	Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language]
ISO/IEC 9945-2:1993	Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities
ISO/IEC DIS 9945-1	Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (Revision of ISO 9945-1:1990)
ISO/IEC 8652:1995	Information Technology — Programming Languages — Ada
ISO/IEC 9899:1990	Information Technology — Programming Languages — C
ISO/IEC 11730:1994	Information Technology — Programming Languages — Form Interface Management System (FIMS)
ISO/IEC 11430:19	Information Technology — Programming Languages — Generic Package of Elementary Functions for Ada
ISO/IEC 11729:1994	Information Technology — Programming Languages — Generic Package of Primitive Functions for Ada
ISO/IEC 13211-1:1995	Information Technology — Programming Languages — Prolog — Part 1: General Core
ISO/IEC 12227:1995	Information Technology — Programming Languages — SQL/Ada Module Description Language (SAMeDL)

Table D-10 International Standards Organization (ISO) Standards

APPENDIX D Software-Related Documents

ISO/IEC DIS 14515-1	Information Technology — Programming Languages, Their Environments and System Software Interfaces — Portable Operating System Interface (POSIX) — Test Methods for Measuring Compliance to POSIX — Part 1: System Interfaces
ISO/IEC TR 10182:1993	Information Technology — Programming Languages, Their Environments and System Software Interfaces — Guidelines for Language Bindings
ISO/IEC 12207:1995	Information Technology — Software Life Cycle Processes
ISO/IEC DIS 14143-1	Information Technology — Software Measurement — Part 1: Definition of Functional Size Measurement
ISO/IEC 12119:1994	Information Technology — Software Packages — Quality Requirements and Testing
ISO/IEC 9126:1991	Information Technology — Software Product Evaluation — Quality Characteristics and Guidelines for Their Use
ISO/TR 9547:1988	Programming Language Processors — Test Methods — Guidelines for Their Development and Acceptability
ISO 9000-3:1991	Quality Management and Quality Assurance Standards — Part 3: Guidelines for the Application of ISO 9001 to the Development Supply and Maintenance of Software

Table D-10 International Standards Organization (ISO) Standards (cont.)

Blank page.

APPENDIX

E

Selected Technical References

Blank page.

APPENDIX E

Selected Technical References

CONTENTS **PAGE**

Tab 1:
Alphebatized Listing and Synopsis of Selected Technical Reports E-1
Tab 2:
Comprehensive Approach to Reusable Defense Software (CARDS) E-11
Tab 3:
GSA Information Resource Management Publications E-16

TAB 1

NOTE: The World-Wide Web has information available at your fingertips, and can provide more listings of technical reports than we could in the limited space here. To find and access reports not listed here, browse the Web sites listed in Appendix B.

Ada83/Ada9X Compatability Guide, Version 6

Aimed at alerting projects currently writing Ada applications where enhancement or maintenance is required beyond 1997 of any incompatibilities between Ada 83 and Ada 95.

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

APPENDIX E Selected Technical References

Ada 95 Adoption Handbook

A comprehensive guide to aid Program Executive Officers and Program Managers understand and implement the transition to Ada 95.

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

Ada 95 Quality and Style: Guidelines for Professional Programmers, Version 1.00.10

October 1995

Prepared by the Software Productivity Consortium. Available through the Ada Information Clearinghouse either electronically or by mail.

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

AdaIC Available Bindings Report

An authoritative reference that describes the status of the major standards and bindings available to Ada programmers, provides a list of relevant reusable resources, and lists vendors supporting commercial implementations. Available electronically through the AdaIC home page or by mail.

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

Ada Implementation Guide (Navy), Vols I and II

March 1992

Naval Information Systems Management Center
Space and Naval Warfare Systems Command
(703) 602-6903

APPENDIX E Selected Technical References

Army Software Test and Evaluation Panel (STEP) Software Metrics Initiatives

Betz, Henry P. and P.J., O'Neill, May 1992

Report documents findings and recommendations of STEP on a set of software metrics that aid program managers in the evaluation of software functionality, maturity, and readiness to proceed to the next development phase, and during test and evaluation. Discusses management metrics, quality metrics, and implementing guidelines. Provides recommendations and concerns on the application of report findings.

DISC4
Pentagon
Washington, DC 20330
(703) 697-6001
DSN 227-6001

CMU/SEI-92-TR-11 *Software Measurement Concepts for Acquisition Managers*

January 1992

Provides basic concept program managers can use to integrate measurement into the process for managing software development. Offers initial measures to help resolve issues that arise in software intensive acquisitions. Contains information on measures data definition and collection, software measures for software development issues and sample techniques on: trend analysis, multiple metric relationship analysis, modeling input data analysis, and data interpretation warnings.

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

CMU/SEI-92-TR-19, *Software Measurement for DoD Systems: Recommendations for Initial Core Measures*

September 1992

Presents recommendations for a set of basic software measures to help plan and manage acquisition, development, and support software systems. Reviews integrating measurement with software process and recommends use of core measures: size (SLOC), effort (staff hours), and quality (counting problems and defects). Provides basic measures implementation guidance. *[Does not address "rework" as a core measure.]*

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

APPENDIX E Selected Technical References

CMU/SEI-92-TR-21, *Software Effort and Schedule Measurement: A Framework for Counting Staff Hours and Reporting Schedule Information*

September 1992

Provides guidance for defining, recording and reporting staff-hours (S-H). Addresses dates concerned with project milestones and contract deliverables and measures of project progress. Discusses how S-H measures can meet the needs of a variety of users and contains recommendations on project applications.

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

CMU/SEI-92-TR-22, *Software Quality Measurement: A Framework for Counting Problems and Defects*

September 1992

Presents mechanisms for describing and specifying software problems and software defect measures. Explains why problems and defects should be measured and their affects on project software quality, cost, and schedule. Checklists, supporting forms, and measurement results are provided. Includes recommendations for application in ongoing, new, and expanding software projects.

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

CMU/SEI-92-TR-29, *Ada Adoption Handbook: A Program Manager's Guide*

Version 2.0, October 1992

Provides guidance on adopting the Ada programming language to include what practices have worked and what pitfalls to avoid.

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

CMU/SEI-93-TR-24, *Capability Maturity Model for Software*

February 1993

Provides an overview of the SEI developed 5-level Capability Maturity Model (CMM) for software process improvement. Describes the CMM process maturity architecture, how CMM is used in practice, and how CMM can be used in the future. Companion report to CMU/SEI-93-TR-25, *Key Practices of the Capability Maturity Model*, February 1993 (below).

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

APPENDIX E Selected Technical References

CMU/SEI-93-TR-25, *Key Practices of the Capability Maturity Model*

February 1993

Companion report to CMU/SEI-93-TR-24 (above). Provides overview of CMM, a description of how to use and interpret key practices associated with the model, and information on how to use the format of the key practices. Key process areas are: requirement management, software project planning, software project tracking and oversight, software subcontract management, software quality assurance, and software configuration management.

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

CMU/SEI-93-TR-23/NIST, Special Publication 500-213, *Reference Model for Project Support Environments*

Version 2.0, 1993

Contains a comprehensive list of tool capabilities to look for in a SEE developed by the Navy's Next Generation Computer Resources (NGCR) Program.

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

CMU/SEI-94-SREv0.2, *Software Risk Evaluation Method*, Version 0.2,

January 1994

Contains a high-level description of the current version of the Software Risk Evaluation (SRE) method. The SRE is as method to identify, analyze, communicate, and mitigate software technical risks.

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

CMU/SEI-95-SR-004, *A Manager's Checklist for Validating Software Cost and Schedule Estimates*

1995

This report provides a checklist of questions to ask and evidence to look for when assessing the credibility of a software cost and schedule estimate. The checklist can be used either to review individual estimates or to motivate and guide organizations toward improving their software estimating processes and practices.

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

APPENDIX E Selected Technical References

CMU/SEI-95-SR-005, *Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations*

1995

This report provides criteria and checklists for evaluating the capability of an organization's software estimating process and the infrastructure that supports it. It also supplies guidelines for good estimating practice. The checklists and guidelines can be used to elicit information for process assessments and to motivate and guide organizations in process improvement efforts.

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

ISO IEC 8652-9X, *Organization for International Standards (ISO)*, Ada Information Clearinghouse

Reference manual for the Ada programming language.

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

Rationale for ISO IEC 8652-9X, the Ada Programming Language

AdaIC
PO Box 46593
Washington, DC 20050-6593
(703) 685-1477
(800) AdaIC-11

RL-TR-92-315, *An Approach to Software Quality Prediction From Ada Designs*

December 1992

Results from and ongoing research project on estimating quality from Ada designs. This research analyzed designs when first represented for their effect on software quality factors relating to reliability and maintainability

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

APPENDIX E Selected Technical References

RL-TR-92-316, Vol. I and Vol II, *A Guide to Total Software Quality Control*

December 1992

Provides techniques and strategies for controlling the quality of software for mission critical systems. Describes techniques to prevent, detect, and correct defects and to improve software processes and resources.

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

RL-TR-92-317, *Procedures for Applying Ada Quality Prediction Models*

December 1992

Describes Ada software quality prediction model applications. Models predict metrics related to software reliability, maintainability, and flexibility. Procedures include use of Ada source code analysis tool and statistical analysis system to abstract data from Ada code and create data sets with numerics needed for the models.

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

RL-TR-92-318, *Estimating Ada System Size During Development*

December 1992

Contains four models for estimating the size of Ada systems at intermediate milestones in a software development project.

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

RL-TR-93-80, *Conflict Resolution (CORE) for Software Quality Factors*

May 1993

Provides a prototype tool for the support of the Rome Laboratory Software Quality Methodology. The tool maximizes quality goals within the cost, schedule and technical areas of software development. Conflicting characteristics are addressed and include: efficiency, expandability, reliability, safety, and availability.

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

APPENDIX E Selected Technical References

RL-TR-94-146, *Rome Laboratory Software Quality Framework Implementation Procedure Guidebook*

August 1994

Provides a step-by-step implementation of the Rome Laboratory Software Quality Framework: a 3-tiered approach which associates a set of user-oriented software quality characteristics *(quality factors)* with their defining software quality attributes *(criterion)* and these attributes, in turn, with measurable software development activities *(elements)* at the lowest level. Contains a set of suggested data collection formats, procedures for beginning a software quality metric program, and methods for selecting and training the measurement team.

RL/C3CB
525 Brooks Road
Rome, NY 13441-4505
(315) 330-4063

Streamlined Integrated Software Metrics Approach (SISMA) Guidebook: Application of STEP Metrics

July 12, 1993

Software Engineering Directorate
U.S. Army Communications-Electronics Command
Research, Development, and Engineering Center
Ft. Monmouth, NJ 07703-5000

STSC, *Project Management and Software Cost Estimation*

April 1995

This report increases awareness and understanding of project management technologies, and provides the first step in transferring effective project management principles, methods and products into practical use. The Software Estimation and Technology Report has been updated, and included as Appendix A. This appendix defines concepts of the software estimation tools domain and identifies their value in improving software quality and productivity. It also explains the capabilities of current project management products available in the marketplace and provides cost, vendor, and acquisition data.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

APPENDIX E Selected Technical References

STSC, *Requirements Engineering and Design Technology Report*

October 1995

Defines upper CASE (UC) products and discusses their value in improving software quality. Explains how features of current UC tools can improve software development and maintenance. Provides information on tool products available in the marketplace along with cost, vendor, and acquisition data.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

STSC, *Process Technologies Method and Tool Report*, Volume I

March 1994

Defines process technologies, identifies tools and software engineering environments that support process technologies, discusses the value of emphasizing process in improving software quality, and examines the effective use of process technologies.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

STSC, *Documentation Technology Report*

April 1994

Presents ideas on the documentation tools domain and discusses their value in improving software quality. Explains the features of current documentation tools products available in the marketplace. Includes data on tools, cost, vendor, and acquisition data.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

APPENDIX E Selected Technical References

STSC, *Software Engineering Environment Technology Report*

April 1994

Defines the concepts of the software engineering environment (SEE) domain and discusses their value in improving software productivity and quality. Explains how features of the current SEE technology can improve software development and maintenance. Includes information on the tools available in the marketplace along with cost, vendor, and acquisition data.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

STSC, *Software Test Technologies Report*

August 1994

Defines the concepts of software testing and identifies their value in improving software quality. Explains how features of current testing tools can improve software development and maintenance. Contains information about current testing tools available in the market place and data on cost, vendors, and acquisition of the tools.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

STSC, *Re-engineering Tools Report*, Volume I

August 1994

Defines the concepts of software re-engineering and discusses their value in improving software quality. Explains how features of current re-engineering tools can improve software development and maintenance. Provides information about tools products available in the marketplaces including cost, vendor, and acquisition data.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

APPENDIX E Selected Technical References

STSC, *Cleanroom Pamphlet*

April 1995

Cleanroom software engineering is a theory-based, team oriented process for on-schedule development and certification of ultra-high-reliability software systems with improved productivity under statistical quality control. This pamphlet is a collection of information on the cleanroom process, including a list of organizations that provide cleanroom services.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 775-5555, x-3024
DSN 775-5555, x-3024

TAB 2

Comprehensive Approach to Reusable Defense Software (CARDS)

The CARDS Program has produced a significant collection of documents detailing its research efforts and experiences in the area of domain-specific software reuse. Some CARDS documents detail the CARDS experience in developing and supporting a domain-specific reuse library. The libraries within CARDS are used as testbeds for the development of like capabilities throughout the Air Force. These documents provide information to organizations interested in implementing their own domain-specific library. Copies of the CARDS documents can be obtained by calling the CARDS Hotline at (800) 828-8161.

1. The **Acquisition Handbook** is aimed towards government program managers and their support personnel, such as contracting officers and administrators, involved in systems, subsystems, and component acquisition and maintenance. The concepts discussed assume the reader has at least three years experience in acquisition. The handbook assists readers in incorporating software reuse into all phases of the acquisition life cycle. It is meant to help develop and tailor reuse programs. Software reuse methods, examples, recommendations, and techniques are presented to implement various reuse strategies throughout the acquisition life cycle. Implications and affects of software reuse on technical, management, cost, schedule, and risk aspects of a program/system during the acquisition process are the foundation of this handbook.

2. The **CARDS Program and Library Course** introduces individuals and organizations to CARDS and the CARDS Command Center Library (CCL). It explains the major goals of CARDS, its products and services, and provides detailed guidance on how to utilize the CARDS CCL. It is assumed the student has a solid understanding of basic reuse concepts, but little or no knowledge of CARDS.

3. The **Command Center Supported Components Report** provides a set of guidelines, initially developed for CCLs, to assist library domain engineers and/or qualification teams in providing information on domain-specific qualified library components. These guidelines consist of templates describing a component from high level technical viewpoints. Examples are provided.

4. The **Component Provider's and Tool Developer's Handbook** provides government software developers and industry vendors (e.g., government contractors and commercial software creators) with guidance for developing domain-specific reusable components and tools supporting software reuse. The goal is to stimulate the development and commercialization of large-scale reusable components and tools for vertical domains, and to complement the CARDS Engineer's Handbook. The end users of these components and tools are system developers on government programs. The main audience is government domain engineers, and component and tool creators (either government or contractors). SPO engineers should be familiar with this handbook.

5. The **Direction Level Handbook** is directed towards all service acquisition executives to facilitate the institutionalization of software reuse. The audience of Program Executive Officers (PEOs), Designated Acquisition Commanders (DACs), and their supporting staff are provided with a framework to assist in establishing plans to manage reuse across their systems and to reach the DoD Software Reuse Vision and Strategy goals. This handbook assist in incorporating software reuse into the initial planning stages of an acquisition, as well as at critical points within the acquisition life cycle. Options are provided to allow the PEOs, etc., to gain the greatest benefits from software reuse while optimizing the use of shrinking resources.

6. The **Engineer's Handbook** provides guidance to government System Program Office (SPO) engineers on envisioned changes to their duties and responsibilities as domain-specific software reuse becomes incorporated into mainstream DoD system/software acquisition and engineering processes. The intended audience is SPO engineers who are responsible for pre-Request For Proposal (RFP) engineering activities, proposal evaluation, monitoring of engineering activities after a contract is awarded, and monitoring of ongoing sustaining (or maintenance) engineering efforts of fielded products. To fully utilize the concepts in this handbook, it is recommended the reader be familiar with software development techniques and methodologies, existing government regulations and standards, and the acquisition process.

7. The **Introduction to Software Reuse Course** provides government, industry, and university students with an understanding of reuse, with emphasis on domain-specific reuse. Domain engineering is defined and the domain analysis products are explored. Reuse library concepts are discussed and an actual library is demonstrated.

8. The **Library Capability Demonstration** documents provide information about the CARDS operational library system capabilities.

9. The **Market Study** provides information regarding the state-of-the-art of software development and maintenance within the military services. Results of analysis of collected data reflect current practices, as well as identified needs within each military service for establishment of a suitable infrastructure to enable reuse. The focus of results has been geared toward facilitating the institutionalizing of software reuse within the DoD by more finely concentrating development, technology transfer and franchising efforts toward satisfaction of identified requirements.

10. The **Training Plan** serves as a comprehensive guide for creating training courses and materials on domain-specific reuse for DoD organizations, DoD contractors, system engineers, and university professors. It provides guidance for conducting three different domain-specific software reuse training courses for the four audiences. The intent of the courses is to demonstrate how software reuse can reduce development and maintenance time and costs, reduce project risks, and increase productivity.

APPENDIX E Selected Technical References

11. A library model (based on the Software Technology for Adaptable, Reliable Systems (STARS) Reuse Library Framework (RLF) technology) has been created and Portable, Reusable, Integrated Software Modules (PRISM) Program components are being added to the CARDS Library System. The **Library System** provides a view of CC requirements presented in the Defense Information Systems Agency (DISA) Command Center Design Handbook and provides a mapping of those requirements to the Generic Command Center Architecture (GCCA). The CARDS Library System also allows users to access ASSET and DSRS reuse libraries through an interoperation mechanism. The CARDS Library System has three libraries and related models:

11.1 The **Command Center Library (CCL)** is an example of a domain-specific reuse library, i.e., based on models of domain-specific, architecture-centric information. It is an organized collection of components and assessments of reusable components which can be used to build CC Information Processing Systems (IPS). A component is any piece of knowledge which can be used to build a system within a particular domain, e.g., code (executable or source), requirements, specifications, and test plans. An assessment of a component describes the suitability of that component for use in building a CC domain system. Both CC and non-CC developers practicing reuse can study the CCL as an evolving application of model-based reuse library techniques and for potential components. The following views are available to users and comprise a domain-specific context for the CCL contents: An architecture view shows the components via their connections within the CC architecture. This view is based on the PRISM GCCA model. A component class view shows groupings of common functionality components. For example, there is a database management system component class which has components with the functionality common to database management systems. This view is based on the PRISM GCCA model. A domain requirements view shows components via the functional requirements of a CC IPS. Requirements are mapped to the functionality of one or more component classes. The requirements view is based on the DISA CC Requirements Handbook (which is also used by PRISM).

11.2. The **PRISM Distribution Library (PDL)** is an organized collection of PRISM products, including reports on: CC prototype demonstrations, the evolving PRISM GCCA, and integration source code (i.e., wrappers) used to build CC prototypes. Users can view abstracts or entire reports, as well as extract reports and wrappers.

11.3. The **CARDS Documentation Library (CDL)** is an organized collection of CARDS documents and papers. These documents describe specific processes, guidelines, and policies of reuse. The CDL is based on specific reuse interest groupings, e.g., DoD acquisition process, reuse library application, reuse library development, reusable component and tool development, and training for reuse. Users are able to view and extract CDL contents.

12. The **Library Development Handbook** provides an overview of the phases involved in developing a domain-specific reuse library: domain analysis, library encoding, and library population and presents a CARDS generic library population process. The handbook enumerates specific instructions and examples for populating a domain-specific reuse library, based on this library population process.

13. The **Library Model Document** describes the current state of each CARDS library, e.g., CCL and PDL. Each library has its own Library Model Document which is updated with every library release. It describes how software engineering relates and feeds back to domain engineering, how domain engineering compares and contrasts to library modeling, and examines modeling concepts and specialization/aggregation hierarchies. The intended audience is anyone desiring an understanding of one or more CARDS library and wanting a view of the current library release. A knowledge of software engineering concepts is assumed.

14. The **Library Operations Policies and Procedures** provides a comprehensive guide to CARDS policies and procedures to implement and maintain its reuse libraries. A high level (non-CARDS specific) view of the policies and procedures for library operations is given for upper level management and technical supervisors (Volume I). Day-to-day CARDS operating instructions needed by technical personnel are also detailed (Volume II). In addition, the processes for developing the contents of a domain-specific library are specified.

15. The **Library User's Guide** describes how to access the CARDS libraries and components, including some of the CARDS specific aspects of Reuse Library Framework (RLF).

16. The **Metrics Concept Report** is based on a February 1993 Software Reuse Metrics Workshop to develop questions indicating what measurements should be collected in support of the DoD Software Reuse Program. Thus, it describes the CARDS efforts to use metrics to measure and improve processes, products, and services; and to monitor and provide possible contributions to the DoD Software Reuse Metrics Plan.

17. The **Model Contracts/Agreements** document is based on an on-going forum of DoD lawyers and contracting officers to examine legal issues and concerns related to software reuse. Sample agreements and associated guidelines for CARDS staff implementation is provided. These are based on the CARDS CCL's business, operational and qualification processes and concepts, as well as recommended policy changes. This was done to help reduce CARDS operating risk, meet its operational and technical goals, and meet the needs of its customers.

18. The **Version Description Document** provides information about each release, including release notes (changes from the previous version and possible problems and known errors), installation instructions, and a listing of computer media files.

19. The **Software Architecture Seminar Report** is based on a CARDS seminar and workshop to increase awareness, explore current research into software architectures as a means of implementing software reuse, and examine current practices and issues involving architectures. Seminar goals were to understand the various meanings of software architecture, research in the field of architecture, and efforts in applying software architectures. This report includes: presentation slides, panel discussions, the workshop, questions and answers, and issues discussed.

20. The **Technical Concepts Document** describes the technical concepts employed towards the development of the CARDS CCL and other domains. CARDS views and its consequences on library development are presented. A distinction is made between domain and library modeling. The CARDS component qualification process is presented, along with the CARDS system composition and component qualification tools. Technical aspects necessary to make the CARDS CCL operational and key technical interoperability and security concepts are addressed.

21. The **Franchise Plan** helps management develop a detailed, tailored implementation plan to prepare the organization to begin the software reuse process. It is targeted toward management personnel and is meant to introduce reuse and obtain assistance from programs such as CARDS to do reuse. The Franchise Plan presents a planning process for building a reuse infrastructure.

22. **Non-CARDS Products**. The following documents were not prepared by CARDS, but are closely related to the CARDS software reuse effort:

APPENDIX E Selected Technical References

The **STARS Conceptual Framework for Reuse Processes (CFRP)** defines a context for considering reuse-related software development processes, their interrelationships, and their composition and integration with each other and with non-reuse-related processes to form reuse-oriented life cycle process models.

The **STARS Reusability Guidelines** proposes guidelines for the design and coding of reusable software components. It offers common capabilities within well-defined application domains that are suitable for installation in a library of reusable components. It is directed to software developers and maintainers of a software reuse library. [STARS89]

SDP-2000: A Guide To Project Implementation of Megaprogramming describes a vision of software industry as it may exist under megaprogramming (when superior practices in software engineering are synthesized) and describes transition steps to be required by the government and contractors alike.

A New Process for Acquiring Software Architecture outlines a process used to ensure system acquisitions include attention to software architecture.

CARDS provides interested parties with technical expertise in the field of process-driven, domain-specific, architecture-centric, library-assisted software reuse. CARDS consulting services include reuse adoption support, domain analysis/modeling, complete site and operation survey, feasibility studies, and operational hardware and software installation. CARDS provides domain engineers with an implementation framework for reuse libraries in their domain of interest. It also guides application engineers through the selection of alternative strategies for reuse based on their domain.

NOTE: To receive CARDS products and services, contact the CARDS Hotline, listed in Appendix A.

APPENDIX E Selected Technical References

TAB 3

GSA Information

NOTE: GSA publications are available from the GSA virtual library listed in Appendix A.

PROCUREMENT	
KMP-90-2-P	Guide for Requirements Analysis and Analysis of Alternatives
KMP-91-1-P	Guide for Contracting Officer's Technical Representatives
KMP-91-5-P	Guide for Acquiring Systems Integration Support Services
KMP-92-2P	Guide for Acquiring FIP Support Services
KMR-92-2P	IDIQ and Requirements Contracts: Lessons Learned
KML-87-2-P	GO-FOR-12 Program: A Report on Testing of Parallel Review
KML-87-3P	GO-FOR-12 Program: Interim Report on the Elimination of Unnecessary Bottlenecks in the Acquisition Process
KML-88-2-P	Better Acquisition Through Education and Training
KML-88-3-P	GO-FOR-12 Program: Final Report
KMP-92-5-P	Source Selection: Greatest Value Approach
SECURITY	
KMP-89-1-S	Information Security
INFORMATION RESOURCES MANAGEMENT	
KCC-92-1-I	IRMS Directory of Client Services
KGD-91-1-I	Managing Information Resources for Accessibility
KMA-87-1-I	Senior IRM Manager: Major Roles and Responsibilities as We Move into the 1990's
KMP-94-1-I	Data Management Issues Associated with Stovepipe Systems

APPENDIX

F

Selected Reading and Reference Material

Blank page.

APPENDIX F

Selected Reading and Reference Material

If you find these Guidelines profitable and useful, stay current with the latest developments in software engineering ***at no cost!*** *If you are not already receiving monthly issues of* ***CrossTalk****, the DoD Journal of Software Engineering published by the Air Force Software Technology Support Center on behalf of DoD, contact their customer service desk at the following address:*

CrossTalk
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
(801) 777-8045
DSN 777-8045

Ackerman, Frank A., Lynne S. Buchwald, and Frank H. Lewski, "Software Inspections: An Effective Verification Process," *IEEE Software*, Vol. 6, No. 3, May 1989

Andriole, Stephen J., ed., Advanced Technology for Command and Control Systems Engineering, AFCEA International Press, Fairfax, Virginia 1990

Arthur, Lowell Jay, Improving Software Quality: An Insider's Guide to TQM, John Wiley & Sons, Inc., New York, 1993

Beizer, Boris, Black-Box Testing: Techniques for Functional Testing of Software and Systems, John Wiley & Sons, New York,1995

Bennatan, E.M., On Time, Within Budget: Software Project Management Practices and Techniques, QED Publishing Group, Boston, 1992

Berlack, H. Ronald, Software Configuration Management, John Wiley & Sons, Inc., New York, 1992

Blum, Bruce I., Software Engineering: A Holistic View, Oxford University Press, New York, 1992

Boehm, Barry, Software Engineering Economics, Prentice Hall, New Jersey, 1981

Booch, Grady, Object Oriented Design, Benjamin/Cummins Publishing Company, Menlo Park, California, 1991

Booch, Grady, Software Engineering with Ada, Third Edition, Benjamin/Cummings Publishing Company, Menlo Park, California, 1994

Brooks, Frederick P., Jr., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, Reading, Massachusetts, 1975

Camp, Robert C., Benchmarking: The Search for Industry Best Practices that Lead to Superior Performance, ASQC Quality Press, Milwaukee, Wisconsin, 1989

Card, David N. and Robert L. Glass, Measuring Software Design Quality, Prentice Hall, 1990

Champy, James, Re-engineering Management: The Mandate for New Leadership, HarperBusiness, New York, 1995

APPENDIX F Selected Reading Material

Charette, Robert H., Software Engineering Risk Analysis and Management, McGraw-Hill Book Company, New York, 1989

Coad, Peter and Edward Yourdon, Object-Oriented Analysis, Yourdon Press, Prentice-Hall, Englewood Cliffs, New Jersey, 1990

Creech, Bill, The Five Pillars of TQM: How to Make Total Quality Management Work for You, Truman Talley Books/Dutton, New York, 1994

Cringely, Robert X., Accidental Empires: How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can't Get a Date, HarperBusiness, New York, 1993

Cusumano, Michael A., and Richard W. Selby, Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People, The Free Press, New York, 1995

DeGrace, Peter and Leslie Hulet Stahl, Wicked Problems, Righteous Solutions: A catalog of Modern Software Engineering Paradigms, Yourdon Press, Englewood Cliffs, New Jersey, 1990

DeMarco, Tom and Timothy Lister, Peopleware: Productive Projects and Teams, Dorset House Publishing, New York, 1987

DeMarco, Tom and Timothy Lister, Software State-of-the-Art: Selected Papers, Dorset House Publishing, New York, 1990

Denton, Lynn and Jody Kelly, Designing, Writing, & Producing Computer Documentation, McGraw Hill, New York, 1993

Dyer, Michael, The Cleanroom Approach to Quality Software Development, John Wiley & Sons, Inc., New York, 1992

Ebenau, Robert G. and Susan H. Strauss, Software Inspection Process, McGraw Hill, New York, 1994

Fagan, Michael E., "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, Vol. 12, No. 7, July 1986

Finkelstein, Clive, An Introduction to Information Engineering, Addison-Wesley Publishing Company, 1989

Firesmith, Donald G., Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach, John Wiley & Sons, Inc., New York, 1993

Fisher, David T., Myths and Methods: A Guide to Software Productivity, Prentice Hall, New York, 1991

Freedman, Daniel P. and Gerald M. Weinberg, Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products, Dorset House Publishing, New York, 1990

Ganti, Narsim and William Brayman, The Transition of Legacy Systems to a Distributed Architecture, John Wiley & Sons, Inc., New York, 1995

Gates, Bill, The Road Ahead, Viking, New York, 1995

Gause, Donald C. and Gerald M. Weinberg, Exploring Requirements: Quality Before Design, Dorset House Publishing, New York, 1989

Glass, Robert L. Building Quality Software, Prentice Hall, Englewood Cliffs, New Jersey, 1992

Glass, Robert L., Software Conflict: Essays on the Art & Science of Software Engineering, Prentice Hall, 1990

Gulledge, Thomas R., et al., eds., Cost Estimation and Analysis: Balancing Technology and Declining Budgets, Springer-Verlag, 1992

Hetzel, Bill, Making Software Measurement Work: Building an Effective Measurement Program, QED Publishing Group, Boston, 1993

Howe, Roger J., Dee Gaeddert, Maynard A. Howe, Quality on Trail: Bringing Bottom-Line Accountability to the Quality Effort, Second Edition, McGraw-Hill, Inc., New York, 1995

Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley Publishing Company, Inc., New York 1995

Humphrey, Watts S., Managing the Software Process, Software Engineering Institute, Addison-Wesley Publishing Company, 1990

APPENDIX F Selected Reading Material

Jacobson, Ivar, The Object Advantage: Business Process Reengineering with Object Technology, Addison-Wesley Publishing Company, ACM Press Books, New York, 1995

Jones, Capers, "New Directions in Software Management," *IS Management Group*, Carlsbad, California, 1994

Jones, Capers, "Software Productivity and Quality Today," *IS Management Group*, Carlsbad, California, 1993

Jones, Capers, Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, Inc., New York, 1991

Jones, Capers, Assessment and Control of Software Risk, PTR Prentice-Hall, Englewood Cliffs, New Jersey, 1994

Jones, Capers, Programming Productivity, McGraw-Hill Book Company, New York, 1986

Katzenbach, Jon R., and Douglas K. Smith, The Wisdom of Teams: Creating the High-Performance Organization, Harvard Business School Press, Boston, Massachusetts, 1993

Keyes, Jessica, ed., Software Engineering Productivity Handbook, Windcrest/McGraw-Hill, New York, 1993

Krell, Dr. Bruce E., Developing with Ada: Life-Cycle Methods, Bantam Professional Books, 1993, ISBN 0553-09093-3

Kriegel, Robert J., and Louis Patler, If It Ain't Broke...Break It!: and Other Unconventional Wisdom for a Changing Business World, Warner Books, New York, 1991

Landsbaum, Jerome B. and Robert L. Glass, Measuring and Motivating Maintenance Programmers, Prentice Hall, Englewood Cliffs, New Jersey, 1992

Lawrence, Measures for Excellence: Reliable Software on Time, Within Budget, Putman, 1992

Ludwig, Mark, The Little Black Book of Computer Viruses, American Eagle Publications, 1992

Lundy, James L., Teams: How to Develop Peak Performance Teams for World-Class Results, The Dartnell Corporation, Chicago, Illinois, 1994

Lynch, Richard L and Kelvin F. Cross, Measure Up! Yardsticks for Continuous Improvement, Blackwell Business, 1991

Marciniak, John J. and Donald J. Reifer, Software Acquisition Management: Managing the Acquisition of Custom Software Systems, John Wiley & Sons, Inc., New York, 1990

Martin, James, Information Engineering, (Book 1 of 3), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989

Martino, Joseph P., Technological Forecasting for Decision Making, Third Edition, McGraw Hill, New York, 1993

McClure, Carma, The Three Rs of Software Automation: Re-engineering, Repository, Reusability, Prentice Hall, Englewood Cliffs, New Jersey, 1992

McMenamin, Steve and John Palmer, Essential Systems Analysis, Yourdon Press, Englewood Cliffs, New Jersey, 1984

Meyer, Bertrand, Reusable Software: The Base Object-Oriented Component Libraries, Prentice Hall, Englewood Cliffs, New Jersey, 1994

Mills, Harlan D, Richard C. Linger, and Alan R. Hevner, Principles of Information Systems Analysis and Design, Academic Press Inc., Harcourt Brace Jovanovich, 1986

Mosely, Daniel J., The Handbook of MIS Application Software Testing: Methods, Techniques, and Tools for Assuring Quality Through Testing, Yourdon Press, Englewood Cliffs, New Jersey, 1993

Mosley, Daniel J., The Handbook of MIS Application Software Testing: Methods, Techniques, and Tools for Assuring Quality Through Testing, Yourdon Press, Englewood Cliffs, New Jersey, 1993

Musa, John D. et al, Software Reliability: Measurement, Prediction, Application, McGraw-Hill Book Company, 1987

Oliver, RADM Dave, Jr., Lead On: A Practical Approach to Leadership, Presidio Press, Navato, California, 1992

Owens, ADM William A., High Seas: The Naval Passage to an Uncharted World, Naval Institute Press, Annapolis, Maryland, 1995

APPENDIX F Selected Reading Material

Pagonis, LTG William G., with Jeffrey L. Cruikshank, Moving Mountains: Lessons in Leadership and Logisitics from the Gulf War, Harvard Business School Press, Boston, Massachusetts, 1992

Pressman, Roger S. & Russell S. Herron, Software Shock: The Danger & the Opportunity Dorset House Publishing, 1991

Pressman, Roger S., A Manager's Guide to Software Engineering, McGraw-Hill, New York, New York, 1993, ISBN 0-07-050820-8

Pressman, Roger S., Software Engineering: A Practitioner's Approach, Third Edition, McGraw-Hill, Inc., 1992

Prezemieniecki, J. S., ed., Critical Technologies for National Defense, AIAA Publications, 1991

Putnam, Lawrence H. and Ware Myers, Measures for Excellence: Reliable Software On Time, Within Budget, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992

Roetzheim, William H., Developing Software to Government Standards, Prentice-Hall, Englewood Cliffs, New Jersey, 1991

Roetzheim, William H., Structured Computer Project Management, Prentice Hall, Englewood Cliffs, New Jersey, 1988

Russell, Glen W., "Experience with Inspection in Ultra Large-scale Developments," *IEEE Software*, Vol. 8, No.1, January 1991

Schulmeyer, G. Gordon and James I. McManus, eds., Total Quality Management for Software, Van Nostrand Reinhold, New York, 1992

Senge, Peter M., The Fifth Discipline: The Art & Practice of The Learning Organization, Currency Doubleday, New York, 1994

Shumate, Ken and Marilyn Keller, Software Specification and Design: A Disciplined Approach to Real-time Systems, John Wiley & Sons, New York, 1992

Strassmann, Paul A., The Business Value of Computers An Executive's Guide, The Information Economics Press, New Canaan, Connecticut, 1990

Tapscott, Don and Art Caston, Paradigm Shift The New Promise of Information Technology, McGraw-Hill, New York, 1993

Taylor, David A., Object-Oriented Technology A Manager's Guide, Addison-Wesley Publishing Co, 1990

Thomas, Brian, The Human Dimension of Quality, McGraw-Hill Book Company, London,1994

Utz, Walter J., Jr., Software Technology Transitions: Making he Transition to Software Engineering, Prentise Hall, Englewood Cliffs, New Jersey, 1992

Utz, Walter J., Software Technology Transitions: Making the Transition to Software Engineering, Prentice Hall, Englewood Cliffs, New Jersey, 1992

Walsh, Mike, Productivity Sand Traps and Tar Pits: How to Detect and Avoid Them, Dorset House Publishing, New York, 1991

Weinberg, Gerald M., Quality Software Management, Volume 1., Systems Thinking, Dorset House Publishing, New York, 1992

Wellins, Richard S., William C. Byham, and Jeanne M. Wilson, Empowered Teams: Creating Self-Directed Work Groups That Improve Quality, Productivity, and Participation, Jossey-Bass Publishers, San Francisco, California, 1991

Whitten, Neal, Managing Software Development Projects: Formula for Success, Second Edition, John Wiley & Sons, New York, 1995

Yourdon, Edward N., Modern Structured Analysis, Prentice Hall, New Jersey, 1990

Yourdon, Edward, The Decline & Fall of the American Programmer, Yourdon Press, Englewood Cliffs, New Jersey, 1992

PART III

Engineering-Related Appendices

Blank page.

Version 2.0

APPENDIX

G

Software Architecture

Blank page.

APPENDIX G

Software Architecture

NOTE: The following two technical reports provide information about the need for a flexible, standards-based software architecture and how to acquire it.

CONTENT **PAGE**

Tab 1:
The Importance of Architecture in DoD Software G-1
Tab 2:
A New Process for Acquiring Software Architecture G-20

TAB 1

The Importance of Architecture in DoD Software

Barry M. Horowitz, PhD
ESC-TR-94-208, September 1994

PREFACE

DoD's automated systems are likely to face more varied military threats than in the past that will require the ability to make system changes rapidly. In addition, defense budgets are likely to continue to decrease. It is important then, for both mission effectiveness and cost savings, that these systems be built with as much flexibility as possible to incorporate new capabilities and new technology. This paper proposes that an increased focus on digital system architecture can markedly improve system flexibility as well as yield significant cost savings. The author, **Dr. Barry M. Horowitz**, is President and Chief Executive Officer of The MITRE Corporation.

ACKNOWLEDGMENTS

Many people at MITRE contributed data and ideas to this document. Special thanks are expressed to Judith A. Clapp, Dr. Richard J. Sylvester, and Gerard R. Lacroix.

TABLE OF CONTENTS

SECTION	PAGE
INTRODUCTION: A New Direction for DoD Software Acquisition	G-3
DoD SOFTWARE: More Important — and More Expensive	G-3
The Value of Software	G-3
The Cost of Software	G-4
ARCHITECTURE: The Invisible Component	G-8
Architecture: A Definition	G-10
Architecture: Ramifications	G-12
The Complexity of Architecture	G-13
ARCHITECTURE: The Waiting Solution	G-14
Technical Focus	G-14
Faulty Emphasis	G-14
Commercial Architecture	G-15
Availability of Tools	G-16
RECOMMENDATIONS: Finding and Applying Architecture	G-16
System Specification	G-17
Early Satisfaction of Architectural Requirements	G-18
Contractor Incentives	G-18
Getting Started	G-18
REFERENCES	G-19

LIST OF FIGURES

FIGURE		PAGE
G-1	Growth of C3 Software Size	G-4
G-2	Software Life Cycle Cost Distribution	G-5
G-3	Software Maintenance Activities	G-5
G-4	DoD Software Expenditures	G-6
G-5A	Structure Versus Cost to Change	G-7
G-5B	Structure Versus Defects	G-7
G-5C	Structure Versus Time to Change	G-7
G-6	Ada Development Versus Modification	G-8
G-7A	Release Interval Versus System Age	G-9
G-7B	Increasing Complexity with Age	G-9
G-8	Hardware and Software Structure	G-11
G-9	Digital System Architecture	G-11
G-10	IBM View of Software Architecture	G-11
G-11	Data Flow Reference Model	G-12
G-12	Joint STARS System Architecture	G-13
G-13	Technical Focus (Estimated)	G-14
G-14	Effect of Control Structure on Errors	G-17

APPENDIX G Software Architecture

INTRODUCTION: A New Direction for DoD Software Acquisition

Our world is changing. The military threat to the United States posed by the Soviet Union for nearly fifty years is diminished, but there are new threats from rapidly evolving Third World countries that require rapid changes to military systems. Crises such as the recent events in the Persian Gulf highlight the need for flexible systems that can be changed quickly to meet the military's unanticipated challenges. In addition, the defense budget continues to be reduced — the government has less money to spend on systems.

The answer to this dual challenge — to make systems more flexible and to reduce the cost of defense systems — lies in the design of the digital system architecture, which includes the composition of hardware and software components, the structure that interconnects them, and the rules by which they interact. All too often, both government and industry focus too narrowly on achieving the initial requirements for systems and give little thought to being able to adjust to what the system may be required to do five or ten years later, or to what happens as hardware may no longer be supportable or advanced technology may become available for incorporation into the system.

Architecture design is the key to achieving the cost savings and operational flexibility inherent in digital systems. If the system is properly structured, then hardware components can be added or upgraded without expensive changes to the rest of the system. A good architecture allows a system designed to counter one threat to counter a different threat through localized modifications to the software that change the functional capability of the system or allow it to interoperate with other systems. What is more, under the right circumstances, these changes can be made very quickly.

DoD SOFTWARE: More Important — and More Expensive

The Value of Software

Software provides modern defense systems with a flexibility that cannot be achieved in any other reasonable way. Operation Desert Storm provides several excellent examples.

Patriot is an Army corps-level missile system primarily designed to counter aircraft. Given the inherent capability of the missile itself, the designers gave some thought to employing it to shoot down incoming enemy missiles. The Scud attacks during the war, however, focused everyone's attention on this threat with much more urgency. Patriot's designers developed a new software package that increased the Patriot's effectiveness to counter the Scud threat. When radar tracks began to show that the Scuds were breaking up on re-entry, the designers further tuned the package to recognize and attack the Scud warhead, and not the debris that accompanied it. Without the modified software, Patriot would have been less effective. Yet the designers were able to implement this capability quickly and at a surprisingly low cost. No new missiles or radars were required. The software improvements could go to the war region in a briefcase.

Another example of this flexibility also involves Patriot, though at the much higher level of command, control, communications, and intelligence (C3I). To improve Patriot's ability to react to the Scud attacks, which proceeded at much higher speeds than the targets normally confronting Patriot, US space satellites were redirected to watch for Scud launches. When a launch was detected, the satellite relayed the targeting cues over a satellite link to the appropriate Patriot battery, leading to a successful interception. Minor software modifications permitted a network to be set up.

There are other, less dramatic examples of the value of software's inherent flexibility that came out of Desert Storm. Navy attack aircraft had been set up for years to attack Soviet targets, either at sea or on land. A cassette data tape provided the attack computers with the information they needed to launch their stunningly accurate attacks on targets that had only recently been identified.

Software was also the key to the effectiveness of Air Force jamming aircraft. Programmed for operations against Soviet-bloc radars, the jammers were faced with a mixture of Soviet, French, British, and Italian equipment. Software changes enabled the equipment to perform its task against this new threat far more quickly — and less expensively — than could have been done otherwise.

Precisely because of its flexibility, the DoD is buying more software in its systems and implementing functions in software that had previously been performed in hardware. Figure G-1 shows this trend in a number of systems. For example, the latest version of the Cobra Dane radar system uses more software than did the original release, and the new Milstar terminal uses more software to perform more functions than did its predecessor, AFSATCOM. Desert Storm demonstrated that the flexibility software offers us is real and of great value to the military. It will become more so if we continue to have crisis scenarios that are a lot harder to predict and cause us to apply our systems in unplanned ways.

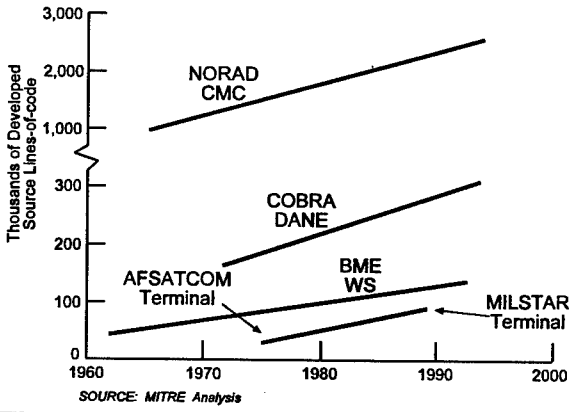


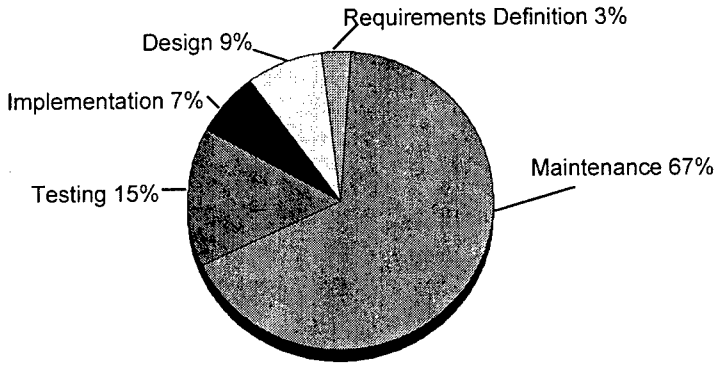
Figure G-1 Growth of C3 Software Size

The Cost of Software

Since the DoD has been buying more and more software, its total expenditure on software has been increasing, and software is expensive. With shrinking military budgets, we have to find ways to use more software and yet reduce its cost. Typically, two-thirds of what is spent on software is believed to be spent after the system becomes operational, during the maintenance phase, as illustrated in Figure G-2.

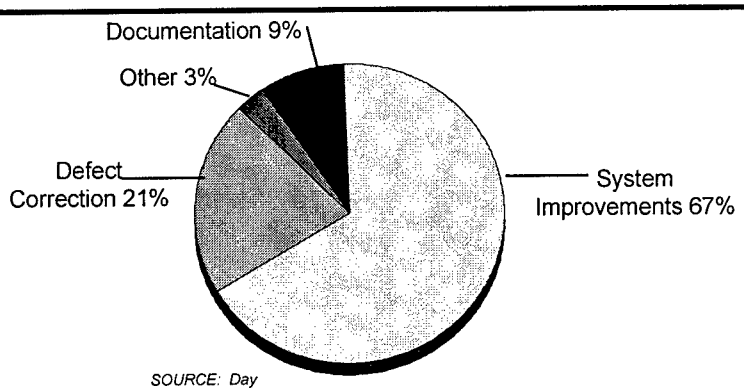
Looking at the distribution of software maintenance activities is itself illuminating. About two-thirds of the software maintenance effort for a system is typically spent on modifying the original system to provide new capabilities and to add new technology, at least twice the effort spent on making repairs. Figure G-3 confirms this data for an Army command and control system. Taking these two sets of data together suggests that about 45% of the effort spent on software is used to change the system after it has been delivered.

APPENDIX G Software Architecture



SOURCE: Arthur, 1988

Figure G-2 Software Life Cycle Cost Distribution

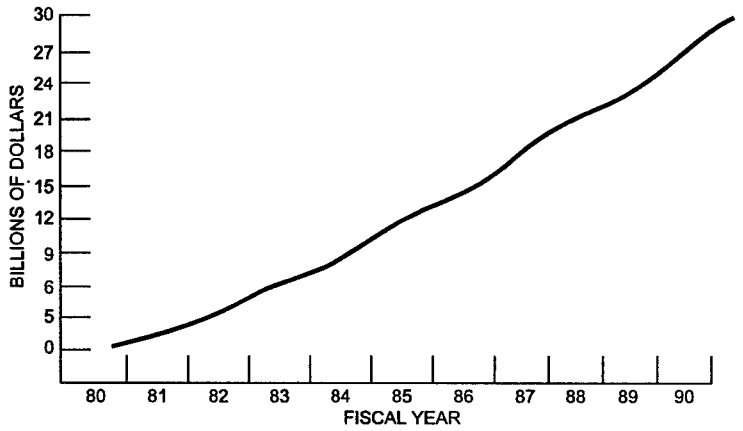


SOURCE: Day

Figure G-3 Software Maintenance Activities

Experience also shows that we often spend part of the system development effort making changes in response to changing or better understood requirements. We probably spend more than 50% of our software effort to change the capabilities of a system over its developmental and operational lifetime. If we can design software systems to take only half as much effort to modify, we can reduce the life cycle cost of the entire software system by 25%. When applied to the total amount the DoD spends on software, this improvement can yield enormous cost savings. While it is difficult to determine accurately how much the DoD spends on software, MITRE staff made a rough analysis that indicates the total amount to be approximately $30 billion per year (see Figure G-4 below). If we can in fact reduce the life cycle cost of software by 25%, the total savings will range between five and eight billion dollars every year.

The example in Figure G-5 (below) illustrates how these savings might be possible. Three thousand lines of new code were required to be added to a system of 50,000 lines. When the changes were made, the cost, time, and number of defects found in the delivered system were measured. Then, the structure of the software was improved, and the changes were again made. It cost only half as much to modify the structured software and it took less than half the time. As an added benefit, there were about one-eighth the number of errors in the structured software.



MITRE estimate based on:
(1) Bureau of Economic Analysis Input-Output GNP and DoD expenditure data;
(2) 1967 Survey of Current Budget, Department of Commerce;
(3) Census of Service Industries, Department of Commerce;
(4) Handbook of Labor Statistics, Department of Labor; and
(5) Census of Population, Occupation by Industry Matrix.

Figure G-4 DoD Software Expenditures

Another indication of increases in productivity that may accrue from well-structured software is shown in Figure G-6 (below). The points on the graph represent software size and productivity for development of some systems programmed in Ada. One of those systems, the Command Center Processing and Display System Replacement (CCPDS-R), was developed with special attention to designing a system architecture and tools to facilitate its modification. The original system was then significantly modified to produce two new versions. Productivity data for the two modified versions of the system are shown in boxes in Figure G-6. The high overall productivity is due in part to the architecture that accommodated these changes and in part to tools that facilitated making changes. Further benefits were realized because the architecture made general system services more accessible and consequently the application modifications were smaller than they might otherwise have been. The productivity data were adjusted for the reused and tool-generated software. This is even more impressive when the usual negative relationship between productivity and system size is taken into account.

While important, the dollar cost of making changes to the system is only one concern; time is another. Operation Desert Storm provided many examples of how well the flexibility of software served the allied cause, but there were also cases where we were not able to exploit software as we would have liked. Requests for changes to systems were made early in the campaign, and estimates were provided that said it would take 18 months to make the desired changes. This was obviously unacceptable, and the military found it hard to understand why it should take so long, given that software is supposed to offer great flexibility.

Software does provide flexibility, but it must be designed from the start with an architecture that allows it to do so. Furthermore, everyone concerned must preserve the integrity of the architecture; otherwise, flexibility can be lost through the process of change. As an example, Figures G-7A and G-7B (below) are plots of the time it took to create each release of an IBM operating system and the number of modules affected in each release. The graphs show a progression; that is, it took longer and longer to modify the system as the system grew older. This was due to the growing complexity of the system — more and more modules had to be

APPENDIX G Software Architecture

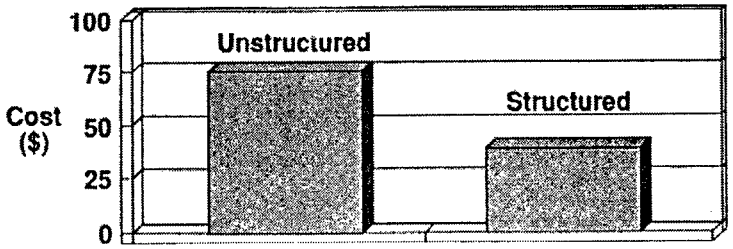


Figure G-5A Structure Versus Cost to Change

changed for each new release. The software structure degenerated, which made it more difficult

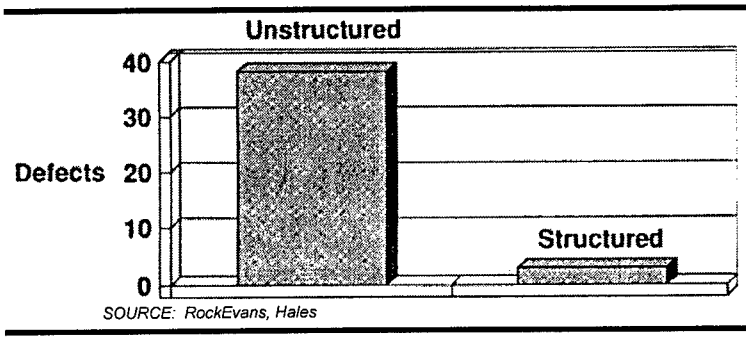


Figure G-5B Structure Versus Defects

to determine which modules had to be repaired. In addition, the pattern of regression testing had

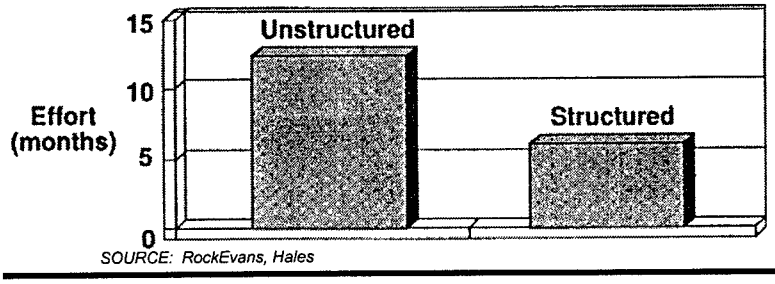


Figure G-5C Structure Versus Time to Change

to be more extensive since so many parts of the system had been affected by the modifications. This complexity and uncertainty translates into more time and money, and this process begins a vicious circle — modifying the system makes the next modification even more difficult, time-consuming, and expensive.

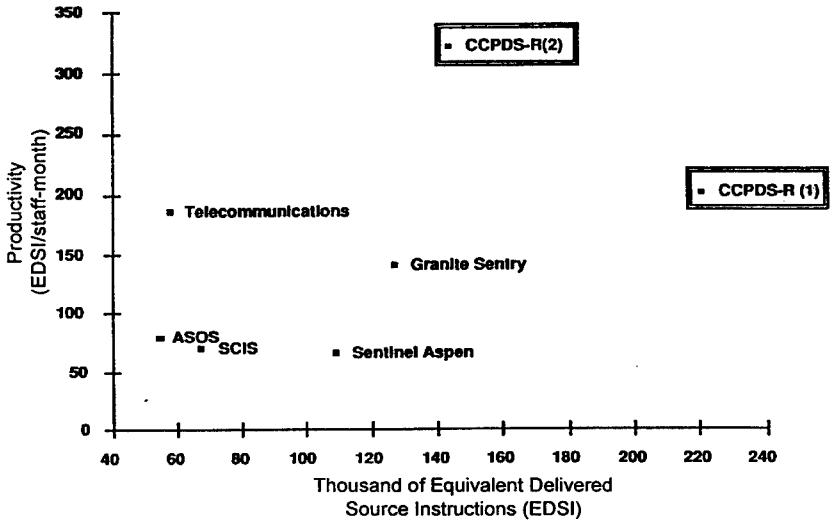


Figure G-6 Ada Development Versus Modification

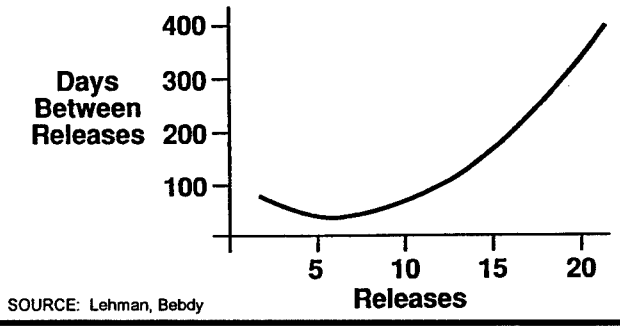


Figure G-7A Release Interval Versus System Age

ARCHITECTURE: The Invisible Component

The DoD does not usually buy architectures — it buys systems that meet explicit functional and performance requirements specified by the user or the acquisition agent. In most cases, the DoD does not ask for an architecture to be delivered; it should therefore come as no surprise that very few architectures are delivered. This is not to say that the DoD does not receive a system architecture. Every system has some form of architecture, but the architecture the DoD receives may be quite convoluted and inflexible by the time the system moves from concept to fieldable implementation. The DoD does not specifically buy an architecture because there are no explicit specifications for its characteristics, no formal tests of its capabilities, and no formal control of its structure to prevent arbitrary changes once it has been defined. This is one reason why architecture is fundamentally invisible — operational users are not often aware that an architecture is even present if it does not directly affect the functional capabilities they are using.

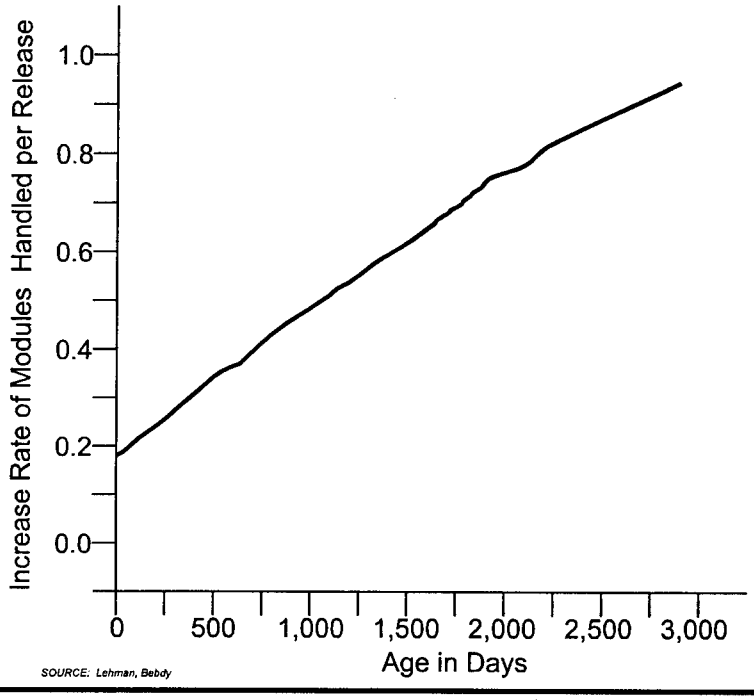


Figure G-7B Increasing Complexity with Age

Yet architecture is the main determinant of a system's characteristics. The efficiency of the system, and thus its performance, depend on how the architecture handles resource utilization; architecture determines how the system sustains operations when parts of the system fail. The architecture also determines how maintainable the system is; that is, (1) how much effort is required to find and fix errors; (2) how easy it is to add new capabilities through software; and (3) how much is required to move the software to different computer hardware. Although they may be invisible to the user, these characteristics, which are all determined by architecture, are very visible to developers and maintainers who must modify and add to the operational capabilities of the system. If the DoD wants to buy architectures, it will first have to know how to ask for them, specify them, test them, demonstrate them, and prevent them from degenerating; in short it will have to perform all the operations that it performs now when buying other products.

In addition, DoD must perform a new task that is currently not part of its acquisition strategy — maintain explicit control of the architecture for the life of the system. One way of accomplishing this is to add architecture to the other aspects of the system that are controlled by the configuration management system. Since the maintenance phase contains a large fraction of the system's software costs, the ultimate maintainer of the system — and thus, the government — must eventually assume control of the architecture. This will require a significant change in the way the government currently views architecture and its importance.

Architecture: A Definition

There is no single, commonly accepted definition of a digital system architecture. In the broadest sense, architecture is a system or style of building having certain characteristics of structure. When applied to digital computer systems, architecture includes the hardware and software components, their interfaces, and the execution concept that underlies system processing.

The simplest level of system architecture defines how the hardware and software that make up the system are partitioned into components, and how software components are assigned to hardware components. Figure G-8 is an oversimplified example (only primary functions are shown) of a fighter aircraft's federated hardware and software structure, which consists of separate computers networked on a standard bus with individual software functions assigned to the individual computers. At this level, the defense industry generally does a fairly thorough job of understanding architecture, mainly because developers need to understand how much hardware of which types they need to buy.

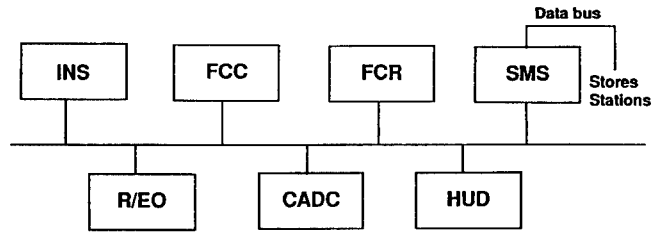


Figure G-8 Hardware and Software Structure

Figure G-9 is another view of the digital system architecture for the same aircraft, showing both the application software in the previous figure and the software that performs system-wide functions. The functions can be described as grouped into layers; in this view, software in any layer may utilize software only in its own layer or the layer below it. The computers in the lowest layer represent the segregation of hardware from software to increase their independence and to enhance software portability. This is an example of the first part of the definition of architecture — the arrangement of hardware and software components (namely, the structure).

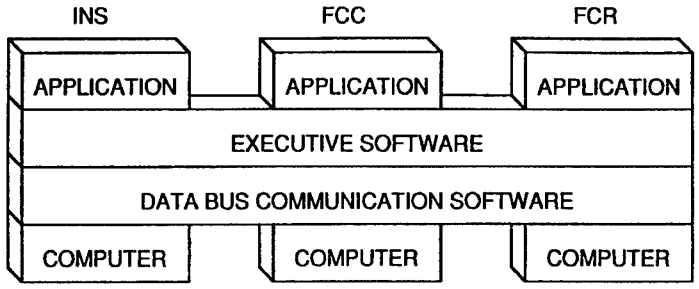


Figure G-9 Digital System Architecture

The second element in the definition deals with the interfaces among key elements—for example, data communications according to a standard protocol (MIL-STD-1553). All computer-to-computer messages in the aircraft's avionics architecture must use this protocol; hence, adding new computers and new functions to the system is relatively simple (from a communications perspective) as long as the data bus has the needed capacity.

The third element in the definition of architecture is the execution concept. In the sample avionics system previously shown, this concept is based on the cyclic execution of each function, precisely timed to repeat the computation on a planned schedule. Taking structure, interfaces, and execution concept together produces one definition of architecture. Of course, different vendors interpret the software part of the architecture in different ways.

Figure G-10 illustrates an IBM view of software architecture. In many cases, commercial companies can provide off-the-shelf components for the general system capabilities of DoD systems; in addition, groups of commercial hardware and software vendors are defining standard interfaces among layers and components. These open system architectures may provide the flexibility necessary to integrate, with a minimum of effort and system disruption, new hardware and software components with improved capability or maintainability. For example, the International Standards Organization (ISO) Open Systems Interconnection reference model defines the functions of each layer and the protocols for peer-level layers of a communications interface. Standards of this type permit the upgrading of elements of the system at particular layers without requiring the alteration of elements at other layers.

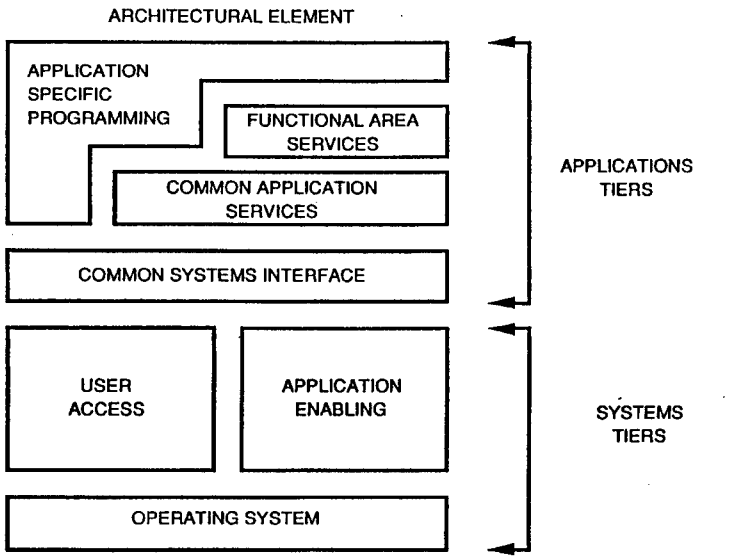


Figure G-10 IBM View of Software Architecture

Figure G-10 also illustrates the concept of service layers in the part of the architecture that is developed uniquely for one class of application (such as command centers or communications systems). These or other services must include error detection and recovery, interprocess communications, scheduling, and synchronization of processes. At this level of architecture, we must rely on the applications designers for standards within their design, as well as the quality control procedures to assure adherence to their standards.

Architecture: Ramifications

The lack of a good architecture has a serious bearing on the cost, effectiveness, and availability of DoD systems. For many applications where high reliability and availability are necessary, the architectural concepts must incorporate failure management as well as other mission requirements. Trouble follows when this is not part of the initial architectural design.

Error handling is a critical component of any system, since errors will inevitably occur. Most systems have software to detect errors and to recover from an error when it is detected (for example, when a numerical value goes beyond expected bounds or when an operator pushes the wrong button). Given the critical nature of most DoD systems, it is crucial to keep the system in operation when errors occur. When we leave it to each programmer who has developed a part of a system to determine how to handle errors, the result is an unintegrated set of sometimes widely varying procedures that are often completely incompatible and even dangerous.

Recently, MITRE scanned the software for a large, safety-critical command and control system, and identified over 200 instances in the code that handled errors incorrectly. In many cases, the system detected the errors and then ignored them, or passed them to another part of the system that could not handle them. What was missing was a consistent, coherent, system-wide error-handling strategy, a critical attribute of architecture. Furthermore, there was no method of ensuring that individual programmers adhered to the failure management standards that should have been established with the architecture.

Data flow diagrams can show the execution concept of the architecture of a system (see Figure G-11). In this view, the sequence of processing, and which hardware and software components are involved as specific data moves through a system, are apparent. This end-to-end view of the system's treatment of an external input is called a string; in actuality, there are many levels of detail that can be represented by a hierarchy of data flows for the same string. A string is useful for assuring users that the system will perform the right functions on their data; it is also useful for estimating and controlling the time the system will take to respond to an input. What complicates the design of an architecture to meet response times is the large number of such strings that may be awaiting execution at the same time (as when many sensor reports are received or must be transmitted) and the contention over which string will use shared resources such as computers and communications lines.

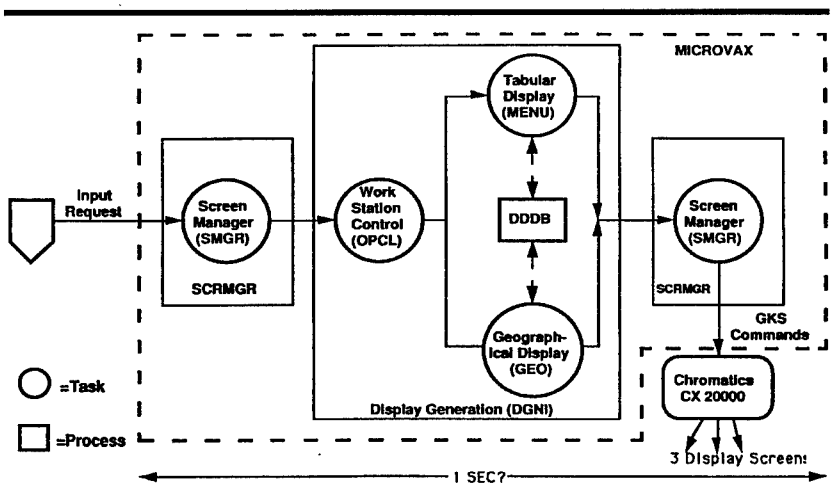


Figure G-11 Data Flow Reference Model

APPENDIX G Software Architecture

To understand the timing aspects of a system, it is often necessary to develop a simulation that models the system architecture and the load on hardware and software components or to execute benchmark software on the actual hardware. The validity of the results depends on how accurately the load, the data flows, the hardware speed and capacity, and the timing of individual processes are represented in the model — even the most elaborate model yields useless results if the parameters are not accurate. The designer of the architecture must be given accurate information to design the architecture and to evaluate its performance; in other words, it is essential that there be good communications between modelers and architects or designers.

Since the demands on the hardware resources will change over time, the architecture must provide the flexibility necessary to upgrade hardware to faster or larger processors to accommodate requirements for increased processing loads or faster response time. Similar increases in bandwidth may be necessary in communications hardware to provide for increased loads. Models that correspond to an architecture can be useful in planning for and evaluating the effect of changes in the hardware configuration of a system architecture to meet new demands.

The Complexity of Architecture

Perhaps the main reason that we fail to address these different aspects of system architecture lies in the increasingly complex nature of the systems we build. Figure G-12 illustrates the top level system architecture of the Joint Surveillance Target Attack Radar System, or Joint STARS. The actual architecture includes many more computers, many different data busses, and a large number of other components (not shown in the figure) to perform its demanding mission. The result is a large, complicated system that makes it difficult for developers to consider the many different aspects of architecture.

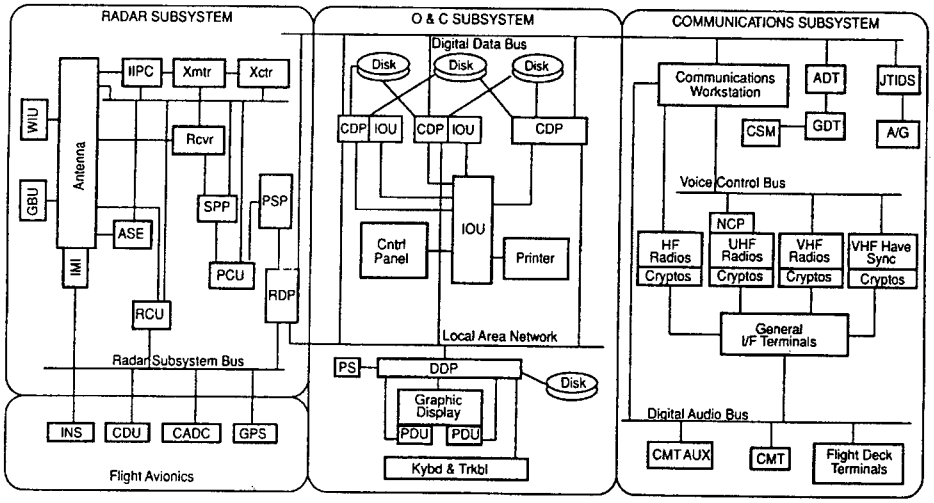


Figure G-12 Joint STARS System Architecture

At the same time, the larger and more complicated the system, the more important good structure becomes. Developing and maintaining structure may be very difficult in a system of such complexity, but the rewards for doing so are even greater. These rewards include higher quality during the initial development, lower life cycle software costs, and the increased likelihood that the system will remain in operation far longer (due to its greater flexibility and ease of upgrading). Furthermore, the reuse of known and expandable architectures will reduce the amount of new software that has to be developed and increase the quality of the systems that use them.

ARCHITECTURE: The Waiting Solution

Technical Focus

At the start of a development program, when considering architecture pays the greatest dividends, the technical focus in the typical DoD program is often not on architecture. Rather, functional and performance requirements are generally focused on by both DoD and the contractor (refer to Figure G-13). This lack of attention to architecture occurs because the government expresses its requirements in terms of specific, measurable system functions and performance requirements that matter to the immediate user, and not in terms of flexibility, which matters to the maintainer and next-generation user. Government standards, such as DoD-STD-2167A, require proof that a design satisfies all functional requirements, not that it is adaptable to change. Design documentation and reviews track individual system and software components, with less attention on the overall architecture until the components are integrated.

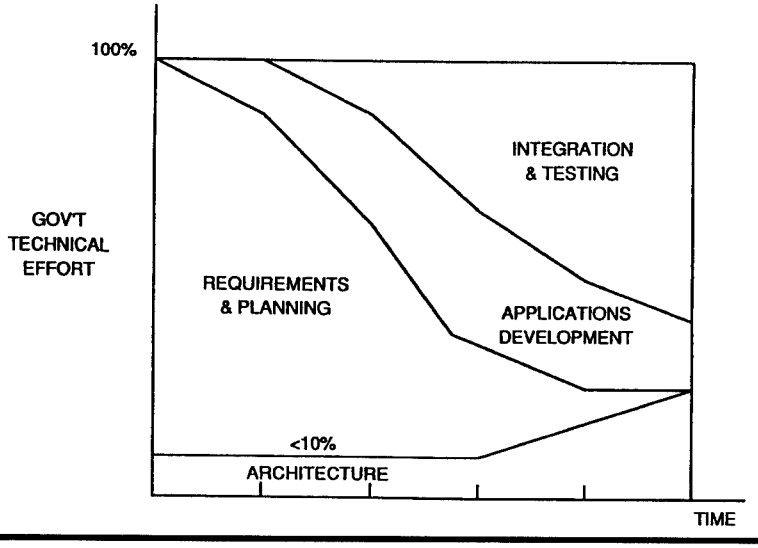


Figure G-13 Technical Focus (Estimated)

As the figure shows, the failure to consider architecture throughout the program's development has serious ramifications as time goes on. The performance and control problems described earlier begin to mount, and the contractor is often forced to call on Red Teams and other desperate measures to modify the original architecture. Since it is done in haste and then only to allow the product to meet the specifications, this last-minute change in architecture does little to ensure the necessary efficiency and flexibility, and usually results in further degeneration of the basic design.

Faulty Emphasis

Both government and industry typically put almost all their efforts into the initial performance and functionality of a program in spite of the fact that these will change substantially over the life of the system. At the same time, there is a near-total lack of attention to an architectural baseline that would form a stable foundation for incorporating the system's

changing requirements. What we do ask for does not address the important architectural issues. For example, we state that the system shall be modular but don't state a good way to partition it into modules that will allow future expansion and change.

We also specify requirements for system growth in an ineffective way that does not relate to operational capabilities such as adding new message types or increasing message traffic. Timing and sizing margins — for example, half the time and twice the memory — cannot ensure that the resources provided are allocated in such a way that they can be used to meet future requirements. With the advent of distributed systems, timing and communications bandwidth margins become important in providing for future growth. The government needs to assure that growth is expressed in operational terms, and not just in physical terms.

Because of the government emphasis on meeting immediate requirements within schedule and cost, even industry perceives that the government is not seriously interested in controlling maintenance costs. In a 1990 Air Force Scientific Advisory Board study of software maintenance, 123 businesses were asked what the government thinks is important when awarding software contracts. Their view of the government's stress on cost and system performance, rather than long term maintenance, is readily apparent.

Overall project cost	6.2 out of 7
Proposed product performance	5.5
Contractor experience in area	5.5
Timeliness	5.3
Last contract an advantage	4.8
Project software development cost	4.6
Contractor software capability	4.4
Ease of software maintenance	**3.4**
Software maintenance cost	**3.3**
Software portability	2.9

Commercial Architecture

It has recently become evident that commercial software users have become more concerned with architecture. As users become familiar with vendors' capabilities, their expectations increase. In turn, many software vendors are now providing software interface standards that enable interoperability across different computer hardware families and allow users to pick and choose among competing software vendors. These commercial architecture trends can do nothing but help DoD software efforts, because DoD is a large buyer of software and hardware that support these interoperability standards. Even embedded, special-purpose militarized systems rely heavily on commercial systems to assist in software support. The DoD cannot try to take the lead because these standards are driven by the commercial marketplace; however, the DoD can use to advantage the opportunities in the commercial market for open architecture standards. Unfortunately, these commercial standards cannot include the service standards that are heavily application-dependent; these must be left to the application designer to establish and implement.

Availability of Tools

The commercial market is also in the lead in providing tools that support the designer in generating and documenting architectures. There are tools to enable developers to analyze the linkage between different software modules, the control flow, the flow of data, and the timing of the various operations, and to assess and improve architectures. Many tools can only perform their analyses after the software is already written. These tools can still be used to understand what has been developed and to evaluate how easily it can be modified, before it is fielded or later. The investment may be small, and the potential payoff large. The following table lists some representative examples of available tools.

Name	Vendor	Analyzes
Logiscope	Verilog	Module structure, path coverage
ACT/BAT	McCabe	Flow graphs, structure graphs
ADAS	CADRE	Dynamic behavior, timing
STATEMATE	i-Logix	Structure, dynamic behavior
CPN	Meta	Dynamic behavior, simulation
Adagen	MarkV	Ada static structure, dynamic behavior
CARDtools	Ready	Timing threads
TAGS	Teledyne, Brown	Dynamic behavior, simulation

The government must acquire these tools and use them if it is going to buy architectures and understand them. In addition to commercially available tools, project-specific tools can improve the productivity of software development for a specific architectural design. Referring to the CCPDS-R program again, the contractor developed a tool to automatically generate the communications software that linked applications. The applications programmer needed only to list the elements of data that were required from each application and were necessary to each application. The tool used this information to generate efficient and correct communications following a standard pattern.

RECOMMENDATIONS: Finding and Applying Architecture

Good architecture potentially can provide significant cost savings as well as greatly increased system flexibility. To obtain these benefits, we must put architectural requirements in system specifications, emphasize the early satisfaction of these architectural requirements, give contractors incentives to use proven architectural concepts, and control the architectural configuration over the life cycle of the system. We believe this can be done.

System Specification

Since contract requirements drive the entire development of a system, the surest way to ensure adherence to a sound architecture is to put architectural requirements in the system specification. This does not mean that the specification will define the exact architecture to be used, but rather that it will specify what the architecture is to do. In cases when the application domain is well understood and a sound architecture is already available, the government may find it in its best interest to be more restrictive than in other situations lacking such a clear precedent.

To specify accurately the criteria architectures must meet, we must also determine how to qualify them. There are few measures of system designs that accurately predict flexibility and expandability. We will have to depend on a combination of techniques, including demonstrations that the system can be modified as well as analyses of features of the architecture. We are beginning to establish relationships between measurable features and rate of errors as well as ease of change. As Figure G-14 shows, the more calls a module makes on other modules, the more errors occur.

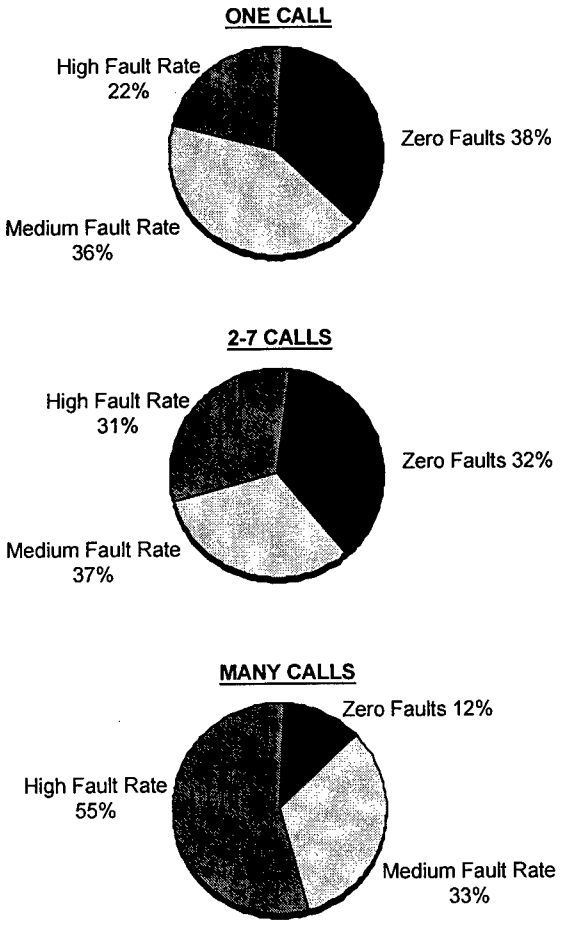


Figure G-14 Effect of Control Structure on Errors

Early Satisfaction of Architectural Requirements

To reap the maximum possible benefit from architectural requirements, we should specify that contractors cannot write large amounts of applications software until they have developed an architecture that the DoD has evaluated and approved. The only applications software that would be written before this point would be that necessary to help evaluate the architecture and reduce other serious risks, not to perform the actual task at hand. We can no longer afford the risk of developing architecture and applications concurrently; on the other hand, if contractors have successful architectures and control procedures that they have used before, they can use them again. In fact, the quality and effectiveness of a previous architecture as well as the tools available to support development of applications within the architecture should be an important factor in the selection of contractors on a program. We should also control these architectures after we evaluate and approve them. Changes would be weighed against the need for future flexibility throughout the life of the program.

Contractor Incentives

Contractors will have to be given incentives to change from their current emphasis on meeting immediate requirements to a longer term view. They will have to set up their own controls to keep applications software writers from corrupting the architecture; in other words, during development, contractors will have the architecture under configuration control. Rules and standards have to be defined as part of the architecture. Tools should facilitate the integration and modification of components within the architecture so we know that the standards of the architecture are observed. Contractors who have good architectural awareness should be treated better than those who do not. The development community needs to start working on architecture with the software maintainers to ensure that we deliver to them whatever is necessary for them to sustain and use the architecture.

Getting Started

It is recommended that the DoD put together a government and industry team to develop specification and contractual language for buying architectures. Approaches for evaluating and testing architecture need to be agreed upon as well. We are confident that this can be done and we have begun to develop an example. This team should also see that we use the experience that we have acquired on programs to determine what the contractors and the government have done to create good architectures, and to define the criteria for evaluating architectures.

We also need to consider buying single architectures for closely similar clusters of systems to reduce the cost of buying and maintaining unique architectures for each. For existing systems, we must work to introduce architectural improvements without disrupting operational use of the systems. It is crucial that industry participate as part of the team that would create the specification language and evaluation criteria. The insight of a joint government-industry working group on architecture will be of considerable benefit to the DoD during this time of changing missions and increased need for flexible systems.

REFERENCES

Arthur, L. J., Software Evolution, the Software Maintenance Challenge, John Wiley & Sons, New York, 1988

Card, D. N., Measuring Software Design Quality, Prentice-Hall, Englewood Cliffs, New Jersey, 1990

Day, R. A., "History of Software Maintenance for a Complex US Army Battlefield Automated System," Proceedings of the Conference on Software Maintenance, *IEEE*, New Jersey, 1985

DoD-STD-2167A, *Defense System Software Development*, 1988

Lehman, M. M., and Belady, L. A., Program Evolution — Processes of Software Change, Academic Press, New York, Academic Press, 1985

Rock-Evans, R., and Hales, K. Reverse Engineering: Markets, Methods, and Tools, Ovum, Ltd., England, 1990

United States Air Force Scientific Advisory Board, *Report of the Ad Hoc Committee on Post-Deployment Software Support*, US Government Printing Office, 1990

TAB 2

A New Process for Acquiring Software Architecture

Barry M. Horowitz, PhD
Thomas F. Saunders
Matt Mleziva
ESC-TR-94-207, September 1994

PREFACE

This paper is the product of a substantial amount of thinking and work on the part of many contributors. Without their help, the topic might have been covered with an extensive, elaborate, but awkward discourse, or it might have been discussed at great length, without coming to any specific implementable recommendations. The quality of contributions from within the government and from the six industrial organizations who participated in developing the concepts and preparing the paper are greatly appreciated. The primary contributors were:

ESC	MITRE
Col. T. Mackey, ESC/AL	A. Buchanan
Boeing	J. A. Clapp
J. H. Hanson	S. W. Dardinski
R. L. Roe	H. G. Goldman
GTE	R. D. Haggarty
C. E. Ellingson	E. J. Hammond
J. Perry	C. G. Hanley
J. Roder	M. R. Kurland
LORAL	C. W. McClure
J. Stanfield	R. Platcow
PARAMAX	H. W. Sorenson
R. Irwin	**Raytheon**
D. G. Perry	T. J. Haley
W. D. Sturm	**TRW**
	Dr. W. W. Royce

APPENDIX G Software Architecture

TABLE OF CONTENTS

SECTION	PAGE
INTRODUCTION: A Process to Control Software Architecture	G-21
BACKGROUND: The Importance of Architecture to Software Flexibility	G-22
Cost Considerations of a Flexible Architecture	G-23
SOFTWARE ARCHITECTURE DEFINED	G-24
PREPARING A REQUEST FOR PROPOSAL	G-25
System/Segment Specification	G-25
Vision Statement	G-25
Statement of Work	G-26
Contract Data Requirements List (CDRL)	G-27
Instructions for Proposal Preparation	G-29
Evaluation Criteria	G-30
ACTIVITIES AFTER CONTRACT AWARD	G-31
Demonstration/Validation Effort	G-31
Modeling Efforts	G-31
Design Reviews	G-31
Documentation and Configuration Control	G-31
ATTACHMENT I — Statement of Work	G-31
ATTACHMENT II — Proposed Amendment of Data Item Description for the System/ Segment Design Document	G-34
ATTACHMENT III — Proposed Amendment of Data Item Description for the Software Development Plan	G-35
ATTACHMENT IV — Instructions for Proposal Preparation	G-36
GLOSSARY	G-37

INTRODUCTION:
A Process to Control Software Architecture

Military systems must be able to change to accommodate new mission needs, threats, and technology. At the same time, military systems are being developed with budgets that are more and more limited. We need to find ways to keep costs down and to maximize adaptability — for new systems and for extensions to existing systems that must be preserved. A system's software architecture is a key determinant of the system's adaptability. A well-conceived and well-maintained architecture allows reusable components to be included in the original development, custom components to be smoothly integrated via standard interface protocols, and improved components to be incorporated as replacements or enhancements are needed. To be useful, the software architecture must first be articulated and include provisions for change; second, it must be controlled and maintained throughout the system's life cycle.

In this paper, we define a process that can be used to ensure that system acquisitions include attention to the software architecture. Attention to software architecture begins with the very first discussions of the system's scope and concept, and extends through system maintenance. The periods in a system's life critical to establishing and retaining a good architecture extend from formal notification to industry of the government's need for the system, through the evaluation of industry's response, the sequence of design reviews during the contractor's design and implementation, and long-term maintenance of the operational system. Before describing the process, we must first identify how to describe software architecture. In general, the attributes of software architecture include:

- Software partitioning,
- Flow of data,
- Flow of control,
- Timing and throughput relationships,
- Interface layering and protocol standards, and
- Hardware/software allocation.

Although these attributes may overlap with concepts considered to be *system* architecture, regardless, these attributes need to be established before software architecture can be evaluated or controlled. The specific content of what is controlled under software architecture must be determined for each program. Similarly, the level of detail contained in descriptions of the software architecture attributes, as well as distinctions between architecture and design, must be determined for each program. Attributes should be considered architecture when they express relationships that contribute to the system's long-term tolerance to changes; they should be considered design when they are implementation-specific.

We provide some background behind the initiative to have architecture defined and preserved in modern acquisitions,* and we define the term *software architecture*. We also provide information that can be used to structure the package the government uses to solicit proposals from industry for systems that strongly depend on computer software, and we offer guidance for monitoring the execution of a software development contract.

We believe that following the process described here for acquiring software will result in systems that better meet user needs by encouraging the development of more maintainable, flexible, extensible software architectures. The acquisition of software architectures will be most effective when there is cooperation between the government and industry, and attention applied by both towards the goal of building systems that can accommodate change.

BACKGROUND: The Importance of Architecture to Software Flexibility

One of the prime benefits originally attributed to the use of software is its inherent flexibility. Unfortunately, for a variety of reasons, the potential benefits of such flexibility have been lost in many systems. In fact, users identify the lack of flexibility and adaptability as two of the major disadvantages of recent systems. Users have been forced to continue using outdated (and sometimes improper) procedures solely because of the expense required (time as well as money) to modify the computer software. The rigidity of some software has even dictated that obsolete hardware be retained because of the difficulty of moving the software to newer machines.

If a system is to have a long, useful life, the allocation of software capabilities to components may need to be altered. For example, technology advances may make available specialized hardware that can substitute for particular software components, new software algorithms may replace hardware devices, or future mission capabilities may be predicted. Buyers should not only anticipate such possibilities, they should provide bidders with an indication of future needs via a vision statement. Likewise, bidders are encouraged to propose structured architectures with appropriately selected components so that the system can be expected to have a long and stable life. Bidders must address how one or more proposed architectures provide for the prospect of a long system life.

With the anticipated duration of modern systems, it is vitally important that systems be able to evolve as application needs change and as users come to appreciate the potential capabilities of systems. Although at the outset of a systems development project the designers

are able to describe desirable characteristics and a logical structure, by the time the systems are delivered their structures are too often constrained and awkward. Instead of receiving well-organized systems with clean interfaces, users receive systems whose distribution of functions appears contrived.

It has been observed that the original organized structure becomes distorted as the detailed designers choose expedient solutions to challenging implementation problems. As these expediencies compound, the software architecture of the delivered system becomes more and more convoluted. This convoluted structure makes it nearly impossible for the implementers or maintainers to find a logical point for introducing needed functions, and leads either to prohibitively rising costs or to ossification and eventual discard of the system. Such an end to well-conceived systems is consistently encountered because modifications, sometimes extensive, are inevitably needed during the course of a system's operational life.

To obtain the benefits of flexibility within a system, the software's structural attributes, which are present at the beginning of the design thought process, must be captured. These attributes are typically called "*architecture*," although the subtle distinction between architecture and design varies among practitioners. What is important is that a management process be established for preserving the architecture. This process must ensure that:

- A vision is established and documented that conveys a sense of direction for future capability growth;
- A structure is defined for the software, and is stable before decisions crucial to the implementation are fixed;
- The structure and its rationale are formally recorded for the implementers and approvers of the design; and
- The structure is revisited and reaffirmed, or modified in a controlled manner and preserved, throughout the life cycle.

For the process proposed here to work effectively, the government and contractor must designate people to oversee the architecture and ensure the system implementation preserves the architecture's desirable aspects.

Cost Considerations of a Flexible Architecture

Cost is sometimes raised as a concern about trying to maintain the integrity of the architecture structure. Although a software acquisition policy may emphasize software architecture, there is a system architecture tradeoff to be made. Hardware capabilities and software functions may need to be included that were not in the initially specified requirements but are necessary to accommodate changes predicted for the future. Offerors and buyers must consider the long-term payoff for preserving software architecture. With the falling price of computer components, it is preferable in most situations to spend more money on hardware rather than sacrifice the system's extensibility because of a shortsighted compromise of the software architecture to reduce hardware costs. In fact, an architecture that allows reuse of software components or use of commercial software products may lower the system's total cost. Appropriate consideration of the full life cycle cost consequences must be included in any decisions regarding the generation, preservation, or abandonment of a software architecture.

SOFTWARE ARCHITECTURE DEFINED

Software architecture refers to the fundamental structural attributes of a software system. Software architecture is a top-level definition of a software design that is defined early in a system's life cycle. It is the result of system design activity to synthesize a software system that will support the system's functions; be in concert with a synthesized hardware system; be responsive to imposed developmental, environmental, and operational conditions; and be demonstrably supportive of a vision for growth and change. The software architecture defines the software components that will provide the required algorithmic functions, their interfaces, and an underlying execution concept for orderly and efficient accomplishment of the algorithmic functions. As design progresses, the architecture captures more specific information, such as a software system execution model that includes specific operating system selections, specific interface and communication protocols, and specific multiprocessing and multiprogramming paradigms.

The distinction between software architecture and software design is not absolute. However, design is considered more detailed; it is a realization of the architecture into a product that fulfills the system requirements. The essential properties that must be described to express a software architecture include:

- **Partitioning software into components**. These should be the major software entities that will be recognized and manipulated by the software developers as the natural partitions within the software. In this context, "*component*" is used in a general sense and does not specifically refer to Computer Software Components (CSCs). Components may be determined by grouping algorithmic functions, objects, and reusable components such as operating systems and database systems, or by any other scheme appropriate to the proposer's development methodology.
- **Flow of data.** This should reveal the mechanisms for managing the flow of data through the components. It should show the major data paths between and among transformation processes as well as to/from data storage. Where the flow of data is controlled by table-driven design, the table data structures and their rules for modifying data flow should be described. It should define data structures and how file management and database management structures will be used.
- **Flow of control.** This should reveal the mechanisms for managing the flow of control among the components. It should show the major execution sequences, where execution sequences may be asynchronous or parallel, and how synchronization is managed. Where execution control uses data-driven design, the mechanisms for accessing the table and managing execution should be described. It should also show how anomalous conditions such as error handling and exception conditions that may dynamically alter the flow of execution are managed.
- **Critical timing and throughput attributes.** The architectural devices used to manage critical timing events, interrupts, throughput demands, and data buffering should be described.
- **Interconnection layers, standards, and protocols.** The layered view should show the grouping of software components into layers containing collections of services whose interfaces to the rest of the system are defined. It should specify the rules for allowable interconnections among the layers. The use of standardized interface protocols and bindings within and between layers, such as SQL, GOSIP, Motif, OSI, IEEE 806.X, etc., should be indicated.
- **Allocation of software to hardware.** The mechanism for assigning software to specific hardware devices should be defined. Where critical design performance is determined by such allocation (e.g., signal processing software might need to be executed on custom systolic array hardware), the allocation should be defined.

APPENDIX G Software Architecture

The definition of what constitutes a "*good*" software architecture is not provided here. Goodness must be judged on a case-by-case basis during source selection and each time modifications or refinements to the software architecture are proposed during a system's life cycle. In general, architectures are judged to be good when they satisfy the objectives outlined in the government's specification and vision statement.

PREPARING A REQUEST FOR PROPOSAL

The government's expectations about a new system are conveyed to industry as a Request for Proposal (RFP), which comprises a number of documents. For software architecture information to be adequately evoked, the RFP package must include specific information that traditionally has not been included in the RFP. The package must make clear to potential bidders that the government's assessment of the proposal will include assessing the software architecture.

System/Segment Specification

The System/Segment Specification (SSS) provided with the RFP package provides functional and performance requirements that must be satisfied by the contractor. The SSS also contains requirements, such as interface standards, language constraints, fault tolerance, and security, that influence architecture and must be considered by the contractor when the architecture is being defined. The SSS should include the requirements necessary for the system to meet the user's known needs. These may include architectural requirements when specific (testable) flexibility and extensibility requirements are known. However, the SSS should not be developed so as to dictate a particular architectural solution. Rather, it should emphasize the need for long-lived viability of the proposed architecture. If an acquisition has a requirement for exact interfacing with (or replacement for) an existing system, then the architecture may be constrained to the extent that architectural properties constitute a legitimate requirement, such as conforming to standards or using standard components. Even in this case, though, the SSS should not impose unnecessary restrictions on the architecture.

Vision Statement

In this new process for acquiring software architecture, acquisitions or developments should include a vision statement in the RFP package. The capabilities described in the vision are not prerequisites for the successful proposal; however, architectural flexibility sufficient to accommodate those capabilities is sought by the government. The SSS contains known requirements the delivered system must fulfill; the vision statement describes new capabilities the system might need to provide in the future.

In the vision statement, the user should provide, to the extent possible, potential future changes in threats and in the role and mission of the system. The acquisition organization should integrate the user's vision of future changes with other potential changes, including technology advances, and prepare the vision statement. Examples of what might appear in a vision statement include new functions (e.g., provide routing directives in addition to tracking targets), anticipated technological improvements (e.g., provide larger or better displays), and radically innovative technology improvements (e.g., direct input through voice recognition). The architecture-relevant topics in the SSS and the vision statement together create the basis for evaluating proposals.

If there are known requirements for flexibility, maintainability, and future growth that must be met in the delivered system and are firm when the RFP is issued, then they must be a part of the SSS (e.g., if the system must accommodate growth and changes in the message catalog), and the software architecture must meet them. The vision statement adds another dimension to

the SSS by helping the offeror make tradeoffs in selecting an architecture that can easily adapt to fulfill potential future requirements at minimum risk to the government, the user, and the maintainer.

A vision statement is important to the architecture process because major system benefits may emerge if the chosen structure is capable of handling different demands. With a robust and flexible underlying structure, a system can absorb tasks different from those identified to meet the initial deployment. The vision statement requires thinking beyond the immediate objectives of the users in acquiring a new system. It should be linked to the user's long-range planning and speculation (10 to 20 years), or to potential alternative users, and should not be restricted by predefined allocations of functions to organizations, systems, or persons.

Statement of Work

The **Statement of Work (SOW)** must reinforce the importance of the architecture in the acquisition by identifying tasks related to documenting the architecture, keeping the documentation relevant as the design progresses, and validating that the architecture is used in the actual software design and implementation and that it allows satisfaction of system requirements.

The **System/Segment Design Document (S/SDD)** is the vehicle for expressing and controlling the architecture. Consequently, a specific task must be included in the SOW to ensure that the S/SDD is produced, updated, and maintained under configuration control by the contractor.

The **Software Development Plan (SDP)** is the vehicle for defining the software development process the contractor expects to use. The SDP must include a description of the processes and the tools or software engineering environment the contractor will use to ensure the architecture is preserved by the software implementation. Consequently, a specific task to develop and maintain the SDP must be included in the SOW.

Although this process is devoted to preserving architecture, it is not devoted to preventing its refinement or modification. Consequently, engineering analysis tasks to evaluate, refine, or demonstrate the validity of the software architecture must be included in the SOW. Results of these analyses should be discussed within design reviews (e.g., at the software walkthrough) and at presentations to the government. These engineering analysis tasks should reveal how the architecture will behave in the proposed application, and how it can be generated, modeled, or prototyped with the contractor's proposed software development process and environment. These tasks should commence prior to the **System Requirements Review (SRR).** Because of the need to perform these studies, the schedule for the contract must allow time before the SRR. The results of the analyses must be presented to the government at the SRR, and again prior to the SDR, as drafts of the architecture portion of the S/SDD. Both the architecture and the analyses that provide the rationale should be documented in the S/SDD and used to reach an agreement between the developer and the government on the architecture. At SDR, the architecture portion of the S/SDD should be placed under configuration control. A task must be defined to conduct at least one Technical Interchange Meeting (TIM) for the contractor and the government to review and agree on the architecture between SRR and SDR.

Additional engineering tasks should be included in the SOW for validation of the architecture itself, and preservation of the architecture throughout the implementation. At a minimum, updates to the S/SDD should be provided at each formal design review (after the SDR, PDR, and CDR), at any time an architectural change is proposed, and at **Test Readiness Reviews (TRRs).**

Most software architectures involve a complex structure that is difficult to understand and therefore difficult to predict. Early in the formulation of the software architecture, a model or prototype is needed; this should be an executable model that could be simulated to yield insights into the software component interface relationships, data flow, execution flow, and timing and sizing performance of the ultimate system. Where complexity of the ultimate system

warrants (to be decided case-by-case) a specific task to develop, document, and use such a prototype/model should be included in the SOW. The prototype/model will validate that the architecture and the implementation coincide and that the architecture can meet system and software requirements. The prototype/model is not required to be an extract of the final version of the system, although it could be; rather, it is prepared to allow meaningful examination of the architecture. Results of analysis of the prototype/model should be delivered to the government at each design review. In some cases, the prototype/model may also be a deliverable for use during the support phase of the system.

Within the context of any implementation, the risk of failing to meet some of the requirements must be carefully managed. To ensure that architectural features are not needlessly abandoned when attempting to resolve other problems, the contractor should be required to analyze the proposed system for features likely to pose implementation difficulties or increased costs. Any requirement, capability, feature, or option that, during the course of the contract, is discovered to fall into the realm of a potential risk to preserving the architecture and its attributes must be identified to the government and a risk management action proposed.

A task must be defined to conduct tradeoff studies whenever a change to the architecture is contemplated. The tradeoff studies should involve simulation modeling or demonstration-based results that provide a quantifiable impact to the architecture attributes. The results of such tradeoff studies must be presented to the government, which must concur before any efforts are initiated that implement the changes. Government concurrence should be managed according to the procedures for managing the S/SDD configuration. Attachment I contains sample wording that should be in the SOW. However, the exact definition of the tasks to be performed must agree with the goals of the system under development.

Contract Data Requirements List (CDRL) System/Segment Design Document

The S/SDD is the vehicle for defining the design of a system/segment and its operational and support environments. Under this new process for acquiring software architecture, the architectural portions of the S/SDD will be expanded. Further, architecture information will be requested from the contractor at the time of the proposal, at SRR, and before SDR. The proposal submittal may be in the form of a draft of the architecture material from the S/SDD or it may be separately formatted. In either case the information necessary to understand and evaluate the software architecture must be presented.

In accordance with DI-CMAN-80534 (S/SDD DID) and Attachment II of this paper, a description of the architecture should be provided with the proposal, a preliminary description should be provided at SRR, and an approved architecture description should be provided before SDR. Obviously, the early submissions will have less detail defined than the later ones. For example, specific choices for software decomposition are not expected in the proposal. Only the components that represent significant software capabilities should be identified as software architecture components (e.g., operating system, database management system, signal processing system, or other breakdown of components consistent with the developer's proposed methodology, such as functional decomposition or object-oriented design). The contractor must also be required to submit revisions to the S/SDD whenever an architecture modification has been approved and the architectural information has become outdated. In general, the software architecture information should be contained in Section 3.4 of the S/SDD.

Many diagrammatic or textual representations are possible to emphasize software attributes such as data flow, execution flow, layering, or interface protocols. The specifics of the notation and format of the information are not as important as recording all the architectural information; for that reason, one or more contractor-preferred representations may be selected to complement one another when assembling an architecture description. Regardless of the forms used, to be acceptable the architecture representation must cover the essential properties listed

in our definition of software architecture. In addition to the architecture description, the S/SDD must be supplemented with explanatory material for the architecture. The contractor must include the following rationales for:

- **Structure of software**. Explain why the partition of software components was selected, emphasizing why it offers tolerance for changes in later stages of the system's life cycle. Similarly, explain why architectural features such as table-driven design were selected and what attributes of tolerance for change, e.g., flexibility and extensibility, are provided by the chosen software structure. Mechanisms for managing the flow of data and the flow of execution should also be described, with the rationale behind how they were selected.
- **Rules for interaction of system components**. Explain why particular standards or de facto standards (e.g., Motif, Windows, POSIX GOSIP, Hewlett Packard New Wave Object Management Facility) are being adopted. Provide the rationale for how an architecture model or profile was selected and how interconnection rules or layered protocols will be enforced in the architecture.
- **Structure of the hardware and software**. Provide the rationale for the choices made to aggregate software components on processors or to distribute them on separate processors.

In addition to discussing how the architecture relates to the needs of the current system, the S/SDD should address how the architecture will handle new capabilities and features as noted in the vision statement. The architecture description should include provisions for exploiting migration paths associated with improved versions of nondevelopmental item (NDI) products, and it should describe the limitations or capabilities to migrate to improved custom or NDI software. It should also explain what aspects of the software are substantially enhanced by the proposed architecture, for example:

- Software testability
- Software maintainability (e.g., isolating and correcting software malfunctions, and
- Software extensibility.

Attachment II contains a candidate rewording of Data Item Description (DID) DI-CMAN-80534 to direct the contractor's preparation of the S/SDD. Caution: when preparing the description of the architecture, the distinction between design and architecture needs to be considered. Since the architecture will be placed under configuration control relatively early in the contract life cycle, implementation detail information is best left out. The mechanisms, rules, standards, and generic structural properties should be included. Specific, low level, component-unique information that is not part of the main structure of the system should not be captured. The distinction between what is design detail and what is structural attribute needs careful consideration by both the government and the contractor. Consequently, the words used to tailor a DID for a specific contract should be carefully considered.

Software Development Plan. The software architecture implementation is carried out by the creation of many software components, modules, and routines. The creation steps must be guided by a process, defined in the SDP, that is fully cognizant of the importance of the software architecture. Therefore, not only is an SDP required from the contractor at the beginning of the contract, it must be resubmitted whenever changes are made to the development process. The SDP must describe the development environment and any tools used to ensure and/or verify the preservation of architectural features during development. The degree to which tools can be used to enforce architectural decisions should be highlighted. Whether or not automated tools are used, the SDP must explain the procedures by which architectural constraints are enforced throughout the design, coding, deployment, and maintenance phases of the system. Information to be included in the SDP includes:

- The extent to which software components are NDIs or reused, and guidelines for determining the suitability of reused or NDI products.
- The process (sequence of activities) and tools that will be used to manage the software architecture, including:
- Generating software in accordance with the rules of the architecture;
- Documenting and controlling the architecture and changes to it;
- Verifying that the architecture will meet its functional requirements and its requirements for adaptability, extensibility, etc.;
- Predicting that the system will meet its performance (timing) requirements;
- Controlling conformance of the software design and software implementation with the architecture;
- Determining how and when the architectural model or prototype will be executed; and
- Creating an organizational structure to support architectural development, preservation, and verification.

Attachment III contains a candidate rewording of DID DI-MCCR-80030A to direct the contractor's preparation of the SDP.

Other documents. There may be other documents (e.g., model/prototype description, timing and sizing reports) that are influenced by architecture. For each one, the DID should be amended to express the need for preserving software architecture.

Instructions for Proposal Preparation

The Instructions for Proposal Preparation (IFPP) must remind bidders about the presence of the software architecture policy and the government's emphasis on software architecture definition, evaluation, and preservation. The IFPP should reiterate the relationship between the specification and the vision statement, the parameters and attributes to be included in the software architecture definition, and the differences intended between architecture and design. It must provide some form of completeness criteria that will enable contractors to know how much of the vision statement characteristics need to be discussed in a proposal. It must also provide for enumerated vision attributes so proposals that offer solutions covering more of the vision can be distinguished from those covering less.

Bidders must be informed of the need to provide a description of the software architecture. This will be a tailored subset of the information contained in the S/SDD. The S/SDD material should be of sufficient depth for the relationship between proposed architectures and the specification and vision attributes to be understood by the buyers. This architecture-relevant information should have a page limitation (e.g., 50 pages, including cover, index, and supporting text). The bidder must adequately describe the architectural approach to be used and must show how a proposed architecture will support the system requirements by including a rationale for the major architectural properties the offeror proposes. Further elaboration on the proposed architecture and its rationale may include descriptive examples of previous systems that used a proposed architecture, results of demonstrations from previous contracts, or results of simulations and models available for a proposed architecture.

The IFPP must also inform bidders of the need to submit a description of the software development processes and the contractor's plan for developing software. This material should define the processes and tools used to support creating and preserving the software architecture. The technical proposal should provide the rationale behind why and how those software development processes and tools ensure preservation of the architecture. Where appropriate, examples from past experience should be cited.

A presentation of the techniques used in a previous project should be encouraged to lend credibility to a bidder's representations about the approach to emphasizing architecture that is proposed. An ideal way of combining past experience and the current effort would be to have the bidder demonstrate the planned system architecture and tools through execution of a high-

level model of a proposed architecture during source selection. On a case-by-case basis, the acquisition organization should determine whether or not it is appropriate to expect or require proposers to be prepared to include a demonstration as part of source selection. The bidders should be encouraged to show how their model reveals whether an architecture supports the specification and vision. For example, the model could show the ability to port software components to multiple hardware platforms, to modify components via the use of computer-aided software engineering (CASE) tools, or to replace one commercial-off-the-shelf (COTS) product with another.

Because the ultimate success of the architectural approach to the system acquisition will be determined by the designers and implementers, the role of the technical leaders of the design team is of much interest to the government. The bidders are encouraged to describe whether a single architect or architect team will be appointed, the scope of this person's or team's authority, and how the architect or team is expected to interact with the rest of the project management team. Examples of previous projects that used the proposed management structure would validate the likely success of the approach. Attachment IV contains sample wording that can be used in the IFPP.

Evaluation Criteria

So that bidders will understand the importance attached to the architectural structure of a proposed system, Section M of the solicitation package must indicate that the proposed architecture, its development process, and its maintenance provisions will be evaluated. Further, it should state that failure to provide an adequate response to the architectural aspects of the proposed system will be deemed to indicate noncompliance with the basic solicitation. The actual factors and standards that the government will use in evaluating the proposals will be based on the statements in Section M, but will not be provided to the bidders. The evaluation criteria included in Section M should be based on a need to assess:

- How the proposed approach and architecture meet system requirements;
- To what extent the architecture can meet the potential long-term needs of the system described in the vision statement; and
- To what extent the proposer's software development approach assures the preservation of the software architecture.

Although these criteria are generic here, they must be made more explicit for the particular system being acquired. The evaluation criteria should be derived from the types of tasks or capabilities the new system may need to accommodate, and they should allow buyers to evaluate whether or not proposed architectures have a flexible underlying structure that will accommodate tasks different from those identified by the initial specification.

ACTIVITIES AFTER CONTRACT AWARD

After the system implementation is under way, the government and the contractor must jointly monitor the effort to ensure that the architectural base for the system is maintained. The SOW defines the appropriate tasks and, for the most part, the contract defines the activities after contract award. However, cooperation is considered essential both in selecting the architecture to be used and in preserving it. Therefore, all parties to the contract should be encouraged to establish open communications and candid evaluation of the degree of success being achieved with respect to defining a sensible architecture and preserving it. TIMs can facilitate communication prior to formal reviews.

Demonstration/Validation Effort

Engineering analyses should allow the contractor and government to identify and quantify the parameters that relate to imposing the chosen architecture on the system. They should be conducted from the outset of the contract to refine the architecture requirements and prepare the contractor and the government for architecture discussions at SRR and prior to SDR. They should also be conducted any time architectural modifications are proposed.

Modeling Efforts

A realistic (but economical) modeling effort should be part of any complex project so that probable performance can be predicted. Without appropriate modeling, the true consequences of preserving the architecture in the face of possible design deviations may not be understood. These modeling efforts should include the architecture aspects of the system.

Design Reviews

As a part of each design review during development, the integrity of the architecture must be explicitly reviewed. Any proposed deviations from the agreed architecture must be explained. Execution threads must be presented as a means of showing that the architecture supports the intended functions of the system.

Documentation and Configuration Control

Between contract award and SRR, the contractor and the government will study the proposed architecture. Based on the results of engineering analyses done during this time, the architecture requirements will become stable. Between SRR and SDR, the contractor will prepare the formal S/SDD for review and comment by SDR. Upon submittal of the S/SDD, reflecting government comments, the architecture will be placed under configuration control. Thereafter, modifications may be made to the architecture but only after tradeoff studies have shown the necessity for change, and the government and the contractor agree to the change. The S/SDD should be updated to reflect the approved changes.

*Horowitz, B. M., *The Importance of Architecture in DoD Software*, The MITRE Corporation (M91-35), Bedford, Massachusetts, July 1991

ATTACHMENT I — Statement of Work

The following material is provided for placement in the SOW to define tasks associated with preserving the software architecture. These paragraphs are general in nature and must be tailored to include specific references to the CDRL. The specific tasks chosen and the degree of tailoring must also match the needs of each program. In particular, the scope and scheduled completion of the tasks must be integrated into the overall plan for the program. The software architecture tasks are intended to supplement those tasks that would be part of an acquisition effort and thus already specified in the SOW; it may be convenient to merge this material into the descriptions of other tasks.

x.x.x System Engineering — General

When performing all system engineering tasks, the contractor shall ensure that the software architectural attributes are considered. All trade studies, design decisions, and implementation actions that impact the software architecture will evaluate whether or not the system's tolerance for change is affected. In the event that it is affected, the contractor shall include the analysis that justifies modifications to the software architecture in terms of life cycle cost of the system.

x.x.x System Engineering — Analysis

Before the SRR, the contractor shall analyze the SSS and vision statement and identify requirements that are judged to be architectural drivers. The contractor shall host a TIM, with government and support contractor participation, to review the software architecture. The purpose of the TIM will be to review the requirements from the SSS being satisfied by the architecture proposed, to assess the ability of the proposed architecture to satisfy the vision statement, and to coordinate revisions of the architecture requirements such that a clear understanding of the software architecture and the criteria or constraints involved in its definition is established. The architecture shall be described in an update of the architecture portion of the S/SDD and shall be provided to the government at the SRR. A formal submission of the architecture description in the S/SDD shall be submitted and placed under configuration control at SDR. [DI-CMAN-80534]

An explanation of the architecture of the system shall be included as primary presentations at the SRR, the SDR, all software and hardware design reviews (SRR, PDR, and CDR), and the TRR. Associated with software architecture descriptions, the design process standards and tradeoff heuristics that were used as criteria or constraints by the contractor, or by the contractor and the government, for selecting the architecture shall be expressed in an Architecture Analysis Report. This report shall contain the essential information for later reviewers to understand why the architecture choices were made that led to the defined architecture. The intent is for later reviewers to be able to identify architectural features that are tied to design assumptions or constraints. In the event those assumptions are refined or the constraints are lifted, preservation of those architecture features may be reexamined. Also included in the report shall be a description of the procedures, tools, and training by which the government can maintain the architecture for the remainder of the planned system life after completion of the contractor efforts. This report shall be delivered at SRR, at SDR, and updated at any subsequent review whenever modifications to the software architecture are requested. [DI-MISC-xxxxx]

x.x.x Software Engineering

x.x.x Software Engineering — General

The contractor shall manage, design, develop, document, control, and qualify performance of the computer software to satisfy the performance, functional, and quality requirements of the SSS. In addition, to the extent technology allows, the contractor shall provide an architectural framework for the software that accommodates the flexibility and extensibility implied by the vision statement. "*The extent technology allows*" will be determined via TIMs where provisions for open system design, software reuse, incorporation of layered architecture reference models, and interface standards are compared against the goals of the program and the state of available commercial products, or the state of contractor-developed architectural products. The software architecture definition shall be governed by the long term cost-effectiveness goals to have the system under development be tolerant to change. Decisions vis a vis architecture selection or modification shall be made on the basis of requirements defined in the SSS, the vision statement, and this goal within the scope of the contract.

x.x.x Software Engineering — Architecture Change Analysis

At each milestone after the SRR (i.e. SDR, PDR, CDR, and TRR), the contractor shall identify any new enhancements or changes to the architecture that affect the long term tolerance for change in requirements, and provide reasons for the differences. Before these changes are presented at the next review, the contractor shall identify the changes at a TIM convened 20 days

prior to the review. The contractor shall update the S/SDD to reflect the changes as agreed within 30 days after the review. If any changes affect portions of the architecture already under government configuration control, the contractor shall be required to generate an administrative engineering change proposal (ECP) describing the changes and their rationales. [DI-MISC-xxxx]

x.x.x Software Engineering — Architectural Model or Executable Prototype

The contractor shall prepare an executable prototype, model, or initial version of the software architecture framework. This model shall demonstrate the flexibility, extensibility, and growth provisions anticipated for the selected architecture. It shall allow testing of the timing and throughput relationships, the interfaces to products using standard protocols, and the fundamental data flow and control flow sequencing, and it shall demonstrate the anomalous condition or error handling mechanisms to be used in the system. The contractor shall present the results of architecture modeling or prototype developments and demonstrations at the SDR, and the results and the model shall be updated at the PDR and CDR. The contractor shall submit an agenda for each demonstration and shall deliver a report after each demonstration to record any action items and decisions. [DI-MISC-xxxx]

x.x.x Software Engineering — Database Design

In the process of designing the logical and physical structure of the database, the contractor shall consider the effects upon the database of the selected architecture (and vice versa) and shall ensure that the database design is consistent with the architecture. The design of the database shall be presented as part of each design review at a level of detail determined by the maturity of the design effort at that time. The contractor shall deliver to the government a Database Design Document (DID-MISC-xxxx). A preliminary version of this document shall be delivered at the (first) PDR and a final version of this document shall be delivered at the (first) CDR.

At the SDR, the contractor shall describe the approach for implementing the database. The contractor shall identify and justify the need for any file management or database management software. The contractor shall discuss the partitioning of the database and shall identify benefits and drawbacks due to the planned database structure. Included shall be a description of the procedures, tools, and training by which the government can maintain the database consistent with the architecture for the remainder of the planned system life after completion of the contractor efforts.

x.x.x Software Engineering — User Interface Design

The contractor shall develop the user system interface (USI) to be consistent with the planned architecture. The impact of the architecture on the USI (and vice versa) shall be considered. At the SDR, the contractor shall produce a description of the sequence and timing of user actions and software responses to illustrate the compliance of the USI with the overall architecture. In particular, the handling of anomalous and error situations and the resultant movement of information to and from the USI shall be discussed. The use of previously developed software in the design or operation of the USI shall be described. The contractor shall provide information sufficiently detailed so that the government can determine that the USI design and the planned hardware and software implementations are consistent with the planned architecture. This information shall be delivered in the form of briefings and portions of the S/SDD. [DI-CMAN-80534]

x.x.x Security Engineering

The design of needed security features shall be accomplished by the contractor in such a manner as to preserve the architecture planned for the system. The impact of security features upon the overall architecture and design shall be presented as part of each design review at a level of detail determined by the maturity of the design effort at that time. No alterations to the security design shall be considered without a thorough determination of the impact upon the future of the system; alternatives that are in concert with the planned architecture shall be given greater weight than permitting changes to the architecture. The contractor shall deliver to the government a System Security Architecture Document [DI-MISC-xxxx] which identifies all components of the system that contain security-relevant functions; this document shall thoroughly describe the integration of security features into the design of the hardware and software.

ATTACHMENT II — Proposed Amendment of Data Item Description for the System/Segment Design Document

1. Amendment to DI-CMAN-80534 (S/SDD DID):

a. 10.1.1-10.1.5.3.4 No change.

b. 10.1.5.4. **System Architecture**. Replace the current wording with the following: This paragraph shall be numbered 3.4 and shall be divided into subparagraphs to describe the internal structure of the system and the software architecture. The system segments, HWCIs and CSCIs, shall be identified and their purpose summarized. The system-level relationships among the segments, HWCIs, and CSCIs shall be described. Paragraph 3.4 shall also identify and state the purpose of each external interface of the system. A system architecture diagram may be used to illustrate the system top-level architecture. There shall be subparagraphs that describe the top-level software architecture structure, timing, and allocation attributes. One or more software architecture diagrams may be used to illustrate the different software attributes that comprise software architecture.

10.1.5.4.1 **Software Structure**. This paragraph shall be numbered 3.4.1 and shall describe the major software entities that will be developed or integrated into the system by the software designers, the mechanisms for flow of data among the major software entities, and the mechanisms for flow of control among the major software entities. Software entities should be the major software entities that will be recognized and manipulated by the system developers as the natural partitions within the software. In this context, component does not specifically refer to Computer Software Components (CSCs). Components may be partitioned by algorithmic functions, objects, and reusable components such as operating systems and database systems, or by any other scheme appropriate to the proposer's development methodology. The entities defining the software structure should reveal the underlying structure of the software rather than hardware allocation or administrative and management controls that are often used as the criteria for defining CSCIs.

This paragraph shall also describe the mechanisms for managing the flow of data through the components. It should show the major data paths between and among transformation processes as well as to/from data storage. Where the flow of data is controlled by table-driven design, the data structures and their rules for modifying data flows should be described. It should define data structure and how file management and database management structures will be used.

This paragraph shall also describe the mechanisms for managing the flow of control among the components. It should show the major execution sequences, where execution sequences may be asynchronous or parallel, and how synchronization is managed. Where

execution control uses data-driven design, the mechanisms for accessing the table and managing execution shall be described. It should also show how anomalous conditions such as error handling and exception conditions that dynamically alter the flow of execution are managed.

Timing. This paragraph shall be numbered 3.4.2 and shall describe the presence of critical timing and throughput processes. The software architectural devices used to manage critical timing events, interrupts, throughput load balancing, and data buffering shall be described.

Interconnection layers, standards and protocols. This paragraph shall be numbered 3.4.3 and shall describe the software architecture rules for interconnecting layered software entities. It shall describe allowable interconnections among layers, and identify the use of standardized interface protocols or bindings such as SQL, GOSIP, Motif, etc.

10.1.5.5 **Operational Scenarios**. Replace the current wording with the following: This paragraph shall be numbered 3.5 and shall describe each operational scenario of the system. For each system state and mode, this paragraph shall identify the hardware and software entities that execute and the manual operations to be performed. A table may be provided to illustrate the states and modes in which each hardware and software entity executes and each manual operation is performed. In addition, this paragraph shall describe the general flow of both execution control and data between hardware and software entities while operating in the different states and modes. Flow diagrams may be used to illustrate execution control and data flow in each state and mode.

10.1.5.6 **Software Architecture Rationale.** This paragraph shall be numbered 3.6 and shall describe the selection criteria and constraints that drove the choice of software architecture. It shall include cross-references to requirements in the SSS, the vision statement, or derived requirements that govern the choice of software architecture.

ATTACHMENT III — Proposed Amendment of Data Item Description for the Software Development Plan

1. Amendment of DI-MCCR-80030A (SDP DID)

a. 10.1 — 10.2.5.11 No change.

b. 10.2.5.12 **Software Architecture**. This paragraph shall be numbered 3.12 and shall describe the development management provisions for ensuring the software architecture is preserved throughout the development life cycle. It shall describe the contractor's procedures and methods for establishing an architecture definition, ensuring software developers understand the architecture, and ensuring software developers follow design guidelines that enforce the preservation of the architecture.

c. 10.2.6 — 10.2.6.2 No change.

d. 10.2.6.2.1 **Software Development Techniques and Methodologies**. Replace the current wording with the following: This subparagraph shall be numbered 4.2.1 and shall identify and describe the techniques and methodologies the contractor plans to use to perform:
Software Requirements Analysis
Software Architecture Analysis and Definition
Preliminary Design
Detailed Design
Coding and Computer Software Unit Testing
CSC Integration and Testing
CSCI Testing

e. 10.2.6.2.2 — 10.2.12 No change.

ATTACHMENT IV — Instructions for Proposal Preparation

The following material is provided for insertion into the IFPP to elicit information needed by the government in the area of software architecture. These words are general in nature and must be modified to match the explicit needs of each program and to be consistent with the remainder of the IFPP and the selection basis (see Section M). The information given below is intended to supplement the information that would conventionally be in the IFPP. It may be convenient to merge the following material into the other information contained in the IFPP. In particular, offerors should not be required to submit additional copies or mere reformulations of information requested by other portions of the IFPP. Offerors should also be required to provide an index that would indicate the location of architecture-related information throughout the proposal.

x.x.x System Engineering

Describe the overall architecture of the system by indicating the partitioning of the system into major hardware and software components. Describe systems with which the contractor is familiar that have similar characteristics and/or are based on similar components. Explain the manner by which information is to be passed among components and the control of execution. Describe how the architecture selected satisfies the requirements in the SSS and accommodates the vision statement. Describe the system services that provide for communication (both control and data) among software processes and tasks, hardware processors, external systems and devices, and the USI. Any other information requested in the DID for the S/SDD, or from other sources, should be presented.

x.x.x Software Engineering

Describe the standards to be followed in the software design and identify any conflicts with the proposed architecture caused by use of these standards. Identify the tools to be used in generating the software. Describe how inadvertent and deliberate departures from the proposed architecture are recognized, reported, and/or prevented. Identify any software components previously developed (COTS or other NDI) and how they may have to be adapted to conform to the proposed software architecture. Describe how and when the execution sequence is determined; include the handling of interrupts, errors, out-of-normal situations, and response to timing constraints. Discuss how the execution sequence can be redefined.

x.x.x Database Engineering

Describe the proposed database architecture with particular emphasis on those attributes that contribute to or detract from the proposed software architecture. Identify any tools to be employed in developing the database design. Identify any available data management packages (COTS or other NDI) and whether the use of these packages affects the proposed software architecture.

x.x.x User Interface Engineering

Describe the proposed USI and associated methods for data display, data entry, sequence control, and user assistance within the architectural framework proposed for the system. Identify any rapid prototyping tools that will be used to support the User Interface Demonstration and how these tools will be used in a manner attuned to the proposed architecture. Describe how COTS/or other NDI software will provide the USI while preserving the proposed architecture. Describe how the results of USI demonstration activities will be incorporated into the design, development, and test of the system.

x.x.x Security Engineering

Describe the proposed security design and the allocation of functions to hardware and software components.

x.x.x Modeling and Prototyping

Describe how the architectural assumptions and conditions are to be imposed on the system models used. Indicate to what extent the performance models will be calibrated against other systems using the same (or similar) architectures. Describe how (and when during the design activities) the architectural model will be subjected to execution and to validation.

GLOSSARY

CDR	Critical Design Review
CDRL	Contract Data Requirements List
CSC	Computer Software Components
CSCI	Computer Software Configuration Item
COTS	Commercial-off-the-Shelf
DI	Data Item
DID	Data Item Description
ECP	Engineering Change Proposal
GOSIP	Government Open Systems Interconnection Profile
HWCI	Hardware Configuration Item
IFPP	Instructions for Proposal Preparation
NDI	Non-Developmental Item
PDR	Preliminary Design Review
POSIX	Portable Operating System Interface for Computer Environments
RFP	Request for Proposal
SDD	Software Design Document
SDP	Software Development Plan
SDR	System Design Review
SRR	System Requirements Review
S/SDD	System/Segment Design Document
SSS	System Segment Specification
TIM	Technical Interchange Meeting
TRR	Test Readiness Review
USI	User-System Interface

Blank page.

APPENDIX

H

How Should Military Software Be Documented?

Blank page.

APPENDIX H

How Should Military Software Be Documented?

Lewis Gray
Ada PROS, Inc.

ABSTRACT

Who is responsible for ensuring that military software is properly documented? This article says, both the acquirer and the developers of the software. The article explains and supports three points that are summarized by Figure H-1.

Competence

•

Normal Software Process

•

Additional Documentation Only If Justified

Figure H-1

First point, before awarding a software development contract, an acquirer should choose a developer that knows how to adequately document their software development effort and their software products. On the other side of the development relationship, before a developer begins to develop military software, they should be sure that they have a mature software process that is capable of properly documenting their software development effort and their work products.

Second, usually, the acquirer should acquire only the developer's normal work products because usually they will be the most appropriate information, in the most appropriate format and the most appropriate media for explaining the development effort and its results. For their part, developers should find development tools that will work well and learn how to use the tools well so they can document their software development effort and their work products in a way that is appropriate for understanding and overseeing their work.

Finally, as Figure H-2 (below) suggests, the acquirer should order additional documentation, supplementary to the developer's normal work products, only when the acquirer has strong evidence that the information is needed and that the developer would not provide it otherwise. For their part, developers should accept that there may be occasions when acquirers will need more information than they can obtain from the normal developer work products, and developers should be prepared to accommodate these reasonable, additional information orders from their customers.

APPENDIX H How to Document Military Software



Figure H-2

In the course of developing these points, the article touches on a wide range of topics including the basic case for software documentation, what documentation acquirers should receive, the situation when code is generated automatically by CASE tools, whether Ada 83 or Ada 95 source code alone is adequate documentation, how Ada software developers want to document their code, the role of software engineering standards in documentation, CASE tools that provide documentation, and benefits and dangers of relying only on normal developer work products.

The article is an extensive update of earlier versions published in *Ada in Europe*, the Proceedings of the 1994 EUROSPACE/Ada-Europe Conference (Springer-Verlag, 1994) and in the September, 1994 edition of the Department of the Air Force Publication, *Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapons Systems, Command and Control Systems, Management Information Systems* (STSC, Hill AFB).

1. Should Military Software Be Documented?

In an influential paper in 1970 that provides a rationale or an inspiration for much of the work to date on life cycle models and software documentation, Winston Royce posed the rhetorical question, *"how much documentation?"* and answered it as follows:

> *"'quite a lot,' certainly more than most programmers, analysts, or program designers are willing to do if left to their own devices. The first rule of managing software development is ruthless enforcement of documentation requirements."*[1]

Royce's emphasis upon documentation is still appropriate. However, today, we have a much broader notion of what documentation is. In 1970, documentation consisted entirely of paper documents. Today, it also includes electronic images of paper documents, diagrams stored in CASE tool databases, drawings stored in CAD/CAM files, hypertext, computer animations and simulations, video tapes, and more.

APPENDIX H How to Document Military Software

Royce argued for six documents: first, a software requirements document; second, a preliminary design document; third, an interface design document; fourth, a final design document developed in stages that, in its final stage, contains a description of the software design as it was built; fifth, a test plan and record of test results; and sixth, a manual of operating instructions. Today, these documents could take many different forms, including, in some cases, the databases of CASE tools. For legacy military Ada software, they are usually paper documents that comply with Data Item Descriptions (DIDs).

According to Royce, documentation serves three purposes. It is a means for a designer to establish an agreement with other designers about interfaces between their designs, and it is needed if designers and their managers are to reach quantitative agreement about the designer's progress. Second, in the early stages of software design, there is no other design product except the documentation. Finally, it is needed after development for three purposes. It is needed for testing the software before release because without it only the developer who wrote the software would understand the software well enough to test it, and at test time the developer would only repeat the same mistakes again that were committed during design and coding. It is needed during the operational phase to tell users how to use the software. It is needed following initial operations to guide programmers in correcting and enhancing the software. Royce's arguments are still valid today.

More arguments from Parnas and Clements. In a persuasive paper on software design published in 1986,[2] David Parnas and Paul Clements presented independent arguments that reinforced the earlier ones by Royce. Parnas and Clements argued for six types of documentation: first, a requirements document, written by end users or their representatives, that captures all software requirements; second, a module guide that states the design decisions that led to each component (module) of the software design; third, a collection of module interface specifications; fourth, a description of which programs in the system depend upon which others; fifth, a module design document that explains the internal structures of a module; and sixth, comments in the source code that supplement (without replicating) the information in the other documents. Their arguments for these documents are still valid. Writing when they did, Parnas and Clements emphasized paper documents. Their arguments apply as well to the broader notion of documentation that is common today.

Software Engineering Institute. The Software Engineering Institute (SEI) greatly extended the argument for documentation when it published its Capability Maturity Model for Software (CMM) in 1991.[3] In the CMM, a developer's software process maturity is inseparably linked to the documents that the developer produces and maintains. At the lowest maturity level, no organizational abilities are assumed. Moving up to the next higher level of maturity depends, among other things, on the organization's ability to document system requirements, its ability to produce and revise a software development plan in accordance with documented procedures, its ability to define and prepare a subcontract statement of work according to a documented procedure, its ability to prepare an software quality plan according to a documented procedure, and its ability to prepare an software configuration management plan according to a documented procedure. At each higher level, an increase in process maturity depends in part on the development of one or more kinds of additional software-related documentation.

Conclusion. Many streams of thought over the past twenty five years have converged on the conclusion in Figure H-3 which makes practical sense today. All software should be documented. Royce argued that documentation describes software requirements, software design, testing, operations, and helps with software maintenance. Parnas and Clements elaborated on its use for requirements capture and design. The SEI adds that it is necessary to describe how software will be developed. "*Documentation*" used to mean paper documents, but now there are many additional forms from databases to movies.

Document All Software

Figure H-3

2. Documentation for Acquirers.

What software documentation should be provided to the acquirer? A simple example shows the answer. Compare the acquisition of commercial, retail software like an e-mail system for a LAN to the contractual development of custom software, for example gateway software to link different e-mail systems on different LANs. The acquirer will receive users manuals with the retail software. What should the acquirer receive for the software under contractual development? In the contractual situation, at least the acquirer should see and consent to the developer's software development plan for the gateway software. If the acquirer intends to maintain the software, they should receive the source code also and an explanation of how the software was constructed, for example a description of the software design and engineering notes on its implementation and test.

Generalizing from examples like this one, I take the position that the acquirer should be provided with the documentation that is appropriate to the acquisition situation. Different situations, different documentation.

Conclusion. No constant, fixed list of documents is appropriate for every acquisition situation.

3. What About Software That is Generated Automatically by CASE Tools?

A few years ago, a prospective client called me for guidance on how to document Ada 83 source code that had been generated by a software design CASE tool. The same type of decision arises on many military projects today. As Figure H-4 from Martin shows, it is inherent in the software process called Rapid Application Development (RAD) since RAD depends upon automatic code generation to achieve high development speed with low rates of defects.

The CASE tool that my caller used had a visual approach to program design through graphical notation supported by a windowing operating system on a personal computer. I advised the caller to use the design graphics rather than the Ada source code as the software documentation since the graphical notation was the most accurate representations of the technical design decisions, and since the caller intended to change the code by regenerating it from new design diagrams.

A similar argument could be made for using pictures generated by a requirements analysis CASE tool to document requirements. There are popular software engineering practices, for example Joint Application Development (JAD), that produce such outputs which are, in some cases, the most accurate representations of the requirements for the software.

In some software life cycles, requirements are transformed directly into source code, skipping the design step entirely. What should be documented then? Certainly the requirements should be documented, and they are documented by the transformation tool that generates code. If the mechanism for transforming requirements into code is not fully trusted, then it should be documented also. However, if the transformational tool is correct, eventually only the tool's developer will bother to look at its documentation.

Conclusion. New development methods may make some types of documentation inappropriate in certain cases.

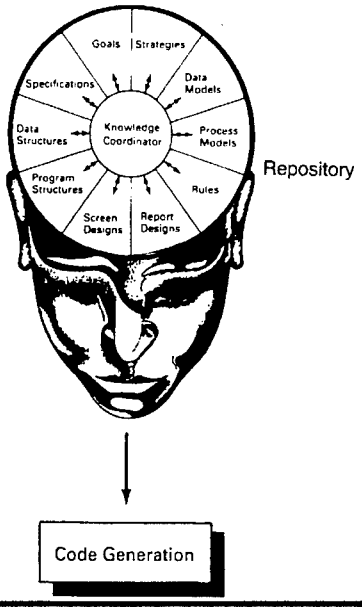


Figure H-4

4. Is Ada Source Code Alone Adequate Documentation?

Ada source code can be easy to read and easy to understand, both Ada 83 and Ada 95. Ada source code, with the proper annotations, could document both the software itself and its design under certain conditions.

Readability and understandability. Ada code that complies with good guidelines for style[1] is readable and understandable. Well-styled Ada could be called self-documenting with respect to what it reveals to an experienced reader about the construction of its program units.

Annotating Ada source code to describe software design. Ada-based program design languages (ADLs) in use today consist of legal Ada source code with special legal Ada comments, like those sketched in Figure H-5 (below), that can be parsed by an ADL parser to extract information about aspects of the software's design.

ADLs are source code. ADLs offer the benefit of allowing developers to use Ada compilers to check their designs for common errors like parameter and type mismatches and compilation dependency mistakes before coders implement the designs. The design annotations contained in ADLs are usually delivered to an acquirer as part of the source code. When there is formal configuration management, the delivery usually takes place in a software product specification that documents the as-built baseline for the software.

ADLs can be altered to suit the circumstances of individual projects. The final ADL specifications are usually documented in a project's collection of design and coding standards and procedures. ADL specifications must match software design methods if the ADL is to successfully present a detailed informative description of software design. This cannot be taken for granted.

A potential problem with ADL is that it can decrease source code reusability. When an ADL that contains certain embedded information about a project like the numbers of paragraphs in textual requirements descriptions is delivered to a software repository, some

```
with Count_Manager;
with Geolocation_Manager;
with Notional_Date_Manager;
with Security_Label_Manager;

package Evacuation_Plan is

-| Source
-| Origin:  Internal Development
...
-| Description:
-| The Evacuation Plan handles all processing described in the following events:
-|    101.1.1.4.19  Update_EPW_Evacuation
-|    101.1.1.4.20  Update_Noncombatant_Evauation
...
 procedure Update_EPW_Evacuation
             (From_JDC_JDA    : in    Prisoner_Evacuation_Orders);

  -| Requirements Satisfied
...
  -| Description
...
 procedure Update_Noncombatant_Evacuation
             (From_JDC_JDA    : in    Noncombatant_Evacuation_Orders);

  -| Requirements Statisfied
...
  -| Description
...
end Evacuation_Plan;
```

Figure H-5

subsequent users will have no use for that information. If the behavior of the software is separately documented elsewhere than in the requirements comments, the comments will not be needed by those who check the software out of a repository who will have to strip them out which will add to the cost of reusing the software. If the behavior of the software is documented only in the comments, in the form of references to paragraphs in a system-level or software-level requirements description, potential users of the software can be expected to avoid it rather than paying the high price to obtain and read the descriptions to understand what the software does before they check it out of the repository.

Conclusion. Well-styled Ada 83 or Ada 95 could be used to document both the design and the implementation of the Ada software products for the project. However, there is no natural reason for software designers and coders to include the information in their source code that is usually found in project management documents such as plans. Well-styled ADL can be an excellent replacement for many technical documents, but even in that case it cannot replace all of the documentation that may be needed by acquirers.

5. How Do Ada Software Developers Want to Document Their Software?

In the Summer and Fall of 1993, approximately one thousand Ada-community members attended over thirty sixty to ninety-minute briefings on MIL-STD-498 presented by the ociation for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda) Software evelopment Standards and Ada Working Group (SDSAWG) at sites in the United States, ada, and Sweden. The purpose of the briefings was to spread information within the Ada unity about the new standard in time to obtain and incorporate developer comments about it not already captured by earlier reviews conducted by the Council of Defense and Space Industry Associations (CODSIA) and Department of Defense (DoD) organizations.

In the course of discussions between the speaker and the audiences at the presentations, Ada software developers (engineers and their technical managers) showed that they were primarily concerned with five categories of documentation, almost to the point of ignoring other categories; first, requirements specifications both at the system level and at the software level; second, design documents at both the system and the software levels; third, collections of technical data called Software Development Files (SDFs) by DoD-STD-2167A and MIL-STD-498. also called Unit Development Folders (UDFs) in other contexts; fourth, data in CASE tool files that represent what their software must do (i.e., the requirements that it must satisfy), represent how it is designed to do it (i.e., its main design elements, calling structure, compilation dependencies, error handling, and so forth), and capture the source code; and finally, software development plans.

Preferred categories of documentation. Developers who attended the SDSAWG presentations on MIL-STD-498 also indicated a strong preference for reducing the categories of documentation to just three; first, the data in their CASE tool files; second, their personal engineering notes not entered into a CASE tool which is the kind of information that has been sought in SDFs; and third, their project's software development plans. While the developers preferred to limit the categories of documentation, they often acknowledged that immature software processes are not likely to adequately document software even with help from CASE tools and technical notes. They accepted that acquirers may only be able to obtain the information that they need about such development efforts by imposing supplementary documentation requirements on the developers.

Conclusion. Developers prefer to limit documentation to three types which are, engineering data in CASE tools, engineering data in development notes not entered into CASE tools, and software development plans. Of the three kinds of documentation that developers prefer, ADL provides only part of the first. Also, developers following immature software processes are not likely to record adequate data for managing the software project or for understanding the software product.

6. What Role Do Software Engineering Standards Play in Adequate Documentation?

Since Secretary of Defense Perry's June 29, 1994 memorandum on, "*Specifications and Standards—A New Way of Doing Business*," the DoD has moved away from the use of military specifications and standards on military contracts. However, at the same time, it has moved to replace some of its software engineering standards with commercial equivalents like the ISO 9000 series of quality standards. Figure H-6 (below) shows some of the organizations whose standards are relevant, the International Organization for Standardization (ISO), the Electronic Industries Association (EIA), and Institute of Electrical and Electronics Engineers (IEEE).

MIL-STD-498. At the time of Secretary Perry's memorandum, there was no commercial equivalent of DoD-STD-2167A. Also, 2167A was known to have some troublesome defects. A replacement standard, MIL-STD-498, was nearly ready to publish, so it was approved for two years, to December, 1996. MIL-STD-498 superseded DoD-STD-2167A, DoD-STD-7935A, and the NSA standard DoD-STD-1703. MIL-STD-498 requires developers to develop and record certain information. It permits whatever means is appropriate for recording the information in a particular development situation. It could be that paper documents are appropriate, but the standard suggests that many other media and formats for the information might be better, for example data files for a CASE tool.

Through the requirements in the standard itself, and additional language in its DIDs, 498 is a lengthy "*checklist*" of potentially relevant information that a developer should consider when planning a software project. This is its value during documentation. The developer and



Figure H-6

the acquirer are responsible for tailoring the "*checklist*" to an actual list of the information that the project will provide. The developer and the acquirer must both be competent at software development to do this. The standard was never intended to be a substitute for technical skill and training. The standard has roughly the same value during software development that a generic preflight checklist for all aircraft would have for a pilot.

Although the earlier DoD-STD-2167A had required developers to document a software design as though it were an hierarchical collection of elements, 498 does not. 498 requires developers to record the actual design which might be a network of components, a repository, a pipeline, or any other reasonable approach to decomposing and satisfying the software requirements. Many different approaches are known, and several are described in the literature.[2]

ISO/IEC 12207. By comparison with MIL-STD-498, this ISO standard approved in 1995 says very little about documentation. 12207 is much broader in scope, and at a higher level of abstraction. Although there are technical differences between the two standards, where the 12207 requirements overlap those of 498 in areas of software documentation, the requirements in 12207 are similar to and can be said to be contained in the requirements in MIL-STD-498.

Replacement for MIL-STD-498 in 1996. A Joint Industry Standard working group led by the IEEE and the EIA is developing a new commercial standard to replace MIL-STD-498 in December, 1996. The new standard, tentatively called J-STD-016, is planned to include ISO/IEC-12207 together with supplementary enhancements derived from MIL-STD-498 and from additional work by industry representatives in collaboration with key DoD personnel. Like MIL-STD-498, J-STD-016 is planned to play the documentation role of a lengthy "*checklist*" of potentially relevant information that a developer should consider when planning a software project. Once again, the standard is not intended to be a substitute for technical skill and training.

Conclusion. From the point of view of choosing documentation, it is reasonable and useful to think of modern software engineering standards as reminders to developers of what they might need to record when they develop software. Standards can be very valuable when they are used in this way, and most of the familiar problems associated with the abuse of military software development standards will never occur.

7. Given that Military Software Should be Documented, Are There Good Ways to Do It?

The short answer to this question is, *"Sure!"* There should not be doubt about this any longer. Developers often document their software in ways that some may not call documentation but that are excellent representations of the developer's work. For example, a class diagram like Figure H-7 from Grady Booch's book on object-oriented design[3] is an excellent representation of design decisions even though it may exist only within a CASE tool.

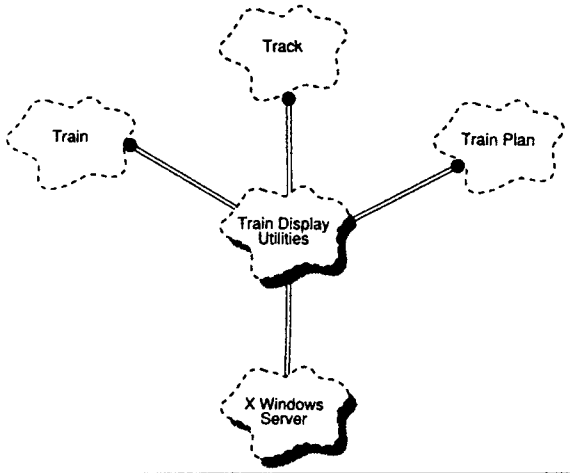


Figure H-7

If a developer followed Booch's object-oriented design method in full, the resulting work products would include not only class diagrams, but also state transition diagrams, object diagrams, timing diagrams, module diagrams, process diagrams, and elaborate textual descriptions of each of them called *"templates."* Taken all together, they would provide a thorough presentation of the developer's design. Rational sells a CASE tool, Rational Rose, that runs on several different operating systems including Unix and Microsoft Windows and supports the development of Booch's work products. A trained user of the tool can easily produce good documentation for recording and evaluating an object-oriented design. There are other useful combinations of methods and tools in addition to Rational's which shows how many different opportunities there are to document software designs well with CASE tools.

There are CASE tools for identifying and documenting system and software requirements, and for tracing requirements to code. There are CASE tools for software testing and for software configuration management, and other tools for source code analysis and evaluation. There are numerous tools for project planning and tracking, including tools that automatically calculate and display such data as cost and schedule variances. And, don't forget all the tools for software process definition and re-engineering.

Conclusion. Given the huge range of CASE tools in the market today, it is probable that for each aspect of software development there is a tool that can document it well. This is not to argue that it would be easy for any project to acquire or learn to use them. Given the excellent tools that are available today, however, it would be easy for many developers to use some of the tools to produce good documentation that was appropriate for their project.

8. Benefits and Dangers of Relying Only on a Developer's Normal Work Products?

Successful, high-quality software development as usual. The big benefit of relying on the developer's normal, mature software process to produce proper software documentation is that it avoids special cases and crises. The software process that has worked well in the past to produce high-quality software can be expected to work well again. The types of documentation and their detail will be predictable because the documentation will resemble documentation produced by previous projects. The effort to produce the documentation will be known. No special training or tools will be necessary. There is no risk of disturbing the developer's mature software process in unforeseen ways as there always is when an acquirer orders information that the developer has never recorded before. On the other hand, there are dangers in depending solely on developer-defined, developer-developed work products. Three of them are described below. Probably there are others.

A poor software process, or poor execution of a good process. When a developer's software process is poor, that is at Level 1 on the SEI's five-level scale of process maturity, the work products that result from the process are likely to be poor as well. For example, a developer with an immature software process will inadequately plan their software development projects. By the definition of an immature process, they cannot be expected to develop adequate planning documentation.

A developer with a process at Level 2 on the SEI scale can be expected to have adequate plans for projects that are similar to projects they have managed successfully in the past. However, planning for projects in new functional domains or for projects that are much larger in scale than anything the developer has managed before, for example, can be expected to be inadequate because capabilities that are key to successful performance in the new areas may not be developed yet, skills such as training, intergroup coordination, and peer reviews, that are needed but not likely to be reflected in the project planning. In the case of software design, a Level 2 developer may have a poorly defined approach. In such a case, software design work products may be haphazard and incomplete because there may be no practices that lead to adequate work products, so chance would dictate their content and organization. If a developer had a software process at SEI Level 3 or higher, most needed documentation could be expected. Even so, latitude in their process definitions, or poor developer personnel, could lead to poor work products that were of limited use to the acquirer for monitoring the contract.

Unskilled or unprepared acquirer personnel. It could happen that the acquirer and the maintenance organization may not be trained or experienced enough to understand the developer's work products. Many of the participants at the SDSAWG presentations on MIL-STD-498 in 1993 were acquirer personnel who were concerned about this. In other cases, even when acquirer personnel and their organizations are trained and experienced in software development, understanding the developer's work products could depend upon knowledge of a proprietary development method that was not fully shared with the acquirer. Many situations could give rise to such a problem.

Skilled acquirer personnel who improperly use developer data. When acquirer personnel are skillful with the developer's CASE tools, checks would have to be placed on them to prevent them from improperly using the developer's work products. The problem is both an ethical one, for example a leak of proprietary data, and a technical one. The ethical problem is obvious. An example of the technical problem is a premature evaluation of certain aspects of a design that have not been developed because they are scheduled for later development.

The technical problem may impede development in surprising ways. For example, if an acquirer insisted on premature completion of areas of software design and source code that developer data showed to be incomplete or defective, it could cause schedule and cost overruns. Since software evolves through many stages of completion, at any time prior to final testing of the software, some aspects of the software or its design can be expected to be wrong or incomplete.

Acquirers should expect developers to have a reasonable schedule for correcting unresolved problems and completing incomplete designs and source code. This is just another way of saying that developers should have a reasonable software development schedule. To pressure developers to complete all defects and open areas of designs and source code at each point in development could be very expensive since it could conflict with the developer's development schedule and could risk forcing premature, poor decisions that would cause major engineering problems and schedule slips when the software was tested or released.

Conclusion. There are risks to relying only on the documentation from a developer's normal software process. In many cases the benefit of doing business well as usual will outweigh the risks. Also, the risk to a project from not having a normal software process, in other words from treating the software development business as a series of exceptional situations, should be avoided whenever possible.

9. Recommendations

My recommendations are summarized in Figure H-8.

Competence

•

Normal Software Process

•

Additional Documentation Only If Justified

Figure H-8

First, consistent success with documentation depends on competence. Before awarding a software development contract, an acquirer should be competent to choose a developer that knows how to adequately document their software development effort and their software products. On the other side of the development relationship, before a developer begins to develop military software, they should be competent to properly document their software development effort and their work products.

Second, stick with the developer's normal software process. Usually, the acquirer should acquire only the developer's normal work products because usually they will be the most appropriate information, in the most appropriate format and the most appropriate media for documenting the development effort and its results. For their part, developers should have a normal software process with tools that will work well for the development effort, and know how to use the tools well, so they can document their software development effort and their work products in a way that is appropriate for understanding and overseeing their work.

Finally, avoid changing the normal software process by orders for additional documentation unless the additional documentation is justified. The acquirer should order additional documentation, supplementary to the developer's normal work products, only when the acquirer has strong evidence that the information is needed for the project and that the developer would not provide it otherwise. For their part, developers should accept that there may be occasions when acquirers will need more information than they can obtain from the normal developer work products, and developers should be prepared to accommodate these reasonable, additional information orders from their customers.

ACRONYMS AND ABBREVIATIONS

ACM	Association for Computing Machinery
ADL	Ada Design Language
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CASE	Computer-Aided Software Engineering
CMM	Capability Maturity Model for Software
CODSIA	Council of Defense and Space Industry Associations
DID	Data Item Description
DoD	Department of Defense
EIA	Electronic Industries Association
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
JAD	Joint Application Development
LAN	Local Area Network
RAD	Rapid Application Development
SDF	Software Development File
SDSAWG	Software Development Standards and Ada Working Group
SEI	Software Engineering Institute
SIGAda	Special Interest Group on Ada
SIGSOFT	Special Interest Group on Software Engineering
STSC	Software Technology Support Center
UDF	Unit Development Folder
WESCON	Western Electronic Show and Convention

[1]Good guidelines for Ada style have been available for many years. Some of the earliest ones were developed by INTELLIMAC, Inc. and the Goddard Space Flight Center. A popular set today was developed by the Software Productivity Consortium, and published by Van Nostrand Reinhold in 1989 with the title *Ada: Quality and Style.*

[2]For example, see Mary Shaws's short article, "*Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging,*" in *Software Engineering Notes*, Special Issue, August 1995 (ACM SIGSOFT, New York).

[3]Booch, Grady, *Object Oriented Design: With Applications,* (Benjamin/Cummings: Redwood City, CA), page 461.

APPENDIX

I

A Detailed Comparison of ISO 9001 and the Capability Maturity Model (CMM)

Blank page.

APPENDIX I

A Detailed Comparison of ISO 9001 and the Capability Maturity Model (CMM[SM])

Mark C. Paulk

CONTENT PAGE

ABSTRACT I-2
INTRODUCTION I-2
The Capability Maturity Model for Software I-3
The Five Maturity Levels I-3
Key Process Areas I-3
Common Features I-4
Key Practices I-5
THE ISO 9000 SERIES OF STANDARDS FOR QUALITY MANAGEMENT SYSTEMS I-5
MAPPING THE ISO 9001 TO THE CMM[SM] I-6
Management Responsibility I-6
Quality System I-6
Contract Review I-7
Design Control I-8
Document Control I-8
Purchasing I-8
Purchaser Supplied Product I-8
Product Identification and Traceability I-8
Process Control I-9
Inspection and Testing I-9
Inspection, Measuring, and Test Equipment I-9
Inspection and Test Status I-9
Control of Nonconforming Product I-9
Corrective Action I-10
Handling, Storage, Packaging, and Delivery I-10
Quality Records I-10
Internal Quality Audits I-11
Training I-11
Servicing I-11
Statistical Techniques I-11
CONTRASTING THE ISO 9001 AND THE CMM[SM] I-12
The Need for Judgment I-12

Version 2.0

APPENDIX I Comparison of ISO 9001 and the CMM[SM]

The Key Process Area Profile of an ISO 9001-Compliant Organization I-13
CONCLUSION I-13
REFERENCES I-14
ACKNOWLEDGMENT I-14
NOTES I-14

ABSTRACT

The Capability Maturity Model for Software (CMM[SM]), developed by the Software Engineering Institute, and the ISO 9000 series of standards, developed by the International Standards Organization, share a common concern with quality and process management. The two are driven by similar concerns and intuitively correlated. The purpose of this paper is to contrast the CMM[SM] and ISO 9001, showing both their differences and their similarities. The results of the analysis indicate that, although an ISO 9001 compliant organization would not necessarily satisfy all of the Level 2 key process areas, it would satisfy most of the Level 2 goals and many of the Level 3 goals. Because there are practices in the CMM[SM] that are not addressed in ISO 9000, it is possible for a Level 1 organization to receive ISO 9001 registration; similarly, there are areas addressed by ISO 9001 that are not addressed in the CMM.[SM] A Level 3 organization would have little difficulty in obtaining ISO 9001 certification, and a Level 2 organization would have significant advantages in obtaining certification.

INTRODUCTION

The **Capability Maturity Model for Software**, developed by the Software Engineering Institute, and the **ISO 9000 series of standards**, developed by the International Standards Organization, share a common concern with quality and process management. The two are driven by similar concerns and intuitively correlated.

- The specific standard in the ISO 9000 series of concern to software organizations is ISO 9001. The questions frequently asked include:
- At what Level in the CMM[SM] would an ISO 9001 compliant organization be?
- Can a Level 2 (or 3) organization be considered compliant with ISO 9001?
- Should my software quality management and process improvement efforts be based on ISO 9001 or on the CMM[SM]?

The purpose of this paper is to compare the CMM[SM] and ISO 9001, identify their differences and similarities, and answer these questions. This paper should be useful to anyone embarking on a software process improvement program where ISO 9001 certification is an important issue in their business environment. Even if the CMM[SM] is not used as the basis for the improvement program, it provides significant guidance over and above that offered by ISO 9001, ISO 9000-3, or **TickIT** for implementing an ISO 9001-compliant software process.

Section 2 of this paper contains a brief overview of the CMM.[SM] Section 3 contains a brief overview of the ISO 9000 family of standards as relevant to software. Section 4 is a clause-by-clause discussion of ISO 9001 and how it relates to the CMM.[SM] Section 5 contrasts ISO 9001 and the CMM;[SM] in particular, it provides a key process area profile for an ISO 9001-compliant organization.

APPENDIX I Comparison of ISO 9001 and the CMM[SM]

The Capability Maturity Model for Software

The **Capability Maturity Model for Software** [Paulk93a, Paulk93b] describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from *ad hoc*, chaotic processes to mature, disciplined software processes. The CMM[SM] is organized into five maturity levels. A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

The Five Maturity Levels

The following characterizations of the five maturity levels highlight the primary process changes made at each level:

- **Initial**. The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
- **Repeatable**. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
- **Defined**. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
- **Managed**. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
- **Optimizing**. Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Key Process Areas

Except for Level 1, each maturity level is decomposed into several **key process areas** that indicate the areas an organization should focus on to improve its software process. Key process areas identify the issues that must be addressed to achieve a maturity level. Each key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. The key process areas and their purposes are listed below. The name of each key process area is followed by its two-letter abbreviation. By definition there are no key process areas for Level 1. The **key process areas at Level 2** focus on the software project's concerns related to establishing basic project management controls, as summarized below:

- **Requirements Management (RM)**. Establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project.
- **Software Project Planning (PP)**. Establish reasonable plans for performing the software engineering and for managing the software project.
- **Software Project Tracking and Oversight (PT)**. Establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.
- **Software Subcontract Management (SM)**. Select qualified software subcontractors and manage them effectively.
- **Software Quality Assurance (QA)**. Provide management with appropriate visibility into the process being used by the software project and of the products being built.

- **Software Configuration Management (CM)**. Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

The **key process areas at Level 3** address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects, as summarized below:

- **Organization Process Focus (PF)**. Establish the organizational responsibility for software process activities that improve the organization's overall software process capability.
- **Organization Process Definition (PD)**. Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.
- **Training Program (TP)**. Develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.
- **Integrated Software Management (IM)**. Integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.
- **Software Product Engineering (PE)**. Consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.
- **Intergroup Coordination (IC)**. Establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.
- **Peer Reviews (PR)**. Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented.

The **key process areas at Level 4** focus on establishing a quantitative understanding of both the software process and the software work products being built, as summarized below:

- **Quantitative Process Management (QP)**. Control the process performance of the software project quantitatively.
- **Software Quality Management (QM)**. Develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

The **key process areas at Level 5** cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement, as summarized below:

- **Defect Prevention (DP)**. Identify the causes of defects and prevent them from recurring.
- **Technology Change Management (TM)**. Identify beneficial new technologies (i.e., tools, methods, and processes) and transfer them into the organization in an orderly manner.
- **Process Change Management (PC)**. Continually improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.

Common Features

For convenience, each of the key process areas is organized by **common features**. The common features are attributes that indicate whether the implementation and institutionalization of a key process area is effective, repeatable, and lasting. The **five common features**, followed by their two-letter abbreviations, are listed below:

- **Commitment to Perform (CO)**. Describes the actions the organization must take to ensure that the process is established and will endure. Includes practices on policy and leadership.
- **Ability to Perform (AB)**. Describes the preconditions that must exist in the project or organization to implement the software process competently. Includes practices on resources, organizational structure, training, and tools.
- **Activities Performed (AC)**. Describes the roles and procedures necessary to implement a key process area. Includes practices on plans, procedures, work performed, tracking, and corrective action.
- **Measurement and Analysis (ME)**. Describes the need to measure the process and analyze the measurements. Includes examples of measurements.
- **Verifying Implementation (VE)**. Describes the steps to ensure that the activities are performed in compliance with the process that has been established. Includes practices on management reviews and audits.

Key Practices

Each key process area is described in terms of the **key practices** that contribute to satisfying its goals. The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalization of the key process area and are described in *"Key Practices of the Capability Maturity Model, Version 1.1."* [Paulk93b].

THE ISO 9000 SERIES OF STANDARDS FOR QUALITY MANAGEMENT SYSTEMS

The **ISO 9000 series of standards** is a set of documents dealing with quality systems that can be used for external quality assurance purposes. They specify quality system requirements for use where a contract between two parties requires the demonstration of a supplier's capability to design and supply a product. The two parties could be an external client and a supplier, or both could be internal, e.g., marketing and engineering groups in a company.

ISO 9000, *"Quality management and quality assurance standards — Guidelines for selection and use,"* clarifies the distinctions and interrelationships between quality concepts and provides guidelines for the selection and use of a series of international standards on quality systems that can be used for internal quality management purposes (ISO 9004) and for external quality assurance purposes (ISO 9001, 9002, and 9003). The quality concepts addressed by these standards are:

- An organization should achieve and sustain the quality of the product or service produced so as to meet continually the purchaser's stated or implied needs.
- An organization should provide confidence to its own management that the intended quality is being achieved and sustained.
- An organization should provide confidence to the purchaser that the intended quality is being, or will be, achieved in the delivered product or service provided. When contractually required, this provision of confidence may involve agreed demonstration requirements.

ISO 9001, *"Quality systems — Model for quality assurance in design/development, production, installation, and servicing,"* is for use when conformance to specified requirements is to be assured by the supplier during several stages, which may include design, development, production, installation, and servicing. Of the ISO 9000 series, it is the standard that is pertinent to software development and maintenance.[1]

APPENDIX I Comparison of ISO 9001 and the CMM[SM]

ISO 9000-3 provides *"Guidelines for the application of ISO 9001 to the development, supply, and maintenance of software."* Annexes A and B in ISO 9000-3 cross-reference ISO 9000-3 and ISO 9001. A British guide for applying ISO 9001 to software [TickIT] provides additional information on using ISO 9000-3 and 9001 in the software arena.

MAPPING ISO 9001 TO THE CMM[SM]

There are 20 clauses in **ISO 9001**, which are summarized and compared to the practices in the **CMM[SM]** in this section. The comparison is based on an analysis of ISO 9001, ISO 9000-3, TickIT, and the TickIT training materials [Lloyd's94]. There is judgement involved in making this comparison, and there are differences in interpretation for both ISO 9001 and the CMM.[SM] ISO 9000-3 elaborates significantly on ISO 9001, and TickIT training provides significant guidance on how to interpret both ISO 9000-3 and ISO 9001. A common challenge for CMM[SM]-based appraisals and ISO 9001 certification is reliability and consistency of assessments, which is partially addressed by strict training prerequisites for TickIT auditors and CMM[SM] appraisers.

Each clause in ISO 9001 will be discussed in the subsections of this section, but not on a sentence-for-sentence basis. A detailed mapping, at the sentence to subpractice level, was performed as part of this analysis and is described in the SEI technical report *A Comparison of ISO 9001 and the Capability Maturity Model for Software* [Paulk94]. (A less detailed discussion was published in [Paulk93c]).

Management Responsibility

ISO 9001 requires that the quality policy be defined, documented, understood, implemented, and maintained; that responsibilities and authorities for all personnel specifying, achieving, and monitoring quality be defined; and that in-house verification resources be defined, trained, and funded. A designated manager ensures that the quality program is implemented and maintained.

Management responsibility for quality policy and verification activities is primarily addressed **in Software Quality Assurance**, although **Software Project Planning and Software Project Tracking and Oversight** assist by assigning responsibility for performing all project roles. Management's responsibility at both the senior management and project management levels to oversee the software project are addressed in the **Verifying Implementation** common feature. More generically, leadership issues are addressed in the **Commitment to Perform** common feature, and organizational structure and resource issues are addressed in the **Ability to Perform** common feature.

One could argue that the quality policy described in **Software Quality Management** at Level 4 is also addressed by this clause, but the Level 4 quality policy is quantitative. ISO 9001 is somewhat ambiguous about the role of measurement in the quality management system, as is discussed for clause 4.20, but ISO 9001 requires that quality objectives be defined and documented, not that they be quantitative (also see the discussion of clause 4.20).

Quality System

ISO 9001 requires that a documented quality system, including procedures and instructions, be established. ISO 9000-3 characterizes this quality system as an integrated process throughout the entire life cycle. Quality system activities are primarily addressed in the **CMM[SM]** in **Software Quality Assurance.** The procedures that would be used are distributed throughout the key process areas in the various Activities Performed practices.

The specific procedures and standards that a software project would use are specified in the software development plan described in **Software Project Planning**. Compliance with

these standards and procedures is assured in **Software Quality Assurance** and by the auditing practices in the **Verifying Implementation** common feature. **Software Product Engineering** requires that the software engineering tasks be defined, integrated, and consistently performed, which corresponds directly to the ISO 9000-3 guidance for interpreting this clause.

One arguable correspondence is to **Organization Process Definition**, which describes a set of software process assets, including standards, procedures, and process descriptions, at the organization level. Addressing Organization Process Definition would certainly contribute to achieving this clause, but the standards and procedures in this clause of ISO 9001 could be addressed strictly at the project level. ISO 9001 specifies the supplier's quality system, but does not discuss the relationship between organizational support and project implementation as the CMM[SM] does. ISO 9000-3, on the other hand, has two sections on quality planning: clause 4.2.3 discusses quality planning across projects; clause 5.5 discusses quality planning within a particular development effort.

Contract Review

ISO 9001 requires that contracts be reviewed to determine whether the requirements are adequately defined, agree with the bid, and can be implemented. Review of the customer requirements, as allocated to software, is described in the **CMM[SM]** in **Requirements Management**. The software organization (supplier) ensures that the system requirements allocated to software are documented and reviewed and that missing or ambiguous requirements are clarified. Since the CMM[SM] is constrained to the software perspective, the customer requirements as a whole are beyond the scope of this key process area.

Software Project Planning describes the development of a proposal, a statement of work, and a software development plan, which are reviewed by the software engineering group and by senior management, in establishing external (contractual) commitments. The CMM[SM] also explicitly addresses the acquisition of software through subcontracting by the software organization, as described in **Software Subcontract Management**. Contracts may be with an external customer or with a subcontractor, although that distinction is not explicitly made in this clause of ISO 9001.

Design Control

ISO 9001 requires that procedures to control and verify the design be established. This includes planning design activities, identifying inputs and outputs, verifying the design, and controlling design changes. ISO 9000-3 elaborates this clause with clauses on the purchaser's requirements specification (5.3), development planning (5.4), quality planning (5.5), design and implementation (5.6), testing and validation (5.7), and configuration management (6.1).

In the **CMM**,[SM] the life-cycle activities of requirements analysis, design, code, and test are described in **Software Product Engineering**. Planning these activities is described in **Software Project Planning. Software Project Tracking and Oversight** describes control of these life cycle activities, and **Software Configuration Management** describes configuration management of software work products generated by these activities.

ISO 9001 requires design control measures, such as holding and recording design reviews and qualification tests. ISO 9000-3 states that the supplier should carry out reviews to ensure the requirements are met and design methods are correctly carried out. Although design control measures are required, the use of the phrasing *"such as"* and *"should"* allows flexibility in what specific control measures are used. In contrast, the CMM[SM] calls out a specific quality control mechanism: peer reviews. The Peer Reviews key process area supports processes throughout the life cycle, from requirements analysis through testing.

TickIT training clarifies this issue by listing three examples of design reviews: Fagan inspections, structured walkthroughs, and peer reviews (in the sense of a desk check). The training also states that *"an auditor will need to be satisfied from the procedures and records available that the reviews within an organization are satisfactory considering the type and criticality of the project under review."* [Lloyd's94, p. 17.10-11] More formal, quantitative aspects of the design process are described in **Software Quality Management**, but this degree of formality is not necessarily required by ISO 9001.

Document Control

ISO 9001 requires that the distribution and modification of documents be controlled. In the CMM,[SM] the configuration management practices characterizing document control are described in Software **Configuration Management.** The specific procedures, standards, and other documents that may be placed under configuration management in the CMM[SM] are distributed throughout the key process areas in the various **Activities Performed** practices. The documentation required to operate and maintain the system is specifically called out in Activity 8 of **Software Product Engineering.**

Purchasing

ISO 9001 requires that purchased products conform to their specified requirements. This includes the assessment of potential subcontractors and verification of purchased products. In the **CMM,[SM]** this is addressed in **Software Subcontract Management.** Evaluation of subcontractors is described in Activity 2, while acceptance testing of subcontracted software is addressed in Activity 12.

Purchaser Supplied Product

ISO 9001 requires that any purchaser-supplied material be verified and maintained. **ISO 9000-3** discusses this clause in the context of included software product (6.8), including commercial-off-the-shelf software.

Activity 6.3 in **Integrated Software Management** is the only practice in the **CMM[SM]** describing the use of purchased software. It does so in the context of identifying off-the-shelf or reusable software as part of planning. Integration of off-the-shelf and reusable software is one of the areas where the CMM[SM] is weak. This clause, especially as expanded in ISO 9000-3, cannot be considered adequately covered by the CMM.[SM] It would be reasonable, though not sufficient, to apply the acceptance testing practice for subcontracted software in Activity 12 of **Software Subcontract Management** to any included software product. A change request has been written for CMM v1.1 to incorporate practices in **Software Product Engineering** that address product evaluation and the inclusion of off-the-shelf and nondevelopmental software.

Product Identification and Traceability

ISO 9001 requires that the product be identified and traceable during all stages of production, delivery, and installation. The **CMM[SM]** covers this clause primarily in **Software Configuration Management**, but Activity 10 of **Software Product Engineering** states the specific need for consistency and traceability between software work products.

APPENDIX I Comparison of ISO 9001 and the CMM[SM]

Process Control

ISO 9001 requires that production processes be defined and planned. This includes carrying out production under controlled conditions, according to documented instructions. Special processes that cannot be fully verified after the fact are continuously monitored and controlled. ISO 9000-3 includes design and implementation (5.6); rules, practices, and conventions (6.5); and tools and techniques (6.6).

The procedures defining the software production process in the **CMM[SM]** are distributed throughout the key process areas in the various **Activities Performed** practices. The specific procedures and standards that would be used are specified in the software development plan, as described in Activity 7 of **Software Project Planning**. The definition and integration of software *"production"* processes are described in **Software Product Engineering**. The tools to support these processes are called out in Ability 1.2 of Software Product Engineering. Process assurance is specified in Activity 4 of **Software Quality Assurance** (product assurance is specified in Activity 5).

Quantitative Process Management addresses the quantitative aspect of control exemplified by statistical process control, but would typically not be required to satisfy this clause. It is also worth noting that clause 6.6 in ISO 9000-3 states *that "the supplier should improve these tools and techniques as required,"* which corresponds to transitioning new technology into the organization as discussed in **Technology Change Management.**

Inspection and Testing

ISO 9001 requires that incoming materials be inspected or verified before use and that in-process inspection and testing be performed. Final inspection and testing are performed prior to release of finished product. Records of inspection and test are kept. The issues surrounding the inspection of incoming material have already been discussed for clause 4.7. The **CMM[SM]** describes testing in Activities 5, 6, and 7 in **Software Product Engineering.** In-process inspections in the software sense are addressed in **Peer Reviews.**

Inspection, Measuring, and Test Equipment

ISO 9001 requires that equipment used to demonstrate conformance be controlled, calibrated, and maintained. When test hardware or software is used, it is checked before use and rechecked at prescribed intervals. ISO 9000-3 clarifies this clause with clauses on testing and validation (5.7); rules, practices, and conventions (6.5); and tools and techniques (6.6). This clause is generically addressed in the **CMM[SM]** under the testing practices in **Software Product Engineering**. Test software is specifically called out in Ability 1.2, which describes the tools that support testing.

Inspection and Test Status

ISO 9001 requires that the status of inspections and tests be maintained for items as they progress through various processing steps. This clause is addressed in the **CMM[SM]** by the testing practices in **Software Product Engineering** and by Activities 5 and 8 on problem reporting and configuration status, respectively, in **Software Configuration Management**.

Control of Nonconforming Product

ISO 9001 requires that nonconforming product be controlled to prevent inadvertent use or installation. ISO 9000-3 maps this concept to design and implementation (5.6); testing and validation (5.7); replication, delivery, and installation (5.9); and configuration management (6.1). Design, implementation, testing, and validation are addressed in **Software Product Engineering**. In **Software Configuration Management**, Activity 8 addresses the status of

configuration items, which would include the status of items that contain known defects not yet fixed. Installation is not addressed in the **CMM**,[SM] as is discussed for clause 4.15.

In the manufacturing world this clause is important because it is sometimes necessary to build products using components that do not conform to all of the requirements. When such decisions are made, the resulting nonconforming products must be carefully controlled. Similarly, in the software world a system may sometimes use tools or reuse software that does not satisfy all of the pertinent standards. For example, reusing Fortran code in an Ada program may be cost-effective if the Fortran code has demonstrated its value in previous applications. That code, however, may pose a significant risk to the Ada system, and the risk must be thoughtfully managed. Nonconforming product is not specifically addressed in the CMM.[SM] In ISO 9000-3, it essentially disappears among a number of related processes spanning the software life cycle.

Corrective Action

ISO 9001 requires that the causes of nonconforming product be identified. Potential causes of nonconforming product are eliminated; procedures are changed resulting from corrective action. ISO 9000-3 quotes this clause verbatim, with no elaboration. A literal reading of this clause would imply many of the practices in **Defect Prevention**. Based upon the **TickIT Auditors' Guide** [TickIT, pp. 139-140] and discussions with ISO 9000 auditors, the corrective action discussed in this clause is driven by customer complaints. The software engineering group should look at field defects, analyze why they occurred, and take corrective action. This would typically occur through software updates and patches distributed to the fielded software. Under this interpretation, an appropriate mapping of this clause would be problem reporting, followed with controlled maintenance of baselined work products, as described in **Software Configuration Management.**

A complementary interpretation described in **TickIT** training [Lloyd's94, section 23] is that the corrective action is to address noncompliances identified in an audit, whether external or internal. This would be addressed in **Software Quality Assurance** in the CMM.[SM] In the current revision cycle for ISO 9001, the draft international standard includes separate requirements for corrective and preventive action. Corrective action is directed toward eliminating the causes of actual nonconformities, and preventive action is directed toward eliminating the causes of potential nonconformities [Durand93, p. 27]. This is a controversial issue in applying ISO 9001 to software. Some auditors seem to expect a defect prevention process similar to that which is found in the manufacturing environment. Others only require addressing user problem reports. It is arguable how much, if any, of the in-process causal analysis and defect prevention described in **Defect Prevention** is necessary to satisfy this clause.

Handling, Storage, Packaging, and Delivery

ISO 9001 requires that procedures for handling, storage, packaging, and delivery be established and maintained. ISO 9000-3 maps this to acceptance (5.8) and replication, delivery, and installation (5.9) Replication, delivery, and installation are not covered in the **CMM**.[SM] Acceptance testing is addressed in Activity 7 of **Software Product Engineering**, and Activity 7 of **Software Configuration Management** describes the creation and release of software products. Delivering and installing the product, however, is not described in the CMM.[SM] A change request has been written for CMM v1.1 to incorporate a practice in **Software Product Engineering** on delivery and installation of the software product.

Quality Records

ISO 9001 requires that quality records be collected, maintained, and dispositioned. The practices defining the quality records to be maintained in the **CMM**[SM] are distributed throughout the key process areas in the various **Activities Performed** practices. Specifically

pertinent to this clause are the testing and peer review practices in **Software Product Engineering**, especially the collection and analysis of defect data in Activity 9. Problem reporting is addressed by Activity 5 in **Software Configuration Management**, and the collection of peer review data is described in Activity 3 of **Peer Reviews.**

Internal Quality Audits

ISO 9001 requires that audits be planned and performed. The results of audits are communicated to management, and any deficiencies found are corrected. The auditing process is described in **Software Quality Assurance**. Specific audits in the **CMM[SM]** are called out in the auditing practices of the **Verifying Implementation** common feature.

Training

ISO 9001 requires that training needs be identified and that training be provided, since selected tasks may require qualified personnel. Records of training are maintained. Specific training needs in the **CMM[SM]** are identified in the training and orientation practices in the **Ability to Perform** common feature. The general training infrastructure is described in **Training Program**, including maintaining training records in Activity 6.

Servicing

ISO 9001 requires that servicing activities be performed as specified. ISO 9000-3 addresses this clause as maintenance (5.10). Although the **CMM[SM]** is intended to be applied in both the software development and maintenance environments, the practices in the CMM[SM] do not directly address the unique aspects that characterize the maintenance environment. Maintenance is embedded throughout the practices of the CMM,[SM] and they must be appropriately interpreted in the development or maintenance contexts. Maintenance is not, therefore, a separate process in the CMM.[SM] Change requests for CMM v1.0 expressed a concern about using the CMM[SM] for maintenance projects, and some wording was changed for CMM v1.1 to better address the maintenance environment. We anticipate that this will remain a topic of discussion as we provide guidance for tailoring the CMM[SM] to different environments, such as maintenance, and begin the next revision cycle for the CMM.[SM]

Statistical Techniques

ISO 9001 states that, where appropriate, adequate statistical techniques are identified and used to verify the acceptability of process capability and product characteristics. ISO 9000-3 simply characterizes this clause as measurement (6.4). The practices describing measurement in the **CMM[SM]** are distributed throughout the key process areas. Product measurement is typically incorporated into the various **Activities Performed** practices, and process measurement is described in the **Measurement and Analysis** common feature.

Activity 5 of **Organization Process Definition** describes the establishment of an organization process database for collecting process and product data. This database is maintained at the organization level, and it seems likely that most auditors would accept project-level data (as described in the project management key process areas at Level 2) to satisfy this clause. At least a few auditors do, however, require an organization-level historical database and the use of simple statistical control charts. If statistical process control is inferred from this clause, it would be satisfied by **Quantitative Process Management** and **Software Quality Management**. Note, however, that statistical techniques are used *"where appropriate."* Some auditors look for use of any statistical tools, such as Pareto analysis. Other auditors are satisfied by any consistently collected and used measurement data. There is a significant degree of interpretation of this clause by auditors.

CONTRASTING ISO 9001 AND THE CMM[SM]

Clearly there is a strong correlation between **ISO 9001** and the **CMM**,[SM] although some issues in ISO 9001 are not covered in the CMM,[SM] and some issues in the CMM[SM] are not addressed in ISO 9001. The levels of detail differ significantly: chapter 4 in ISO 9001 is about five pages long, chapters 5, 6, and 7 in ISO 9000-3 comprise about 11 pages, and the CMM[SM] is over 500 pages long. There is some judgment involved in deciding the exact correspondence, given the different levels of abstraction.

The clauses in ISO 9001 with no strong relationships to the CMM[SM] key process areas, and which are not well-addressed in the CMM,[SM] are purchaser supplied product (4.7) and handling, storage, packaging and delivery (4.15). The clause in ISO 9001 that is addressed in the CMM[SM] in a completely distributed fashion is servicing (4.19). The clauses in ISO 9001 for which the exact relationship to the CMM[SM] is subject to significant debate are corrective action (4.14) and statistical techniques (4.20). The biggest difference, however, between these two documents is the emphasis of the CMM[SM] on continuous process improvement. ISO 9001 addresses the minimum criteria for an acceptable quality system.[2] It should also be noted that the CMM[SM] focuses strictly on software, while ISO 9001 has a much broader scope: hardware, software, processed materials, and services [Marquardt91].

The biggest similarity is that for both the CMM[SM] and ISO 9001, the bottom line is *"Say what you do; do what you say."* The fundamental premise of ISO 9001 is that every important process should be documented and every deliverable should have its quality checked through a quality control activity. ISO 9001 requires documentation that contains instructions or guidance on what should be done or how it should be done. The CMM[SM] shares this emphasis on processes that are documented and practiced as documented. Phrases such as conducted *"according to a documented procedure"* and *following "a written organizational policy"* characterize the key process areas in the CMM.[SM] The CMM[SM] also emphasizes the need to record information for later use in the process and for improvement of the process. This is equivalent to the quality records of ISO 9001 that document whether or not the required quality is achieved and whether or not the quality system operates effectively [TickIT, p. 120].

The Need for Judgment

When making a more detailed comparison, some clauses in **ISO 9001** are easily mapped to their equivalent **CMM[SM]** practices. Other relationships map in a many-to-many fashion, since the two documents are structured differently. For example, the training clause (4.18) in ISO 9001 maps to both the Training Program key process area and the training and orientation practices in all of the key process areas. Satisfying a key process area depends on both implementing and institutionalizing the process. Implementation is described in Activities Performed; institutionalization is described by the other common features.

In general, practices in **Commitment to Perform** (policies, leadership) can be considered addressed under ISO 9001's clause on management responsibility (4.1). Practices in **Ability to Perform** (training, resource allocation, tools, and organizational structures) can be considered addressed under ISO 9001's clauses on management responsibility (4.1) and training (4.18) and ISO 9000-3's clauses on rules, practices, and conventions (6.5) and tools and techniques (6.6). Practices in **Measurement and Analysis** can be considered addressed under ISO 9001's clauses on quality records (4.16) and statistical techniques (4.20) and ISO 9000-3's clause on measurement (6.4). Practices in **Verifying Implementation** (senior management oversight, project management review, and audits) can be considered addressed under ISO 9001's clauses on management responsibility (4.1) and quality system (4.2).

As this illustrates, the element of judgment in making this comparison is significant. A preliminary comparison of the concepts in ISO 9001 and the CMM[SM] would suggest that an organization with an ISO 9001 certificate should be at Level 3 or 4. In reality, there are Level 1 organizations with certificates. One reason is variability of interpretation; it is absolutely clear

that the design reviews in ISO 9001 correspond directly to the CMM[SM]'s peer reviews if one has gone through the TickIT training. Another reason, however, is that achieving Level 2 implies mastering the Level 2 key process areas. Due to the high level of abstraction in ISO 9001, it is unclear what degree of sophistication is required to satisfy an auditor.

The Key Process Area Profile of an ISO 9001-Compliant Organization

What would be the maturity level of an **ISO 9001** compliant organization, if it implemented no management or engineering practices not called out by ISO 9001? This is an extreme case, but it gives a lower bound for the maturity of an ISO 9001 compliant organization. The key process area profile of an ISO 9001-compliant organization has no quality practices beyond those directly called out in ISO 9001. Where there may be a matter of judgement involved, the judgment interpretation is also illustrated in the profile. Key process areas may be partially or fully satisfied, satisfied under some interpretations, or not satisfied.

Based on this profile, a Level 1 organization according to the CMM[SM] could be certified as compliant with ISO 9001. That organization would, however, have significant process strengths at Level 2 and noticeable strengths at Level 3. Private discussions indicate that many Level 1 organizations have received **TickIT** certificates. Surveillance audits may identify deficiencies later that result in loss of certification. Other organizations have identified significant problems during a CMM[SM]-based assessment that had not surfaced during a previous ISO 9001 audit [Coallier94]. Given a reasonable implementation of the software process, however, an organization that obtains and retains ISO 9001 certification should be close to Level 2.

Can a Level 3 organization be considered compliant with ISO 9001? Even a Level 3 organization would need to ensure that the delivery and installation process described in clause 4.15 of ISO 9001 is adequately addressed and should consider the use of included software product, as described in clause 6.8 of ISO 9000-3. This would be comparatively trivial for a Level 3 organization; even a Level 2 organization would have little difficulty in obtaining ISO 9001 certification.

CONCLUSION

Although there are specific issues that are not adequately addressed in the **CMM,**[SM] in general the concerns of ISO 9001 are encompassed by the **CMM.**[SM] The converse is less true. ISO 9001 describes the minimum criteria for an adequate quality management system rather than process improvement, although future revisions of ISO 9001 may address this concern. The differences are sufficient to make a rote mapping impractical, but the similarities provide a high degree of overlap.

Should software process improvement be based on the CMM,[SM] with perhaps some extensions for ISO 9001 specific concerns, or should the improvement effort focus on certification concerns? A market may require ISO 9001 certification, and Level 1 organizations would certainly profit from addressing the concerns of ISO 9001. It is also true that addressing the concerns of the CMM[SM] would help organizations prepare for an ISO 9001 audit. Although either document could be used to structure a process improvement program, the more detailed guidance and greater breadth provided to software organizations by the CMM[SM] suggest that it is the better choice (a perhaps biased answer). In any case, building competitive advantage should be focused on improvement, not on achieving a score, whether the score is a maturity level or a certificate. We would advocate addressing the larger context encompassed by the CMM,[SM] but even then there is a need to address the still larger business context, as exemplified by **Total Quality Management.**

REFERENCES

Coallier(94), Francois, "How ISO 9001 Fits Into the Software World," *IEEE Software*, Vol. 11, No. 1, January 1994, pp. 98-100.

Durand(93), Ian G. Durand, Donald W. Marquardt, et al., "Updating the ISO 9000 Quality Standards: Responding to Marketplace Needs," *ASQC Quality Progress*, Vol. 26, No. 7, July 1993, pp. 23-30.

Lloyd's(94) Register TickIT Auditors' Course, Issue 1.4, Lloyd's Register, March 1994.

Marquardt(91), Donald, et al., "Vision 2000: The Strategy for the ISO 9000 Series Standards in the '90s," *ASQC Quality Progress*, Vol. 24, No. 5, May 1991, pp. 25-31.

Paulk(93[a]), Mark C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

Paulk(93[b]), Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush, *Key Practices of the Capability Maturity Model, Version 1.1*, Software Engineering Institute, CMU/SEI-93-TR-25, February 1993.

Paulk(93[c]), Mark C., "Comparing ISO 9001 and the Capability Maturity Model for Software," *Software Quality Journal*, Vol. 2, No. 4, December 1993, pp. 245-256.

Paulk(94), Mark C., *A Comparison of ISO 9001 and the Capability Maturity Model for Software*, Software Engineering Institute, CMU/SEI-94-TR-12.

TickIT: A Guide to Software Quality Management System Construction and Certification Using EN29001, Issue 2.0, U.K. Department of Trade and Industry and the British Computer Society, 28 February 1992.

ACKNOWLEDGMENTS

I would like to express my appreciation to the many people who commented on the early drafts of this paper and who discussed the relationships between ISO 9001 and the CMM.[SM] In some cases, we have agreed to disagree, but the discussions were always interesting. I take full responsibility for any errors in this comparison. I would like to specifically thank Peter Anderson, Robert Bamford, Kelley Butler, Gary Coleman, Taz Daughtrey, Darryl Davis, Bill Deibler, Alec Dorling, George Kambic, Dwight Lewis, Stan Magee, Helen Mooty, Don O'Neill, Neil Potter, Jim Roberts, John Slater, and Charlie Weber.

NOTES

1 There are several other standards and guidelines in the ISO 9000 series, including ISO 9002, ISO 9003, ISO 9004, and ISO 8402. ISO 9002, *"Quality systems — Model for quality assurance in production and installation,"* is for use when conformance to specified requirements is to be assured by the supplier during production and installation. ISO 9003, *"Quality systems — Model for quality assurance in final inspection and test,"* is for use when conformance to specified requirements is to be assured by the supplier solely at final inspection and test. ISO 9004, *"Quality management and quality system elements – Guidelines,"* describes a basic set of elements by which quality management systems can be developed and implemented. ISO 8402, *"Quality – Vocabulary,"* defines the basic and fundamental terms relating to quality concepts, as they apply to products and services, for the preparation and use of quality standards and for mutual understanding in international communications. There are also a number of guides, such as ISO 9000-3, which are additional parts to standards in the ISO 9000 series.

2 This statement is controversial in itself. Some members of the international standards community maintain that if you read ISO 9001 with insight (between the lines so to speak), it does address continuous process improvement. There is faith that weaknesses will improve

over time, especially given regular surveillance audits. Corrective action can be interpreted in this way, although that may not be consistently done today. This will undoubtedly be one of the major topics for the next revision cycle for ISO 9001.

Mark C. Paulk
Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

Blank page.

APPENDIX

J

Counting Rules for Function Points and Feature Points

Blank page.

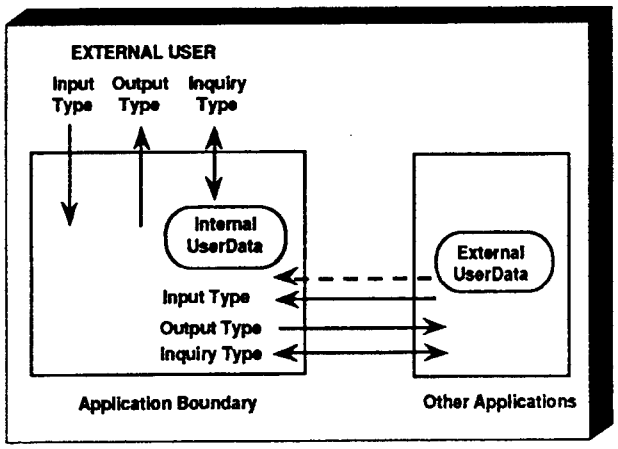
APPENDIX J

Counting Rules for Function Points and Feature Points

Software Productivity Research, Inc.

INTERNAL USER DATA GROUP (IU)

Count each major logical group of user data or control information in the application as an internal user data group, include each logical file, or within a database, each logical group of data from the view point of the user that is generated, used, and maintained by the application. Count each logical group of data as viewed by the user and as defined in the external design or data analysis rather than on the physical implementation. IUs maintained by more than one application are counted as IUs by each.



APPENDIX J Feature/Function Point Counting Rules

Counting Recommendations

Do ***not*** include logical internal files that are not accessible to the user through external input, output, or inquiry types.

DESCRIPTION	COUNT
Logical entity of data from user viewpoint	1 IU
Logical internal file generated or maintained by the application	1 IU
User maintained table or file	1 IU
File accessible to the user through keyword(s) or parameter(s)	1 IU
File used for data or control by sequential (batch) application	1 IU
Each hierarchical path (leg) through a data base, derived from user requirements	1 IU
Intermediate or sort work file (temporary)	0 IU
File created by technology used (e.g., index file)	0 IU
A "master file" only read by application	0 IU, 1 EU

EXTERNAL USER DATA GROUP (EU)

Count each major logical group of user data or control information used by the application (but maintained by another application) which crosses the application boundary. Include each logical file or logical group of data from the viewpoint of the user that is used by the application.

Counting Recommendations

Count each major logical group of user data or control information that enters the application from another application as an external user data group.

DESCRIPTION	COUNT
File of records from another application	1 EU
Data base read from other applications	1 EU
Logical internal file from another application used as a transaction	0 EU, 1 IT
Each hierarchical path (leg) through a data base, from another application derived from user requirements	1 EU

INPUT TYPE (IT)

Count each unique user data or user control input type that enters the external boundary of the application being measured, and adds, changes, or deletes data in a logical internal file type. Also count control information which enters the application boundary and assures compliance with business function specified by the user. An external input type should be considered unique if it has a different format, or if the external logical design requires a processing logic different from other external input types of the same format.

Counting Recommendations

The recommendation most closely describing each input type should be used in counting each input type.

APPENDIX J Feature/Function Point Counting Rules

DESCRIPTION	COUNT
Data screen with add, change, and delete	3 IT
Multiple screens accumulated and processed as one transaction	1 IT
Two data screens with the same format and processing logic	1 IT
Two data screens with the same format and different processing logic	2 IT
Input that may result in error message(s)	1 IT, 1 OT
Data screen with multiple functions	1 IT/Func
Automatic data or transactions from other applications	1 IT
User application control input	1 IT
Input forms (OCR) with one transaction	1 IT
An update function following a query	1 IT
Individual selections on menu screen	0 IT
Update of user maintained table or file	1 IT
PF Key duplicate of a screen already counted as input	0 IT
Light pen duplicate of a screen already counted as input	0 IT
External input types introduced only because of the technology used	0 IT

OUTPUT TYPE (OT)

Count each unique user data or control output type that leaves the external boundary of the application being measured. An external output type should be considered unique if it has a different format, or it the external design requires a processing logic different from other external output types of the same format. External output types usually consist of reports or messages to the user.

Counting Recommendations

The recommendations most closely describing each output type should be used in counting each output type.

DESCRIPTION	COUNT
Data screen output	1 OT
Batch report	1 OT
Error message(s) format associated with input	1 OT
Transaction file crossing the application boundary (at least)	1 OT
Automatic data or transactions to other applications	1 OT
Single error message on a screen	0 OT
Error messages sent to an operator as a result of an input transaction	1 OT
Backup files (only if requested by the user)	0 OT
Output to screen and to printer	2 OT
Output files created for technical reasons	0 OT
Bar chart as well as pie chart graphical displays	2 OT

APPENDIX J Feature/Function Point Counting Rules

EXTERNAL INQUIRIES (OT)

Count each unique input/output combination, where an input causes and generates an immediate output, as an external inquiry type. An external inquiry type should be considered unique if it has a format different from other external inquiry types in either its input or output parts, or if the external design requires a processing logic different from other external inquiry types of the same format.

Counting Recommendations

The recommendation most closely describing each inquiry type should be used in counting each inquiry type.

DESCRIPTION	COUNT
Online input and online output with no update of data in files	1 QT
Inquiry followed by an update input	1 QT, 1 IT
Help screen input and output	1 QT
Online input with immediate printed output of existing data and no update of data	1 QT
Note: A major query facility or language should be decomposed into its hierarchical structure of IT(s), OT(s), and QT(s) using the existing definitions and current practices.	

FEATURE POINT METHODOLOGY

The SPR Feature Point metric is a superset of the IFPUG Function Point metric and introduces a new element (algorithms) in addition to the five standard Function Point parameters. The Feature Point method also reduces the Internal User Data Group weight from IFPUG's average value of 10 to an average value of 7. Since Feature Points include algorithmic complexity, a definition of "*algorithm*" is appropriate. An algorithm is defined as the set of user required rules which must be completely expressed in order to solve a significant computational problem. For example, a square root extraction routine, a Julian date conversion routine, or an overtime pay calculation routine are all considered algorithms.

FUNCTION POINT METHODOLOGY

The primary difference between the IFPUG and SPR Function Point methodologies is in the way they deal with complexity. The IFPUG techniques for assessing complexity are based on weighing 14 influence factors and evaluating the numbers of field and file references for transactions or the numbers of fields and record element types (user views) for data groups. The SPR technique for dealing with complexity separates the overall topic of "*complexity*" into two distinct questions that can be dealt with intuitively: (1) How complex are the algorithms or equations or problems in the software? (2) How complex is the data structure of the application? The SPR methodology is usually utilized at or prior to Requirements.

With the SPR Function Point method, it is not necessary to count the number of data element types, file types referenced, or record types. Neither is it necessary to assign low, average, or high values to each specific input, output, inquiry, data file, or interface. The SPR complexity questions can be answered quickly by anyone familiar with an application, and they deal with the entire application, rather than with its elements.

APPENDIX J Feature/Function Point Counting Rules

COMPLEXITY FACTOR

The complexity factor and multiplier is used to compute the Function Point Count measure. SPR uses a "*quick-fire*" method which can adjust the function point count by ±40%. IFPUG and SPR use the same principles to identify the five elements (six with Feature Points) of counting. IFPUG then applies a weighting factor of high, low, or average, depending on the number of data elements, file types referenced, and record types invoked. SPR's quick-fire method assumes average weight in its methodology. To adjust the raw FP count, IFPUG looks at 14 variables of complexity that will adjust the count by ±35%. SPR simply requires answers to two questions which summarize the intent of IFPUG's 14 complexity factors. By answering 1 through 5 on both questions and adding the two values together, a complexity multiplier is then obtained using the simple chart provided.

IFPUG METHODOLOGY

IFPUG METHODOLOGY

Function Point Counting

Element	Count Weights Low	Average	High	Total
Input	___x3	+ ___x 4	+ ___x 6	= ____
Output	___x4	+ ___x 5	+ ___x 7	= ____
Inquiry	___x3	+ ___x 4	+ ___x 6	= ____
IU	___x7	+ ___x10	+ ___x15	= ____
EU	___x5	+ ___x 7	+ ___x10	= ____

Total Unadjusted Function Points ____

Input Complexity Matrix

FTRs	1-4 DETs	5-15 DETs	16+ DETs
0-1	Low	Low	Avg.
2	Low	Avg.	High
3+	Avg.	High	High

Output Complexity Matrix

FTRs	1-5 DETs	6-19 DETs	20+ DETs
0-1	Low	Low	Avg.
2-3	Low	Avg.	High
4+	Avg.	High	High

User Data Group Complexity Matrix

RETs	1-19DETs	20-50 DETs	50+ DETs
1	Low	Low	Avg.
2-5	Low	Avg.	High
6+	Avg.	High	High

- FTR = File Types (User Data Groups) Referenced
- DET = Data Element Type Field
- RET = Record Element Type (User View)

IFPUG METHODOLOGY (Cont.)

Fourteen General System Characteristics are rated from 05 based upon their degree of influence on the application. The fourteen characteristics are:

- Data Communication
- Distributed Function
- Performance
- Heavily Used Configuration
- Transaction Rates
- On-Line Data Entry
- Design for End-User Efficiency
- On-Line Update
- Complex Processing
- Usable in other Applications
- Installation Ease
- Operational Ease
- Multiple Sites
- Facilitate Change

The Unadjusted Function Point count of an application is adjusted by the total of the General System Characteristics using the following equation to determine the application Function Point count:

[.65 + (.01x total of General System Characteristics)] • [Unadjusted Function Point Count]

Users should refer to the IFPUG Counting Practices Manual for more complete definitions.

"*BACKFIRE*" METHOD

The "*backfire*" method for estimating Function Points is based on empirical relationships discovered to exist between source code and Function Points in all known languages. This method is based on tables of average values. It is useful for doing retrospective studies of projects completed long ago, and for easing the transition to Function Point metrics for people who are familiar with lines-of-code metrics.

Assembler	320*
C	128
COBOL	107
Ada	71
DB Languages	40
Object Oriented	29
Query Languages	25
Generators	16
*Statements per Function Point	

APPENDIX J Feature/Function Point Counting Rules

Function Point Counting

Element	Count	Weight		Total
Input	___	x 4	=	____
Output	___	x 5	=	____
Inquiry	___	x 4	=	____
Internal User Data Group	___	x 10	=	____
External User Data Group	___	x 7	=	____
		TOTAL		___ •

OR •

Feature Point Counting

Element	Count	Weight		Total
Input	___	x 4	=	____
Output	___	x 5	=	____
Inquiry	___	x 4	=	____
Internal User Data Group	___	x 7	=	____
External User Data Group	___	x 7	=	____
Algorithm	___	x 3	=	____
		TOTAL		____

Function/Feature Count (FC)

Total Unadjusted Function/Feature Points = ____

Complexity Factor and Multiplier

Problem Complexity?
1) Simple algorithms and simple calculations
2) Majority of simple algorithms and calculations
3) Algorithms and calculations of average complexity
4) Some difficult or complex algorithms
5) Many difficult algorithms and complex calculations

Data Complexity?
1) Simple data with few variables
2) Numerous variables, but simple data relationships
3) Multiple files, fields, and data intersections
4) Complex file structures and data intersections
5) Very complex file structures and data intersections

Sum of Problem and Data Complexity	2	3	4	5	6	7	8	9	10
Complexity Multiplier	.6	.7	.8	.9	1.0	1.1	1.2	1.3	1.4

FP Count Measure:
FC X Complexity Multiplier = ____

Figure J-1 SPR Methodology

APPENDIX J Feature/Function Point Counting Rules

Blank page.

APPENDIX

K

Software Support

Blank page.

APPENDIX K

Software Support

NOTE: The following papers were written by the F-22 Life Cycle Software Support (LCSS) Integrated Product Team (IPT). These papers discuss how to manage the different aspects of Post-Deployment Software Support (PDSS), and are found in the 1994 and 1995 Software Technology Conference proceedings.

CONTENT **PAGE**

Tab 1:
How Post-Deployment Software Support Starts in the Design Phase K-2
Tab 2:
LSA for Software: A Practical Approach K-8
Tab 3:
Computer Resource Integrated Support Document (CRISD) Foundation to Post-Deployment Software Support (PDSS) K-12
Tab 4:
Software Quality Assurance Impact on Post-Deployment Software Support Cost K-18
Tab 5:
The DoD Generic Fighter: F-22's Historical Foundation K-28
Tab 6:
How More is Less: The F-22 Streamlined Block Change Cycle K-39

TAB 1

How Post-Deployment Software Support Starts in the Design Phase

Philip C. Gould
Lockheed Fort Worth Company
John E. White
ASC/YFSL, SM-ALC/YFLBA
Integrated Product Team Managers
F-22 Life Cycle Software Support

Presented at the Sixth Annual Software Technology Conference
Salt Lake City, Utah, April 15, 1994

INTRODUCTION

"*When does Post-Deployment Software Support (PDSS) start?*" Traditionally, it has started when the weapon system has passed the Initial Operational Capability (IOC) milestone. From that point forward, "*maintenance*" is performed by someone — either by the contractor or a government support agency. Now, let's rephrase the question to "***When should** PDSS start?*" Not only has the question changed but the logical answer is also different. What follows is the story of the approach being taken by the F-22 program — where PDSS starts in the ***design*** phase with the result being effective, efficient ***Life Cycle*** Software Support (LCSS).

The Team

The F-22 Weapon System is under Engineering and Manufacturing Development (EMD) by a team consisting of the United States Air Force (USAF), two prime contractors (Lockheed Aeronautical Systems Company [LASC], with Lockheed Fort Worth Company [LFWC] and Boeing Military Airplanes [BMA] as principle subcontractors, and Pratt & Whitney), and numerous subcontractors. The USAF contingent is further structured as the System Program Office (SPO) at Wright-Patterson AFB, Sacramento Air Logistics Command (SM-ALC, also known as SPO-West) at McClellan AFB, Air Combat Command (ACC) represented by Langley AFB, Eglin AFB, and Nellis AFB, and Air Education and Training Command (AETC) personnel. The program is being directed utilizing Total Quality Management (TQM) principles which results in an Integrated Product Team (IPT) for each major portion of the F-22 Weapon System. The LCSS IPT has representation from all of the above groups, for only through cooperation of all parties involved can the concepts and products of EMD be transitioned and maintained throughout the remainder of the F-22's life expectancy.

APPENDIX K Software Support

The Challenge

The Cold War is over and the Big Bear has died. Unfortunately, some of our industrial defense base has also dissolved — either by closing or merger or layoff. The defense budgets of the present are only a fraction of the past and competition for those dollars is intense. Some of that competition is between programs, between services, and between support organizations within a given service as well as between government support agencies and industry. A different type of competition, but just as intense, consists of acquisition dollars versus funding for support. In the present scenario, estimated excessive support costs can limit the number of units to be procured — or even cancel a program. In the software world, it is often said that, over the life of a system, 30% of the software cost is in development and 70% is in "*maintenance*" (let's say PDSS). In other words, if you spend $2 billion developing a system's software, you'll spend about $4.6 billion to support that software over the system's life. If you take a life cycle of 20-30 years, that's roughly $150 to $230 million dollars per year — which makes a very large target for the budget cutters. Looking at it another way, if you take the pseudo-standard rule that 10% of a given program will change per year and apply that to a weapon system of about 5 million lines-of-code, you've obviously found a way to keep a major portion of the free world's Ada programmers employed — if Congress will give you that annual $150-$230 million. The F-22 LCSS IPT was created to find new and creative ways to reduce life cycle software costs by "*designing in*" supportability.

The History

In the past, PDSS has typically been addressed by some lone voice crying in the wilderness who at one time was given somebody else's code to maintain and who had to learn software supportability the hard way. This lone voice was traditionally a staff person who had no budget, people, or clout and, therefore, nobody listened. In the 1985/86 time-frame, some of those lone voices (Sylvia Blair, Jon Floyd, and Phil Gould) banded together in Ft. Worth and created a brand new entry in the Work Breakdown Structure (WBS) dictionary of the Dem/Val Advanced Tactical Fighter proposal from General Dynamics (now LFWC) to address life cycle software issues. Although the GDFW proposal did not survive the down-select, the LCSS concept impressed the appropriate people (Thank you, Col. Bob Lyons!) and was added to the WBS dictionary for the follow-on RFP. Never before had a major weapon system addressed this problem with an appropriate method of funding the task this early in the life of the system.

The Mission

The mission of the F-22 LCSS IPT is very succinct — ***To insure the efficient support of all F-22 Weapon System software.*** Upon closer review, however, the enormity of the task looms ominously. We have endeavored to approach this task with a five-fold plan of attack :

1. **Analyze EMD development requirements.** Software Maintenance is nothing more than multiple, small iterations of the entire life cycle. If you think of a block update, you start with analysis of the new or modified requirements, proceed through design, code, test, and integration just as if you were implementing a small, autonomous project. Therefore, the requirements for software support are very similar, if not identical, to the requirements necessary to develop the original software. Unfortunately, these requirements often change from phase to phase of development and are not usually documented adequately to reconstruct for life cycle support. We are capturing this data as part of the requirements analysis.
2. **Influence development for maximum supportability.** Anyone who has modified someone else's code can attest to the fact that supportability is a design factor. Structured coding concepts have calmed the nerves of many software maintainers — until they look (in vain) for adequate documentation, or design rationale, or that little test program that the developer "*whipped up*", or..., etc. Our goal is to insert supportability considerations into the code, the documentation, and the environment of each piece of F-22 Weapon System software. This thrust must begin during design or else the cost of re-engineering, rewriting, etc. will be considered prohibitive.

3. **Develop post-deployment support concepts.** Many government agencies have inherited software systems for which no support concept was ever developed. If concepts for support are not considered during system development, there is no way to have organic support available at the earliest need dates considering government procurement cycles, especially if facility construction is required. This traditional approach defaults to contractor support for the first few years of PDSS. By developing these support concepts from the beginning of development and maturing them during EMD, intelligent, efficient decisions can be implemented.
4. **Predict F-22 software changes.** Once again, the approach driving this concept is a departure from tradition. Most (all?) programs have taken the direction that "*software is software*" and it all must be supported in the same fashion — by throwing money at it after the fact. This "*lack of support*" concept often leads to software support capabilities being (expensively) developed for software that may never change over the life of the system. Sometimes this inefficiency precludes money from being available to create support where it is really needed. To combat this inefficiency, we are looking at past programs to review the systems that were most prone to change, the extent of and reason for those changes, and the cost involved. Also, if you could predict the software that was most likely to require modification, you would know where to concentrate more effort during development to insure supportability, right? Well, that's exactly what we're trying to do.
5. **Identify post-deployment support requirements.** If you can determine what software is most likely to change and develop the concepts to support that software, you should be able to accurately identify the requirements necessary for support in a timely manner — at least that's how we envision it working. At that point, all that's left is to document the required capabilities for procurement.

The Products

Contractually, we have only two products — the **Computer Resources Integrated Support Document (CRISD)** and a specification for the software support facility. A Weapon System level CRISD is being developed as a formal CDRL in accordance with DoD-STD-2167A and in compliance with a coordinated tailoring of DI-MCCR-80024A. As far as we can tell, this will be the first fully 2167A-compliant CRISD for a major weapon system. Even here, we're breaking new and challenging ground. (A later presentation by Ms. Fleming will discuss CRISD development.) The proposed software facility has been named the Integrated Weapon System Support Facility (IWSSF). Our second product is the specification from which that facility can be produced. It is viewed as a "*build to*" spec for the entire support facility, although not all of it may be in the same building, or the same city, or operated by the same agency. It will be submitted in contractor format and is not a CDRL.

The CRISD can be thought of as the formal documentation of the concepts with the IWSSF spec documenting the resource requirements necessary to implement those concepts. The preliminary versions of the documents will be available to assist in the decision to begin Low-Rate Initial Production (LRIP). A second version of each is also funded under EMD which will assist in a final production decision. We anticipate funding a final update under the LRIP contract for delivery just prior to IOC. While the contractor team is on the hook for these documents, USAF has to develop the Computer Resources Life Cycle Management Plan (CRLCMP) to give us an idea of their desires. Just as the government representatives of our team will assist in the review and comment of the CRISD and IWSSF spec, the contractor members of the IPT are assisting in review and comment of the CRLCMP. The stronger the communication and coordination, the less chance of surprise. As we progress, various concepts and processes are being documented in informal White Papers for the edification of both industry and the government. Since much of what we are attempting to do has never been done before, the broader we distribute lessons learned, the less pain others will endure by not being required to reinvent the wheel.

APPENDIX K Software Support

THE DILEMMA, THE DIRECTION, & THE SMART APPROACH

The argument over organic versus contractor support has raged for years. The reason that the debate still exists is that no answer is correct for all cases. Levels of management and technical risks and utilization of existing facilities, personnel, and expertise at both government and industry facilities are factors that vary over time as well as from program to program. The F-22 program has already experienced both extremes of opinion. An original Program Management Directive (PMD) stated that we should *"...plan for total organic support."* On the other end of the spectrum, some people within USAF felt that if the contractor developed a product, then the contractor should be in the best position (cost, expertise, etc.) to support that product. This led to one general's comment of *"Why do I need a depot?"* While this may originally sound like music to the ears of a contractor, the truth is that nobody can possibly know the right answer without proper investigation. Having been hit over the head with two diametrically opposing viewpoints, our IPT approached MGen. (sel.) Raggio for a heading check. The first thing that he made clear was that one reason for our existence was to preclude business as usual and to replace past practices with Integrity, Teamwork, and Logic. His direction was clear — there was no correct answer at this point in time. We were tasked to go determine what was smart for the program. But how? AFR 66-7 outlines a decision process for determination of post-deployment support. As far as we can determine, no one has ever tried to apply this process to software support. Our IPT has utilized this process to develop a decision tree which will assist in arriving at the optimal contractor/organic work split for F-22 Weapon System software support. Currently, we are investigating the history of recent similar programs in an effort to quantify the factors which constitute the branches of the tree. Once quantified, the criteria will be applied to each subsystem to determine a preliminary breakdown of contractor versus organic support. This will be documented in a White Paper for use in long-range depot planning.

THE FOUNDATION, THE INFRASTRUCTURE, & THE PROCESS

The foundation for development of F-22 software support is based on the functional capabilities required for F-22 software development and how these capabilities are currently implemented. Our IPT's infrastructure to capture this data is through active participation in the IPTs responsible for developing software. Through participation in their meetings and design review and evaluation of the documentation that they produce, we can collect the data necessary to determine the functional capabilities and concepts required for PDSS while raising issues and concerns, as required, to enhance supportability.

While the CRISD and IWSSF spec are our main products, a portion of the CRISD will be devoted to a transition plan detailing how the required capabilities and resources may be implemented at the desired site(s). The key, as we see it, is that the ***functionality*** required for support must be in place when required, but not necessarily in the exact form used for development. A simple example is that right now I'm producing this document in Microsoft Word for Windows, Version 2.0c. If this document were essential for PDSS, would I be required to transition this exact version of Word (and Windows, and the hardware) to the support facility or, rather, the functionality necessary to read, edit, and print this document when required? Hopefully, the answer is obvious. If PDSS implementation is approached simply as a duplication of all the development environments and labs, the task is much easier — in fact, it's a no-brainer. All you have to do is accumulate all the equipment lists and find a copy machine. Unfortunately, many people have this view of the task and the result is an overwhelming conglomeration of sometimes redundant and often obsolete equipment. That approach certainly isn't efficient, effective, or smart. The LCSS IPT plans to change that outcome.

THE SOURCES & THEIR DATA

As you may have already deduced, the development of documents which apply to an entire weapon system does not lack for a shortage of source data. Our sources of data are the IPTs — all IPTs that develop, use, integrate, or test any software on or in support of the F-22 Weapon System (Air Vehicle, Support System, Training System). This applies not only to the prime contractors but also to the subcontractors, their subcontractors, etc. The sources of our data literally extend from coast-to-coast and border-to-border. Now, concerning the nature of the data; all that we're interested in is everything! We need to know what was used (facilities, equipment, tools, people, etc.) in every phase (analysis, design, code, integration, test) to manufacture every piece of F-22 software. All of this information goes into a database (can you think of any other way to analyze it?) which was hammered out by a subset of our IPT (contractors and government) locking themselves in a room for three days and brainstorming on two walls of a large conference room which had been lined with flip-chart paper. The result was formulated into a relational database design which is still being updated as required, even as data entry proceeds. Through analysis of this database, we plan to distill the essential requirements for efficient, cost-effective PDSS.

THE CHANGE MODEL

Earlier we mentioned an initiative to attempt the prediction of software changes during F-22 PDSS; let us elaborate. The LCSS IPT is collecting data from past fighter programs in an attempt to identify trends which may be used to either influence software design in traditionally volatile areas or focus areas where software support is more crucial. We currently have plans for visits to the F-14, F-15, F-16, and F/A-18 software support agencies in an attempt to collect data for use in the development of this change model. We have developed a series of questions which have preceded us to these sites and which provide the support agencies appropriate lead time to prepare the answers. The data that we feel is necessary to develop this model involves the reasons driving the changes, the systems and subsystems which were required to change, the size of the change, the length of time required to make the change, the quantity and types of people required to make the change, etc. This data will be analyzed as part of our historical change model.

The four programs for which we have visits scheduled have been extremely cooperative in our efforts. (Of course, an introductory letter from the desk of our MGen. doesn't hurt.) There are also other programs which we may tap for data in an attempt to refine and/or verify our model (B-1, B-2, F-111, F-117, A-10, E-3). The other half of the process involves an analysis of the software currently being developed. We produced a second set of questions for the IPTs who are developing the software for the F-22 which asked them, the system and subsystem experts, where they would anticipate future software changes to occur, an estimate of the size and impact of the change, and what could be done now, during design and code, to minimize the change. Even though the IPTs are extremely busy with their own development, they have been gracious enough to cooperate. The answers to these questions have not only helped us to begin the F-22 portion of the model but, in some cases, have led to adjustments in software design to minimize potential impacts in areas of higher volatility.

THE ACCOMPLISHMENTS

While the CRISD and IWSSF spec are products for which the LCSS IPT has total responsibility, there are many additional processes and products which we are constantly influencing. The F-22 has developed a **Weapon System Software Development Plan** which complies with -2167A in an attempt to define and standardize the software development process at the highest possible level. While the LCSS IPT was not tasked with development of the document, we have authored sections of the document and have had major influence on many other sections.

APPENDIX K Software Support

As with the SDP, all software documents have been subjected to Software Product Evaluations (SPEs). The LCSS IPT has been a major contributor in this process, which has produced documentation which is an accurate reflection of the development and should actually be useful in the post-deployment scenario. The SPEs are being done via electronic conferencing among multiple locations with meetings held only to kickoff the process and to summarize the results. We are utilizing a series of evaluation guidelines which were developed by Draper Labs under contract to SM-ALC. These guidelines assess the technical content of the documents for their usefulness rather than the traditional reviews to catch spelling and grammar errors.

The F-22 received much publicity early in the program for its attempt to utilize a common, COTS-based development environment, which was labeled the **System/Software Engineering Environment (S/SEE)**. The S/SEE is in use and is the common foundation for software development, documentation, configuration management, etc. As systems identify requirements not available through the core S/SEE which are peculiar to their needs, these are added as extensions to the core which are not procured in mass quantities. Wherever possible, COTS has been used. This has led to some inevitable interface problems, some problems in timing of new version releases from multiple vendors, etc. On the whole, however, the S/SEE has been successful when compared to past methods. The LCSS IPT will be tasked with the transition of the ***functionality*** of the S/SEE tools to the IWSSF for PDSS. Transition of the exact tools currently in use, however, is very doubtful. After all, what tool won't be obsolete in 8-10 years?

Another area of major effort has been in support of development of the OFP build concept for the air vehicle. (For the air vehicle, Mr. Lax will present more later.) Suffice it to say that with three large contractors developing, testing, and integrating software and numerous subcontractors adding pieces at all three sites, the combination of MSLOCs of code into a loadable entity is nontrivial — and that's even without considering the security implications. Our interest in the development of this concept is grounded in the reality that whatever concepts are used during the flight test program typically become the *de facto* processes for post-deployment. Therefore, our IPT has a vested interest in helping get this process off on the right track and has contributed much time and expertise to this effort.

Finally, the core of PDSS is centered around the block change cycle. While software doesn't *"break"* in the traditional sense, no successful major weapon system spends its entire life cycle in the same mode and configuration as its original release. Typically, this is accomplished by a structured block change program which involves multiple organizations from both the government and the contractors. The development of the process and the identification of the players for a block change is a building block for the identification of support requirements. The block change process involves not only the technical aspects but the political considerations of contractor and government work, definition of responsibilities within the government, and the color of money which must be used for various efforts. Therefore, support of this effort is crucial to the success of the IPT and, therefore, the success of PDSS.

THE FUTURE

As you should be able to conclude, the LCSS IPT certainly does not lack for work. While continuing to try and influence development for the good of supportability, we are deeply involved in the analysis of the data which we are accumulating (development requirements, historical and predictive volatility, etc.). Additionally, our government members are responsible for the updating of the Computer Resources Life Cycle Management Plan (CRLCMP) which will serve as a companion document to the CRISD. By the end of the year, the IPT should have quantified the criteria of the decision tree and released a preliminary contractor/organic work split. Based on this work split and the preliminary change model, a preliminary cost estimate for software support for the F-22 life cycle can be produced. All of this leads to the development of the preliminary CRISD and IWSSF spec which will be utilized as support documentation for the LRIP decision. After all, the bottom line is that if you can't afford to support it, why bother to build it. That is precisely why *"Post-Deployment Software Support **MUST** Start in the **DESIGN** Phase!"*

TAB 2

LSA for Software: A Practical Approach

Jerry A. Raddatz
F-22 Life Cycle Software Support IPT
Lockheed Fort Worth Company
Mike Donlon
F-22 Life Cycle Software Support IPT
ASC/YFSL, SM-ALC/YFLBA

Presented at the Sixth Annual Software Technology Conference
Salt Lake City, Utah, April 15, 1994

Much has been written and discussed about the possible documentation of software support requirements in accordance with MIL-STD-1388, Logistics Support Analysis. While this standard is definitely geared toward hardware development, MIL-STD-1388 does state that the standard applies to both hardware and software. Additionally, all Department of Defense (DoD) programs are required to adhere to DoD-STD-2167A for software development, which uses other methods for the documentation of software development data. Much of the software support data appears in DI-MCCR-80024A, the Computer Resources Integrated Support Document (CRISD). This paper addresses the approach being taken by the F-22 Life Cycle Software Support (LCSS) Integrated Product Team (IPT) for the collection and analysis of the software development process source data — the LCSS Analysis Database (LAD).

The F-22 Weapon System is divided into three parts — Air Vehicle, Support System and Training System. Approximately 5 million lines-of-code will be required for the F-22 Weapon System. The LCSS Analysis Database has been created to help define and collect the software support requirements for all three parts of the F-22 Weapon System. The LCSS IPT has recognized that analyzing all the documents that define the F-22 software would be a monumental and costly task. After several meetings, it was agreed that a database was needed to capture all the pertinent software developmental data needed to define the software support concepts and environments necessary for the F-22 life cycle. In these meetings it was decided that the basic purposes of the LCSS database are:

- Provide a summary of the support requirements for each CSCI through each phase of software development,
- Provide source data for analysis which results in an optimum integrated Weapon System Software Facility (IWSSF) configuration,
  - Provide source data for the CRISD,
  - Provide a historical basis for life cycle studies to be used for future acquisition decisions, and
  - Provide applicable LCSS data to LSAR.

It was also recognized that some of the existing LSA definitions could be used for the LAD with little or no modification. Wherever possible, LSA definitions such as "*Skill Specialty Code*" or "*Vendor*" were used in the LAD so that any information common to both the LSA database and the LCSS database could be easily exported from the LAD to the LSA database. In

some cases, the LSA definition needed to be extended or slightly modified for use by the LCSS IPT. Most of these modifications are simply adding more codes to the existing list for a given LSA data element and should be easily incorporated into the LSA database. Examples of this type of data element are "*Classification*" and "*End Item Id*". The "*Classification*" element list of codes was increased to include F-22 specific security classifications. The "*End Item ID*" element list of codes was increased to include more types of system/subsystem/operational flight program (OFP) designators. The LAD was developed by breaking down the software development process for each phase of a generic CSCI into the elements required to complete each phase of development. Each CSCI goes through seven phases during its development process:

- Software requirements analysis,
- Preliminary design,
- Detailed design,
- Coding and computer software unit (CSU) testing,
- Computer software component (CSC) integration and test,
- CSCI testing, and
- System integration and test.

Each phase may have different personnel requirements; different in the number of personnel required or different in type, educational requirements, training, or years of experience. Each phase may require different facilities; different in size, power usage, furnishings, security classification level, and location. There are many other types of software development phase-related support elements contained in the LAD which will not be detailed here. Obviously, the LCSS Analysis Database will be large since there are over 400 CSCIs on the F-22 program, each with seven phases of development and each phase containing many data elements. A relational database was deemed the best way to handle the large amounts of data to be analyzed and an entity relationship diagram (ERD) was created that defined the known relationships between all the data elements. The basic ERD was created as a result of two days brainstorming with the government. The walls were covered with charts, drawings and lists, several meetings were held, work was done on a PC and out popped the first ERD. After several discussions, the ERD was approved by the LCSS IPT membership. However, the ERD was recognized to be a living document and minor updates will occur as required.

After the ERD was finalized, a database tool was selected. Several existing database tools were studied and tested. Microsoft Access was chosen after several months of analysis and testing because it was deemed the easiest to use of those tools available at that time **AND** the database could be easily ported to the F-22 System/Software Engineering Environment (S/SEE) when appropriate.

The next step in the database development process was to develop a questionnaire that could be given to the various F-22 IPTs. The questionnaire covered all the data elements required for the LAD and yet would be easy for the IPTs to use without taking up large amounts of their time. The questionnaire was developed largely in tabular form and was successfully used to gather software development data prior to Preliminary Design Review (PDR). The questionnaire will continue to be modified as necessary and used to gather software development data throughout each phase of the developmental life cycle of the F-22 CSCIs. The database allows us to map all the software support requirements across all the development phases. All of this data gathering begs the question—"*How does this database relate to the CRISD*?" The CRISD is the repository for the collective wisdom for all the software development processes for all of the F-22 Weapon System software. All of this "*wisdom*" must be analyzed and condensed into the support concepts and resource requirements necessary to maintain the F-22 Weapon System throughout its life cycle — 30 years or more.

The F-22 program has recognized that software supportability cost is a large part of the weapon system cost over the life cycle of the aircraft fleet. The F-22 Team has placed a great deal of emphasis on the CRISD as **THE** document that will provide cost effective software supportability direction for the program. Logistics Support Analysis (LSA) is a subset of the

system engineering process that includes supportability as a design parameter equal to cost, schedule and performance. Most of the data elements and parameters used by the LSA community are designed for hardware use only, even though MIL-STD-1388 applies to the entire system, including software.

Software is a logical, rather than a physical, part of a system. As such, software does not "*wear out*", "*break*", or "*fail*" in the traditional sense of the words. Properly developed and tested software will execute in exactly the same manner every time it is executed. Software is a set of instructions that tell hardware what to do. Since software is a logical element, not a physical element, software has no "*weight*" but does occupy space. Since software occupies space, it must reside in something (memory), therefore, software has an inherent weight. This type of seemingly illogical description of the attributes of software results in much of the discussion and confusion concerning the software development process and how to support that process.

Mean-time-to-repair (MTTR) and mean-time-between-failure (MTBF) are typical hardware parameters that have little or no relationship to software. Time to "*repair*" software is an inaccurate phrase since properly operating software never "*fails*" or needs "*repair.*" However, sometimes new requirements are levied on the software, unanticipated operational conditions are imposed or latent errors appear which require that the software be updated. When this type of change is required, two basic situations result. The first situation is that in which a change is an emergency, (i.e., a war scenario is involved and changes need to be made immediately). This type of change can be made in as little as two or three days (sometimes) and then distributed to the fleet for use. The other types of changes result in non-emergency changes to the software. Typically these changes wait until the next scheduled block release for incorporation into the fleet base line. Neither one of these types of change can be graded against time since the change and the time it takes to implement the change are not related.

The LSA database is slanted towards typical hardware data elements; weight, size, shelf life, etc. The LSA database uses LSA Control Numbers (LCNs) to track and status each item in the database. The LCN system of hardware configuration control can be adapted for software by relating LCNs to computer software configuration items (CSCIs). CSCIs are already used for software configuration management and control of the software development process. Many data elements of the CRISD analysis database are similar to existing LSA elements and require only minor modifications to the LSA database in order to provide common definitions.

By using the software IPTs as sources for the LAD data, the LCSS IPT will acquire the software developmental information in a timely manner before it is forgotten or lost. The software product IPTs have been very helpful in providing their software development process information. In addition to the questions asked of them, the IPTs have also provided additional insight into the software development process that was missed in the initial LAD database design. The LCSS IPT has enhanced the questionnaire and the database to include the IPT recommendations. So far, the data collection process has gone well.

When the F-22 Team acquires a database tool for the System/ Software Engineering Environment (S/SEE), the LCSS IPT will move the LAD from its current PC based environment to the S/SEE so that other members of the Team will have on-line access to the database. The LCSS IPT will analyze, condense, shuffle, fold, spindle, mutilate and sort the LAD data until it is determined that the most cost effective software support environment has been defined. Unnecessary duplicate resources will be eliminated and facilities consolidated. Personnel requirements will be studied and a best possible mix of disciplines and experience will be recommended. The results of this analysis will be turned into CRISD input (both textual and tabular) and will form the basis for deciding the best support concepts and support requirements for the F-22 Weapon System Software. All of this analysis is made possible because of the existence of the LCSS Analysis Database. It provides an efficient and logical way to organize, analyze, and merge the various data elements that are used in the software development processes for the F-22. The use of the LAD:

- Supports collection and analysis of large volumes of data,
- Helps identify resource dependencies in the software development and test processes,

APPENDIX K Software Support

- Helps to ensure that all systems are reviewed in order to determine each systems impact on the software support concepts and resources, and
- Results in the optimum Post-deployment Software Support concepts and resources for the life cycle of the F-22.

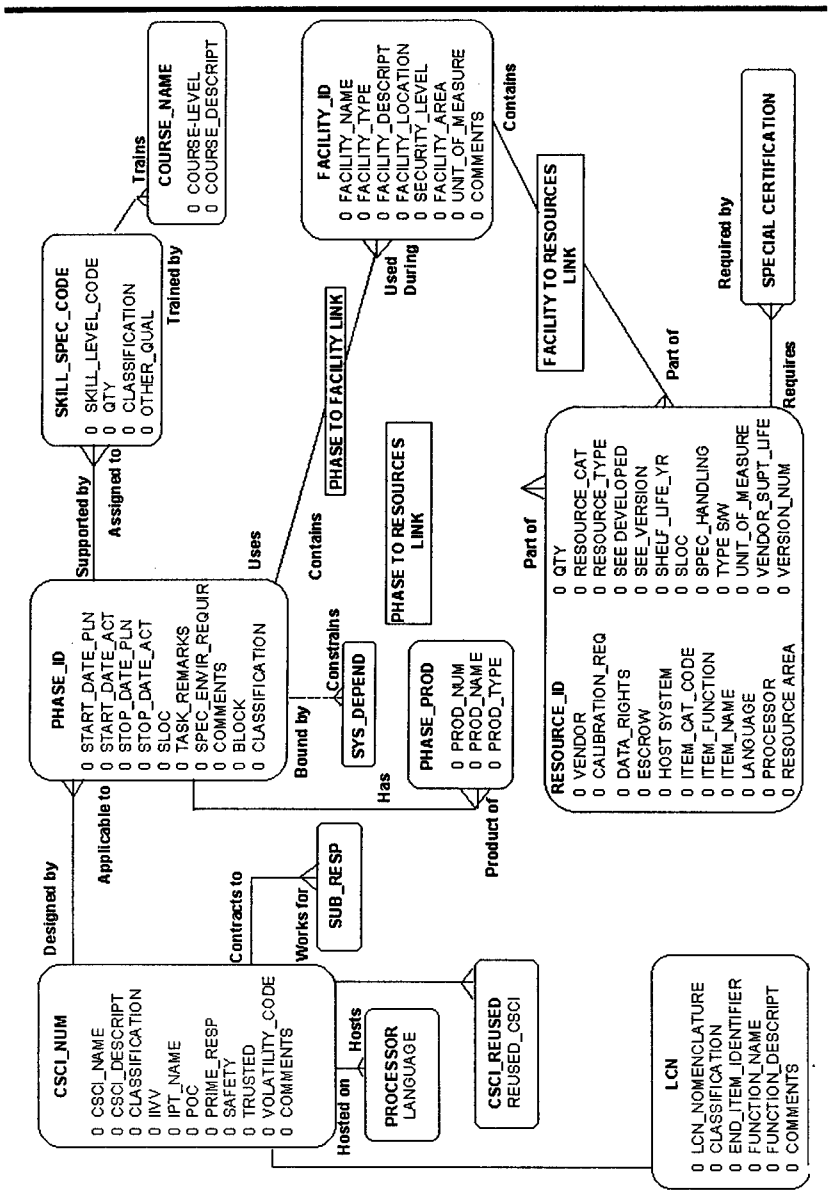


Figure K-1 LCSS Analysis Database Entity Relationship Diagram

TAB 3

Computer Resources Integrated Support Document (CRISD) — Foundation to Post-Deployment Software Support(PDSS)

Judy Fleming
F-22 Life Cycle Software Support IPT
Lockheed Fort Worth Company
Phil Mastrolia
F-22 Life Cycle Software Support IPT
ASC/YFSL, SM-ALC/YFLBA

Presented at the Sixth Annual Software Technology Conference
Salt Lake City, Utah, April 15, 1994

Looking at software history from a life cycle perspective, 30% of its total cost is attributable to software development. The remaining 70% of the software life cycle cost is used for its maintenance. Generally, maintenance costs are estimated by multiplying the development cost by 2.3. This premise leads one to believe that software maintenance is always addressed early on in the life cycle. From a historical perspective, early detailed analysis and planning in support of post-deployment software support (PDSS) have not been embraced as a part of software development. The number of software archaeologists and re-engineering activities attest to this. The **Computer Resources Integrated Support Document (CRISD)** is designed to capture all aspects of software support and maintenance requirements and is the one document that can provide early planning information for PDSS. It provides the basis for the software development organization to establish the software support posture for the system under development. Surprisingly enough, the CRISD is usually "*tailored out*" of major contracts or it is tailored down, or pushed so far to the right in the schedule that it adds little benefit to the overall program. There are only a couple of explanations as to why the CRISD is often eliminated or diluted: "*no one really understands what it is and what information it should provide*" or, "*they do understand and realize what a large task it is.*" As long as these misunderstandings persist, transition to post-deployment software support will be riddled with problems. Contractor Logistics Support (CLS) is often the only option available to the receiving agency when the CRISD has been tailored out of a contract.

Let's look at some of the reasons the CRISD is often eliminated from a Demonstration/ Validation (Dem/Val) or Engineering and Manufacturing Development (EMD) contract. DoD-STD-2167A is CSCI driven. This CSCI structure leads to the requirements for a piece of code being well defined and documented but the same type of requirements for a function or system

is segmented into bits and pieces. Unfortunately, the CRISD has often been generated at a CSCI level, too, rather than as a system document. Because of this segmented view, its value to a program is very little known, much less understood. We've heard many, many times that "*it's too early for the CRISD!*" "*This shouldn't be addressed until right before Program Management Responsibility Transfer (PMRT)*" and "*if it's not required for first flight, why are we doing it?*" In times of extreme budget cuts, this is one of the first CDRLs to be deleted. By eliminating the CRISD up front, the program may initially save some money in the short term; however, the negative impact on the life cycle cost can be quite significant. This Scarlet O'Hara syndrome *of "I won't think about that today, I'll think about that tomorrow"* is going to have to stop if we are to successfully minimize the overall cost of software development and support.

UNDERSTANDING THE IMPORTANCE OF THE CRISD

The CRISD is not just another document. It contains an on-going process that is crucial to the planning and management of 70% of the software life cycle activity. The CRISD process provides the basis for realistically determining the software support concept. Among other things, the CRISD captures the software support environment, its functionality, its limitations, and the processes required to build and load the software on the target system. The CRISD documents the dependencies of the support environment and the order in which system integration must be performed. It is the sole management tool that can accurately map the software support requirement over time. Whether your program is large or small, simple or complex, the software life cycle planning and analysis process must be well defined and executed in order to produce and maintain the CRISD. The importance of fully understanding the role the CRISD plays, and just what is involved in developing and maintaining it cannot be overemphasized.

The CRISD is really the tangible result of the Life Cycle Software Support (LCSS) process. As the pyramid illustration shows, the bottom tier, *Early Analysis and Supportability Decisions During Design*, is the cornerstone for encouraging commonality, modularity, reuse, and simplification of the CSCIs/systems. On past programs, the CRISD would be started in the last two years of the development contract. Since maintenance costs constitute the majority of the overall software costs, the F-22 inserted LCSS tasks up front so that the supportability and

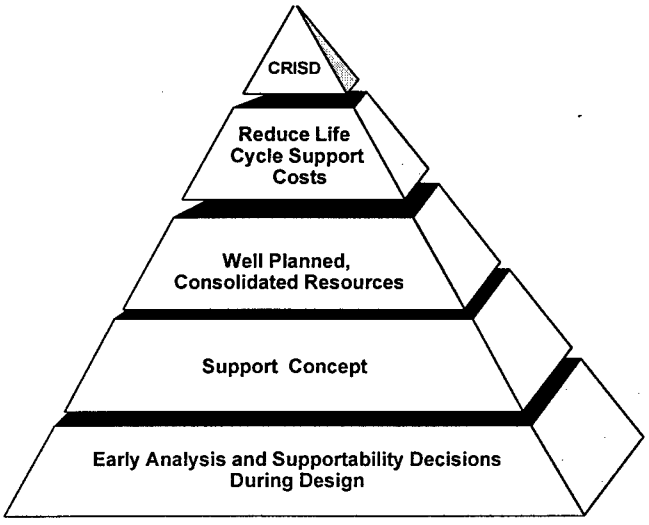


Figure K-2 Understanding the Importance of the CRISD

maintainability features and their processes could be an integral part of the development. This gives us the benefit of identifying all the support resources in each phase as they occur rather than trying to reconstruct the deep past at a critical time. It allows us to specifically identify the tools that are utilized at each phase (and by whom) in order to facilitate the sizing and number of licenses needed to meet organic maintenance schedules. By participation in the software product IPTs, PDRs, CDRs and software product evaluations, supportability issues are addressed up front and are factored into the design.

Early interaction with the developers facilitates the identification of volatile areas in the software. On the F-22 program, the LCSS IPT is conducting an historic change analysis and a predictive change analysis for development of an F-22 change model. This analysis activity will directly effect the determination of CSCIs and systems for organic or contractor software support. The initial support concept is then derived as a result of knowledge gained from performing early predictive analysis and outlining system dependency maps, processes and resources. Once you have a grasp of the anticipated change rate for the CSCIs, their prime functions, dependencies, and required support resources, decision support criteria can be applied to disposition the organic/contractor mix of software support.

Support resources can then be allocated in a very efficient manner. Based on the identified organic CSCIs/systems and the anticipated block change rates, consolidation of the support resources can occur. Subjective guessing is then replaced with well planned consolidated resources. The application of DoD-STD-2167A, MIL-STD-1815A, and a common System/ Software Engineering Environment all greatly contribute to reducing the overall life cycle cost. These potential savings can be fully realized, and even extended, by providing a consolidated and well structured vehicle for PDSS planning and management. Each of these efforts is built upon the previous activity and is fully documented in the CRISD. In reality, the CRISD is a living document that must reflect the configuration of the products, the development environment and test/integration environment. In this manner, the CRISD builds the foundation for post-deployment software support and is an essential key in reducing the life cycle software support costs.

F-22 APPROACH TO THE CRISD

To our knowledge, the F-22 Weapon System is the first major contract to have a CRISD that is fully compliant with DoD-STD-2167A and its attendant Data Item Description (DI-MCCR-80024A). One of the first stumbling blocks we had to overcome is that DoD-STD-2167A is CSCI driven. In order for the CRISD to fulfill its roles and purposes, we had to structure it in such a way as to provide overall support requirements for the Air Vehicle, Support System and the Training System. Since the F-22 CRISD is being developed at the Weapon System level, it will coherently document the software support requirements for the CSCIs, functions, subsystems and systems. Considering that there are currently over 100 CSCIs for the Air Vehicle alone, this is a very research intensive task. The following provides a view of the steps we took:

In order for the CRISD to be a useful document, we first defined all the known users and uses of the CRISD at that point in time. We then created a detailed process flow for development of the CRISD. This process flow allowed us to identify all the information sources and data items we needed, the analyses to be performed and a critical path for the evolutionary development of the CRISD data to support the different users. We determined early on that the only way we could systematically gather and analyze all the needed information was to utilize a relational database. With this approach, a consolidation of support resources can be realistically determined and sized to meet post-deployment needs. On smaller projects, the complete development and test environment could just be duplicated; however, with three major primes and numerous subcontractors, the state of Rhode Island might not be big enough for the Integrated Weapon System Support Facility (IWSSF) if we took that approach. Because development of the F-22 Weapon System software is occurring at many different sites, a consolidation of software support requirements will be performed to define the needs for each phase initially by functional requirement and then by specific items. This analysis allows us to eliminate any redundancies in functionality and define an efficient support suite for the IWSSF.

- Defined Purpose Of Each Issue And The Required Content Level of Detail
- Defined User Roles And Functions
- Developed Detailed Process Flow Resulting In Definition Of Analyses To Be Performed
- Identifed Factors Affecting The Content Of The CRISD
- Identified All Data Items (Over Time) To Be Gathered For Analysis
- Developed Analysis Data Base
- Developed CRISD Template To Be Used By All Team Members
- Tailored DI-MCCR-80024A and Evaluation Criteria

Figure K-3 F-22 Approach to the CRISD

Another important activity was the detailing of the CRISD outline and the building of the CRISD template. DI-MCCR-80024A is largely misunderstood. Although the DI does give some guidance, it can be interpreted in a myriad of ways. Once you understand the roles of the CRISD, interpretation of the DI is a much simpler task. The level of detail is still up for interpretation, so it is important that the contractor work very closely with their customer to ensure consistent expectation levels. To further this process, SM-ALC in conjunction with Draper Laboratory developed clear evaluation criteria for each paragraph in the DI. We then added the evaluation criteria to the CRISD template to facilitate the authors in the creation of the document. This was a definite proactive step which provides built-in quality. A copy of the CRISD template can be obtained from ASC/YFSL, Wright-Patterson AFB, OH 45433.

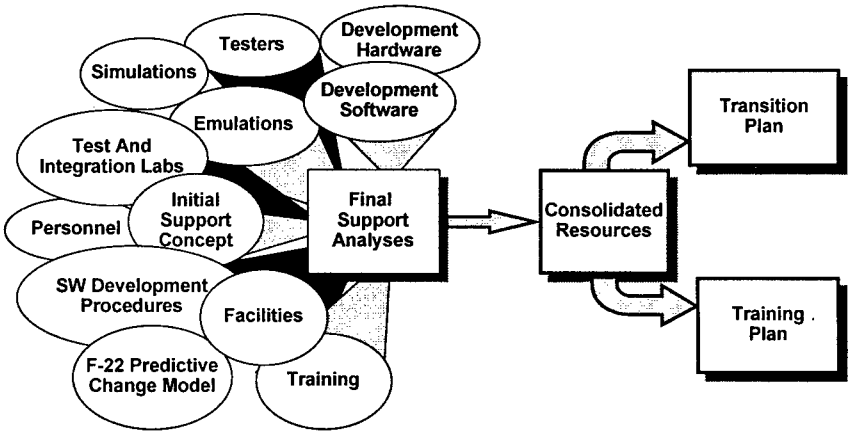


Figure K-4 Source Data and Factors Involved in CRISD Development

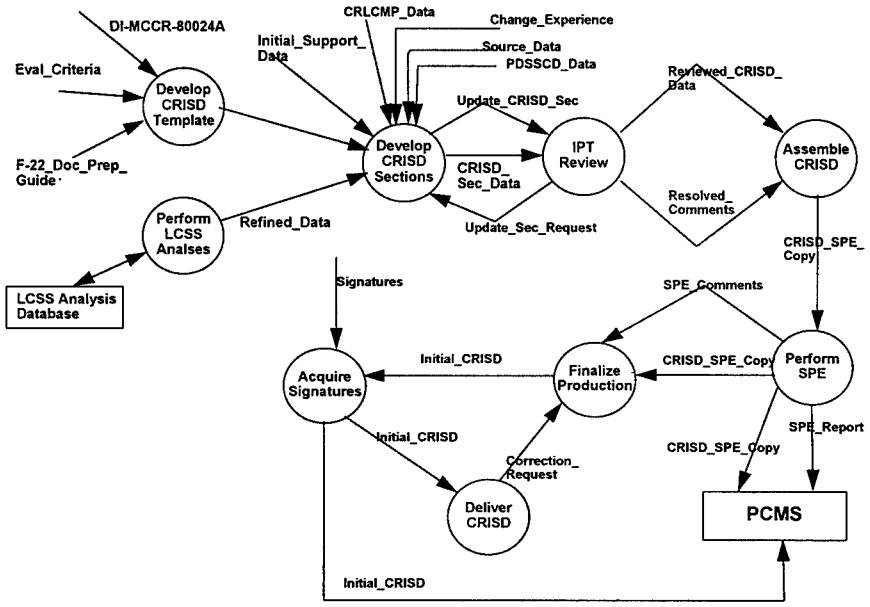


Figure K-6 CRISD: Foundation to PDSS

UNDERSTANDING THE ROLES OF THE CRISD

The LCSS IPT worked directly with SM-ALC to detail the uses, structure and information needed to support not only life cycle planning but to continually support post-deployment software maintenance. Whether the Weapon System is 100% organic or a mix of organic and CLS, the software manager must have the information detailed in the CRISD in order to effectively manage the enormous task. These are the specific roles we defined for the F-22 CRISD:

- **Planning**. The CRISD is the main source for planning PDSS. It will detail the support concept, all the support requirements (hardware, software, emulations, simulations, special test tools, number and skill code/level or personnel, facilities, security requirements, required procedures, and documentation), the transition plan, and the personnel training plan. This information is essential for the program objective memorandum (POM) cycle to ensure that the facilities, budget, equipment, personnel and training will be available in time for transition to the receiving agency.
- **Management**. Methodologies and procedures are contained in the CRISD to provide the support agency with a minimum of management activities that must be performed in order to successfully maintain configuration management of the software and efficiently manage the activities whether organic or contracted. (There is often a misunderstanding that only those items that will be organic need be included in the CRISD. **THIS IS AN ERRONEOUS ASSUMPTION** totally unsupported by the DID!) In order for the supporting agency to effectively manage contracted support, they must have all the necessary support information and a good understanding of the software/system to be maintained.
- **Catalog of resources for the maintainer.** When a maintainer is assigned a specific CSCI or system, the CRISD must provide an index to all the necessary information the maintainer needs to accomplish his or her tasks. The CRISD provides an overview of how the CSCI or system fits into the overall architecture and provides a listing of all the software documentation

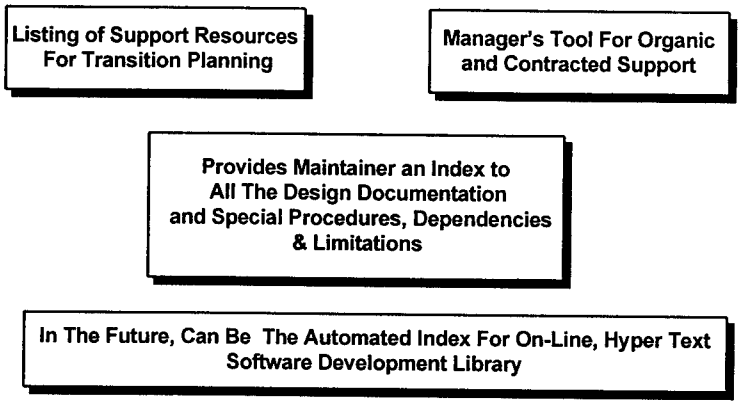


Figure K-5 Get the Purpose(s) Firmly Understood and Agreed to Between Contractor and Customer

and designs as well as a listing of the specific tools and procedures that will be required for the job.

- **Special procedures.** Because of the integrated nature of the F-22, the only place certain important procedures will be found is in the CRISD. These procedures will provide essential guidelines for specific dependencies and processes that must be followed in order to update an OFP. The Appendixes of the CRISD will provide identification of specific activities that are required for each system.

As you can see, the CRISD is not just another document that is delivered with the system at the end of the development phase of a program. It is a living document that should be maintained to reflect the configuration of the systems and environments it is supporting.

CONCLUSION

Understanding the information that the CRISD contains and the diverse set of organizations that will use it throughout the system life cycle will raise its importance among system developers and managers. Even though CRISD development is no easy task (if it is, it probably is not being done correctly), its value becomes apparent when PDSS concerns and reducing life cycle costs are the top priorities of those who manage the total budget. Looking at a system from a life cycle perspective and considering the CRISD as a process for a living document can provide the system manager with the software support options and rationale to facilitate the lowest maintenance and support costs for PDSS.

TAB 4

Software Quality Assurance Impact on Post-Deployment Software Support Cost

Joseph J. Stanko, Capt, USAF
F-22 Life Cycle Software Support IPT
ASC/YFSL, SM-ALC/YFLBA

Presented at the Sixth Annual Software Technology Conference
Salt Lake City, Utah, April 15, 1994

ABSTRACT

Post-Deployment Software Support (PDSS) continues to be the most expensive period of the software life-cycle. Steps can and must be taken during the Engineering and Manufacturing Development (EMD) phase of system acquisition to reduce PDSS costs. One effective step is establishment of and adherence to rigorous software development processes. Such processes are usually found in the program Software Development Plan (SDP); however, SDPs are not necessarily followed during software development. To ensure compliance to such rigorous processes, the government and contractor must accept and implement the SDP. Integrating the Software Quality Program Plan (SQPP) into the SDP provides Software Quality Assurance (SQA) both a process compliance evaluation charter and the criteria for such an evaluation. Indeed, if SQA evaluates and reports contractor process compliance (and noncompliance), there will be a high probability of supportable code and reduced PDSS cost. This paper addresses both the need to have SQA involved from the beginning of the software development effort, and the results of such an effort on the F-22.

INTRODUCTION

Post-Deployment Software Support

Post-Deployment Software Support (PDSS) is the software support activities and functions that take place after initial deployment of a weapon system. PDSS occurs during the Operation and Support (O&S) phase of the acquisition life-cycle; however, all previous ...isition life-cycle phases lay the foundation for PDSS. The most crucial of these foundation-...ing phases is Engineering and Manufacturing Design (EMD). Several major events happen during the EMD phase: the most promising design solution is translated into a stable, producible, ...t effective system design; manufacturing processes are validated; and system capabilities ... ...aluated against both user and contract specification requirements.[3] Since weapon system design stabilizes during EMD, this is the most cost effective time to influence the design for supportability.

The two major factors that influence PDSS costs are changing user requirements and latent errors (from "*bugs*"). As long as there are users, there will be changing user requirements. Developing quality software products and effective PDSS processes and environments during EMD can minimize the life-cycle cost of future changes in requirements. For reducing the possibility of latent errors, the most cost effective method is error prevention efforts during the software design phase.[2] Software design typically occurs during EMD, although the previous Demonstration/ Validation (Dem/Val) phase could also require software development. A rigorous software quality program during EMD would lessen the impact of both changing user requirements and latent errors on PDSS and weapon system O&S costs.

The F-22 Weapon System

The F-22 Weapon System is an advanced, complex weapon system currently in the EMD phase of the acquisition life-cycle. The F-22 Weapon System has three major parts: the air vehicle system, the support system, and the training system. The air vehicle system alone contains approximately 1.6 million source lines-of-code, while the support and training systems are estimated to contain much more source lines-of-code than the air vehicle. The F-22 System Program Office (SPO) is using the Integrated Product Team (IPT) approach for all product development. This teaming arrangement integrates the functional aspects from both the government and contractor sides into one cohesive unit, focusing on developing a product and adding value to the F-22 program. The SPO, Defense Plant Representative Office (DPRO), user, and contractor team members are making and effort now, during EMD, to ensure delivery of quality software products and reduce the PDSS and overall life-cycle costs of the F-22 Weapon System.

Software Quality Assurance

Software quality is "*the ability of a software product to satisfy its specified requirements.*"[5] Therefore, software quality must be an important consideration during software and systems development. Both the Department of Defense (DoD) and industry recognize the need to develop quality software as part of their system acquisition and development process.[1,3,7] There are two end users of every system: the operational user, who will operate the software and is concerned with its operational quality; and the logistics user, who will maintain the software and is concerned with its supportability quality. Both users deserve a quality product and the assurance they will receive one. Quality assurance is "*all those planned and systematic actions necessary to provide adequate confidence that a product or service will satisfy given requirements for quality.*"[8] Therefore, planned and systematic actions form the basis of assuring software quality to the user. The Software Quality Program Plan (SQPP) describes these actions, which target three areas: the software development process; deliverable software and its associated documentation (software products); and all non-deliverable software.[5] The contractor Software Quality Assurance (SQA) organization performs these actions from a traditionally independent viewpoint. The following sections discuss the SQA organization and integration within the F-22, the software development process and process compliance, software products and software product evaluations, non-deliverable software that really is deliverable later in the acquisition life-cycle, and the impact SQA has on PDSS.

SQA Organization and Integration within the F-22

The quality assurance organizations for each of the F-22 contractors are matrixed, with reporting structures independent of the F-22 program. Thus, SQA is an independent organization; however, SQA is also an integral member of each integrated product team (IPT). Independence provides SQA the necessary resources, responsibility, authority, and organizational freedom to perform objective evaluations during software development, and allows SQA to initiate and verify corrective actions as needed.[10] This is a traditional SQA role, and is typically the basis for the relationship with the software developers. Unfortunately, such a basis usually fosters attitudes of mistrust and anger in both software developers and SQA personnel.[2,6] To

overcome this, SQA personnel on the F-22 program are included as integral members of the IPTs. This encourages positive working relationships, and adds value to the software development effort. An IPT is a heterogeneous collection of personnel necessary to develop a software product (see Figure K-7). Grouping the various software disciplines promotes effective communication and removes/reduces barriers between functional identities. The SDP requires this integration, and states SQA personnel shall be active members of their IPT and *co-located with their IPT.*[10] Other programs have successfully implemented this concept.[6] Such an arrangement puts the software developers and quality personnel together from the start, fostering a proactive and cooperative approach to designing quality in to the software products while they are being developed, instead of checking the quality at the end of the development effort. It also permits the other IPT members to participate in quality evaluations of their products during development.

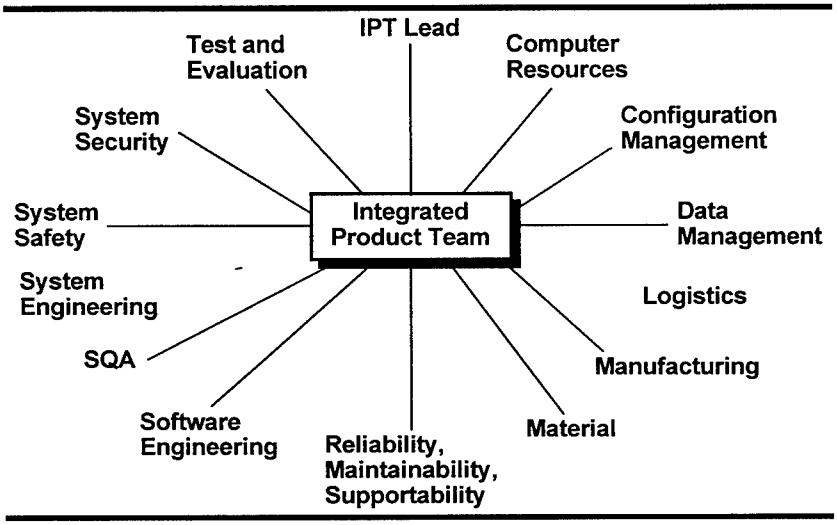


Figure K-7 Integrated Product Team Membership

SOFTWARE DEVELOPMENT PROCESS AND PROCESS COMPLIANCE

Software Development Process

The software development process is the heart of a mature software development organization. If the developing organization does not have a stable process implemented on a project, the cost and schedule risk will greatly increase.[7] Given the trend of reduced funding for DoD projects, software development organizations must have a mature software development process and follow that process in order to reduce cost and schedule risk during software development. One place to describe the software development process is the software development plan (SDP). The F-22 SPO incorporated a team-wide software development process into the Weapon System (WS) SDP. Product-level SDPs provide further clarification and adaptation for their specific software product. The product-level SDPs must include the content of the WS SDP, and may contain additional guidance needed for a particular software product. In some cases, the product-level SDP may tailor out parts of the WS SDP; however, the SPO must agree to this tailoring. The statement of work (SOW) calls out the SDPs, and there is no additional contractual language requiring a specific version of any SDP. This allows both WS and product-level SDPs

to undergo continuous process improvement throughout the EMD phase of the acquisition process, with all changes being implemented (as much as practical) during the phases of software development. Since the F-22 Program uses the IPT approach for product development, the F-22 team must agree to the new revision of the WS SDP before following it. After reaching this agreement, the IPTs must update their product-level SDP to acknowledge the changes to the team's plan.

Determination of Processes to Evaluate

While most programs have an SDP, not all programs follow the processes it describes. The F-22 SPO addressed this issue through the SQPP. Instead of developing a separate SQPP, the WS SDP includes the intent of each SQPP paragraph. Placing the SQPP and SDP in one singular document provides an SQA charter to assure process compliance with both the SDP and contract requirements. This integration also provides the processes and criteria for process compliance evaluations.

Software development processes consist of subprocesses, which in turn contain process elements. The F-22 Weapon System has 27 software development processes, 85 subprocesses, and 237 process elements for process compliance evaluation during EMD. The processes fit into either the phase-dependent or phase-independent categories (see Table K-1). SQA evaluates those processes in the phase-dependent category only during that software development phase. The phase-independent processes must be evaluated during all software development phases. This provides SQA personnel a way to select and evaluate software development processes in a systematic manner, covering all processes required during any specific software development phase.

PHASE-DEPENDENT PROCESSES	PHASE-INDEPENDENT PROCESSES
Acceptance inspection and delivery	IPT management functions and technical controls
Coding standards	Non-CSCI software
Computing resource control program	Non-developmental software
Configuration control	Risk management
Configuration status accounting	Schedules and milestones
Continuous process improvement	Software development library
Corrective action process	Software development process
Design standards	Software engineering environment
Documentation/media distribution	Software product evaluations
Formal qualification testing	
Formal reviews	
Formal test and evaluation	
Reusable software	
Software development files	
Software safety — process requirements	
Subcontractor product management	
System-level requirements and specification development	
Verification and validation	

Table K-1 Software Processes to Evaluate[10]

Planning for process compliance evaluations begins prior to the subject development phase, with the SQA person scheduling process evaluations throughout the phase. During the software development phase, the SQA person conducts process compliance evaluations using methods such as monitoring, sampling, flowcharting, identifying past trends, and identifying opportunities for continuous process improvement.[11] The results of each evaluation show whether or not the IPT is following the software development processes. If the IPT is following the processes, it is compliant, and the SQA person continues with the regular planned evaluations. If the IPT is not following the processes, it is non-compliant, and the SQA person will re-evaluate

the non-compliant process at regular intervals (e.g., every month) until the IPT is found compliant. Non-compliances are categorized as either major *("an error in implementation of process requirements that would have an impact on cost, schedule, or performance"*) or minor *("an error in implementation of process requirements that would have no impact on cost, schedule, or performance; but could affect the effectiveness or efficiency of the process").*[11] The IPT must have a formal resolution plan for all major process non-compliances. While not required to have a similar plan for minor process non-compliances, the IPT must take some action to correct the deficiency.

SQA personnel report the results of process compliance evaluations in a chart format similar to the prototype one shown in Figure K-8. The process compliance chart tells two stories: how well the IPT is complying with software development processes; and how many evaluations the SQA person has performed. Percent compliance (the top half of the chart) shows the percentage of total *planned* process evaluations where the IPT is compliant. The expected growth lines (the dotted lines) show different percentage levels of compliance (50%, 75%, and 100%). If an IPT is 100% compliant with processes evaluated to date, their actual line will be less than 100% on the scale; however, it will match the 100% expected growth line, indicating progress. Also, the actual line will never reach 100% on the scale if SQA evaluates less than 100% of the processes. In other words, the expected growth line shows 100% of the processes evaluated to date and not 100% of all processes planned for evaluation during the software development phase. The only place where the expected growth line shows 100% of all processes evaluated and compliant is at the very end of the line, where the line reaches 100% on the scale. Similarly, the 75% and 50% expected growth lines represent that percentage of process compliance *to date*, and will eventually reach the corresponding scale on the right side of the chart. The 75% and 50% expected growth lines are flags. If an IPT's compliance level drops below one of these, the SPO and contractor IPT leads will be alerted and can assess the situation and determine the appropriate level of assistance to give the IPT.

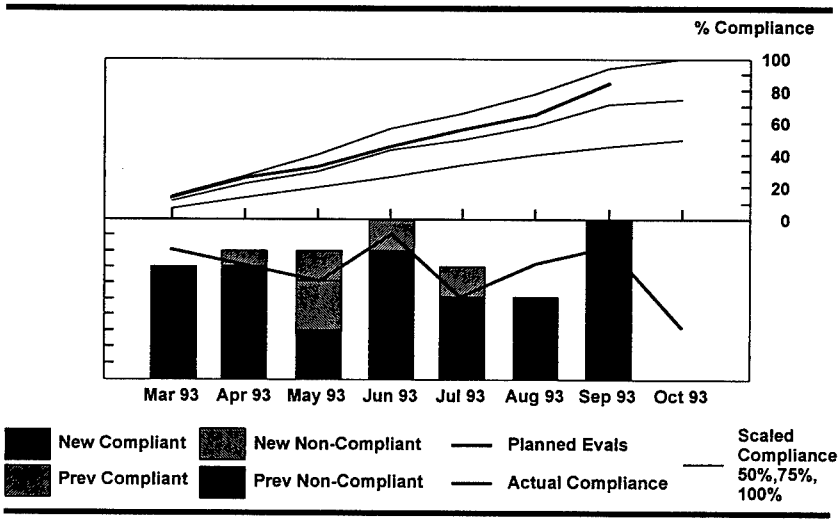


Figure K-8 Prototype of Software Development Process Compliance Metric

The chart's lower half provides another indication of IPT performance. While the chart's top half shows overall process compliance trends, the bottom half describes process trends in four categories: newly evaluated processes found compliant; newly evaluated processes found non-compliant; previously evaluated processes found compliant; and previously evaluated

processes found noncompliant.11 Trend analysis of recurring non-compliances could pinpoint process problems the IPT must solve to ensure a quality product. Trend analysis could also indicate problems between the IPT and external functions (such as documentation management) where both the IPT and external functional processes must adjust and work together. The chart's lower half also provides insight into SQA effectiveness. The dotted plan line shows how many process evaluations the SQA person intends to do each month. The histogram bars show the actual evaluations performed. Previous non-compliant processes will create additional unplanned workload to re-evaluate, potentially making the histogram higher than the plan line. Places where the histogram is lower than the plan line could indicate problem areas with SQA coverage (e.g., not enough SQA resources to evaluate the processes).

SOFTWARE PRODUCTS AND SOFTWARE PRODUCT EVALUATIONS

Software Products

Typically, software products are the deliverable software and documentation of that software.[4] Automation efforts usually center on the software code itself, ensuring code development in a configuration managed environment.[7] With the advent of larger DoD systems, software documentation needs the same level of automation. The purpose of software product automation should be to capture the systems analysis and design rationale during software development. This shifts traditional software product evaluations from late in the EMD phase to a proactive evaluation during their development. For computer software configuration items (CSCIs), this involves reviewing analysis and design products created during the software development process. DoD-STD-2168 identifies additional areas to evaluate: software development processes; all software development libraries; any non-developmental software; all non-deliverable software; software engineering and test environments (the deliverable parts); subcontractor management; and software acceptance inspections and preparations for delivery.[5] Integrating the traditional software product definition with these additional areas provides a modified list of software products to evaluate (see Table K-2).

Software Development Plan(s)
Software Development Process Compliance Status
Electronic Databases/Model Data
Software Development Folders
Source Code
Software Documentation

Table K-2 Software Products to Evaluate

The SDP is the key product from a quality standpoint. Without a good plan, it will be impossible to develop good software code and documentation. Next, the team must follow defined processes. This is why Software Development Process Compliance Status is next in order of importance. A quarterly status of how all IPTs are doing is an important indication of the software development effort.

After establishing the processes (SDP) and evaluating compliance (process compliance status), the focus should turn to actual development data. The electronic databases and model data are the core from which all other software products result. IPTs generate the electronic databases and model data through the use of automated tools during software development. The F-22 has several automated tools integrated into the common System/Software Engineering Environment (S/SEE). One such tool is Teamwork,[8] a software analysis and design tool that

generates both Ada diagrams and data dictionaries. Another important tool is the Interface Definition Tool (IDT). IDT provides a standard format for defining software-to-software and software-to-hardware interfaces. The S/SEE tools provide information for development of source code, and also form the basis for developing software documentation. For example, the IDT database provides data for the Interface Design Document (IDD), maintaining some consistency between the software documentation and the actual state of the system.

Software development folders (SDFs) are the next product to evaluate. SDFs contain the engineering notes made during development of each software module. Traditionally, these notes exist as hard copies in binders or folders; however, for most of its software development the F-22 program is using an electronic form of the SDF on the S/SEE. The last two software products, source code and documentation, will result from all the previous products. If there is no established plan, if the development teams are not following the proper processes, if the developmental data is not properly created, and if the design decisions are not adequately captured, the source code and documentation will not provide the necessary information to perform PDSS.

Software Product Evaluations

Each software product requires a specific evaluation to ensure its quality. The following sections discuss software product evaluations (SPEs) for the software products identified previously.

Software Development Plan

The standard SPE for a SDP is a document review by peers. While this will suffice for some onetime documents, the software development processes should always be as current as possible during a multi-year system acquisition. This requires a continuous process improvement (CPI) effort. The CPI effort should allow for either incorporation or rejection of write-ups against the SDP. The F-22 program uses CPI process action teams (PATs) to address individual processes identified as deficient. The results of each PAT are rolled into the next revision of the SDP, which then undergoes a SPE before release. This iterative loop provides a set of quality processes and fine-tunes them as EMD progresses.

Software Development Process Compliance Status

The easiest way to evaluate software development process compliance is from a process compliance metric, which tells how both the IPT and SQA are doing. Evaluation questions for the process compliance metric include what is the severity of the non-compliances, what is being done to resolve non-compliances, and why is the actual number of process evaluations less than the planned number. Anyone reviewing the process compliance status can, and should, ask these where applicable. For example, trends might indicate the IPT is not complying to agreed-upon software development processes. From this, SPO and contractor IPT leads must find out the reason and take appropriate action to help the IPT. Likewise, if SQA personnel are not able to complete their planned evaluations, some investigation could reveal areas that need to be addressed. Finally, if the IPT is meeting the expected line and the SQA person is meeting the planned line, both would deserve positive recognition.

Electronic Databases/Model Data

Evaluation of the electronic databases and model data will be more challenging than evaluating the SDP and process compliance status. The F-22 program databases and model data reside on the S/SEE, and require S/SEE access to evaluate. One possible method of evaluation is manually searching all the fields of selected databases to ensure information is there and usable. Another approach is to use the host tool (or create a utility program) to verify all fields contain information; however, this would still require manual verification of the quality of that information. Currently (as of the writing of this paper), no definitive questions or criteria exist for evaluating the database and model data. Since this data captures both analysis and design information, this is the next area being defined.

APPENDIX K Software Support

Software Development Folders

The F-22 program is using an electronic form of the SDF on the S/SEE for most of the software development. Therefore, it is important to evaluate the electronic information stored in the SDF (both text files and data). SQA and SPO personnel have conducted initial audits of both electronic and paper versions of SDFs; however, there is no single event scheduled for evaluation of an entire SDF. The F-22 approach is to evaluate an SDF incrementally while it is being assembled. Such incremental audits should occur during the detailed design, code, computer software unit (CSU) test, and computer software component (CSC) test phases. If audits are not done until the end of development, the SDFs will be too large to effectively evaluate.

Source Code

The F-22 program uses walkthroughs to evaluate source code during the coding phase, focusing on the source code's technical quality.[10] Walkthroughs are also recommended for the requirements, preliminary design, and detailed design phases. This is to ensure the software is being developed to meet the correct requirements. The IPT will present walkthrough results at the milestone events, with successful completion of a walkthrough being necessary for the source code to progress to the next level of integration and test. One supplement to a walkthrough is the use of automated tools. Automated tools are useful for both test coverage and quality and style checks prior to module integration. The F-22 program is working in this area to define quality and style thresholds for the source code that are automatable.

Software Documentation

Traditionally, two review groups perform software document evaluation: the systems engineers doing the development work (both SPO and contractor); and the operational test and evaluation (OT&E) test team. One disadvantage of this two group approach is the familiarity the engineers have with the system they are developing. Development engineers do not always communicate what is important in a PDSS environment. Another problem is the OT&E evaluation time frame, which occurs late in the EMD phase. Problems found this late in development are too costly to fix, or can not be fixed do to unavailable funds. Therefore, a pro-active effort must address both evaluation of software documentation and the *development* of software documentation.

DoD-STD-2167A identifies data item descriptions (DIDs), which then identify what information should go into a given document. Development engineers understand the content (and context) of the system they're writing about; however, the PDSS personnel do not have that level of insight and understanding. Unfortunately, engineering document reviews do not usually take this into account. In contrast, the OT&E personnel are representative of the PDSS personnel, yet cannot have the documents corrected in a cost effective manner. The F-22 SPO Logistics Support Division, located at Sacramento Air Logistics Center (SM-ALC) decided the content of a document (e.g., information on analysis and the design decisions) is more important than the document format (e.g., centering of page numbers, margin widths, white space, etc.). Working with The Charles Stark Draper Laboratory, this team jointly developed software document evaluation guidelines (DEGs) for all 17 DoD-STD-2167A documents. The DEGs focus on the documentation content, and serve as a paper "*knowledge base*," addressing both developmental and support concerns. The DEGs ask specific questions for each document part, specifically in six key areas: is the information there; is it understandable; is it consistent; is it traceable; does it add value to software support; and (to the extent possible) is it correct. If one can develop a document and pass a DEG evaluation, that document will provide the information necessary for PDSS.

The DEGs were originally developed for SM-ALC software supportability engineers to evaluate software documents before their acceptance by the government. Building upon this, the F-22 SQA organization took a more pro-active approach and incorporated all the DEG questions into the SPE effort as evaluation criteria. Now, government IPT members must use the DEG questions as criteria in evaluating documents during a SPE. Taking this concept one step

further, the F-22 program IPTs have incorporated the DEG questions into the original document templates. This fosters creation of acceptable information content during document development before the SPE takes place. Then, the SPE occurs prior to formal delivery to the government. This total effort builds quality into the documents from their inception, instead of inspecting it in later.

NON-DELIVERABLE SOFTWARE

DoD-STD-2167A identifies software products as the deliverable software and documentation of that software.[4] Typically, deliverable software during EMD means only the software the EMD contract requires; however, there is a substantial amount of software developed during EMD that will be deliverable *after* EMD. The production contract identifies this software and its documentation. The software development time frame (and not the software delivery time frame) should drive SQA efforts. Therefore, software that is deliverable at any time during the EMD and production phases, and is developed during EMD, must be treated as deliverable even during the EMD phase. This necessitates the same rigorous SQA efforts for both EMD deliverable and post-EMD deliverable software. The F-22 is taking this approach, and has identified three categories of software: deliverable during EMD; software developed during EMD that is required for post-deployment support of the weapon system; and software that is not required for post-deployment support of the weapon system.[9] All software required by the EMD contract is considered deliverable during EMD. All other software is considered required for post-deployment support of the weapon system, unless the developing IPT requests a reclassification as not necessary for post-deployment support. Thus, SQA will be evaluating the processes and products for all deliverable software, regardless of when it is deliverable. Non-deliverable software also requires SQA assessment; however, this effort is much less than that for the "*deliverable*" software.

IMPACT ON PDSS COST

The F-22 is still early in its EMD phase, and therefore has not begun any formal PDSS activities. Although data is not available on the impact of the new document SPE approach and criteria on PDSS cost, initial indication shows they are having a positive influence on document quality and first time acceptance, both of which are key factors in reducing life cycle costs. The other product evaluation efforts are in development, and when completed are expected to have a similar effect on their respective products. Initial findings from software process compliance evaluations indicate this approach is having a positive influence on product quality. While all these results are preliminary, in general they indicate the pro-active SQA efforts have a positive impact on product quality and will reduce PDSS cost.

REFERENCES

[1] ANSI/IEEE Standard 1074-1991, *IEEE Standard for Developing Software Life Cycle Processes*. New York: The Institute of Electrical and Electronics Engineers, Inc., 1992

[2] Beizer, Boris. *Software System Testing and Quality Assurance*. New York: Van Nostrand Reinhold Company, 1984

[3] Department of Defense. *Defense Acquisition Management Policies and Procedures*. DoD Instruction 5000.2. Department of Defense, 1991

[4] Department of Defense. *Defense System Software Development*. DoD-STD-2167A. Department of Defense, 1988

[5] Department of Defense. *Defense System Software Quality Program*. DoD-STD-2168. Department of Defense, 1988

[6] Feinstein, Steven. "Barnacles on the Software Ship." *IEEE Software*. 1991, 8(4), 92-93

[7] Humphrey, Watts S. *Managing the Software Process*. USA: Addison-Wesley Publishing Company, Inc., 1990

[8] International Standard ISO 9000, *Quality Management and Quality Assurance Standards — Guidelines for Selection and Use*. Switzerland: International Organization for Standardization, 1987

[9] Lockheed Corporation, and The Boeing Company. *F-22 Non-CSCI Software Development Requirements*. Unpublished Work, 1992

[10] Lockheed Corporation, and The Boeing Company. *F-22 Software Development Plan for Engineering and Manufacturing Development*. Unpublished Work, 1993

[11] Lockheed Corporation, and The Boeing Company. *Software Quality Assurance (SQA) Process Evaluations and the Process Evaluations and Compliance Metric*. Unpublished Work, 1993

TAB 5

The DoD Generic Fighter: F-22'sHistorical Foundation

Jon Floyd
Lockheed Fort Worth Company
F-22, LCSS IPT
Phil Gould
Lockheed Fort Worth Company
F-22 LCSS IPT Manager
Phil Mastrolia
SM-ALC, F-22 LCSS IPT
John White
F-22 SPO, F-22 LCSS IPT Manager

Presented at the Seventh Annual Software Technology Conference
Salt Lake City, Utah, April 14, 1995

INTRODUCTION

The "*generic fighter*" referred to throughout this paper is an invention of the F-22 Life Cycle Software Support (LCSS) Integrated Product Team (IPT). This generic fighter is an amalgamation or normalization of four modern front line fighter programs currently in service in the US Navy and the US Air Force — the F-14, F-15, F-16 and the F/A-18. The generic fighter was invented to allow reasonable comparison with the forecast software changes in the F-22 weapon system. The result of this comparison will allow an optimized support structure for F-22 software resulting in reduced life cycle costs. The data analyzed to create the generic fighter was gathered in response to 54 questions generated by the LCSS IPT during the first quarter of 1994.

GENERIC FIGHTER ARCHITECTURE

The generic fighter of the 70s and 80s utilized a federated avionics architecture (see Figure K-9). The Air Force fighters typically used MIL-STD-1750 processors while the Navy pursued the UYK series of standard processors. Bus systems evolved from one HOO-9 to one to five MIL-STD-1553 A or B bus systems. As time passed, more functions were automated and more analog functions were digitized resulting in a growing number of processors. By the late 80s, more functions were being merged in a single processing element as miniaturization occurred.

The generic fighter had four hardware unique baselines, with ever increasing computing resources. Each hardware baseline had multiple software variations. Even more variants existed if the aircraft was sold overseas. A single software baselines, compatible with

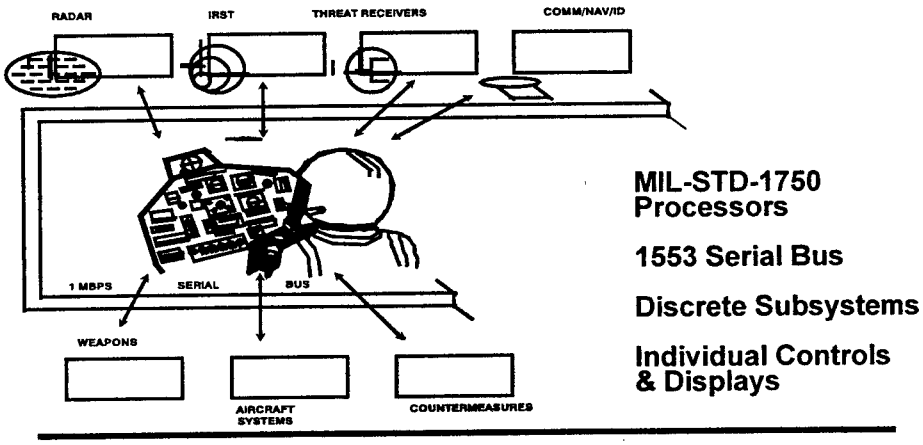


Figure K-9 Federated Systems

all hardware configurations, was always the common goal but rarely achieved. The major impediment was shortage of retrofit funds caused by diminishing value of older aircraft retrofit.

VARIANTS

From the original hardware baseline there were new hardware baselines approximately every five years. New navigation, radar, electronic combat, and weapons were added at each new hardware baseline. Software baselines and releases followed a schedule that often was hardware driven (see Figure K-10). There was an average of three software releases the first year of service after PCA. The second year of service typically had two software releases. Eventually software releases settled to one every 18 to 24 months. As hardware sensitive software baselines matured, the release frequency decreased. Eventually software changes saturated the hardware capability and hardware upgrades were required. For each new hardware baseline, there were two to three corrective software updates released within the first 12 to 18 months.

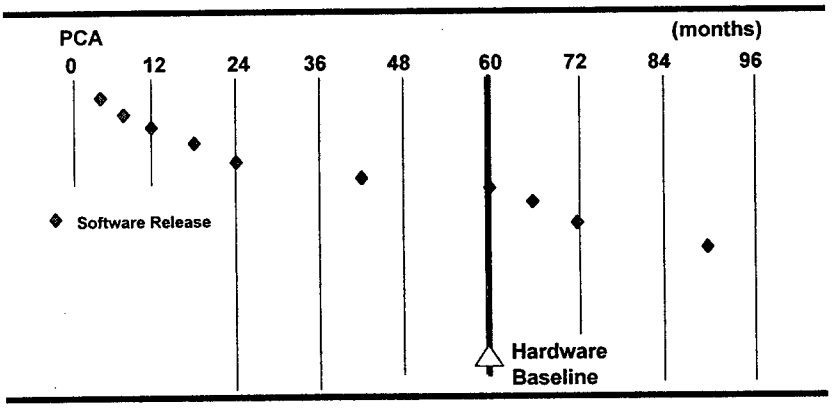


Figure K-10 Frequency of Software Releases

APPENDIX K Software Support

As the generic fighter matured, many variants of the hardware remained in the field. Each of these variants required a supporting software release. In addition, foreign military sales (FMS) typically required a software release of varying limited capabilities with each country requiring separate configuration management. This resulted in multiple versions of the software being maintained concurrently as the example shows in Figure K-11.

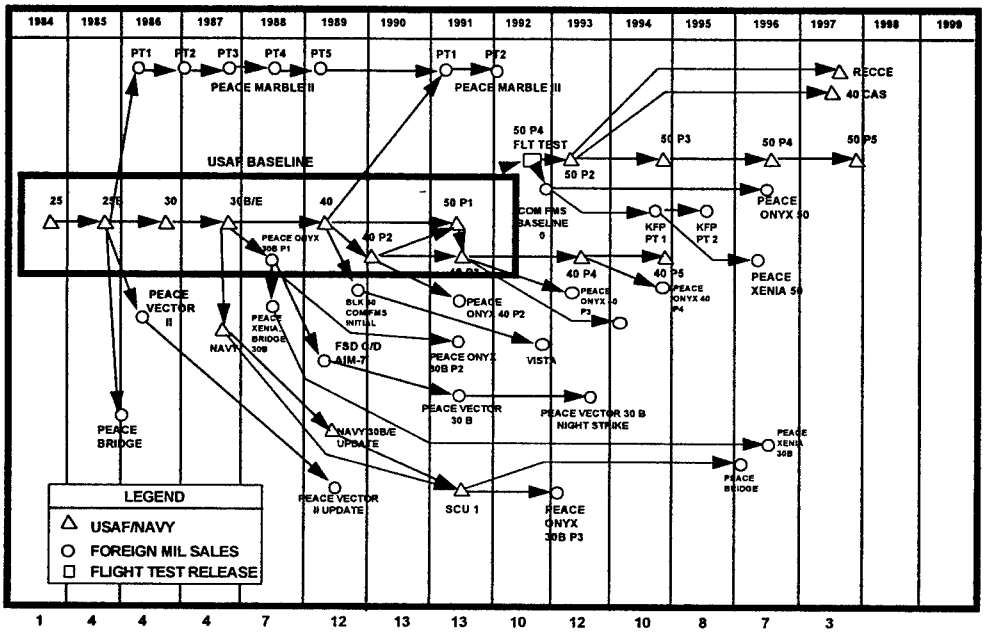


Figure K-11 Software Baseline Proliferation

The basic generic fighter started as an Air to Air fighter with less than 100K words of on-board memory. As more roles were assigned to the generic fighter, more systems were added and on-board memory was forced to expand. By the late 70s, the generic fighter had grown to approximately 300K words of on-board storage but with only 40 to 60% memory utilization. (These numbers are for avionics systems software only.) The mission related functions grew from 3 to 5% per year until the memory was forced to expand. By the end of the 80s, the on-board memory had grown to 2 million words with 70% used. (See Figure K-12.) Also in the 80s, non-avionics software began to appear. Digitized engine controllers, digital flight controls and diagnostic software were installed on the generic fighter.

SOFTWARE CHANGES

After the first software baseline was established, typical avionics software maintenance changes can be categorized using the three basic types of software changes as defined by Swanson. Swanson's categories of software changes include corrective (fixing bugs, no new requirements), adaptive (adapting working software to hardware changes, no new requirements), and enhanceable (modifications/improvements of working software, new requirements). The percentage of change, by type, is illustrated in Figure K-13.

APPENDIX K Software Support

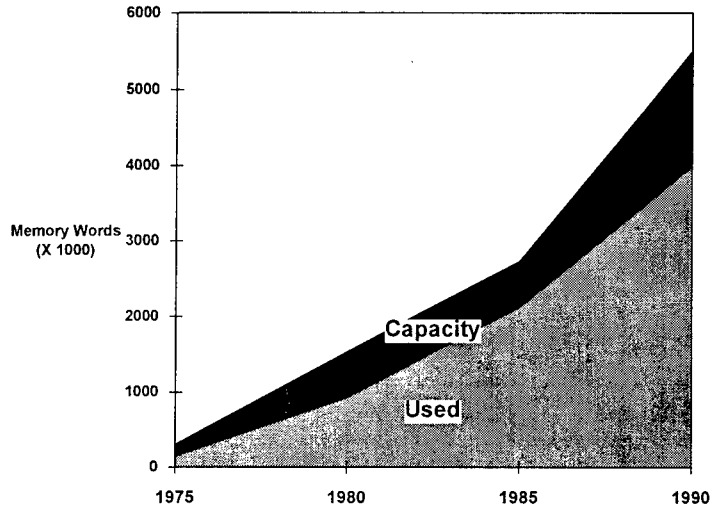


Figure K-12 Memory Growth Over Time

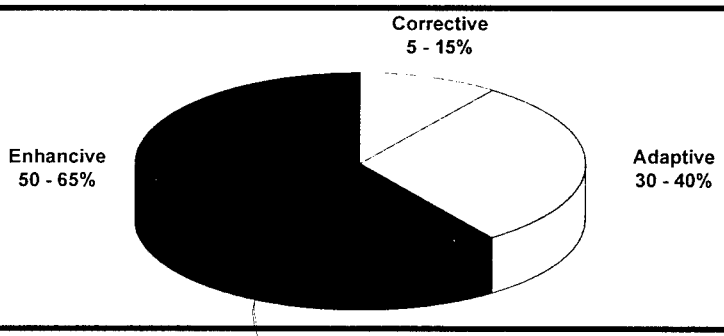


Figure K-13 Types of Changes

For a typical software block change, where no hardware was being changed, approximately 10% of the software was changed. This 10% was broken down as follows: 3-5% new code. 5-7% modified code. Approximately 88-93% of the software remained unchanged. The 10% rate of change's impact on the operational flight program (OFP) software configuration is reflected in Figure K-14 (below). Software changes which were driven by hardware changes, either the addition of new hardware or the upgrade of existing hardware systems, had a much lower percentage of reusable software. Obviously, the amount of reusable code depended on the extent of the hardware modifications; expanding a memory board has a minor software impact compared to changing the processor type.

Often the amount of software that was changed in a block cycle was limited by available funding or by available schedule. The larger and more complex software configuration items were typically most impacted by funding and/or schedule constraints. This is due to the increased integration and test requirements associated with large and complex software configuration items. Not all software changed at each block change. The generic fighter's primary areas of change were mission and human interface related (see Figure K-15 below). These primary areas of change included controls and displays (pilot/vehicle interface improvements accounted for up to 60% of the changes), stores addition, launch envelope improvements, radar

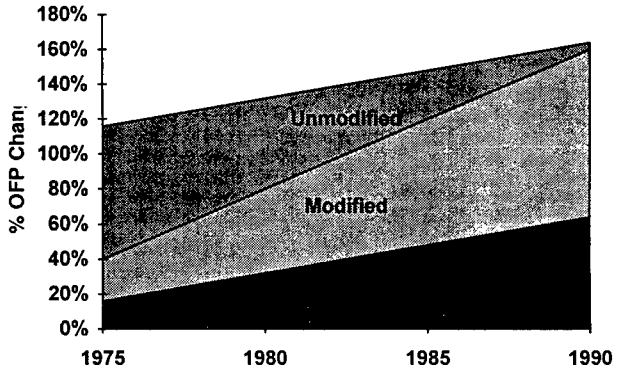


Figure K-14 Impact of Software Change on Operational Flight Program

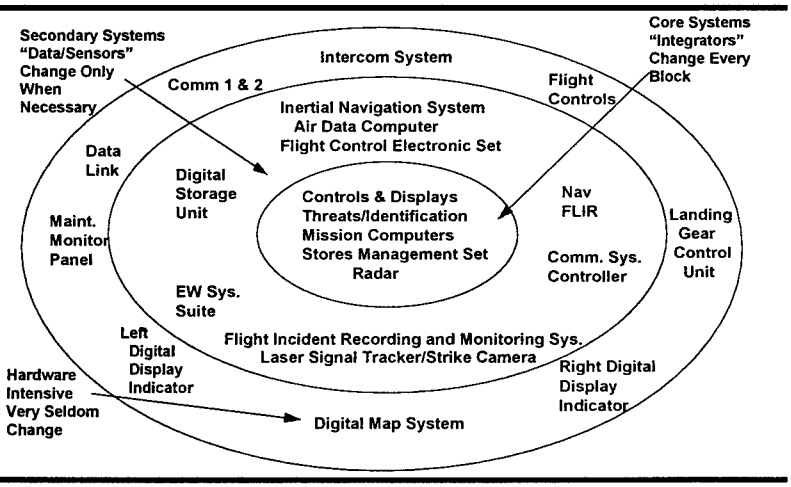


Figure K-15 Subsystems/Functions Affected by Software Releases

improvements, electronic defense systems addition and improvements, and identification capabilities and improvements.

Some systems were very stable and exhibited little or no change to date. Examples of this were the inertial navigation system, air data computers, and digital data storage units. Software which was closely bound to hardware typically changed only when the hardware was upgraded. Examples of this were landing gear control, anti-skid systems-communications, and digital flight controls systems.

SOFTWARE SUPPORT PROCESS

Software changes are categorized by the user as either routine (block change cycle), urgent (six to eight weeks), or emergency (within 72 hours). The goal of the generic fighter was to field a routine software release every 18 months; however, a single block change cycle required 36 to 44 months (see Figure K-16) from requirements definition to fielding. To accomplish this, as many as three or four software baselines were in work concurrently (see Figure K-17).

The block cycle is divided into two phases. The first phase is the requirements analysis phase where the depot team, with the support of the user, develop an understanding of the types of system changes required for the next block change. This phase was typically undocumented and undisciplined. It required from 6 to 10 months for system definition/analysis and 9 months for establishing a contract with a software developer. The contracting timeline in the requirements definition phase was the primary driver in deciding whether the block change process was 36 or 44 months long. The second phase included allocating the changes to software configuration items, design, coding, integration, testing, and fielding the new release. This phase was in the process of maturing during the F-22 round robin with most programs actively pursuing a Software Engineering Institute CMM Level of 2.

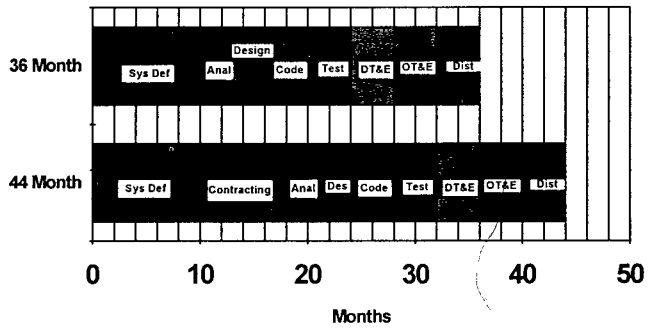


Figure K-16 Block Change Cycle

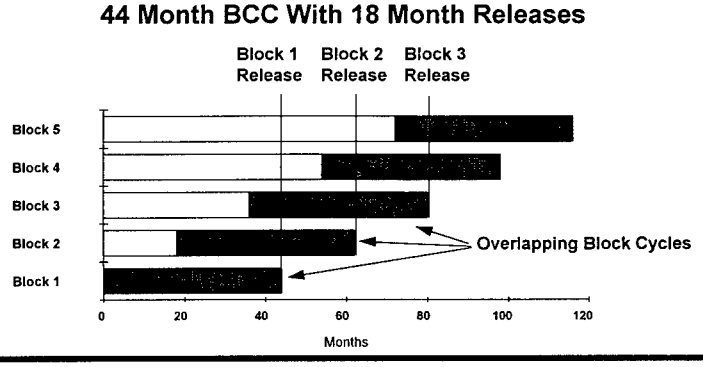


Figure K-17 Block Change Releases

SUPPORT COSTS

The generic fighter block change cycle required 400 to 500 man years and ranged from $22M to $200M with the normalized average of $112M. This difference was caused by the length and size of the effort for the block change. All costs are included except for requirements elicitation and business development. The costs start with requirements identification through distribution of the software to the field. It also includes technical order (TO) and technical manual (TM) updates. The block change costs include systems engineering, software development, lab costs, flight test, TOs, and miscellaneous and are reflected in Figures K-18 and K-19.

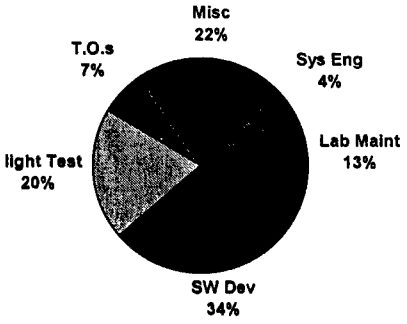


Figure K-18 Block Change Cost Allocation

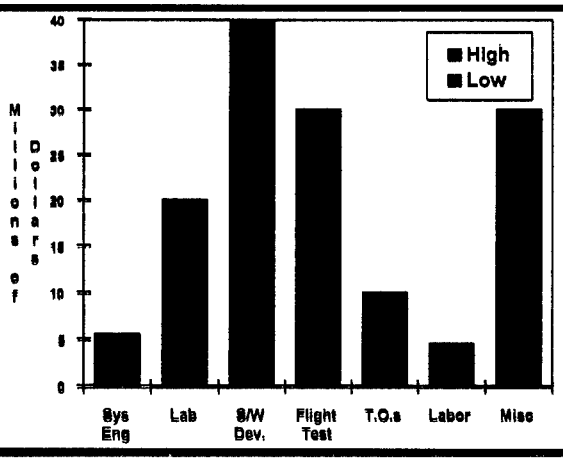


Figure K-19 Block Change Cost Ranges

The software development phase includes software requirements analysis, design, code, and unit test. For new hardware configurations, integration and system level tests took up to 40 percent of the total cost of development effort. For software only updates (without changing hardware) integration costs were less than 15 percent. Flight test costs were dependent upon location and duration. Typical flight test costs ran from $30K to over $50K per hour depending on chase plane and drone requirements. Other costs, included overhead, mechanical replacement, security, stores for test, and communications.

APPENDIX K Software Support

SUPPORT STRATEGY

The developing contractor has continued to provide critical skills at the government and contractor software support facilities. The production baseline of software was "*owned*" by the government at first production aircraft flyaway (FCA/PCA). Organic support for the first production baseline was not complete phased in until 5 to 8 years later. The initial program management directive (PMD) typically dictated a total organic effort for software support very early in the program. Over time it became apparent that this was technically impractical and undesirable from a program view. The contractor was continually adding capability, correcting errors, and optimizing the pilot vehicle interface. As a new avionics hardware baseline appeared, software support for the older baseline was transitioned to the government location.

Ratios of manpower mixes of 40%/60% to 60%/40% (organic/contractor) were common over the life of the program. The typical work split between organic and contractor resources by skill area is reflected in Figure K-20. This figure reflects the approximate mix at the ten year point of the generic fighter. Requirements analysis was done at the contractor's facilities on simulators, reproduction and distribution is organic but was outsourced to subcontractors. Technical Orders/Technical Manuals were done at the Contractors facility.

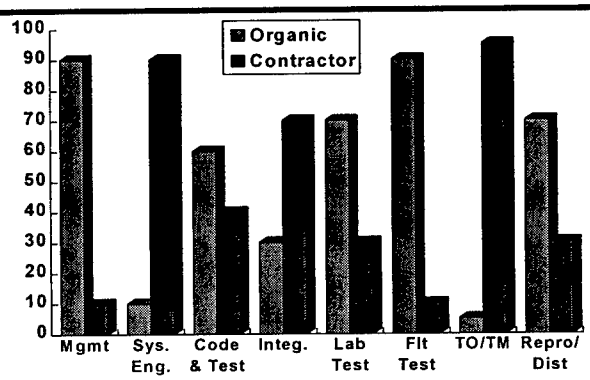


Figure K-20 Organic/Contractor Personnel Split (%)

Labor rate costs were as typically follows: $110/hr for contractor personnel costs in a contractor-owned contractor-operated (COCO) facility; $65/hr for contractor personnel costs in government-owned contractor- operated (GOCO) facility; and $55 - 65/hr for organic personnel costs in a government-owned government-operated (GOGO) facility. The lower rates for labor performed in a government facility did not include total burdening.

SUPPORT ENVIRONMENT

The generic fighter had few software development tools at the beginning of the 70s. In the OEM aerospace companies, an in-house software development tool making capability was developed. These "*homegrown*" tools were developed to fill in the blank spots between the available tools of the time. This was because few off the shelf software development tools existed during the early to mid 70s. In the labs little or no allowance was made for either growth or multiple configuration support. Non-reconfigurable lab equipment was a problem in the early period. Systems level testing was done without the aid of extensive simulation/stimulation and much of systems testing was done during extensive and costly flight tests.

By the end of the 80s (see Figure K-21), non-integrated computer-aided software engineering environments and multisystem labs with some simulation/stimulation capability were in use. "*Homegrown*" software development tools began to fall from favor because of the availability of COTS tools of higher quality, more functionality, and at lower costs. The short life span of these COTS development tools caused continual upgrade for development hardware and software. Simulations improved and began to replace trial and error, test and fix methods. This led to integration environments which were impacted less by each new hardware baseline, with very little impact to the support environment caused by software only changes. Occasional bottlenecks were encountered in the labs due to support of multiple support. Labs often cut holes in walls to test radar systems and hung weapons on the outside of the labs to simulate actual flight conditions. However, extensive flight testing was still required to ensure systems were meeting user requirements.

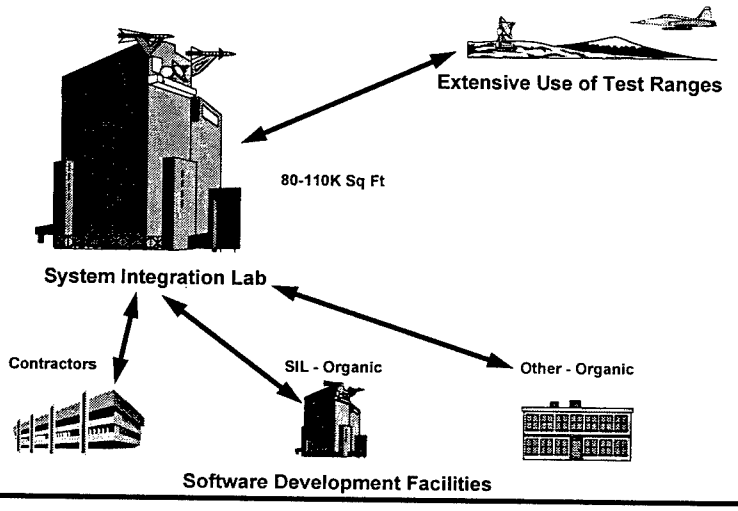


Figure K-21 Support System

TRAINING SYSTEM IMPACTS

Training was separately contracted and, furthermore, was typically not integrated with the rest of Weapon System. Trainer and training systems typically lagged behind the fielded configuration. This was known by the user as "*version skew.*" The problem was that the acquisition strategy for trainers and training was different from, and at a lower priority than, the rest of the weapon system. Even if the trainers were at the same priority and received the OFP at the same time as the user, the training system required re-engineering before it could be used. Air vehicle software was not designed for trainer reuse. A lag of 2 years between new software installed in the fleet and trainer reconfiguration to match was not uncommon.

PROGRAM MANAGEMENT

The generic fighter allowed patches but discouraged use of them. No patches were allowed for flight controls due to safety critical considerations. Flight Controls had the highest reliability but often took longer to develop. (Flight Controls typically had less than 1% error rate). Very few weapon system productivity figures were developed. However, the productivity numbers that were developed, varied from weapon system to weapon system and did not match

IEEE productivity ranges for embedded applications. Mission related software changes had the highest productivity, safety critical software had the lowest.

Very little use of parametric models for project estimation. Developers tended to rely on past experience and rough orders of magnitude. Those projects using models such as COCOMO or LOCOMO tended to apply their own "*corrective factors*" to come up with numbers that matched their own experience. In many cases, very little data other than C/SCSC data was available which forced managers to make rough orders of magnitude estimations. Portions of the Block Change that tended to be fixed (e.g., lab overhead) were generally close to the budget. Portions of the Block Change that were product related (e.g., integration and test) tended to be several months late and over budget. If the project was schedule driven and a release was "*forced*" because of that schedule, two to three corrective updates were required and the total project cost increased by over 15%.

LESSONS LEARNED

These lessons learned were provided by the visited weapon systems during the round robin. Many of the weapon systems were experiencing the same types of problems and made the same suggestions. Below is a summary of the most repeated and critical suggestions provided to the round robin team.

1. Distributed support concepts (where maintenance is done on the same OFP in different locations) experienced major communications problems which led to misunderstandings of requirements and interfaces and thus to schedule delays and cost overruns.
2. "*Pure*" organic support of OFPs was technically impractical except for mature configurations no longer experiencing instability.
3. The maintenance planners must plan and provide resources for multiple concurrent software development efforts.
4. Technology insertion must be factored into the weapon system as early as possible.
5. Block changes will typically request maximum changes at the sacrifice of throughput and memory. Spare throughput and memory will typically be expended by the fifth block change (if not earlier). Plan for growth early or be caught in the trap of developing new capabilities at the cost of existing functionality.
6. Plan for changes in the support and training environments that are sympathetic to the OFP change. OFP changes can result in major changes to simulations and stimulations.
7. Training and technical orders must remain current with the OFP release.
8. Ease of system/software change depends on the quality of the documentation and when it was received. Development and design rationale is often more important than formal documentation.
9. There is a definite difference between the user and the supporter for the time requirements associated with a block change. The user expects to see an approved change fielded within 18 months. The supporter seems to feel that he meets the user's requirements by releasing a new version of software every 18 months. This often results in user priority requirements taking as long as 44 months to reach the field.
10. The amount of up-front time required to place a contractor on contract for a block change seemed excessive and often was the major cause for an eight to nine month slip in release. Contracting should be done in parallel with the requirements definition for the block update.
11. Every effort should be made to decrease the number of flight tests required for validation of a block change. Investments in comprehensive simulations and stimulations have a rapid return on investment by decreasing the number of flight tests required. However, these simulations and stimulations must also be updated as part of the block change process.
12. Labs that are given the same prioritization as operational aircraft have the capability to obtain necessary spares and replacement parts. Labs that are not given this prioritization suffered schedule slips and cost overruns due to a lack of operational equipment.
13. Those programs that co-located their integration and test facilities with their flight test capabilities significantly reduced flight test costs while improving schedule adherence and communication of requirements.

ACKNOWLEDGMENTS

The F-22 Life Cycle Software Support Integrated Product Team thanks those who sacrificed their valuable time and participated in the F-22 Software Supportability Trade Survey: Col Ron Bischoff, SM-ALC/YFL; Capt Jeff McElroy, ASC/YFP; Capt Shawn Shanley, HQ USAF/AWC/28TS/TXBF; Capt Dennis Fleming, HQ ACC/SCTA; and Mr. Art Rindell, SM-ALC/TIS. The IPT would also like to thank project directors and those individuals residing with them at the SPO offices and system support facilities from each of the four aircraft weapon systems for accommodating our entire team and arranging their schedules to discuss the support details of their respective fighter aircraft: F-15 — Mr. Bob Anderson, WR-ALC/LFE; F-16 — Mr. Bruce Kress, ASC/YPVC; F-14 — Mr. Brad Gilmer, Code: P2205; and F/A-18 — Mr. Rich Bruckman, Code: C2107. Finally, a special thanks to the System Program Directors (SPDs) and Program Management Authorities (PMAs) for their support in this entire effort: Col. Rutley, WR-ALC/LF; Col. Kenne, ASC/YP; Capt Richard Evert, Program Executive Officer (PMA 241); and Capt. Joe Dyer, Commander (PMA 265).

TAB 6

How More Is Less: The F-22 Streamlined Block Change Cycle

Dean Hipwell, Capt, USAF
SM-ALC/YFSL
F-22 Life Cycle Software Support IPT
Jerry Raddatz
Lockheed Fort Worth Company
F-22 Life Cycle Software Support IPT

Presented at the Seventh Annual Software Technology Conference
Salt Lake City, Utah, April 14, 1995

INTRODUCTION

Within the F-22 Advanced Tactical Fighter Program, the Life Cycle Software Support (LCSS) Integrated Product Team (IPT) has been leading the effort to streamline the software block change process in order to reduce the time and cost required to update embedded operational flight software. The F-22 fighter is expected to have a life-cycle of at least thirty years after production begins. During these 30 years, air vehicle software is required to be updated every 18 months. This update is called a Block Change Cycle (BCC) and implementing consecutive block changes is called the Block Change Cycle process. Streamlining the Block Change Cycle process is the focus of this paper.

BACKGROUND

In order to understand the Block Change Cycle for the F-22 program, one must have some knowledge of how the weapon system is organized. The F-22 Weapon System is composed of three major systems: an air vehicle, a support system, and a training system. Each major system contains both hardware and software. This paper only addresses the software change process for the F-22 Air Vehicle.

Air Vehicle software, collectively called the Operational Flight Program (OFP), controls the operation of three systems: the Avionics System, the Vehicle Management System, and a collection of functions called Utilities and Subsystems. Each system within the Air Vehicle is further divided into subsystems. For instance, the Avionics System includes radar, communications, electronic warfare, and stores management subsystems. The Vehicle Management System includes flight controls and kernel services subsystems. Utilities & Subsystems includes environmental, landing gear, and braking subsystems. The F-22 OFP contains software load images for all three systems in a single load file. Traditional federated avionics architectures use separate software load images for each subsystem. In the integrated architecture world of the F-22, when we say "*THE OFP*" we mean the entire load image for all Air Vehicle software.

Support System and Training System software will be updated at the same time as the Air Vehicle, but may not be part of the OFP block change process. Support System includes an integrated maintenance information system, a system/software engineering environment, and all tools needed to support the Weapon System. Support System software will be upgraded using government, contract, and commercial off-the-shelf approaches depending on the system being upgraded. The Training System consists of pilot simulators, computer-based training systems, and other training devices needed to teach people how to operate and maintain the Weapon System. The current software support concept for Training Systems is to separately contract for each software upgrade.

ANALYSIS

In order to predict change magnitude, the LCSS IPT investigated the OFPs for previous fighter aircraft: F-14, F-15, F-16, and F/A-18. OFPs for these aircraft contained software of multiple languages and did not have an integrated architecture. Code sizes varied depending on aircraft complexity and operational mission requirements. The F-22 OFP will contain approximately 1.6 million source lines-of-code (SLOCs) when first fielded. After normalizing OFP sizes to equivalent Ada SLOC using function point analysis as an intermediate step, we estimate that the F-22 OFP will be at least twice the size of any previous fighter aircraft software. The historical rate of change for previous aircraft programs is approximately ten percent of the SLOC per change cycle. Considering the estimated OFP size, the F-22 OFP change magnitude could exceed 160k SLOC. In previous fighter programs, a 10% change to the OFP took 36 months to implement with some taking as long as 48 months. The F-22 program goal is to accomplish block changes in 18 months, one-half of the time required for previous fighter aircraft, while retaining the same capacity for changing the capabilities of the weapon system. Based on a software volatility analysis performed by Jon Floyd of the Lockheed Fort Worth Company (LFWC), we anticipate that only five percent of the F-22 code will change during each cycle. Floyd analyzed each subsystem to determine its expected frequency of change and then these individual volatility indices were combined to result in an overall five percent change prediction. The exact changes to be implemented for each cycle will be determined during the initial phases of the block change process.

TRADITIONAL PROCESS

The block change process for software updates is fairly well defined. Each cycle begins with a Requirements Definition phase which includes all of the negotiations necessary to define and fund selected changes for the OFP. These requirements are then incorporated into code in a Software Development phase. After the software is designed, developed, integrated and tested at the module level, the various software modules are integrated and tested at the subsystem and then the system level. System Integration and Test activities include the OFP Build Process where all source code comes together into a single load image. Next, there is a Development Test and Evaluation (DT&E) phase where the software is functionally tested to ensure that the software can be safely flown in a real world environment. DT&E is followed by an Operational Test and Evaluation (OT&E) phase in which all operational capabilities are verified. After OT&E, the software update is packaged, duplicated and distributed to the fleet. The distribution package contains all of the associated documentation required to support the new OFP. Over the expected 30 year life cycle of the F-22 there will be at least 20 block changes.

APPENDIX K Software Support

F-22 STREAMLINED PROCESS

Now that the software development process has been briefly explained, the question is, "*How will the F-22 program improve the Block Change Cycle timeline to meet the 18 month requirement without sacrificing quality or capability?*"

Four factors account for the F-22 Block Change Cycle streamlining effectiveness. The first factor is the use of Integrated Product Teams (IPTs) for the entire Weapon System support process. IPTs enable the customer, designer, tester, technical publications writer, quality engineer, manager and others necessary to the design and support process to remain in close contact at all times. This synergy results in better understanding of design and programmatic issues that affect the software by all participants. This synergy also demands "*buy-in*" by all parties at each step of the change process so that there are no surprises at delivery. Since understanding and communication are better, time is saved and a higher quality product is obtained, ensuring that our product will match the customer's needs the first time.

The second factor that helps streamline the BCC process is the use of Ada. By Department Of Defense (DoD) direction, all software for weapon systems must be developed in Ada in order to improve maintainability throughout the life cycle of the weapon system. Ada features, such as modularity, strong typing, and information hiding result in easier updates and modifications. Additionally, the use of a single high-order language minimizes training, complexity and software interfaces.

Third, the F-22 program will store parameter definitions in mission data tables instead of "*hard coding*" values into the OFP so that the number of OFP changes are reduced. Many Weapon System changes are driven by requirements from air vehicle operators, i.e. pilots. Historically, the Pilot Vehicle Interface (PVI) is one of the most volatile pieces of fighter aircraft software because of perfective change requirements. Coding PVI parameter definitions into data tables gives pilots the opportunity to customize the aircraft without going through a formal OFP block change process. Such implementations include everything from navigational way point definition to enemy identification parameters and have been defined with the help of Air Combat Command — the customer. We anticipate that placing parameters in data tables will drastically reduce the number of OFP code changes while increasing the flexibility afforded to customers.

The fourth streamlining factor that the F-22 program will use is performing the Requirements Definition phase concurrently with the previous Block Change Cycle. During each cycle, Avionics System Requirements Review Conferences (ASRRCs), will generate requirements for mission data and OFP changes for the following cycle. Requirements will be analyzed, defined, prototyped, and funded during each ASRRC period. This process allows up to 18 months for funding to arrive before the start of software development and, therefore, prevents any delays due to unforeseen budgeting issues. By always having requirements defined and funded prior to the start of software development, at least 6 months will be saved in each cycle.

ORGANIZATIONAL PLAYERS

A typical block change process will have four major players performing the key functions:

- Users from the field identify deficiencies and generate change requests.
- A Technical Focal Point acts as an initial screening agency and presents a consolidated list of change requests.
- Major Command Headquarters (HQ) reviews the change list, prioritizes changes for investigation, and develops a final list of requirements for implementation.
- The Software Support Facility (SSF) conducts feasibility analyses on requested changes, develops and tests final change requirements, makes the changes, builds and distributes the changed operational flight program.

The major portion of the BCC will occur in the SSF, since the SSF personnel will be responsible for feasibility and trade studies, technical development, and change distribution. OFPs and mission data will be changed in the SSF in the technical development process. In the F-22 Program, mission data table development is treated as a separate process under the oversight of ACC personnel co-located in the SSF.

CHANGING THE PROCESS

In order to help with the streamlining of the BCC process, the LCSS IPT will move the Feasibility Study out of the Technical Development phase and added it to the Requirements Definition phase. Then we will provide parallel paths for the mission data changes at two points: the HQ approval process and in the SSF implementation process. Third, we will provide a "*flyable*" copy of the OFP to both DT&E and OT&E so that testing can begin as soon as it is safe to do so, which allows OT&E to begin prior to the end of DT&E. Finally, since mission data tables, by their very nature, do not require the extensive testing and documentation of traditional code changes, mission data changes that fall within allowable ranges will not require testing and certification. Mission data updates may be distributed on a nine month schedule.

RISKS

There are several risks areas associated with streamlining that must be addressed. The size of individual software modules may dictate either a redesign or the use of additional resources in order to change the code within an 18 month period. Redesign into smaller, more modular pieces allows concurrent development to occur. Additional resources (people, equipment, etc.) could be assigned to support a larger design, but this option follows the law of "*diminishing returns*". Multiple, smaller modules would isolate changes and reduce the amount of code to be analyzed, integrated, and tested for each functional change.

Another risk area is the coordination of hardware modifications with the Block Change Cycle schedule. Hardware modifications may impact the schedule since the highly integrated nature of the air vehicle is sensitive to just about any modification (except perhaps the color of the tires). Hardware modifications, however, come with separate funding and schedules, so that software releases accompanying each hardware modification must be coordinated with planned OFP block change releases.

Surge capability will be able to handle most schedule anomalies. Surge capability is the "*management reserve*" that is built into the SSF planning process in order to adjust for anomalous events. Surge capability consists of second shift usage, equipment reallocation, overtime work, etc. For example, equipment failures in the SSF may force a delay of development activities. Emergency and urgent change requests will require the full dedication of SSF resources for short periods of time. Congressional funding debates may impact the block change schedule in strange and mysterious ways. Once an anomaly is over, and, depending on the length of the delay, surge capability can be used to recover lost schedule letting the SSF catch up.

CONCLUSION

There is a significant historical challenge being attacked by the F-22 program in the area of software supportability. We are using a single high order language in an attempt to simplify design and maintenance of the entire software package used on this program. We are planning an 18-month Block Change Cycle for the update of software. Our aggressive management approach of using IPTs for the entire process, requiring tight, concurrent scheduling of all activities, and performing aggressive risk management will, we believe, significantly reduce the overall life cycle cost of the F-22 software and provide more timely updates as required by the customer. The utilization of integrity, teamwork, and logic in planning for and implementing the Block Change Cycle should ensure a quality product that meets the customer's needs in a timely manner.

APPENDIX

L

STARS and I-CASE Tools and Services

Blank page.

APPENDIX L

STARS and I-CASE Tools and Services

CONTENTS **PAGE**

Tab 1:
Software Technology for Adaptable, Reliable Systems (STARS) L-2
Ada Applications and Insertion L-3
Bindings L-3
Technologies L-5
Service Lab Research & Development L-6
Operational Reuse Libraries L-7
Commercial Off-the-Shelf (COTS) Products L-7
Unisys COTS Products L-7
Hewlett-Packard (HP) COTS Products L-8
Digital COTS Products L-8
IBM COTS Products L-9
CASE Vendors Products L-10
Reuse L-11
FAR/DFAR Clauses L-13
Domain Analysis L-13
Software Engineering Environment (SEE) L-14
Process L-15
Standards L-17
Tab 2:
I-CASE L-19

APPENDIX L STARS and I-CASE Tools and Services

TAB 1

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

The following discussion conveys the impact of the STARS program from technology development to transition to commercial practices. Some earlier STARS efforts, such as the early Foundation efforts, are generally not included, as the focus here is on products and programmatic efforts supporting megaprogramming. This narrative provides a detailed description of each STARS accomplishment, STARS role, value-added, and an IMPACT statement. These IMPACT statements, enumerated below, reflect an increased level of value-added. For IMPACT statements that are not self-evident, definitions are provided.

- New technology development.
- Laboratory demonstration — technology developed in research environment and demonstrated for proof of concept.
- Enhancement of existing technology.
- Application of technology by end user.
- In use on _____ program.
- Became an industry standard — a de facto standard widely used due to adoption by a number of suppliers.
- Influenced/enhanced a commercial product.
- Developed into a commercial product — a tool, methodology, technique, or service available from an established company.
- Influenced a national/international standard.
- Became a national/international standard — standards developed through a rigorous, formalized, consensus-based process by national/international user/supplier representatives for industry-wide use.

Many of the items in this narrative (with the general exclusion of commercial products) are available from Asset Source for Software Engineering Technology (ASSET) *[discussed in Volume 1, Chapter 9, Reuse]*. For the availability of individual items, see the STARS/ASSET catalog available from ASSET.

NOTE: See Appendix B for the STARS Web address.

Ada APPLICATIONS AND INSERTION: STARS Contribution to Ada Technology Development, Application, and Insertion Efforts

SHADOWS

Early (1988-89) STARS-funded software efforts to demonstrate the advantages of using Ada in real-time DoD systems were instrumental in acceptance of Ada in this segment of defense industry.

AMMWS (Advanced Millimeter Wave Seeker)

The Air Force AMMWS shadow program performed an independent, parallel software development effort to demonstrate the feasibility of Ada in small, high-speed, mission-critical systems.

IMPACT: *Application of technology by end user.*

C2P (Command & Control Processor)

This was a Navy shadow program to demonstrate the use of Ada in Navy Command and Control systems. The **C2P** program used parallel development to compare a CMS-2 and Ada implementation and to demonstrate Ada as a design method.

IMPACT: *Application of technology by end user.*

F-111 DFCS (Digital Flight Control System)

This Air Force shadow project used parallel development efforts by the same contractor but with different teams. The program approach was to develop the **F-111 DFCS** Operational Flight Program in Ada for the 1750A ISA.

IMPACT: *Application of technology by end user.*

BINDINGS

This allows application programs written in Ada to use and interface with products implementing functional standards (such as query language, graphical user interface systems, operating systems) and software written in other languages. STARS has both defined individual bindings, as well as accelerated the formation of consensus through evaluation activities.

GKS (Graphical Kernel System)

GKS provides a two-dimensional graphics system for the application programmer. It offers basic tools for software development that allow control and efficient use of workstations. In 1985 GKS was published as an International Standards Organization (ISO) standard. The STARS-developed GKS/Ada binding provides a set of strictly defined graphical procedures that serve as an interface between GKS and the application program written in the Ada language. GKS facilitated technology usability within the Ada community and provided public domain implementation of Ada binding to national/international standard. It influenced a standard with respect to general language bindings as well as specific Ada binding.

IMPACT: *Influenced a national/international standard.*

PCTE (Portable Common Tools Environment)

PCTE provides a framework for data integration, which potentially allows CASE tools to use a common data definition and structure. PCTE is currently a standard promulgated by European Computer Manufacturers Association (ECMA) and is a candidate ISO standard. Emeraude, a STARS Prime Affiliate, has a commercial implementation of PCTE based on an earlier standard, to which STARS implemented a set of Ada bindings compliant with ECMA's Ada binding standard (ECMA 162). As a result of this implementation, flaws were uncovered in the ECMA 162 standard; suggested corrections have subsequently been incorporated into the standard. In addition, STARS rehosted several STARS-developed tools onto the Emeraude PCTE implementation using these bindings and demonstrated that they function correctly and with adequate performance. This facilitated the technology usability within the Ada community. As a result, developers of tools written in Ada can rehost to PCTE at lower cost and with higher assurance that things will work as advertised.

IMPACT: ***Enhanced new technology. Influenced a national/ international standard.***

X (A de facto graphical user interface standard developed at MIT)

Bindings to X-Windows were developed under the STARS program and provided the first full working Ada specification of the X-Window system. This binding hid the actual C implementation of MIT's X-Window from Ada developers and allowed them to concentrate on building Ada applications using X. The STARS X-Window bindings were adapted and made available commercially by Systems Engineering Research Corporation (SERC) of Mountain View, California and ATC. SERC has also added Motif support. This facilitated technology usability within the Ada community, and provided new technology development that evolved into commercial products. Other Ada compiler vendors such as IBM and Rational now offer compatible X/Motif Ada bindings.

IMPACT: ***New technology development. Influenced/enhanced a commercial product.***

SQL (Standard Query Language)

SQL or *"Database Language SQL"* is the American National Standards Institute (ANSI), ISO, and Federal Information Processing Standard (FIPS) standard language for accessing relational databases. STARS representatives participated in the SEI working group that drafted the Ada binding to SQL, subsequently adopted as the standard. The SEI group developed the SAME (SQL/Ada Module Extensions) method for developing an Ada-appropriate abstract layer upon the standard binding. STARS contractors IBM and Lockheed developed tooling that automates the SAME method, thus facilitating technology usability within the Ada community.

IMPACT: ***New technology development. Influenced a national/ international standard.***

APPENDIX L STARS and I-CASE Tools and Services

TECHNOLOGIES

Technology developments in software verification capabilities, hardware test languages, intermediate languages, semantic interface specifications, and development environments to further extend the IMPACT of Ada.

Penelope

Penelope is a workstation-based tool set for mathematically verifying properties of interest about Ada programs. Tool development was sponsored by Rome Labs and STARS and implemented by Odyssey Research Associates (ORA). The tools have been applied within National Security Agency (NSA) and National Aeronautical and Space Administration (NASA). Penelope directly supports reliable Ada software and will be provided to the Army STARS Demonstration project.

IMPACT: *New technology development. In use on DoD and other government agency programs.*

UATL (Universal Ada Test Language)

UATL is hardware test language, originally developed as part of the STARS Foundation Project, which provides a consistent framework for testing complex systems at all stages of the software/system development production and maintenance cycle. UATL test procedures have been modeled after ATLAS constructs to retain familiarity with this current standard DoD/IEEE test language, and this makes the transition to Ada relatively straightforward and cost effective. The IEEE Ada Based Environment for Test (ABET) standard was an outgrowth of UATL. Using seed money from STARS, ITT, under subcontract to Boeing, provided the leadership to establish the ABET language as an IEEE standard. As part of the ABET standardization effort, the language has grown into an even more powerful tool called ABBET (A Broad-Based Environment for Test). Boeing built a test environment based on this standard to support the F-22 that is now being used by 26 F-22 suppliers. ABBET enables the use of the same test environment from the factory to the depot. The Air Force estimates that ABBET will result in a $186 million dollar savings on the F-22 program.

IMPACT: *New technology development. Influenced national/ international standard.*

IRIS (Intermediate Representation Including Semantics)

IRIS is an intermediate computer language representation that was initiated under the ARPA Arcadia consortium and further developed and trial-tested under STARS. IRIS-Ada is an instance of IRIS specialized for the Ada language. It is used to facilitate Ada-based tools by providing an intermediate representation of semantically analyzed Ada so that each tool does not have to do its own language analysis. IRIS continues to be used in the Arcadia consortium, which is experimenting with IRIS-based analysis tools.

IMPACT: *New technology.*

ACE (Ada Command Environment)

ACE is an interactive Ada environment coupled with a set of Ada abstract data types (ADTs). This interactive environment allows users to rapidly prototype general Ada applications, while the ADTs allow prototyping of applications for particular domains, such as X-window systems applications. In addition, the ADTs provide an Ada view of underlying applications, which when combined with the interactive environment, replaces the traditional role of a command language. When using ACE, Ada becomes the command language as well as the programming language.

IMPACT: *New technology.*

SERVICE LAB RESEARCH & DEVELOPMENT

STARS funded support to in-house DoD laboratory projects.

CAMP (Common Ada Missile Packages)

CAMP is a missile guidance and control functions, developed with partial STARS support, written in Ada, and designed as generic components for reuse in guided missiles across the Services. Over 450 reusable Ada parts were generated by CAMP, making it an early prototype of domain-specific reuse.

IMPACT: *New technology development.*

ASTS (Ada Symbolic Test System)

ASTS provides a symbolic debugger capability for Ada. Two incremental software prototypes were built: one to demonstrate symbolic debugging and a second to integrate tools for symbolic debugging. ASTS was developed at Rome Air Development Center (RADC).

IMPACT: *New technology development.*

Math Functions

Math functions is a set of Generic Elementary Math Functions packages for Ada that conform to the SIGAda Numerics Working Group's proposed standard, developed by Argonne National Laboratories. An advantage of this package is that the results may be truncated to obtain a desired accuracy. Later, these math functions were incorporated into Ada 9X language specification.

IMPACT: *New technology development.*

ATVS (Ada Test and Verification System)

ATVS provides tools and tests for static and dynamic analysis. ATVS also provides test and verification support, including automated generation of data for inclusion in DoD-STD-2167A documents. ATVS was incorporated into SLCSE.

IMPACT: *New technology development.*

ABICS (Ada-Based Integrated Control System for the F-15)

ABICS demonstrated Ada for real-time computer applications. ABICS integrated flight/fire control weapons delivery system with aircraft systems, and was the first use of Ada for flight control application.

IMPACT: *Technology demonstration.*

SLCSE (Software Life Cycle Support Environment)

SLCSE is a software engineering environment, consisting of an integrating framework and an extensible toolset, allowing the integration of organization-specific tools. STARS provided funding for the early SLCSE development work that demonstrated the viability of a central repository as a data integration mechanism. The repository information model was based on DoD-STD-2167A. Production of this new technology is being funded by Electronic Systems Center in a program called PROSLCSE.

IMPACT: *New technology development. Developed into commercial product.*

OPERATIONAL REUSE LIBRARIES

STARS reuse tools, in particular library mechanisms and interoperability capabilities, have been operationally used by organizations providing reuse library services to the software community. These reuse libraries are expected to be operational well past the projected end of the STARS program with one planned to become a commercial enterprise.

COMMERCIAL OFF-THE-SHELF (COTS) PRODUCTS

This section identifies various STARS technology efforts that are being commercialized by Unisys commercial and HP. Both Unisys Commercial and HP are the commercial counterparts* of Unisys Government Systems Group.

Unisys COTS Products

This section identifies both STARS technology efforts that are being commercialized by Unisys and Unisys commercial products that are being leveraged and used as part of the Unisys STARS SEE.

Re-engineer

Re-engineer is an internally developed Unisys tool to facilitate analysis, reverse engineering and restructuring of legacy code in a variety of languages. Re-engineer includes language anatomizers and regenerators that are integrated with IDE (a STARS Prime Affiliate) Software Through Pictures (StP). STARS is conceptually extending the usage scenario of Re-engineer to directly support a domain engineering life cycle working in conjunction with StP, NASA/CTA's Knowledge Acquisition for Preservation of Trade-offs and Underlying Rationale (KAPTUR) tool, and STARS RLF.

IMPACT: ***Influenced/enhanced a commercial product.***

ReQuire

ReQuire is a Unisys requirement traceability tool that allows the tracking of requirements. ReQuire was developed independently by Unisys to support requirements traceability in a traditional application engineering life cycle. STARS has conceptually extended the usage scenario of ReQuire to support a domain engineering life cycle. ReQuire is a Unisys product that has been licensed for use on the Army STARS demonstration project.

IMPACT: ***Influenced/enhanced a commercial product.***

*[*The term commercial counterpart, as used here, refers to the active partnerships established by the STARS Prime contractors with leading commercial vendors to promote the commercialization of megaprogramming technologies. These firms referred to as the Prime's commercial counterparts are: Digital with Boeing, IBM with Loral, and Unisys Commercial and HP with Unisys.]*

RLF (Reuse Library Framework)

RLF is a toolset to support domain modeling and the development of domain-specific reuse libraries. It is intended to be used by system/software engineers involved in domain analysis, domain modeling, reuse library development and administration, and reuse-based applications engineering. RLF is in use on the Army STARS demonstration project, the Air Force

CARDS program at NUWC, NRAD, Air Force Sacramento ALC, NSA, as well as on other programs. RLF was first prototyped under STARS Foundations beginning in 1988, has been further developed under STARS Prime, and extended with internal Unisys funding toward a commercial product.

IMPACT: *New technology development. In use on several service programs.*

Hewlett-Packard (HP) COTS Products

This subsection identifies various HP products that have been leveraged and improved due to their use in STARS.

VUE (Visual Users Environment)

VUE supports X/MOTIF-based presentation integration within a SEE. VUE also support transparent access to tools within a heterogeneous SEE. HP's VUE product has been licensed by many vendors and is available on a variety of equipment. STARS has used VUE to provide a consistent user interface and transparent access to capabilities in a heterogeneous HP and SUN environment. STARS is one of the first validators of VUE/Softbench within a heterogeneous environment. VUE is in use on the Army STARS demonstration project.

IMPACT: *Influenced/enhanced a commercial product. In use on the Army demonstration program.*

Synervision

Synervision is an HP process enactment product that is deeply integrated with Softbench. STARS was one of the alpha users and provided significant feedback for this product. STARS continues to exercise the product as the base of process technology within the Unisys STARS SEE. Unisys STARS, ASR, and SDA (Prime Affiliates) are working with HP on overlap/underlap/best ways of integrating multiple process technology products with Synervision. It also fosters cooperation among different providers of process technology products.

IMPACT: *Influence commercial product.*

Softbench

Softbench is a SEE framework product that supports a message paradigm for control integration. HP's product has been licensed by many vendors and is available on a wide variety of equipment. HP has provided access to Softbench libraries that has enabled STARS to create Ada bindings to Softbench and deeply integrate Ada tools using those bindings (e.g., RLF). STARS has extended example Softbench tool integrations to *"fully"* integrated, robust message-based integrations. HP plans to *"share"* STARS bindings and integrations with other Softbench users. Softbench is the key integration mechanism within the Unisys STARS SEE and is in use on the Army STARS demonstration project. It facilitated usage of COTS SEE framework and influenced usefulness of framework commercial product via contributions to third-party library for sharing among users.

IMPACT: *Influenced/enhanced a commercial product.*

Digital COTS Products

This section identifies various Digital products that have been leveraged, extended, and improved due to their use in STARS.

APPENDIX L STARS and I-CASE Tools and Services

CDD/R (Common Data Dictionary/Repository and Administrator)

CDD/R provides a single, shared store of data about projects, systems, and data, as well as procedures and rules for managing them. It is the foundation of the Boeing STARS SEE and has been extended and enhanced to accommodate the requirements of the STARS megaprogramming approach to software engineering. The extensions added by Boeing include definitions of work products identified by DoD-STD-2167A, and DoD-STD-498, and system object support for process definition and enactment, making the Boeing SEE adaptable to other DoD projects.

IMPACT: *Influenced/enhanced a commercial product.*

DECPlan

DECplan is Digital's versatile project management, planning and time management tool providing distributed management capability for the Boeing STARS SEE. Using a special application program interface (API) supplied by Digital, Boeing was able to integrate DECplan with CDD/R (see above). The integration enabled Boeing to demonstrate the power of one aspect of megaprogramming — a process-driven SEE. Boeing is currently encouraging Digital to make the API more widely available.

IMPACT: *In use on the Navy demonstration project.*
Influenced/enhanced a commercial product.

IBM COTS Products

This section identifies various technology efforts developed or influenced by STARS that are being commercialized by IBM or SET and their business partners/subcontractors.

PCTE (Portable Common Tools Environment)

IBM commercial is developing technology that implements the ECMA PCTE standard. **PCTE** provides tool integration through data sharing and execution control by following an integration schema. IBM STARS has fostered and encouraged this development to meet the government's need for software engineering environment data integration. STARS has influenced IBM commercial to proceed with this technology development and has facilitated beta testing of the technology in conjunction with other ARPA programs.

IMPACT: *Influenced/enhanced a commercial product.*

CMVC/6000 (Configuration Management Version Control)

CMVC/6000 allows software developers to better manage their software development process and reduce complexity. CMVC/6000 is especially suited for configuration management of large projects. Design/defect management is integrated with configuration management and version control, building upon existing version control. Project communication is provided by a notification facility, and change history is recorded so that previous work can be recreated. This product is being used by Loral FSC on government projects. Because of STARS involvement in WorkBench/6000, IBM began marketing CMVC/6000 two years earlier than would have been the case without STARS involvement.

IMPACT: *Influenced/enhanced a commercial product.*

SDE Workbench/6000 (Software Development Environment WorkBench/6000)

This is IBM's framework for the AIX operating system. **SDE WorkBench/6000** provides the basis for a customized, integrated software development environment. It includes a set of core development tools for software development, such as language sensitive editors and interfaces to standard operating system features. SDE Integrator/6000 is a companion product that enables users to integrate new and existing tools so that they can use the SDE WorkBench/6000's services and be integrated with other tools. IBM product development credits the IBM STARS team for influencing the licensing of Hewlett-Packard Softbench and Encapsulator technology that was ported to the IBM Reduced Instruction Set Computer (RISC) System/6000-AIX platform to provide the basis of the product set. IBM agrees that it would have been two or more years beyond the Workbench product availability before a framework product would have been available on IBM machines. The technology is sufficiently similar that tools integrated using one environment can be migrated to the other with trivial effort.

IMPACT: ***Influenced/enhanced a commercial product.***

Rediscovery/2

This is a new COTS product announced by IBM Software Solutions and soon to be available on IBM's OS/2. (OS/2 runs on large IBM-compatible PCs, such as 486/33, 16MB, and 340MB RAM configurations, but not on open platforms, like UNIX). The product uses the RIG's Basic Interoperability Data Model (BIDM), which was largely derived from the ALOAF model developed in the STARS program.

IMPACT: ***Influenced/enhanced a commercial product.***

COMPUTER-AIDED SOFTWARE ENGINEERING (CASE) VENDORS PRODUCTS

This section identifies various technologies and products developed under STARS that have been commercialized by independent COTS vendors.

PEAKS (Process Engineering and Analysis Kernel System)

PEAKS is a tool for modeling, simulating, and managing processes and projects, and used by project managers and process engineers to design and refine processes before enacting them. PEAKS is a second generation system that evolved from the Software Process and Management System (SPMS). SPMS was initiated as one of three STARS *"breakthrough initiatives."* PEAKS, an enhanced version of the SPMS prototype, has been used on the STARS demonstration project in a process-driven software development environment. Experience in developing and supporting this tool has demonstrated its power and the practicality of the approach. PEAKS is an enhanced and industrial-strength version of SPMS.

IMPACT: ***New technology. Developed into a commercial product.***

InQuisiX (previously known as AMS — Asset Management System)

InQuisiX is a set of tools supporting software domain analysis and recording. One use is as a reuse library mechanism for both classifying and cataloging reusable assets as well as searching for and extracting them from the library. InQuisiX supports classification using hierarchical taxonomic, faceted, and textual organization, and searching schemes. While InQuisiX development was initiated via a small business initiative research (SBIR) program

funded by DoD and NASA, STARS support significantly extended the requirements and implementation. InQuisiX is marketed by SPS of Indiatlantic, Florida.

IMPACT: *Enhanced new technology and influenced/enhanced a commercial product.*

Ada/X

Ada/X is a tool kit that allows one to develop a user interface in Ada using the MIT X-window standard. Ada/X was built by SERC as a commercial product based on the STARS Ada/X bindings.

IMPACT: *New technology development. Developed into a commercial product.*

SGML (Standard Generalized Markup Language)

SGML is an international standard for representing the elements and structure of electronically stored text. SGML and SGML Electronic Review provide the enabling technology required for integrating software documentation development with the software design and development process. SGML uses Document Type Definitions (DTDs) to unify the structure of various kinds of documents. STARS has developed DTDs and FOSIs (formatted output specification instances) for informal technical reports and for briefing chart applications. As a result of STARS encouragement and sponsorship, Arbortext, the vendor, now offers an SGML-based authoring tool that supports multiple electronic reviewers collaborating across a wide-area network to produce a finished product.

IMPACT: *New technology development. Influenced/enhanced a commercial product. Influenced a national/international standard.*

REUSE

Concepts, processes, methods and tools that define and support capabilities for and transition to a reuse-based software development approach.

RLF (Reuse Library Framework)

Significant enhancements increasing support for domain analysis, modeling, and composition were made to **RLF**. Additionally, RLF is being integrated with Knowledge Acquisition for Preservation of Trade-offs and Underlying Rational (KAPTUR) techniques and more closely linked with the Organization Domain Modeling (ODM) analysis techniques.

IMPACT: *New technology development. Influenced/enhanced a commercial product.*

ROAMS (Reusable Object Access & Management System)

ROAMS is a reuse library mechanism for an object-oriented, repository-based software reuse library. Boeing designed and developed ROAMS and integrated it with the Boeing STARS SEE.

IMPACT: *New technology development. In use on the Navy/STARS demonstration project.*

Reusability Guidelines

STARS developed sets of guidelines for developing reusable software components. They identify the characteristics of a component that makes it reusable and provides high-level guidelines for producing and selecting components with those characteristics. These guidelines form the basis for ASSET certification.

IMPACT: *New technology. In use by ASSET. Influenced/enhanced a commercial product.*

Synthesis

Synthesis is a methodology for constructing software systems as instances of a family of systems that have similar descriptions. It was originally developed by the Software Productivity Consortium and was recently placed in the public domain by the Virginia Center of Excellence. Synthesis is structured around the concept of two life cycles: domain engineering and application engineering. Domain engineering focuses on identifying the commonality and the variabilities of the product line. The commonality and variability are expressed for both the reusable components of the domain and the *"instructions for assembly"* or the associated process to use them. Application engineering focuses on how to use the reusable components and their associated processes to build a member of the product line to meet the specific needs of a system. Synthesis forms a fundamental part of the megaprogramming concept.

IMPACT: *New technology development. In use on the Navy/STARS demonstration project.*

ALOAF (Asset Library Open Architecture Framework)

ALOAF is a research and prototyping effort focused on the exchange of reusable assets among diverse libraries and the development of reuse tools that are portable among asset libraries. The ALOAF effort in general and the ALOAF specification in particular provided the technological basis for the BIDM (Basic Interoperability Data Model) developed by the RIG (Reuse Interoperability Group) standards group. IBM Software Solutions has announced a new product, ReDiscovery/2 (intended to catalog collections of renewable assets that may physically reside on a number of different platforms) which uses the RIG's BIDM.

IMPACT: *New technology development. Influenced/enhanced standards development and a commercial product.*

KAMEL (Knowledge-based Application Model EvaLuation tool)

KAMEL uses NASA's CLIPS to implement the application modeling process of Synthesis for the Boeing STARS SEE. It can be used to support question-based selections over a set of objects.

IMPACT: *New technology development. In use on the Navy/STARS demonstration project.*

CFRP (Conceptual Framework for Reuse Processes)

The STARS **CFRP** document provides a common context for understanding domain-specific, reuse-based software engineering in terms of the process involved. CFRP also provides additional technology transition guidance to STARS customers/affiliates.

IMPACT: *New technology development. In use by several STARS technology transfer affiliates to guide development of their reuse plans.*

FAR/DFAR CLAUSES (FEDERAL ACQUISITION REGULATION/DEFENSE FEDERAL ACQUISITION REGULATION CLAUSES)

Suggested revisions to the FAR and DFARs were developed by STARS to help remove barriers to software reuse created by current acquisition regulations. These revisions have been forwarded to the FAR Council for consideration. Subsequent development is being continued by the CARDS program.

IMPACT: *New technology for acquisition.*

DOMAIN ANALYSIS

Domain Analysis is a process that captures critical domain information such as product requirements, product architectures, and product design. Domain analysis enables engineers to define how domain assets can be adapted and reused. Domain analysis also identifies the processes for building new products based on reusing assets. STARS has documented and demonstrated repeatable domain analysis processes, such as the Domain Analysis Process Model (DAPM) and Organizational Domain Modeling (ODM).

IMPACT: *New technology development. In use on all STARS demonstration projects. In use by Hewlett-Packard. Became an industry standard.*

ASSET CERT (Asset Certification)

ASSET CERT is a 4-level process used to determine the quality and completeness of assets stored in a reuse library. Determining the quality of assets is essential for widespread reuse to be effective.

IMPACT: *New technology. In use by ASSET to certify assets in their library.*

SRL (STARS Reuse Library)

SRL is a prototype library mechanism that provides functions to search, browse, and extract reusable assets.

IMPACT: *New technology development. Influenced/enhanced operations at ASSET since September 1992.*

RSM (Reuse Strategy Model)

RSM is a planning aid for use by organizations and projects in defining business objectives. RSM: (1) assesses for an organization or team the extent of its transition to reuse-based development with respect to engineering management practices, and the infrastructure supporting those practices; (2) identifies candidate reuse transition goals that are reasonable with respect to the current assessment; (3) aids in defining metrics to gauge the progress against these goals. Thus, RSM provides planning guidance for those organizations pursuing continuous

improvement within their business processes while simultaneously introducing reuse as a new technology. The three STARS demonstration teams did a trial application of RSM and concluded that RSM helped them identify management and infrastructure issues and goals that they had not discovered in their top-down decomposition of strategic business objectives.

IMPACT: *New technology development.*

SOFTWARE ENGINEERING ENVIRONMENT (SEE)

Technologies and guidelines providing the underlying technology foundations for the Software Engineering Environments (SEEs) that embody megaprogramming. STARS SEEs represent integration of commercial products with the results of STARS-sponsored research and development (R&D) activities. They are significantly more advanced than the current state of practice.

CAIS-A (Common APSE Interface Set-Version A)

Early in the STARS program, both Boeing and Unisys committed to using and supporting **CAIS-A** because it was an Ada-based virtual operating system that promised portability of tools and data across operating systems on different platforms. Considerable technical effort was expended in extending CAIS-A concepts and implementations, and porting and enhancing various tools to work on CAIS-A platforms. In 1989 STARS changed its directions to require the use of commercial products that would be supported by major vendors upon the completion of STARS contracts. None of the commercial participants in the STARS program believed there was a business case for CAIS-A, which upon historical reflection has proven correct.

IMPACT: *Enhanced new technology.*

AFS

AFS is a distributed file system developed largely by Carnegie Mellon University. In AFS, because of its global name space, location independence, and wide-area performance capabilities, a user perceives no difference between accessing a local file and one that is stored remotely. AFS was commercialized by Transarc Corp., and has been used by STARS to support network-based collaborative development and document preparation. STARS was a key participant in the ARPA-sponsored experiment to understand both cultural and technical implications of AFS usage. As part of the experiment, several key documents were developed using AFS as the distributed store/versioning capability for development between three geographically distributed sites, Washington, DC, Paoli, Pennsylvania, and Seattle, Washington. STARS feedback to Transarc has assisted in the evolution of the product.

IMPACT: *Laboratory demonstration. Enhancement of existing technology. A national/international standard.*

Info Model

Info Model defines the types of objects found in an object-oriented repository. The model defines how repository objects relate to one another, how they respond to messages, and what attributes they have. The Boeing SEE has an information model based on ATIS (A Tool Integration Standard). The Boeing model has been extended to support the integration of STARS reuse and process technology as well as specific DoD objects.

IMPACT: *New technology. In use on STARS/Navy demonstration project.*

APPENDIX L STARS and I-CASE Tools and Services

ATIS/PCTE

A Tool Integration Specification (ATIS) and Portable Common Tool Environment (PCTE) are two prominent technologies that support the integration of Computer-Aided Software Engineering (CASE) tools. Though they are intended to solve similar problems, ATIS and PCTE use different core technologies. STARS is developing environments using these and other integration mechanisms and emerging standards with the intent of providing lessons learned to CASE tool vendors and of aiding in standards convergence.

IMPACT: ***Influenced/enhanced a commercial product. Influenced a national/international standard.***

RGB (Reusable Graphical Browser)

RGB is a graphical user interface for browsing the contents of an object management system. It is intended to facilitate the construction of various browsing tools by serving as a user interface component for those tools. RGB also serves to promote the portability of graphical browsing tools by insulating them from the details of the underlying graphics system.

IMPACT: ***New technology development. In use at all RLF sites as its graphical user interface.***

PCTE Browser (Portable Common Tools Environment Browser)

The **PCTE** browser (and the browser portion of RLF) is built upon the RGB. The browser permits visualization of data structure definition and provides a graphical depiction of the information model defined for a PCTE object management system, the objects contained therein, and their relationships to each other. In addition to viewing this object base, the browser can be used to invoke tools necessary to examine or manipulate objects through point-and-click user actions. As a result, vendors of tools that rehost their products to PCTE can examine PCTE data structures generated by their tools and other tools, facilitating the rehosting and correct integration with other tools.

IMPACT: ***New technology development. In use on an Ada/X-based reusable graphical browser as an object management system, validating reuse in a different context.***

PROCESS

Technologies and guidelines that define and support the process aspects of STARS megaprogramming approach to software engineering.

Enactment: Control Points and Policies

Enactment was developed by Honeywell originally to support an object-oriented environment called the Engineering Information System (EIS) and further refined under a subcontract to Boeing on STARS. In an object-oriented repository, control points and policies are basically programs that act as objects controlling the execution of other programs (called methods). This control, which can be specified by a process specification language, allows users to insert process definitions and process control into an object-oriented SEE.

IMPACT: ***Enhancement of existing technology. Application of technology by end user, Boeing.***

AAA (Agent, Activity, Artifact)

AAA is a process programming language for defining activities so that they can be encoded in the persistent programming language, Denali, for automated enactment. Boeing's STARS SEE is process-driven via enactment of AAA activities. AAA provides an integrated approach to activity definition for both automated and human agents. The AAA run-time support provided by Denali allows access to Digital's CDD/Repository so that the state of project artifacts can be retrieved and used for checking activity entry and exit conditions. An interface to Amadeus is also provided to the activity programmer to explicitly define metrics that can be collected for ongoing activity and product improvement. AAA activities implement work packages that are coordinated and integrated with DECPlan's project management product thus insuring that the enactment of an activity is consistent with the overall project plan. The Software Productivity Consortium's reuse-based process, Synthesis, has been defined in AAA. This process defines the core activities for creating and managing a product line domain and the associated applications. The Synthesis process is enacted by the Boeing STARS SEE to guide the Navy demonstration project.

IMPACT: ***In use on the Navy/Boeing demonstration program.***

Denali

Denali is a persistent programming language and system for providing enactment support for AAA activities. The Denali language provides the syntax and semantics for encoding activity definitions. The Denali compiler has extensive syntax and semantic checks that provide pre-runtime analysis to minimize improper constructs and usage. The Denali system provides an extendable interface to system services and persistent information. The Denali System provides features typically found in large-scale Ada development environments. These features include: (1) an Object Library with post-processing tools, (2) on-line access to previously developed Denali process programs, and (3) on-line information about the current Denali primitives. During execution, the Denali System handles: (1) interpretation of the Denali object code, (2) management of the runtime contexts of the executing AAA activities, (3) calls to the primitive layer, (4) debugger support, and (5) invocation of the generic language capabilities. All automated processes in Boeing's STARS SEE are implemented in the Denali language and supported by the Denali System. This technology is incorporated in the process-driven SEE to guide the Navy demonstration project.

IMPACT: ***In use on the Navy/Boeing demonstration program.***

Process Engine

Process Engine is an extension to the Denali System to manage the synchronization and integrity aspects of activities in a multi-user SEE. The Process Engine is a tailored version of the Denali Runtime System to support AAA Activities. It is integrated with an Object-Oriented repository for capturing persistent process information and provides unobtrusive metrics collection for analysis to support process improvement. The Process Engine extends the foundation provided by the generic Denali System. The added capabilities include: (1) handling the repository interface, (2) handling activity state changes, (3) supporting synchronization of activities by enforcing pre- and post-conditions, (4) handling multi-user visibility of activities, (5) integration of user defined metrics during activity enactment, (6) automated metric capture of process events, (7) implementation of a repository level interface to provide access to persistent process variables and provide activity creation and deletion, and (8) implementation of a Process User Interface for activity interaction and visibility. The Process Engine provides the man/machine interface for automated activity enactment in Boeing's STARS SEE that is being used to guide the Navy demonstration project.

IMPACT: ***In use on Navy/Boeing demonstration program.***

APPENDIX L STARS and I-CASE Tools and Services

PEM (Process Engineering Methodology)

PEM is a methodology for creating processes for a process-driven environment that incorporates elicitation of process information, functional descriptions, architectural representations, project-management driven activities, and automated process enactment. The PEM activities describe the development and support activities to produce a process-driven SEE. It specifies the Application Engineering activities for the SEE Process product line domain. Domain Engineering of the family of process-driven SEEs provides a Process Asset Library (PAL) for the product line of SEE processes. When the PEM methodology (SEE product line Application Engineering process) is exercised the process-driven SEE components are instantiated for one of the members of the family of process-driven SEEs maintained in the PAL. PEM provides a consistent definition of the activities to create and maintain a process product line domain for creating process-driven environments. This methodology is used to create the process-driven aspects of the Boeing STARS SEE.

IMPACT: *In use on Navy/Boeing demonstration program.*

Process Definition Guidelines

These are guidelines for process definition and enactment, from manual to automated enactment. A definition for enactable process is provided along with a reference model for assessing capabilities to support process enactment. Provides technology transfer assistance.

IMPACT: *New technology development.*

Software First

Software First defines an approach to system development where software is developed from the system requirements first, and hardware decisions are postponed until later in development. This software-first approach was defined by Loral and used early in the STARS program.

IMPACT: *New technology development.*

STANDARDS

Software standards activities directly influenced by STARS.

POSIX/Ada (Portable Operating System Interface/Ada)

POSIX/Ada provides an Ada interface to the POSIX standard. STARS (Loral) participated in the standards working group that developed POSIX/Ada.

IMPACT: *New technology development. Influenced a national/ international standard.*

SGML/CALS (Standard Generalized Markup Language/Computer-Aided Logistics System)

SGML/CALS is a common electronic medium for sharing documentation. STARS pioneered work in both the specification and prototyping of electronic document review capabilities with SGML. The electronic review specifications were included in MIL-M-28001B, the DoD text processing standard. STARS developed tailored SGML DTDs and FOSIs for informal technical reports and standard STARS briefing charts.

IMPACT: *Influenced a national/international standard.*

ATIS (A Tool Integration Standard)

ATIS is an object-oriented model for information stored in a repository. STARS (Boeing) with Digital (a commercial counterpart) and Atherton (a Prime Affiliate) were actively involved in developing the standard. It is uniquely suited for the broad definition of a repository system and is incorporated in Digital's CDD/Repository.

IMPACT: *Influenced a national/international standard.*

X3H4

This is the ANSI Technical Committee on Data Dictionary Interface Definition. STARS (Loral) actively participates on the technical standards committee and aids in the evolution of the standard.

IMPACT: *Influenced a national/international standard.*

X3H6

This is the ANSI Technical Committee on CASE Tool Integration Models. STARS (Loral, Boeing) actively participates on the technical standards committee and aids in the evolution of the standard.

IMPACT: *Influenced a national/international standard.*

RIG (Reuse Library Interoperability Group)

RIG is a volunteer, consensus-based organization composed of members from government, academia, and private industry. RIG drafts standards for the interoperability of reuse libraries in areas such as nomenclature, interchange protocols, and system components exchange formats. STARS initiated formation of the RIG, chairs multiple technical committees, funds secretarial support, provides draft proposals based on STARS experiences, and provides proof-of-concept prototypes. The RIG recently completed the first of its proposed standards as well as its first technical report. The RIG's first proposed standard is called the Basic Interoperability Data Model (BIDM). Two planned uses relate directly to STARS: (1) the trilateral index central to the planned interoperation of CARDS, ASSET, and DSRS is defined in terms of the BIDM data model; and (2) the data to be stored in ASSET's National Software Reuse Directory (NSRD) are defined in terms of the BIDM. An IBM Software Solutions product, ReDiscovery/2, implements BIDM. The RIG's first technical report is a glossary defining terms necessary to discuss reuse interoperability issues and forming the basis for creating a comparison framework for library mechanisms and reuse libraries. The glossary clarifies important distinctions between reuse libraries and the library mechanisms used by the libraries. The articulation of this distinction owes much to work done within the STARS program.

IMPACT: *Influenced/enhanced a commercial product. Became a national/international standard.*

ABBET (A Broad-Based Environment for Test)

ABBET is an IEEE standard (IEEE 1226) environment for test. The proposed ABBET standard began as a STARS project dedicated to establishing a common method of developing test programs for hardware systems, including replaceable subsystems and components. Common Ada packages for ABBET (IEEE STD 1226.1 - 1993) and Ada-Based Atlas-Level Test Procedure Interface for ABBET (IEEE 1226.2 - 1993) have been published by IEEE as trial use standards. UATL was the proof of concept for ABET.

IMPACT: *New technology development. Influenced a national/ international standard.*

Common APSE Interface Set-Version A (CAIS-A)

Boeing funded Softech to complete a Virtual Memory System (VMS) version of CAIS-A which was used by both Boeing and Unisys in further development. Boeing ported the WIS SDME to run on top of X-windows and CAIS-A. (Boeing demonstrated this running on a Digital VMS host from an X-window server.) Unisys ported CAIS-A to Sun UNIX. (Boeing recompiled WIS-SDME on the Sun and demonstrated that it would run without any source code changes.) Unisys developed and integrated CAIS-A and CAIS-A-based applications into its initial prototype SEE. This included:

- Porting CAIS-A to a UNIX environment and re-engineering the AJPO-sponsored implementation to a distributed client-server base;
- Porting to a SUN/MACH environment;
- Developing a graphical browser based on the STARS Reusable Graphical Browser;
- Tailoring the STARS Ada Command Environment as the command interpreter for the prototype CAIS-A-based SEE;
- Developing a prototype version of a Configuration Management System (CMS) for managing versions and variants of software libraries and a prototype Configuration Management Assistant (CMA) for managing configurations of systems that stored all their information in CAIS-A nodes; and
- Integrating the Reusable Library Framework (RLF) to use CAIS-A file nodes to store all RLF persistent data.

Inter-Tool Communication Facility (ITCF) was a VAX/CAIS-A facility designed to allow concurrently executing tools to cooperate in an integrated manner while maintaining a high degree of modularity and functional independence. (ITCF was developed to ease the integration of tools into the very early STARS environment developed by Boeing.)

TAB 2

I-CASE

CASE Tools Available With I-CASE

Each CASE tool available with the I-CASE SEE is described below. Each of these tools has an X Window graphical user interface, usually Motif. The LOGICORE architecture was designed to support the I-CASE objective of migrating to new functionality. CASE tools that have become outdated or obsolete will be replaced with upgraded COTS components, not necessarily from the same vendor. In adding a new tool to the environment, the most time-consuming effort is the development of a data conversion software utility to read the semantic data from the tool's output file or database and then write these data to the repository. If a tool outputs to a format for which no data conversion utility is available, such a utility must be developed. This is necessitated by the lack of a standard for the interface between CASE tools and a CASE repository. LOGICORE currently includes conversion utilities for such common CASE tool output formats as CDIF.

Project Management

Cost and Schedule Planning and Tracking

AUTOPLAN II by Digital Tools performs the standard project management functions including staff load leveling and critical-path calculations to assist in initial planning to develop a schedule of tasks. Periodically throughout the project, actual cost and schedule are input. AUTOPLAN II compares these data with planned cost and schedules using a variety of graphical presentations such as Gantt and PERT charts. The task network managed by AUTOPLAN II is automatically made to be the same network enforced by the automatic workflow control feature of I-CASE.

Software Size Estimation

SIZEPLUS by Marconi Systems Technology computes estimates of software size as function and feature points by a variety of methods. Feature points are an extension of function points from information management to real-time software. Input to a function or feature point calculation is information about the software that is known at the business and functional requirements analysis stages of software development. This overcomes a major problem of the COCOMO software cost estimation model of requiring an input estimate of lines of source code. SIZEPLUS includes algorithms to convert function points to lines-of-source code for a variety of programming languages.

Software Cost Estimation

GECOMOPLUS by Marconi Systems Technology inputs the SIZEPLUS-computed estimates of lines-of-source code and user estimates for various complexity factors and computes estimates of staff level by development phase. GECOMOPLUS implements the standard COCOMO cost estimation model.

Configuration Management

CMVision by Expertware performs baseline management and change control for UNIX files. It includes a capability for the user to define forms that are then tracked by CMVision. A number of such forms are available with I-CASE, including forms for problem reports, action items, engineering change proposals, and acceptance sign-off by quality assurance. Baseline management and change control of data in the relational database part of the LOGICORE repository are accomplished by tasks in the framework-controlled process/workflow network. Tasks that establish a configuration-managed baseline provide access to software that moves data from user workspace tables in the relational database to baseline tables. Once a baseline is established, change access is only granted via tasks in a framework-managed change-control subprocess network.

Management Metrics

The **I-CASE SEE** integration framework, InConcert by Xerox, automatically generates an audit log in the LOGICORE repository of all process/workflow actions — such as when activities are ready to be performed, started, and completed — and all changes to the process network managed by InConcert. This log includes date/time, action type, and person taking the action. AUTOPLAN II by Digital Tools generates a number of management metrics reports such as activities on the critical path and a graphical Gantt chart comparison of actual versus planned schedule.

APPENDIX L STARS and I-CASE Tools and Services

Business Process Improvement

Business Analysis From the Process Perspective

ASAEDIT by VERILOG enforces the IDEF0 methodology for analyzing the operations of an enterprise. This methodology identifies all processes performed in an enterprise operation and the input, output, control, and mechanism for each. A process mechanism is a resource used in the performance of a process. A process transforms input to output in response to controls. Input (output) may come from (go to) another process or an external.

Business Analysis From the User Interface Perspective

ETIP/X by AT&T and **ezX** by Sunrise Software International are used to graphically develop the graphical user interface of the application. These tools are used in the Business Analysis phase of development to allow functional people to model the enterprise from the perspective of the user interface of the application software. Fields in screens of this GUI further identify data needs of the enterprise. Data translation software exports to the repository the names of these fields as candidates for data requirements.

Business Information Input Via Forms

DART forms by Logicon are used to input data about the enterprise that are not output from a CASE tool. These data are input directly to the repository. They could include business objectives, sites, transactions and transaction rates by site, hardware and software inventories by site, organization, people including staff authorized to work on a project using I-CASE, identification and summaries of enterprise and customer policy documents, contract requirements including statement of work, deliverables and terms/conditions, and external systems and their interfaces.

Business Data Reconciliation Forms

DART forms by Logicon are used to retrieve from the repository potentially related business data from the three business analysis CASE tools so the user can identify alias and parent-child relationships. For example, each of IDEF0, IDEF1X and the user interface prototype identify information used or needed by the enterprise. These data are reconciled using these forms.

Requirements Analysis

Requirements Traceability

RTM by Marconi Systems Technology used to allocate, categorize, and trace requirements. RTM also includes a capability to strip requirements from an electronically scanned document and import this into RTM as a requirement. RTM has a capability to declare two requirements as aliases (i.e., different names for the same requirement) and to allocate a requirement at any level to two or more derived child requirements. RTM also has a capability to define keywords for categories of requirements and then link any requirement to one or more keywords. Multiple links between requirements are allowed. For example, a functional requirement may be allocated both to a unit of source code and to a test procedure.

Software Design

User Interface Design

ETIP/X by AT&T and **ezX** by Sunrise Software International combine to provide a capability to graphically design the screens and screen navigation of an X Window graphical user interface for the application. ETIP can generate both Open-Look and Motif versions of X Windows. ezX provides a full set of Motif widgets which are imported by ETIP. ETIP can also generate screens for a text only display.

Forms Design

APT by Sybase and **SQL*Forms** by Oracle are tool kits that assist the user in designing forms for direct access to relational databases implemented, respectively, with the Sybase and Oracle database management systems.

Automatic Generation of Source Code

Automatic Generation of SQL Source Code

DBStar by DBStar automatically generates SQL/DDL with the unique extensions for one of a number of user-selected COTS relational database management systems. ETIP/X by AT&T includes a facility to assist in the generation SQL queries that correspond to fields in ETIP-generated screens.

Automatic Generation of Documentation

Complex Format Documents Including Graphics

Interleaf 5 with Active Link Toolkit by Interleaf and SMARTLEAF by Database Publishing combine to automatically generate documents from data in the repository. The format and content of the document are determined by a document template input to Interleaf 5. A document template consists of standard Interleaf Markup Language plus two additional codes. One code encloses SQL language database queries which are executed to retrieve data from the repository into the document. The second code identifies a CASE tool and data for input to the CASE tool. When this code is encountered, the CASE tool is automatically put into execution to generate a graphic from the input data and the graphic is inserted into the document. A document template for a DoD-STD-2167A Software Requirements Specification is currently available. New templates are planned when Data Item Descriptions for MIL-STD-498 are approved. Users have the ability to create their own document templates.

Simple Format Documents

`Report Workbench` by Sybase and **SQL*ReportWriter** by Oracle are the report writers that are bundled with these relational database management systems. They can be used to write reports from data in the LOGICORE repository, or to generate reports that are part of the application software.

Quality Assurance

Static Metrics

Logiscope by Verilog inputs source code of the Cobol or Ada programming languages and outputs metrics at the component or architecture level. The output of the static analyzer function is a number of different metrics including basic counts, Halstead textual complexity metrics, McCabe control graph metrics, and Mohanty and Schutt call-graph metrics.

Test

Linking Test Cases to Requirements

RTM by Marconi Systems Technology traces test procedures to requirements and requirements to test procedures.

Automatic Recording and Playback of Test Scripts

XRunner by Mercury Interactive is based on the creation and replay of test scripts. A test script can by automatically created when a test is manually conducted, or it can be written in Mercury Interactive's Test Script Language. A test script includes mouse movements and clicks, keyboard input and benchmark intermediate results against which intermediate results are compared on playback. XRunner also includes enhanced debugging capabilities.

Dynamic Test Coverage Metrics

Logiscope by Verilog includes a dynamic test coverage analyzer that measures the exhaustiveness of the tests that have been conducted. Test coverage metrics include instruction blocks, decision-to-decision paths, linear code sequence and jump and procedure-to-procedure paths.

Compilers, Linkers, and Debuggers

Ada Programming Language by Sun Microsystems provides the Ada compilers, linkers and debuggers.

Reuse Library Manager

ReuSE by Westinghouse is a general-purpose manager of electronic libraries. It has a capability for the user to define library cataloging schemes for one or more types of objects such as source code, functional models, design, algorithms and test descriptions. It also has the capability to access remote libraries of reusable software

Environment Management

SEE Reconfiguration

Tivoli Management Environment by Tivoli Systems allows the SEE UNIX administrator to view, change, delete, organize or relocate SEE resources on the network. It allows the administrator to remotely configure a workstation, control login access, or reboot.

Job Accounting

JobACCT by Brain Tree Technology provides resource accounting and charge back by user, group, project/charge number and cost center. It tracks and allows charging for connect time, CPU usage, disk I/O operations, memory usage, disk usage, pages printed, network I/O and miscellaneous charges.

Backup/Recovery

NetWorker by Legato Systems provides backup and recovery support for the SEE. Backup features include automatic network-wide backups; full, incremental, and differential backups; live file system backup; and simultaneous backup of multiple clients. Recovery features include online index of backups, selection of multiple versions of a file and automatic determination of tape volumes required for recovery.

I-CASE Hardware Configuration

The **I-CASE SEE** is a client/server network of Sun SPARCStations configured so that the user can initially purchase an I-CASE to support a small group and then "*grow*" this SEE to support increasingly large development or maintenance teams. When a user grows a SEE, each hardware component in the smaller configuration is used in the larger configuration.

I-CASE Migration

A plan is in place for the migration of I-CASE from its current capabilities to new versions of current hardware and software items, new functionality and methodologies, and additional standards. A new version of I-CASE is planned roughly every six months. The most important new capabilities that Logicon plans to add to the environment within the first year are a Cobol source code reverse-engineering CASE tool, the Rational Apex Lower CASE environment, and one or more CASE tools for object-oriented analysis and design. A reverse-engineering tool for the Cobol, Fortran, C and Ada programming languages has been tentatively selected and is currently undergoing test. The current lower CASE Ada Programming Support Environment is based on the Verdix Ada compiler and associated tools. Verdix is part of the more fully featured Rational Apex.

NOTE: See Appendix A for information on how to contact the I-CASE program office and I-CASE vendor.

PART IV

Management-Related Appendices

Blank page.

APPENDIX

M

Software Source Selection

Blank page.

APPENDIX M

Software Source Selection

CONTENT **PAGE**

Tab 1:
SOURCE SELECTION UNDER ACQUISITION REFORM M-2
Tab 2:
SOFTWARE CAPABILITY EVALUATION (SCE) M-3
SCE Implementation Guidelines M-3
Evaluating Ada Experience During SCE M-6
Subprocess Area Selection Tables M-8
SCE Text for Inclusion in Instructions to Offerors M-15
Tab 3:
SAMPLE RFP PREPARATION CHECKLISTS M-17
Sample Questionnaire for Site Visit Preparation M-17
Program Profile Outline M-18
Proposal Evaluation Checklist M-19
Tab 4:
SAMPLE PARAGRAPHS FOR RFP INCLUSION M-21
Software Quality Requirement M-21
Software Testing Requirement M-21
Software Life Cycle Development and Support Environment Requirement M-22
Software Life Cycle Development Technology Scalability Requirement M-22
Reusable Software Requirement M-22
Tab 5:
SOURCE SELECTION FOR SOFTWARE SUPPORTABILITY M-23
Instructions to Offerors (Section L) M-23
Supportability Issues M-23
Additional MIL-STD-498 Considerations M-23
Additional AFSCP/AFLCP 800-45 Considerations M-24
Sample Section L M-25
Proposal Evaluation Supportability Criteria M-30
Source Selection Evaluation Considerations M-30
DoD-STD-1467 (AR) Considerations M-31
AFOTEC Pamphlet 99-102, Volume 3, Considerations M-32
Additional Considerations M-32
Software Language Considerations M-33
AFSSI 5100 Considerations M-33
Other Supportability Source Selection Considerations M-34

TAB 1

Source Selection Under Acquisition Reform

Under acquisition reform, the offeror's process and past performance are considered as significant criteria. Therefore you, as software acquisition managers and software engineers participating in source selection, must evaluate the contractor's processes and experience to select the best offeror capable of providing a quality system with the lowest development and life cycle risks. The newly revised DoD Directive 5000.1, *Defense Acquisition*, 15 March 1996, describes "*broad management principles that are applicable to all DoD acquisition programs.*" It states the following about acquiring software-intensive systems:

> *Software is a key element in DoD systems. It is critical that software developers have a successful past performance record, experience in the software domain or product-line, a mature software development process, and evidence of use and adequate training in software methodologies, tools, and environments.*

Regardless of acquisition size, you should evaluate these areas during source selection. Remember, while it is necessary for a contractor to have a mature software development process, you should examine the process that particular division or component within the organization proposes to use on your program. The parent organization as a whole might have a mature process; however, certain divisions or components within the organization might not be as experienced in or knowledgeable of that process. To assess this, you should ask if the division or component you are evaluating has successfully adopted the parent organization's process. You should also examine the division's or component's specific experience in using these processes in your application domain.

One last consideration is source selection for post-deployment software support. If you perform a source selection at this time, you must make sure the proposed post-deployment software support process will provide at least the same level of quality software as the development process provided. To do this, you should use the same rigorous methods for source selection of the post-deployment software support organizations that you used for the development organization.

APPENDIX M Software Source Selection

TAB 2

Software Capability Evaluation (SCE)

As mentioned in Volume 1, Chapter 7, *Software Development Maturity*, there are two software development capability assessment methods effective for determining the maturity of an organization's software development (and support) process — the Software Development Capability Evaluation (SDCE), developed by Aeronautical Systems Center, and the Software Capability Evaluation (SCE) developed by the Software Engineering Institute. While this section provides information on how to implement the SCE, if you are acquiring a C3 or ground electronics system, your are encouraged to contact Electronic Systems Center (ESC) for assistance in conducting the SCE *[see Appendix A, "Evaluating C3 Systems"]*.

While this section does not address the SDCE, you can find out more information (including about training) by contacting Aeronautical Systems Center (ASC) *[see Appendix A, "Evaluating Embedded/Avionics Systems"]*. For additional assistance with MIS acquisition, please contact the Standard Systems Group (SSG) *[see Appendix A, "Evaluating MIS Systems"]*. For Air Force in-house software development organizations with questions on Software Process Improvement and Software Maturity Assessments, contact the Air Force C4 Agency *[see Appendix A, "Software Process Improvement and Software Maturity Assessments"]*.

SCE IMPLEMENTATION GUIDELINES

General

Software Capability Evaluation (SCE) offers a means to evaluate an organization's software process capability, that is, how well an organization manages the process it uses to create software. SCE provides a way to compare a development organization's software process against a predefined standard. The purpose of these guidelines is to standardize the application of SCE on source selections. The Software Engineering Institute's (SEI) Capability Maturity Model℠ (CMM℠) is the basis for this SCE appraisal. The SCE appraisal is intended to be considered as an integral part of the source selection evaluation, however, the SCE evaluation team may operate independent of other area/factor source selection evaluators. SCE results should be evaluated consistent with evaluation criteria specified in the Request For Proposal (RFP).

Applicability

SCE applies to all source selections for Management Information Systems (MIS) and Command, Control, Communications, Computer, and Intelligence (C4I) Systems with software development costs greater than $10 million. Software development includes: development of new code, modification of existing code, and integration of software modules. Source selections with software development costs less than $10 million should consider the use of SCEs based on a cost/benefit tradeoff and the goal of acquisition streamlining. An SCE should always be performed on prime contractor Offerors. The only exception is when the proposed prime does not do, and never has done, software development, and is acting as only a "*general contractor*." In this exception, the proposed prime must not impose any process guidance on proposed subcontractors that affect the sub's software development activities.

Conducting multiple SCEs on an Offeror's team is encouraged if one or more proposed subcontractors are to perform significant software development. All proposed subcontractors performing more than $10 million or 35% of the software development using their own processes should be evaluated. SCEs are normally not applicable for source selections for the acquisition of commercial-off-the-shelf (COTS) or non-developmental item (NDI) software. SCEs should be performed when NDI software modifications to satisfy Air Force requirements or "*glue code*" development to link COTS packages will cost more than $10 million.

SCE Appraisal

The source selection related SCE appraisal should investigate/cover, as a minimum, all Key Process Areas (KPA) and goals for the Repeatable and Defined maturity levels described in the CMM.[SM] The appraisal should not tailor the KPAs or goals specified in the CMM.[SM] The SCE appraisal team may be independent but is part of the source selection team. The SCE appraisal should be conducted as soon as possible after a competitive range decision is made and discussions with contractors are authorized. Normally, each SCE includes a 4-5 day site visit, in addition to preparation and wrap-up time for each site visit for a total of up to ten (10) days.

The SCE appraisal should be conducted by a team trained in the CMM[SM] version to be used and comprised of members from the Air Force and/or an Air Force approved independent organization. If a contractor is used to conduct the SCE appraisal, FAR Subpart 9.5, Organizational and Consultant Conflicts of Interest, shall be adhered to. In selecting programs for review, the priority should be programs currently being worked, or recently completed, by the Offeror rather than programs most similar to the acquisition. The programs must be from the same organization. The reviewed programs should be approximately the same size, from the same development site, and from the same broad domain. Offeror must be put on notice in the Request For Proposal (RFP) that process documentation from selected programs must be available at the unclassified level.

An Offeror's software engineering/development practices should be considered validated if: (1) a written and approved procedure for a practice exists; (2) the procedure implementation is effective for the organization; (3) evidence exists showing that procedures are followed; (4) evidence exists that training for the procedure is planned, funded, scheduled, required, and accomplished in a timely manner; and (5) the procedure has been institutionalized. An institutionalized practice or procedure is one that has been in place and practiced for greater than 12 months. Practices and procedures less than six months old may be considered a process improvement activity. Practices and procedures in place and practiced for less than 12 months but longer than 6 months may, by SCE team consensus, be considered institutionalized.

There are three components of the CMM[SM] reference model that can be rated: goals, KPAs, and maturity level. SCE results are documented as KPA findings of general observations, strengths, weaknesses, and process improvement activities. A *strength* is a particular part of the software process capability that is sufficiently robust to mitigate the development risk due to software process. A *weakness* is a particular part of the software process that has characteristics that increase the risk due to software process. A *process improvement activity* is a practice or procedure that is not yet institutionalized and indicates potential mitigation of risk due to software process. Maturity level ratings are optional since the rating itself provides minimal visibility into the state of an appraised contractor's software process. All findings are determined by team consensus.

- **Goal.** A goal is satisfied when the associated findings indicate that the goal is implemented, as defined in the CMM,[SM] with no significant weaknesses or that an adequate alternative exists, and is institutionalized, as defined in paragraph 4.5 above.
- **KPA.** A KPA is satisfied when all goals for that KPA have been investigated/covered and rated as satisfied. A KPA is weak if one or more goals for it are not satisfied. A KPA is assigned "Not Rated" if any of the goals for the KPA are not investigated/covered.
- **Maturity Level.** A maturity level is achieved when all KPAs for that level and all of the levels below it have been investigated/covered and rated as satisfied.

APPENDIX M Software Source Selection

At the conclusion of each SCE, a *findings exit brief* should be provided at the site. The exit brief is to provide a courtesy one way information flow of the draft findings to the Offeror before the SCE team leaves the site, and allow the Offeror to provide information and/or artifacts that may have been overlooked. The exit brief should include the PCO name, address, contact method, and instructions for the Offeror to respond or comment on the draft findings presented by the SCE team. The RFP should explain the exit briefing rules.

Source Selection Evaluation of SCE Results

Normally, the SCE will be designated as a factor in the management area. The source selection evaluation team should evaluate the SCE results using the following procedures.

- The SCE shall be a significant factor. Do not include in the SCE factor any augmentation elements. Any augmentation elements should be evaluated in a separate factor.
- In addition to affecting the SCE factor rating, the SCE results may affect proposal risk assessments relevant to other factors.
- Evaluation of SCE subfactors should be standardized. When converting the SCE subfactors into a factor color, the following criteria should be used:
  - **BLUE**. There are no weak KPAs at the Repeatable and Defined maturity levels.
  - **GREEN**. There are no weak KPAs at the Repeatable maturity level *and* four (4) or less weak KPAs at the Defined maturity level.
  - **YELLOW**. No KPAs at the Repeatable maturity level are rated weak *and* five (5) to seven (7) KPAs at the Defined maturity level are rated weak; or
    - -- One (1) KPA at the Repeatable level is rated weak *and* six (6) or less KPAs the Defined level are rated weak; or
    - -- Two (2) KPAs at the Repeatable level are rated weak *and* five (5) or less KPAs at the Defined level are rated weak; or
    - -- Three (3) KPAs at the Repeatable level are rated weak *and* four (4) or less KPAs at the Defined level are rated weak.
  - **RED**. There are four (4) or more weak KPAs at the Repeatable maturity level *or* eight (8) or more weak KPAs in the Repeatable and Defined maturity levels.
    - When multiple SCEs for an Offeror — proposed subcontractor team are conducted, the following standards should be used to determine the team's SCE factor color rating and proposal risk.
    - Where the proposed prime's color rating is lower than one or more of its proposed subcontractors, the team's color rating should always reflect the rating of the proposed prime.
    - When the team members each use their own processes, the lowest color rating among them should determine the proposal's color rating.
    - If a higher color rated proposed prime imposes its higher level processes on the proposed subcontractor(s), the higher color rating may be used, with proposal risk being assessed to indicate that one or more lower rated proposed subcontractors have never used the process.
    - Where the team members are equally color rated and they will integrate their processes, the color rating should be the color rating of the individual team members and the proposal risk should be other than LOW.

Post Contract Award

Contracts should be structured to allow the performance of one or more SCEs subsequent to contract award to assure the evaluated level is maintained and/or to verify progress against Software Process Improvement Plans (SPIPs).

EVALUATING Ada EXPERIENCE DURING SCE

Objective. This provides an outline of the issues that should be addressed when assessing a contractor's ability to develop programs in Ada during a Software Capability Evaluation (SCE).

Background. A SCE can provide a snapshot of a contractor's past process implementation, current process activities, and futures process potential. SCE's are based on the Software Engineering Institute (SEI) Capability Maturity Model[SM] (CMM[SM]) assessments. The SEI CMM[SM] Version 1.1 is a good starting point for assessing the capability of a contractor. This can provide the basis for a similar assessment of a language capability. Once the language is known (be it Ada, Fortran, or C), we can ask the contractor a new set of questions focused on that language. Most of the issues are not specific to any particular language, e.g., Ada. The approach to take is to identify the issues needed to access the contractor's capabilities in a given language and then to fill in the details when the language is Ada.

Proposal. There are six SEI CMM[SM] Key Process Areas (KPAs) that can be tailored to include assessment of the contractor's Ada capability. The first two KPA's, Software Tracking and Oversight, Software Quality Assurance (SQA), can be used to get a snapshot of how a contractor performs on Ada Programs. The following are those KPA's and some issues that can be addressed during the evaluation.

KPA — Training Program. The SCE team could first learn about the contractor's Ada capabilities from reviewing their training program. The training plans should show what type Ada training is planned. For example, is there Ada training for other than programmers (e.g., program managers, SQA, and SCM personnel). The employee's training records should reveal who has been trained and was the training taken at the appropriate time (i.e., prior to the start of working on an Ada program).

KPA — Software Program Planning. This KPA allows the SCE team to see how a contractor develops estimates for schedules, manpower, facilities, and sizing (lines-of-code). When reviewing the contractor's estimating methodologies and procedures, the team would look for an Ada influence. The contractor's management should show an understanding of the Ada language and its use. The people selected for programs should have used the language within the domain of their program. Domain experience is more important than language experience. If the language used in previous programs is similar, then this should be an advantage. While reviewing the contractor's program plans, the team can see if the selected hardware meets the need of the developers. Does the contractor plan for adequate file server processing capacity and disk storage to support the program? Ada tends to need more computer cycles and disk storage than other languages. Finally, the management must demonstrate a commitment to doing what is necessary to make the program a success (e.g., additional training, software tools, hardware, etc.).

KPA — Software Program Tracking and Oversight. Once the team has seen how the contractor plans for Ada programs, then they should see how the plans work on those programs. The tracking metrics should be tailored for the use of Ada. The data collected from these metrics should be reported to management. The management should show a commitment to taking any corrective actions necessary based on the results from the metrics.

KPA — Organization Process Definition. This KPA's force is on the company standards and procedures. The SCE team would be looking for Ada programming standards and procedures. These standards should be up to date and easily available to all programmers. The Unit development folders should show signs that the standards are being used.

KPA — Software Product Engineering. Under the Software Product Engineering KPA, the SCE team should be looking at how the software environment is set up for building the Ada software. For example, are the following tools used?

- CASE tools,
- Configuration management tools,
- Compilers,

- Integration tools, and
- Code generators that support Ada development.

These tools should be integrated with the contractor's overall software development methodology and software development process. It is also important for the contractor to have experience with these tools.

KPA — Software Quality Assurance (SQA). The final KPA that can be used by the SCE team in assessing a contractor's Ada capability is the SQA KPA. Under this KPA, the SCE team should review the SQA procedures to see if any are covering the company Ada standards. The SQA training records should be reviewed for some type of Ada training.

Summary. The above KPAs and related issues are just a starting point for a SCE team to use in assessing a contractor's Ada capabilities. Additional information may be learned under the other KPAs not listed here. To better help a SCE team in their review, the following sample questions related to the above listed KPAs has been developed.

SCE Key Process Areas (KPAs) for Ada Evaluation

The following lists some of the Key Process Areas (KPAs) from the Software Engineering Institute (SEI) Capability Maturity Model[SM] (CMM[SM]) and questions for a Software Capability Evaluation team to use in assessing a contractor's Ada capabilities. In addition are some non-SEI CMM[SM] KPAs and questions that should also be considered.

KPA Questions

Training program. Does the training planning include Ada training and is the training provided? Is there Ada training for other than programmers (e.g., program manager, SQA, and SCM personnel)? Is there any on-the-job training? Are experienced programmers assigned to work with the under experienced programmers? Is follow-up training provided? When are the people trained? Have they taken all required training prior to being assigned to a program using Ada? Are they encouraged and do they take additional or follow-up training?

Software program. Does the contractor's estimating methodologies and procedures for schedules, manpower, and sizing have an Ada influence?

Planning. Are the people planned for the program those who have used the language within the domain of the program? Have they used the proposed tools? How well does the management understand the language and its use? Is the management committed to doing what is necessary to make the program a success? Does the detailed software development process support the contractor's management techniques? Is there adequate hardware available to meet the needs of the developers? Does each developer have a workstation? Is there adequate file server processing capacity and disk storage to support the team?

Software program tracking and oversight. Are the tracking metrics tailored for the use of Ada? Does management review the metrics and are corrective actions taken?

Organization process definition. Are there company standards and procedures for Ada? Are they tailored for each Ada program? Are they used by the programmers? Are they reviewed on a regular basis and updated as needed? Do the Unit Development Folders show signs the standards are being used?

Software product engineering. Are the following tools used?

- CASE tools,
- Configuration management tools,
- Compilers,
- Integration tools, and
- Code generators that support Ada development.

How are these tools integrated with the contractor's overall software development methodology and software development process? What experience does the contractor have with these tools?

Software quality assurance. Review SQA procedures for any covering the company Ada standards. The SQA training records should be reviewed for some type of Ada training.

Non-KPA Questions

Reuse. Do they have a reuse component in their process? Does it support the language being used (e.g., Ada)? Do they have and use a corporate reuse library? How is reuse coupled back to the development process? How are reusable components tested and validated?

COTS. Do they have experience in integrating COTS products in general and with products they are using on this program? Do they have experience integrating COTS products written in other languages with the program's language (e.g., Ada)?

SUBPROCESS AREA SELECTION TABLES

The tables in this appendix are provided as an aid to help SCE teams select critical subprocess areas during Step 5. The tables were created by the SCE program members at the SEI for guidance only. SCE teams are expected to use their experience and judgement to select critical subprocess areas based on the requirements of the particular development. Factors considered in selecting critical subprocess areas are the following:

- What processes would an organization need to manage the aspects of the program which are new to the organization?
- If the product being developed is new to the end user, what processes will the development organization need to manage the anticipated requirements changes?
- What are the basic processes that a development organization would need for any software development effort?

How to Read the Tables in this Section

This appendix contains a table for each key process area (KPA) in the *Repeatable* and *Defined* levels. The tables contain the following columns.

- **Subprocess areas column**. Each row under this column corresponds to a subprocess area associated with the KPA. Some of the subprocess areas contain other subprocess areas. These "*higher-level*" subprocess areas are indicated by boldface type.[1]
- **Major attributes columns (ApD, Pt, Ps, Tw, and Sub).** An "*X*" in the column for an attribute indicates that the subprocess area listed in that row may be important to the development organization for managing the risk associated with a lack of experience relative to that attribute. These columns correspond to the five major attributes from the Experience Table created in Step 4. The Experience Table shows where any of the development organizations may lack experience with regard to some attribute of the new program.
- **Operational precedence (Op) column.** An "*X*" in this column indicates that the subprocess area listed in that row may be important for managing the level of requirements changes which may be anticipated if end users do not have experience with similar products. The Op column corresponds to the *operational precedence* attribute from the Target Product Profile developed by the sponsor. This attribute indicates the degree to which the product being developed may be new to the end user.
- **Nucleus capability (*) column**. An "*X*" in this column indicates that the subprocess area listed in that row is part of the recommended *nucleus capability*. Nucleus capability refers to a basic set of subprocesses which are needed for almost any software development.

Repeatable Level Key Process Areas (KPAs)

Key to Abbreviations:

ApD Application Domain
Tw Type of Work
Op Operational Precedence
Pt Product Type
Sub Subcontracting
***** Nucleus Capability
Ps Product Size

Repeatable Level Key Process Area: Program Management

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
General Management Functions							
Committed management process	X	X	X			X	X
Compliance to organizational standards							
Taking corrective action; issue/action item tracking			X				X
Review and oversight: oversight by senior management and management reviews							X
Tracking; actual vs. estimate comparison; commitment evidenced by reviews of compliance							X
Customer interface	X					X	
Usage and collection of performance data							X

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Integrated Software Management							
Risk management; recognition of risk events; cost, software technology, resources, and schedule	X	X	X			X	
Tailoring and selection of project process and its support environment							
Maintenance of process performance database							
Coordination between project groups	X	X	X			X	

APPENDIX M Software Source Selection

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Requirements Management	X	X				X	
Requirements allocation							
Requirements change							X
Requirements implication evaluation			X				
Matching software architecture to requirements; transforming requirements into top-level design							

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Subcontracting							
Subcontractor selection							
Contracting: subcontract process							
Coordination of work with subcontractor							
Subcontractor monitoring							

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Testing							
Preparing to carry out testing; test procedures							
Carrying out test operations							
Reviewing test scenarios, testbeds, and test cases							
Regression testing							X

APPENDIX M Software Source Selection

Repeatable Level Key Process Area: Program Planning

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Project Planning							
Size estimation: software development resources, costs, and critical target and host computer resources; the scope of work and effort has a basis in reality	X	X	X			X	X
Cost estimation; cost has a documented correspondence to estimate size and schedule; software responsibility, software engineering technical direction	X	X	X			X	X
Planning: resource planning and management for project's software size, cost, and schedule, software development plan, the software life cycle model, planning schdedules, software schedules							X
Commitment process during change	X	X	X			X	X
Project manager's participation with the project proposal team	X	X	X			X	
Usage of software process database							
Integration of technical direction, engineering tools and methods into planning process, engineering and technical reviews of plans	X	X	X			X	
Product capacity tracking, critical target computer resources							

Repeatable Level Key Process Area: Configuration Management

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Configuration Management							
Status report, monitoring, configuration responsibility							X
Change control process, standard forms for reporting errors			X				X
SCM plan; baselining of software engineering products and process specifications; a configuration management repository for the software baselinesz; software baseline audits	X	X					X
Release of software baseline products							
Library support system			X				
Configuration control board							

Repeatable Level Key Process Area: Software Quality Assurance

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Software Quality Assurance							
Auditing: SQA objective evidence of audits							X
Noncompliance resolution	X	X	X			X	X
Reporting chain: SQA group reports, independent authority							X
SQA plan							X
SQA concurrence on milestone progress	X	X					
SQA group participation							
Oversight for all process support systems; e.g., corrective action system; data collection of defects; earned value of system deviation handling	X	X					

APPENDIX M Software Source Selection

Defined Level Key Process Area: Software Engineering Process Group

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Software Engineering Process Group							
Assignment of full-time resources, establishing and supporting							X
Coordination of review with senior project technical staff, analysis, and evaluation of software process definition, responsibility assignment	X	X	X			X	X
Planning systems and software process improvement; review of existing and proposed process standards							
Defining training requirements	X	X				X	

Defined Level Key Process Area: Standards and Procedures

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Standards and Procedures							
Planning standard software process development	X	X	X			X	
Implementing standard software process development							
Process assets; a process library system; library of software process specifications; software process database maintenance; tailoring the organization's standard software process	X	X	X			X	
Standards for software development folders							X
Review standards							
Human-machine interface standards							

Defined Level Key Process Area: Software Product Engineering

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Software Product Engineering							
Integrating the project's process with the software architecture: process change and technology transition review	X	X				X	X
Investigating software engineering tools and methods; tool selection and use with gathering of performance data							
Developing and maintaining the project's software architecture							
Reviewing the system/software testing							
New technologies		X				X	

Defined Level Key Process Area: Training

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Training							
Planning/procuring training courses for training curriculum, courses	X	X	X			X	X
Job analysis to support each project's training needs							
Communicating and keeping track of delivered training; schedules for all professional and technical staff; records of training							X
Delivering training; management support							
The organization's training program; training requirements							

APPENDIX M Software Source Selection

Defined Level Key Process Area: Peer Reviews

SUBPROCESS AREAS	MAJOR ATTRIBUTES						
	ApD	Pt	Ps	Tw	Sub	Op	*
Peer Reviews							
Planning/assigning peer reviews; technical review	X	X	X			X	X
Schedule, process for technical reviews							
Conducting peer reviews							X
Review assignments							
Peer review performance; organizational database of review activities; cost; peer review result handling							

Notes

1. Most of these became KPAs in the Capability Maturity Model℠ (CMM℠) Version 1.1 [Paulk 93a], and were established in anticipation of that version of the CMM℠. Some of the subprocess areas distinguished in this manner are at the wrong maturity level relative to CMM℠ Version 1.1; however, this does not affect how an SCE is conducted, because maturity level scores are not calculated. It does alter the category the findings are reported under, because findings are consolidated by KPA.
2. The abbreviation Ps stands for "*Product Size*." Product Size refers to the "*Size*" attribute.

SCE TEXT FOR INCLUSION IN INSTRUCTIONS TO OFFERORS

Section L

The following sample text illustrates how SCEs might be inserted within Section L or M of the RFP. These examples assume the SCE will be used as a specific criterion for source selection.

Sample 1

Software Engineering Capability. The Government will evaluate the software process by reviewing the offeror's Software Process Improvement Plan and by using the Software Engineering Institute (SEI) developed technique, the Software Capability Evaluation. The Government will determine the software process capability by investigating the offeror's current strengths and weaknesses in key process areas defined in the SEI report CMU/SEI-TR-11 "*Characterizing the Software Process: A Maturity Frame-work.*" The Government will perform an SCE of each offeror by reviewing current programs at the site proposed on this contract. The evaluation will be an organizational composite. It will be substantiated through individual interviews and reviews of documentation, of the offeror's strengths and weaknesses in key process areas relative to maturity level three; i.e., the extent to which an offeror meets or exceeds maturity level three criteria. The on-site evaluators may be separate and distinct from

the proposal evaluation team and may include a government contracting representative. The evaluators will have been trained and experienced in conducting SCEs.

SCE Text for Inclusion in Instructions for Preparation of Proposals (IFPP)

NOTE: Instructions for Preparation of Proposals provide guidance to offerors as to how they should prepare their proposal. The following text requests the offeror to provide program profiles, organization charts, sample documentation, and a software process improvement plan. It also requests the offeror to provide the SCE team with facilities during the site visit.

The technical proposal shall include the offeror's response to the software evaluation process. The offeror shall provide the following information to assist the Government's preparation for the Software Capability Evaluation of each offeror:

1. The offeror shall complete the Program Profile form for 7-9 major software engineering development programs. All programs should be drawn from the same site and organization (e.g., profit center) bidding on this solicitation. One of these programs must include the (proposed) software development effort and the others should be programs that are near completion or completed within the last three years. These programs should be as similar as possible in scope and magnitude to the (proposed) effort. The programs should be from programs where the offeror was the prime contractor, at least one program should include a development where another subcontractor developed portions of the software, and as least one program should be an Ada program, more if applicable. Program Profiles from Special Access Programs are discouraged. For offerors with fewer than 7 programs at the bidding site, submit information for as many programs as are available.
2. Section C, Tab 1, contains the questionnaire outline and report form that should be used to generate the evaluation profiles for each of the programs. Respond to the SEI questions with a Yes or No answer. For each "**yes**" response, please note the mechanism or document for justifying the response on a separate form.
3. The offeror shall provide program-level and higher-level organization charts. The organization charts should contain individual's names and job titles and indicate how the programs above are related to each other. If there are departments that the software programs rely on, these too should be positioned on the organization chart (e.g., training, Software Engineering Process Group, quality assurance, configuration management, standards, policy and procedures).
4. The offeror shall provide a draft Software Development Plan (SDP) and a Software Standards and Procedure Manual (SSPM). If there are "*generic*" SDPs and SSPMs those are preferred; otherwise, select a sample SDP and SSPM from the program that has the most representative SDP.
5. The offeror shall submit their site's Software Process Improvement Plan, in the form of their choosing, with their proposal. The document shall be no longer than 15 pages. The Software Process Improvement Plan shall be detailed enough for the offeror to communicate their current software process capability, specific planned improvements, dedicated resources, effort estimates, and a time phasing of those improvements to bring the offeror's software process maturity to the organization's desired maturity level.
6. After the proposal is received, the Government will coordinate a site visit with the offeror to discuss the questionnaire responses and conduct the Software Capability Evaluation (SCE) at the offeror's location. The offeror shall provide a point of contact and phone number for the coordination of all SCE activities. So that the site visit will go smoothly, the Government will list details about the site visit during the coordination process; e.g., interview schedules, documentation requests, facilities for the evaluation team. The offeror shall be notified

approximately two working days prior to the site visit of the programs to be examined. The site visit dates selected by the Government are not open for discussion.

7. During the site visit, the SCE team will need a secure meeting room capable of accommodating at least eight people. The offeror shall have a copy of the organization's software standards, procedures and/or operating instructions, and organizational charts for the programs being reviewed in the meeting room when the SCE team arrives. All interviews conducted as part of the SCE shall be done in private, one individual at a time. The SCE team may be separate and distinct from the proposal evaluation team.

8. If security authorization is necessary for the members of the evaluation team, a Fax number and telephone number of the contractor's security office should be provided along with a list of any other pertinent information required to obtain security approval.

TAB 3

Sample RFP Preparation Checklists

SAMPLE QUESTIONNAIRE FOR SITE VISIT PREPARATION

The following questions are examples of what you should consider as you develop your site visit checklist. *[SOURCE: Yourdon, Edward, Decline and Fall of the American Programmer, Yourdon Press, Englewood Cliffs, New Jersey, 1992].*

- Does this company *care* about software quality? Does it care enough, for example, to delay putting a new system into production because its software reliability models indicate an unacceptable number of latent errors? Does it have software reliability models?
- Does this company care about its people? Has it invested time and money to train its software development managers to do a better job in hiring people? Does it invest an adequate amount of time training its technicians, or does it assume that its software engineers are replaceable commodities? Does it use modern "performance management" methods to ensure that its corporate goals are aligned with personal consequences of those goals? Do the people in the organization understand what the organizational goals are, and how they are supposed to fit into those goals?
- Does this company use modern programming tools, languages and methodologies, as opposed to assembly language and the waterfall life cycle.
- Does this company measure everything it does in the software arena? Does it measure the *process* of software development as well as rate the final *product*? Does it have a separate software metrics group? Are size, effort, schedule, defects, and rework measured routinely? Are the metrics used in a positive way, so that everyone in the organization can see how they improve?
- Does this company support the concept of software reusability? More important, does it provide some incentive (for example, cash royalties) to its software engineers to create reusable components? Has it considered a separate "*Software Parts Department*" whose *only* job is to create reusable components? Does it estimate the degree of expected reusability at the *beginning* of programs and base its schedules and resource requirements on that estimate?

- Does this company have CASE tools? Does it believe that CASE tools are like toothbrushes, that is, they're not meant to be shared? Does it provide an adequately equipped PC or workstation for everyone?

PROGRAM PROFILE OUTLINE

The following outline is a sample program profile that can be referenced in the RFP. Six to nine of these forms, for different programs, should be filled out by each contractor.

- **Program Name:** (name of program listed on the contract)
- **Program Number:** (unique identifying number on the contract)
- **Program Type:** (e.g., scientific, human-machine, business, control, support software)
- **Customer:** (the agency that procured the software and a point of contact within that agency)
- **Subcontractors/Prime Contractors:** (list any subcontractors employed on the program or list the prime contractor if the offeror was a subcontractor)
- **Current Phase:** (identify the current phase of the software development process; e.g., requirements definition, detailed design, code & unit test, integration test, maintenance)
- **Location:** (primary site of the software development effort)
- **Start Date:** (starting date of the contract)
- **Design Completion Date:** (estimated or actual)
- **Code Completion Date:** (estimated or actual)
- **End Date:** (contract completion date)
- **Team Size:** (peak man-month period and average man-years over the contract period)
- **Estimated KSLOC and Function Points:** (estimated/actual thousand source-lines-of-code (KSLOC)) and function points.
- **Programming Languages:** (percentage of KSLOC in languages (e.g., Ada, FORTRAN, Pascal, C, Assembly))
- **Target Hardware System:** (computer on which software executes)
- **Development Hardware System:** (host computer for the compiler and support environment)
- **Applicable Standards:** (e.g., MIL-STD-498)
- **Cost:** (actual/estimated dollars spent to date/completion)
- **SEI Questionnaire:** (the attached questionnaire and its answer sheet should be completed for each of the programs)
- **Organization Chart:** (Most recent organization chart for each program with titles and individual names. This chart should identify the individual responsible for the following activities: program management, system engineering, software program management, software engineering, software quality assurance, software configuration management, subcontractor control, simulation, integration and testing and other technical software activities.)

PROPOSAL EVALUATION CHECKLIST

The following checklist is provided as an aid for software development proposal evaluation.

Program Management

- What was the software manager's involvement with the proposal?
- How is software progress tracked? Management reviews? Frequency?
- Who will approve software schedules? Cost estimates?
- How are issues raised, tracked, and conflicts resolved?
- What will be the software manager's reporting chain?
- How does the software requirements team relate to the software design team?
- How much manager visibility into integration and test will be necessary?
- What will be the relationship between the System Engineer and Software? How will tradeoffs be made?
- Is senior management briefed regularly on software status?

Subcontractor Management

- What is the subcontractors' development process?
- How will qualified software subcontractors be selected?
- Do the subcontractor's standards, procedures, process comply with the prime contractors'?
- How should the results and performance to commitments be tracked?
- Is the subcontract manager knowledgeable of and trained in the software?
- Are there periodic technical reviews & interchanges with subcontractor?
- Does the prime's Software Quality Assessment and Configuration Management monitor sub's SQA & CM?
- Do the prime's senior management review the status of the subcontractor regularly?

Metrics Management

- Is design progress, test progress and staffing measured?
- Is integration progress measured?
- Is software size overtime and memory utilization measured?
- Is throughput and I/O channel utilization measured?
- Is progress tracked and reported to the PM regularly?
- Are technical, schedule, cost, and resources plans prepared?
- How are software size, cost and schedules established? How are document procedures established?
- Document Commitments: Who commits, size, cost and schedules?
- Are there policy exits for resource planning and commitments?
- Are the software managers trained on software estimation?
- Are actual versus planned estimates recorded and compared?
- Is there a central estimation manager and data base for accuracy?

Software Quality Assurance Management

- Is there an independent reporting chain?
- Are audits conducted at all phases of life cycle and line activities?
- How is it ensured that audits are representative?
- Does SQA have adequate resources?
- Does SQA audit subcontractors?
- Are deviations handled according to documented procedures?

- Does senior management review SQA activities regularly?
- Is SQA authority and concurrence required?

Configuration Management

- How can requirements, design, and code changes be controlled?
- How can interface changes be controlled?
- Is there traceability for requirements, design and code?
- Is there a tool to help control versions and builds?
- Are parameters established for regression testing?
- Are baselines established for tools, change log, and modules?
- Does the CM plan include staff, schedule, response, resources, tools, and facilities
- Does the library system store work products and prevent unauthorized change?
- Does the document change request process include check in/out, review and regular testing?
- Is there a document Change Control Board and a change proposal process?
- Is there a change log that tracks open/closed change requests?

Peer Reviews Management

- Are design, code, and test case peer reviews conducted?
- Who and how many people attend?
- Are documented procedures and checklists used?
- Are the peer reviews included in the Software Development Plan and are they published?
- Are statistics compiled on the type, severity, and location of errors?
- Are statistics compiled on the time to prepare, review, and correct elected errors?
- How are errors tracked to closure?
- Does SQA audit peer review activities?

Training

- How are CM and Quality Assurance leaders trained?
- Are moderators and developers included in peer reviews?
- Do program managers participate in software estimation and peer reviews?
- Do software supervisors participate in QA, CM, estimation and peer reviews?
- Do software developers participate in peer reviews, software development process and tools?
- Do training resources include money, facilities, tools and schedules?
- Is there a corporate training policy supported by a training manual?
- Are program training needs identified and planned?
- Are job functions mapped to training?
- Do training records include people and courses?

Standards Management

- Do standards include coding, unit development folders, and man-machine interface standards?
- Do standards include generic SDP, a QA plan and a CM plan?
- How are standards enforced?
- How and when are standards updated?
- What is the assigned response for updating standards and policy?

TAB 4

Sample Paragraphs for RFP Inclusion

SOFTWARE QUALITY REQUIREMENT

Software quality requirements will be specified for the program. The development of these requirements shall be the responsibility of the program office. The program office will work together with the end-user of the system to generate requirements based on an analysis of the system requirements, life expectancy, development costs and user concerns. Example user concerns to consider are performance (e.g. reliability, usability and efficiency), design architecture (e.g. maintainability and correctness) and re-engineering (e.g. reusability, interoperability and portability). Software quality requirements will be specified and documented within the baselined Software Requirements Specification (SRS). A hierarchical quality model of quality factors, criteria and metrics will be used to predict software quality. Factors representing the user's concerns will be decomposed (using relevant standards and guidebooks) into software oriented characteristics. Measures of these characteristics (i.e. metrics) will also be defined. The specified model will apply to all software development phases and products. Quality progress will be reported and reviewed at each major program milestone. All open and closed software quality problems will be tracked and reported. The achievement of software quality requirements will be demonstrated, using industry accepted measures of operational quality (e.g. reliability = mean-time-to-failure), during integration testing. Failures will be categorized according to an Government approved some severity standard.

SOFTWARE TESTING REQUIREMENT

In addition to functional testing of the software to assure compliance with requirements, the software will be tested such that 100% of the software branches (i.e., decision to decision statements) are exercised prior to release in the field. Reasons for not achieving 100% execution coverage must be formally documented in the Software Test Report.

Software tools (i.e., test coverage analyzers) to automate the branch testing process are available. Intrusive analyzers insert software code into the software under development to capture and record the execution coverage and are appropriate for non-real-time software developments. If a software product under development must operate in real-time, if it is highly memory constrained, or if the software units are very large, non-intrusive analyzers should be used. Non-intrusive analyzers use a separate hardware processor to capture and record this same execution coverage information.

SOFTWARE LIFE CYCLE DEVELOPMENT AND SUPPORT ENVIRONMENT REQUIREMENT

An automated computer-based software life cycle development and support environment will be used by the contractor. Development of the environment's requirements shall be the responsibility of the program office. The environment should provide the following capabilities: 1) specification of the life cycle software development process and the monitoring/ enforcement of that process, 2) integration of Computer-aided Software Engineering (CASE) and other tools supporting the various interphase activities of the life cycle, and 3) interphase support including program management, configuration management and baselining, document/ specification generation, traceability and change impact analysis.

SOFTWARE LIFE CYCLE DEVELOPMENT TECHNOLOGY SCALABILITY REQUIREMENT

An automated, computer-based software life cycle development and support environment will be used by the contractor. Development of the environment's requirements shall be the responsibility of the program office. The ability of the environment's hardware/ software complex (including each of its associated CASE tools) to adequately and efficiently support the breadth of software under development (i.e., scalability to the size of the problem) will be a primary consideration.

REUSABLE SOFTWARE REQUIREMENT

As part of the SDP, reuse software engineering and planning shall be addressed. The SDP shall contain a WBS that includes the establishment and implementation of a reuse program. Reuse shall be an integral part of software development planning, review, audit and reporting. As part of the contractor's SEE, a Software Reuse Library shall be established and maintained after appropriate review and approval by the Government.

TAB 5

Source Selection for Software Supportability

INSTRUCTIONS TO OFFERORS (SECTION L)

In addition to specifying proposal form and content, the Instructions to Offerors should require submission of a Software Development Plan and Software Quality Program Plan as part of the proposal. The SDP will include the offeror's software development and management concepts, procedures, and metrics for controlling and assessing progress during the development process.

Supportability Issues

The following supportability issues must be covered in the Instructions to Offerors:

- The methodology used to perform software sizing and cost estimating and the approach to be followed during software development
- The rationale used for computer resource timing and sizing estimates and description of how spare I/O utilization (channels or data rates), CPU throughput utilization, memory utilization requirements will be met;
- A description of any teaming and subcontractor arrangements;
- The skill levels required for computer resources development and their availability within the corporate structure;
- The method to be used for risk control;
- Any planned use of firmware;
- Any plans for reusing or modifying existing software;
- A clear definition of all assumptions used during proposal preparation;
- Plans for the development of prototype software;
- Plans and procedures for generating and using software metrics.
- A disclosure statement of defect removal efficiency. This should include their definition of defects and what defects are included in the metric and the method of calculating the metric.

Additional MIL-STD-498 Considerations

- The offeror should address the manner in which they will comply with their Requirements for Software Standards, how this will be achieved and how such compliance will be measured. The offeror should describe proposed software development methodologies to be incorporated in any resultant contract.
- The offeror should document the manner in which compliance with Category and Priority Classifications for Problem Reporting, will be achieved and describe the problem reporting system to be used in any resultant contract.
- The offeror's proposal to the items above should be part of the technical volume of the proposal and not be required as part of the contract.
- The offeror should document the manner in which compliance with Evaluation Criteria will be achieved in the software development effort if the offeror is awarded the contract. These

include: internal consistency; understandability; traceability to indicated documents; consistency with indicated documents; appropriateness of analysis, design, and coding techniques used; appropriateness of allocation of sizing and timing resources; adequacy of test coverage of requirements. *[NOTE: The offeror may propose, subject to government approval, additional criteria or alternate definitions for any of the criteria.]*

- The offeror shall also provide examples of software documentation (e.g., software specifications, source code listings, software test reports) prepared on other software development efforts (the Government's source selection team can then evaluate the supportability of the proposed documentation.) The offeror should describe the process, techniques, methodologies, and metrics to be used and define acceptable (i.e., pass) criteria (minimum, range, or maximum) for each proposed evaluation criteria test environment (STE) (including tools therein) (see definitions below) proposed to develop software for the system. The offeror should also describe the environment proposed to be delivered to (assuming the contract requires such delivery) or to be used by the Government to support the system's software. *[The offeror should be required to submit metrics on this issue to help government evaluators determine the quality of the environment proposed for delivery to the Government.]*
- Offeror should document the factory software engineering environment (SEE), including tools therein. The plans should address how the offeror will evolve the factory environment into the supporting environment. This should not include the concept of developing a separate support environment. The evolution should include the constant updating and refining of the factory environment to meet all needs of the supporters and then be transitioned to the supporting/maintaining organization.
- The plans should also describe how the offeror will install the support environment at the supporting/maintaining organization, load the environment with all program software/data and hardware (e.g., operational software/data, all development/ test tools, hardware configurations, master engineering data repository, and administrative practices to be used for software support) and use the environment as the only source of information/tools to support the initial operational test and evaluation (IOT&E), as well as initial block changes to the system (while under interim contractor support).
- The plans should describe any differences in tools between the factory environment and that envisioned for the software support activity and plans to ensure that tools differences will not adversely impact the supportability of the software. *[NOTE: If too much documentation is required for submission to the Government, it may exceed page count restrictions.]*
- The offeror should document the approach to be used in evaluating the quality of software and software development processes; i.e., how the offeror will comply with proposed evaluation criteria during the period of the contract. In addition, the offeror shall identify, explain (with rationale), and provide pass (as in pass/fail) criteria for each process and product metric used. This document shall contain a step-by-step sequence of quality-related activities to include the data collection process, scoring algorithms, reporting, and corrective action.

Additional AFSCP/AFLCP 800-45, *Software Risk Abatement,* Considerations

The offeror should document the approach to be used in managing risk in developing software and integrating it in the system. The offeror should be required to quantify performance, support, cost, and schedule risk factors (this should be part of the offeror's Software Development Plan).

APPENDIX M Software Source Selection

Sample Section L

The following information is useful for developing Instructions to Offerors (ITO) (as related to software supportability concerns):

1. Submit Volume XXX and completed questionnaires from A Method for Assessing the Software Engineering Capability two weeks prior to submission of Volumes XXX.

2. Volume I. TECHNICAL

Volume I shall describe the complete proposed Reliability and Maintainability Plan and engineering programs and shall not exceed xxx pages. Volume I shall be divided into two books, marked and placed in separate three-ring or spiral binders. Each book shall be arranged as described below.

2.2. Volume I, Book II. Engineering Program and Design

Book II shall detail the proposed engineering program. As a minimum, the following information shall be included.

2.2.1. Describe the overall engineering development and design program including major activities and an integrated schedule.

2.2.1.1. Identify the overall engineering development schedule and specific integration program activities such as design requirements analysis, testing, software development, support equipment development, and management processes for controlling the development effort.

2.2.1.2. Provide an overall technical description of the total program. Identify significant benefits of design features proposed including commonality considerations among subsystems (including support equipment, maintenance trainers, and aircrew training devices), between aircraft types, and between aircraft mission design series. Include, as a minimum:

2.2.1.2.1. Software design, development, and integration efforts for each subsystem. Include the top-level description of each computer software configuration item (CSCI), identify and justify the computer languages used, and estimate the size of each CSCI. Identify if the CSCI currently exists, will be modified, or will be developed.

2.2.1.2.2. Describe the overall built-in-test (BIT) approach for each subsystem and how it will test subsystem and subsystem-to-aircraft interfaces.

2.2.1.3. Define the draft subsystem specification and development plan for the major subsystems. The specifications shall be sufficiently detailed, as a minimum, to include descriptions of:

2.2.1.3.1. Growth potential of each LRU with respect to the number of circuit card assembly (CCA) expansion slots available and the type of functional enhancements (such as additional memory, processor, or input/output CCAs).

2.2.1.3.2. Significant components at the SRU level, such as embedded computers and memory devices. Identify and justify the intended processors to be used, estimated lines of code, throughput, memory, and growth capacity requirements.

2.2.1.3.3. Identify the significant benefits of design features proposed including commonality considerations among CCAs or SRUs.

2.2.1.4. Within the draft subsystem specification and development plan for the xxx subsystem, provide additional detailed descriptions of the following:

2.2.1.4.1. Identify commercial software to be used. Describe the level of documentation available, to be developed, and how the Government will support the commercial software.

2.2.1.4.2. Describe how the XXX subsystem software development and design approach will allow modification of display page formats or information, incorporate additional pages, provide for growth in number of display units and display avionics management units.

2.2.2. Describe relevant engineering development experience of the technical personnel proposed for this program. Include specific information on planned contribution to

this program for each person identified. Be specific about team members with experience on at least two programs of similar scope, and where similar engineering tasks were accomplished.

2.2.4. Describe engineering development facilities (laboratories), staffing, and equipment planned for use on this program. Identify the respective availability of each resource and plans to acquire resources not currently available.

2.2.5. Define the preliminary support equipment (SE) program. Describe the overall program for designing, developing, and testing the proposed support equipment for the XXX system.

2.2.5.1. Describe how the support equipment selection and development processes integrate with the BIT software development effort and maintenance procedures development.

2.2.5.2. Describe test program sets (TPSs) to be used with both existing and newly developed test stands. Include a description of the TPS hardware and software requirements and identify compatible automated test equipment (ATE).

2.2.6. Describe the overall software development and management program.

2.2.6.1. Describe the software development approach, analysis methods, and integrated schedule for completing the software configuration items.

2.2.6.2. Define all software development tools that will be used including such applications as compilers, assemblers, debuggers, editors, linkers, loaders, and configuration management programs. Define the computer and operating systems on which each software tool will be used. Describe how these tools will be made available to the Government.

2.2.6.3. Describe relevant software development experience of the technical and management personnel proposed for this program. Include specific information on planned contribution to this program for each person identified. Be specific about certifications held by inspection personnel.

2.2.6.4. Describe software development facilities, staffing, and equipment planned for use on this program. Identify the respective availability of each resource and plans to acquire resources not currently available.

2.2.6.5. Describe how your software development program will support the independent validation and verification (IV&V) effort. Describe the data and documentation which shall be provided as part of the IV&V effort. Describe how IV&V personnel will be accommodated.

2.2.6.6. Each offeror may be visited by a government software assessment team (SWAT) as part of a site survey to assess software engineering capabilities. The survey will be conducted using A Method for Assessing the Software Engineering Capability, provided as an attachment to this RFP. Complete the questionnaire below to prepare for the SWAT survey.

2.2.6.6.1. Provide a completed software assessment program Form 01 (programs profile summary) and Form 02 (answers to the software assessment questionnaire) for six on-going software development programs (representative of all phases of software development) and the proposed (program name) software development efforts. The type of information required is indicated on the forms provided and shall be used to prepare responses (attachment following this section of the RFP). Provide the completed forms to the program contracting officer (PCO) separately from the proposal (address listed in paragraph XXX). The forms shall be delivered in accordance with the letter from the PCO coordinating the dates for the SWAT survey at each contractor's facility.

2.2.6.6.2. Each offeror will be notified by separate letter from the PCO to coordinate the SWAT survey visit. The team will conduct interviews with software program leaders, quality personnel, system integrators (software testing), and configuration management personnel to discuss the answers provided on the forms and assess software engineering

capabilities. Additional documentation will be requested to validate responses to the questionnaires. Documentation may include, but not limited to, cost estimating worksheets, unit development folders, software development procedures, organizational charts, software quality audit reports, and software change requests.

2.2.7. Describe the overall test and evaluation (T&E) program.

2.2.7.1. Describe all computer models, test stands, and hot mock- ups needed to ensure accurate integration and interface requirements analysis and design verification. Include the basic concept of operation for each test stand and hot mock-up. Provide a description of your modeling tools for structural and stress analysis. Identify the availability of each resource and plans to acquire resources not currently available.

2.2.7.2 Describe the integration of software development and management activities with detailed test and evaluation activities.

2.2.7.3 Describe test and evaluation facilities, staffing, and equipment planned for use on this program. Describe capability to provide supply support and maintenance to the T&E level of flight testing. Identify the respective availability of each resource and plans to acquire resources not currently available.

3. REFERENCES

a. Department of Defense

(1) Directives/Instructions

(2) Standards

[The following standards should only be cited in accordance with DoDD 5000.1 and DoD 5000.2-R]

(a) MIL-STD-498, *Software Development and Documentation*

(b) DoD-STD-1467 (AR), *Software Support Environment*

(c) MIL-HDBK-347, *Mission-Critical Computer Resources Software Support*

(d) DoD-STD-1703, *NSA/CSS Software Product Standards Manual*

(3) Other

(a) Defense Systems Management College (DSMC), *Mission Critical Computer Resources Management Guide*

b. Air Force

[See Appendix D for applicable Air Force documents]

c. Other

[The following references provide additional guidance, and should come from industry first, then (if applicable) government sources. Again, refer to the 5000 series for guidance.]

(1) AFOTEC Pamphlet 99-102, Volume 3, *Software Supportability Evaluation Guide (Contact AFOTEC/SAS)*

(2) AFOTEC Pamphlet 99-102, Volume 5, *Software Support Resources Evaluation Guide,*

(3) AFSCP/AFLCP 800-45, *Acquisition Management Software Risk Abatement (Contact HQ AFMC/EN)*

(4) AFMCP 800-51, *Software Development Capability Assessment (Contact HQ AFMC/EN)*

(5) ASC Pamphlet 63-103, *Software Development Capability Capacity Review (Contact ASC/EN)*

(6) CMU/SEI-94-TR-06, *Software Capability Evaluation (SCE), Version 2.0, Method Description (Contact Software Engineering Institute at Pittsburgh PA)*

(7) RADC-TR-85-37, *Specification of Software Quality Attributes*, Volumes I-III,

4. DEFINITIONS

a. Software supportability: characteristics of software and computer support resources that affect the ability of software support activities to correct errors, add system capabilities, delete features, and modify software to be compatible with hardware changes. It should be noted that as the Air Force moves toward truly open systems, the need to modify software to be compatible with hardware changes should no longer exist.

(1) Organization: Software possesses the characteristic of organization when the documentation is logically partitioned into sets of volumes and document development conventions have been followed. It also measures how easily specific information is located within the documentation. Another factor is how well the documents have been divided along functional lines. A hierarchical partitioning of the system's documentation of less detail to descriptions of more detail should reflect the partitioning of software.

(2) Descriptiveness: Software documentation possesses the characteristic of descriptiveness when it contains information about its intent, assumptions, inputs, processing, outputs, components, and revision status. Documentation should have a descriptive format and contain useful explanations of the software program design.

(3) Traceability: Software documentation possesses the characteristic of traceability when information about all program elements, and their implementation, can be traced between all levels of lesser and greater detail (up and down in the system hierarchy). Program elements consist of, but are not limited to, data flow, control flow, algorithms, variables, and constants. Software may be well written and well described but still lack a clearly defined trail between top level requirements and detailed implementation. The software maintainer must be able to trace any particular element from higher levels of program description down to executable code, and from executable code to higher levels of program description. Traceability should also be evident from requirements through the design to the tests which verify the design.

(4) Modularity: Software possesses the characteristic of modularity when the software design is based on a logical partitioning/grouping of software and its parts/logically related abstractions and based on minimized module/unit interdependence. Software that is the easiest to understand and change is composed on independent modules. The fewer and simpler the connections between modules, the easier it is to understand each module without reference to other modules. Reducing connections between modules also minimizes the paths along which errors can propagate into other modules of the system. Modularity also implies that a module consists of only a few easily recognizable functions which are closely related with a minimal number of links to other modules.

(5) Consistency: Software possesses the characteristic of consistency when products correlate and contain uniform notation, termination, and symbology. The use of standards and conventions in documentation, flow chart construction and certain conventions in input/output processing, module interfacing, naming of modules/variables, etc., are typical indicators of consistency. This characteristic permits for the software maintainer to concentrate on understanding the true complexities of an algorithm, data structure, etc.

(6) Simplicity: Software possesses the characteristic of simplicity when it reflects the use of singularity concepts and fundamental structures in organization, language, and implementation techniques. The use of high order language as opposed to an assembly language makes a program relatively simpler to understand because there are fewer discriminations that have to be made. The number of operators, operands, nested control structures, nested data structures, executable statements, statement labels, decision parameters, etc., will determine to a great extent how simple or complex the source code is.

(7) Expandability: Software possesses the characteristic of expandability when a physical change to information, computational functions, data storage, or execution time can be easily accomplished once the nature of what is to be changed is understood. The design should allow for flexible timing, reasonable storage margin, parameterized constants, and indentured numbering scheme for source listings that easily accommodate changes.

(8) Testability: Software possesses the characteristic of testability when it contains aids which enhance testing. The documentation should describe how well the program has been designed to include test aids (instruments), while the source listings should illustrate how the code is implemented to allow for testing. The software should be designed and implemented so testability is either embedded within the program or can be easily inserted into the program or is available through a combination of these capabilities.

(a) Testability provides information on the logical build of functions or processes of the designed/developed software from the development phase of its individual computer software units (CSUs), into its integration phase of CSUs into computer software components (CSCs), and CSCs into computer software configuration items (CSCIs).

(b) Testability includes the testing of security software requirements for compartmented, and/or multilevel security modes of operation.

(c) Testability reflects the "*as-designed*" requirements of the software as they are developed into the "*as-built*" capabilities of the final software product.

(9) Convention: Software possesses the characteristic of convention when the software products correlate and contain uniform notation, terminology, and symbology. The use of standards in documentation, flow chart, or program design language construction and certain conventions in input/output processing, error processing, module interfacing, naming of modules/variables, etc., are typical reflections of convention.

(10) Design: Software possesses the characteristic of design when programs are formed using a structured method consisting of functional parts which are interrelated, yet independent of one another.

[NOTE: Software requirements traceability is inherent in software supportability; that is, all requirements should be traceable through the documentation to the appropriate test procedure and area of code for each specific requirement.]

(11) Reusability: Software created during the development process that posses the potential for reuse within the same program or other programs.

b. Other Software Supportability Characteristics

(1) Portability: Software possesses the characteristic of portability when it is relatively easy to rehost software from one hardware platform to another hardware platform. This characteristic will require initial software development to consider future rewriting for adaptation to new hardware platforms.

(2) Machine Independence: Software possesses the characteristic of machine independence when it can be run on any hardware platform without needing to be modified to do so.

(3) External documentability: Software possesses the characteristic of external documentability when documentation (e.g., hierarchy charts, flow control charts, compilation sequence, data flow diagrams, and general explanations of what and how the software is used) matches the as-built software exactly.

(4) Coupling: Minimum degree of interaction between CSUs.

(5) Cohesion: Maximum degree of interaction within a CSU.

(6) Structured: One entry and exit per CSU.

(7) Standardization of Naming Conventions: Use of uniform notations for naming data elements.

(8) Parameterization: A measure of the use of a minimum of unnamed constants.

(9) Style: The appropriateness and use of standard conventions to aid in visual presentation of structures (e.g., numbering scheme, indentation of structures, blank lines between procedures and function definitions, and other factors which affect the readability of the software.

(10) Documentation: Availability, completeness, and correctness.

(11) Complexity: Degree to which module flow can be traced (typically measured using a McCabe's value).

PROPOSAL EVALUATION SUPPORTABILITY CRITERIA

NOTE: Evaluation criteria will cover all requirements within the request for proposal (RFP), including computer resources development and management activities and the offeror's software management plans contained in the SDP and other applicable documents. The key to achieving supportability is by defining contractual processes, performance requirements, and metrics to which the contractor will commit and adhere during software development. These are important evaluation factors and must be included in the RFP.

- Availability of software, documentation, and rights necessary to meet life cycle needs.
- The compatibility of the proposed design with the support concept in the CRLCMP.
- For systems where software changes will be frequent and are critical to overall mission capability, quantitative criteria should be established to ensure the design is modifiable and proposed support resources and methods are adequate. The offeror should describe how to identify and reestablish a previous software configuration.
- When processing of sensitive or classified information is involved, ensure computer security is an evaluation criterion.
- Correctness and reliability (or their supporting criteria of traceability, completeness, error tolerance, accuracy, and simplicity) should be measured, over the entire life cycle, on every program.

Other Evaluation Criteria

Other evaluation criteria should include:

- Throughput and memory capability of the proposed computer;
- Future vendor support for commercially supplied items such as tape drives, disk drives, and controllers;
- Computer resources interfaces to the rest of the system architecture and human operators;
- Adequacy of the operating system or software executive;
- Availability, currency, and usage of software development plans;
- Organic supportability of computer hardware and software;
- Offeror's software development plan and software development standards and procedures;
- Offeror's software development capability and capacity.
- Defect-removal efficiency (e.g., rate of 95 % or higher is acceptable). *[CAUTION: defect-removal efficiency can be manipulated. Changing the definition of a defect from a development defect to a production defect can affect the metric.]*

SOURCE SELECTION EVALUATION CONSIDERATIONS

MIL-HDBK-347 is geared to DoDD 5000 series documents and should be used in conjunction with top level software support guidance provided government-directed documents, which you should follow throughout the period of the contract to ensure a supportable and supported system is fielded. Not only should the attached factors be included in the Sections L and M of the RFP and the offeror's proposal, but they should also be incorporated as requirements in the Statement of Work. Much of the issues addressed in the following can be addressed in the

offeror's SDP, therefore the Instruction for Proposal Preparation should require the offeror to submit a draft SDP for the system being acquired.

- **CMU/SEI-94-TR-06** (This assessment is normally conducted by the offeror with SEI assistance before source selection occurs.) The Instructions for Proposal Preparation may inform the offerors that only those proposals from offerors who have received a Level 3 rating or higher will be evaluated.)
- **AFMC Pamphlet 63-103** (Use of the tool requires a competent government team and a significant amount of time to complete, but it provides the program office an estimate of the level of risk that can be expected in the software development process using each offeror.) [NOTE: A SQM proficiency audit or SEI audit may also provide the desired assessment results.]

Use of these tools will produce ratings in the following areas which should be reported to the source selection authority: program management, planning and execution, configuration management, quality assurance, quality measurement, training, process focus, and overall.

DoD-STD-1467(AR), *Software Support Environment,* Considerations

Contractually specifying the exact SEE prior to initial development is not a good approach to acquisition. Instead, the offeror should specify the top-level requirements for the SEE, and the detailed implementation of this environment should be allowed to evolve.

- The offeror should identify any proposed software or documentation with limited or restricted rights. The offeror should identify any licensing agreements that apply to the software engineering environment or software test environment to be delivered to the Government. The offeror should describe how in-house personnel or a third party contractor can accomplish software support within constraints imposed by the rights and/or licensing agreements. *[NOTE: Although not realistic to obtain in an offeror's response to a proposal, it is also desirable to know what data rights restrictions/licensing arrangements apply to the SEE/STE to be used by the offeror (or proposed subcontractors) and how those restrictions/arrangements will apply to the SEE/STE delivered to the Government.]*
- If the government's SOW requires the offeror to use the government's designated life cycle software support environment (LCSSE), have the offeror describe how the resources of the government's LCSSE will be used.
- The offeror should respond to the government's desire that the offeror's environment, identified in the proposal, shall be used in the subsequent contract performance, and that the offeror agrees to notify the Government of any changes required in the environment, with rationale given for the changes throughout the period of the contract.
- Have the offeror describe how all delivered products, both mission and support (e.g., products used in the factory development environment), will be integrated in and perform with the government designated LCSSE.
- Have the offeror identify the proposed sources for all software to be delivered in the SEE/STE.
- The offeror shall describe how the designated LCSSE might be used by the Government or the government's designee to evaluate, generate, install, integrate, test, modify, and operate the formally delivered software.

AFOTEC Pamphlet 99-102, Volume 3, *Software Maintainability Evaluation Guide,* Considerations

- Have the offeror document the approach to ensure supportable software (i.e., the software has supportability characteristics), using the definitions in paragraph 4 of the basic document.
- Have the offeror provide samples of software documentation from a program of similar scope and effort and use evaluation procedures of this pamphlet to assess its supportability. Ask the offeror to identify any changes made since that software was produced which might be relevant to the current effort.

CAUTION: When this approach is used, you run the risk of not treating all offerors equality since some may not have a viable documentation base to be evaluated. Also, the evaluation of the documentation may not include the Government influence in the decisions and direction that led to the particular software and software documentation that was produced in the previous effort.

Additional Considerations

- Have the offeror describe how the software development effort and costs will be visible to assure the effort is on track. The offeror's description of development/cost tracking should include a detailed explanation on the use of software work packages. The offeror should also be required to describe how the quality of software will be measured and maintained during the development process and over the life cycle.
- Have the offeror describe ways they will keep COTS items in their latest configuration and upward compatible to changes after delivery of the system without affecting system performance. The offeror should also address contingency plans for support of COTS products in the event the COTS vendor drops support or goes out of business. Also have the offeror describe how licenses and titles for COTS items will be transferred to the LCSSE. *[NOTE: Decisions to upgrade configurations must be a joint decision between the Government and contractor, with the Government having ultimate control.]*
- Have the offeror describe how software functionality will be allocated, traced, and its quality measured between mission and system software computer software configuration items (CSCIs). Recognizing that specific design details are not set in concrete at the time of source selection, this information should describe what intrinsic system services (i.e., services contained in system software) will be needed and what the planned COTS utilization will provide these systems. By extension, the offeror can describe in general terms what non-system service functions (i.e., functions contained in mission software) will also be needed.
- Have the offeror describe procedures for performing independent verification and validation (IV&V) (if required in the contract) and ensuring the IV&V agent access to software and associated documentation. Also, have the offeror describe how duplication of effort between the IV&V agent and SQM agent will be avoided.
- Have the offeror describe how they will minimize/eliminate use of different type/manufacturers for processors used in the system, unless those different types would make use of existing workstations/server resources at government operational and support locations. *[NOTE: Limiting processor types is a specification issue, not a source selection issue. Source selection is not used to impose requirements.]*
- Have the offeror describe how compliance will be achieved with the requirement to deliver or provide government access to all software documentation (deliverable and non-deliverable) on electronic media or in digital format (i.e., paperless, computer-aided acquisition logistics systems (CALS) compliant). Include in "*Software Documentation*" software quality

measurement data, including raw data, score sheets, tiered scores, problem trouble reports, and corrective actions. *[NOTE: The Government should specify what form of electronic media is acceptable and have the offeror describe how compliance with that form will be accomplished.]*

- Have the offeror describe how software will be loaded into storage media. The offeror should document how and where software will be uploaded into the equipment (e.g., at what maintenance level (on/off-equipment), and with or without requiring removal of processors from the equipment) and describe the memory technology proposed (e.g., programmable read only memory (PROM), ultraviolet PROM (UVPROM), electrically erasable PROM (EEPROM)). Additionally, the offeror should provide the rationale behind the decisions made to determine the support concepts/maintenance levels. *[NOTE: It is inappropriate to require the offeror to identify the memory technology planned for use.]*
- Have the offeror describe how specific critical design requirements (e.g., spare memory, timing, standardization of processors within system, etc.) will be met.

Software Language Considerations

Have the offeror describe how they plan to comply with DoD 5000.2-R and DoDD 3405.1 software development language requirements. If they can not comply with these software development language requirements, the offeror must provide a rationale based on *life cycle* (and not just developmental) cost evaluation.

AFSSI 5100, *The Air Force Computer Security (COMPUSEC) Program,* Considerations

Trusted Computing Base (TCB). Have the offeror describe how they will address each of the evaluation criteria set forth in DoD-STD 5200.28, *DoD Trusted Computing System Evaluation Criteria*, for the appropriate trusted computing base, depending on the sensitivity of the data and the clearances of the users.

Risk Management. Have the offeror describe how they will address risk management requirements, including risk analysis, security test and evaluation, and certification for facilities, software development center processors, and embedded software used or developed under the contract. Systems must be accredited by the Defense Audit Agency before they are placed in use.

Have the offeror describe how all the automated computer security provisions (identification and authentication, audit trails, and file protection and control) will be met.

MIL-STD-498 (or Industry Equivalent) Documentation Requirements Considerations

The offeror should address adequate completion of the appropriate documents listed (by DID title) in paragraph 6.2 of MIL-STD-498 (or industry equivalent), and describe how documentation adequacy will be evaluated.

AFOTEC Pamphlet 99-102, Volume 5, *Software Support Resources Evaluation,* Considerations

- The offeror should describe their approach to addressing the software support environment (i.e., software support concept
- Software support resources should address the required personnel, support systems, and facilities required for supporting software during its life cycle.

Other Supportability Source Selection Considerations

- Require that the offeror states conditions for software licensing. Specifically addressing the ability of the Government to process under a single site license with the right to copy for large quantity systems (e.g., Z-248 personal computers).
- All commercial-off-the-shelf software obtained for general purpose information systems processing equipment is required to be approved through the computer systems requirement board (CSRB) for management information systems.
- The offeror should provide data for applicable software, indicating any software attained under public domain libraries.
- The offeror should describe how contractor proprietary rights to proposed software will be minimized. While it may be difficult to control rights to commercial off the shelf or third part software, in-house developed software should be the property of the Government and be delivered as part of the life cycle software support environment.
- The offeror should describe the approach for transitioning the software process, products, and documentation to the supporting activity.
- The offeror should describe the approach for preparation, including training, of software support activity personnel for accomplishing the software support mission.
- The offeror should program PDSS personnel, facility, and equipment costs up front and include these in calculating system life cycle costs. Facility costs should include location, site preparation, construction, and installation.
- The offeror should make recommendations as to the optimum support concept (contractor only, Government only, or contractor/ Government mix) for each proposed computer software configuration item, and justify the recommendation based on operational requirements and life cycle costs.
- Have the offeror describe how the system/software engineering environment will meet all trusted database and multilevel security requirements.
- Have the offeror describe the level and sources of training (skills) required for support of each of the delivered software products.
- Have the offeror describe how the software will be implemented without serious impact to the operating system (if applicable).
- Have the offeror describe how the developed software will fulfill requirements and yet meet RFP interface requirements.
- Have the offeror describe how impacts of the newly developed software on other operating systems will be assessed.
- Have the offeror describe the strategy for reuse of existing and newly developed software.
- Have the offeror describe plans for software disaster storage and recovery.

APPENDIX

N

AIS ORDs Recommendations

Blank page.

APPENDIX N

Recommendations for Automated Information Systems (AIS) Operational Requirements Documents (ORDs)

NOTE: The source for this material is AFOTEC Technical Paper, C4I Systems Directorate, Automated Information System (AIS) Operational Test and Evaluation (OT&E) Planning Handbook, by Nickolas P. Angelo, AFOTEC/ TKA.

CONTENTS **PAGE**

1. General Description of Operational Capability. N-3
   a. Mission Area. N-3
   b. Mission Area Need. N-3
      (1) Planning Task Needs. N-4
      (2) Organizing Task Needs. N-4
      (3) Directing Task Needs. N-4
      (4) Controlling Task Needs. N-5
   c. Joint Potential and Multinational Applicability. N-5
2. Threat N-5
   a. Threat Engagement. N-5
   b. Threat Vulnerability. N-5
3. Shortcomings of Existing Systems N-5
   a. Status Quo. N-5
   b. Modify Doctrine Option. N-5
   c. Modify Operational Concept Option. N-6
   d. Modify Tactics Option. N-6
   e. Modify Organization Option. N-6
   f. Modify Training Option. N-6
4. Capabilities Required. N-6
   a. System Performance. N-6
      (1) Effectiveness Critical Operational Issues (COIs). N-7
         (a) Planning Task Scenarios. N-8
         (b) Organizing Task Scenarios. N-9

(c) Directing Task Scenarios. N-10
(d) Controlling Task Scenarios. N-11
(2) Effectiveness Metrics — Measure of Effectiveness/Performance. N-11
(a) Task Timeliness. N-12
(b) Information Accuracy. N-12
(c) Information Currency. N-12
(d) Information Completeness. N-13
(e) Information Relevancy. N-13
(f) Information Format. N-13
b. Logistics and Readiness. N-13
(1) Suitability Critical Operational Issues (COIs). N-13
(a) AIS Readiness. N-14
(b) AIS Logistic Support N-14
(2) Suitability Metrics — Measure of Suitability/Performance. N-14
(a) Operational Availability (Ao). N-14
1. Mean Time Between Downing Events (MTBDE). N-15
2. Mean Downtime (MDT). N-15
(b) Operational Dependability (Do). N-15
1. Mean Time Between Operational Mission Failures (MTBOMF). N-16
2. Mean Corrective Maintenance Time for Operational Mission Failures N-16
(c) Mean Time Between Maintenance (MTBM) MOS N-16
1. Mean Time Between Unscheduled Maintenance (MTBUM). ... N-17
2. Mean Time Between Scheduled Maintenance (MTBSM). N-17
(d) Maintenance Ratio (MR) MOPs N-17
1. Mean Corrective Maintenance Time (MCMT) MOP N-17
2. Mean Preventive Maintenance Time (MPMT) MOP N-18
(e) System Survivability. N-18
1. Administrative and Physical Controls. N-18
2. Communication Controls. N-18
3. Data Integrity. N-19
4. Post-processing Controls. N-19
(f) Human Supportability. N-19
1. Manpower and Personnel Support Adequacy. N-19
2. Training and Training Support Adequacy. N-19
3. Technical Data Adequacy. N-20
4. Human Factors Engineering Adequacy. N-20
5. Safety and Health Hazards Adequacy. N-20
(g) Infrastructure Supportability. N-20
1. Transportation and Basing Adequacy. N-20
2. Facility Support Adequacy. N-21
3. Supply Support Adequacy. N-21
4. Support Equipment Adequacy. N-21
(h) Software Supportability. N-21
1. Software Maturity Adequacy. N-21
2. Software Maintainability Adequacy. N-21
3. Software Support Resources Adequacy. N-22
4. Software Life Cycle Support Adequacy. N-22
c. Critical System Characteristics N-22

APPENDIX N Recommendations for AIS ORDs

5. Integrated Logistics Support. N-22
   a. Maintenance Planning. N-22
      (1) Organizational Maintenance Concept. N-22
      (2) Depot Maintenance Concept. N-23
   b. Support Equipment. N-23
   c. Human Systems Integration. N-23
      (1) Manpower and Personnel. N-23
      (2) Training and Training Support. N-23
      (3) Technical Data. N-23
      (4) Human Factors Engineering. N-24
      (5) Safety and Health Hazards. N-24
   d. Computer Resources. N-24
      (1) Software Maturity. N-24
      (2) Software Maintainability. N-24
      (3) Software Support Resources. N-24
      (4) Software Life Cycle Support. N-25
   e. Other Logistics Consideration. N-25
      (1) Supply Support. N-25
      (2) Facilities and Land N-25
6. Infrastructure Support and Interoperability. N-25
   a. Command, Control, Communications, and Intelligence (C3I). N-25
   b. Transportation and Basing. N-26
   c. Standardization, Interoperability, and Commonality. N-26
      (1) Standardization and Commonality. N-26
      (2) Interoperability. N-26
   d. Mapping, Charting, and Geodesy (MSG) Support. N-26
   e. Environment Support. N-26
7. Force Structure N-27
8. Schedule Considerations. N-27

REQUIREMENTS CORRELATION MATRIX. N-27

NOTE: **For AIS, the ORD represents a formatted statement addressing the operational effectiveness and suitability performance requirements and parameters for an identified AIS. The following guidance for AIS applies.** [Always use the verb "*shall*" in a statement that indicates an operational requirement.]

1. **General Description of Operational Capability**. The general description of operational capability pertains to the mission area as it relates to the proposed AIS, and anticipated operational and support concepts for program and logistics support planning.

a. **Mission Area**. The mission area pertains to associating the need with the major planning objective found in the Defense Planning Guidance document. Address the following information in this paragraph.

- **Identify** the major program planning objective or Defense Planning Guidance section addressed by the need.
- **Reference** DoD or Military Department long range investment plans, where applicable.
- **Identify** for top-down directed needs who and what drove the requirement.

b. **Mission Area Need**. Mission area need pertains to identifying and describing the required assigned tasks for achieving mission needs to support the national strategy. Managers plan, organize, direct, and control the performance of assigned tasks to achieve mission objectives. To perform these management processes, managers require information to make

decisions that ultimately direct the effective use of personnel and resources to support the national strategy. Managers use management information systems (MIS) — automated, semi-automated, or manual information systems — as a means of generating information that managers need. The effectiveness of an MIS to achieve assigned tasks depends largely on the capability of the MIS to provide quality information in a timely, usable, and reliable manner. Address the following information in this paragraph.

- **Identify** the overall MIS mission need (objective) and operational task(s).
- **Describe** the deficiency(ies) to meeting the MIS mission need and operational task(s).
- **Comment** on the timing for achieving the mission need relative to other needs in the mission area.
- **Comment** on the priority of the need relative to other needs in the mission area.

(1) **Planning Task Needs.** Planning task needs pertain to setting goals and defining policies, procedures, and programs to achieve mission objectives. The three types of planning activities include strategic planning, tactical planning, and forecasting. Strategic planning tasks involve determining organization objectives and formulating long term organizational policy to meet mission needs. Tactical planning tasks involve the allocation of total resources of the organization to meet mission needs. Forecasting tasks involve determining or predicting possible outcomes of proposed strategic and tactical plans. Planning describes and explores the external operational environment with regard to the mission of the organization. Address the following information in this paragraph.

- **Identify** the planning tasks performed by managers to support the mission need.
- **Describe** current MIS deficiency(ies) to perform effectively planning assigned tasks.
- **Define** MIS requirement(s) to perform effectively planning assigned tasks.

(2) **Organizing Task Needs.** Organizing task needs pertain to grouping activities to be performed as well as establishing organizational forms and relationships to meet mission needs. The two types of organization activities include personnel organization and resource organization. Personnel organization tasks involve the selection and training of organization personnel, and the allocation of duties and workloads to personnel in a logical manner to meet mission needs. Resource organization tasks involve the use of logical methods to categorize and arrange necessary resources to meet mission needs. Organizing describes and explores the internal operations of the organization with regard to the mission of the organization and the external operational environment. Address the following information in this paragraph.

- **Identify** the organizing tasks performed by managers to support the mission need.
- **Describe** current MIS deficiency(ies) to perform effectively organizing tasks.
- **Define** MIS requirement(s) to perform effectively organizing assigned tasks.

(3) **Directing Task Needs.** Directing task needs pertain to leading, guiding, and motivating people in the organization through information facilitation and flow of knowledge to achieve mission needs. The three types of directing activities include leadership, communication, and coordination. Leadership directing tasks involve the ability to provide effective guidance that motivates personnel to meet mission needs. Communication directing tasks involve the psychology, language structure, and physical communication paths used to convey information to meet mission needs. Coordination directing tasks involve the integration of specific departments or divisions within the organization to meet mission needs. Directing communicates decisions for executing the internal operations of the organization with regard to the mission of the organization and the external operational environment. Address the following information in this paragraph.

- **Identify** the directing tasks performed by managers to support the mission need.
- **Describe** current MIS deficiency(ies) to perform effectively directing tasks.
- **Define** MIS requirement(s) to perform effectively directing assigned tasks.

(4) **Controlling Task Needs**. Controlling task needs pertain to monitoring, measuring, and modifying (where necessary) policy, procedures, and programs to achieve mission needs. The three types of controlling activities include monitoring activities, measuring planned performance, and modifying activities. Monitored activities controlling tasks involve the supervising of actual activities to meet mission needs. Measuring planned performance controlling tasks involve the comparison of planned performance with actual performance to meet mission needs. Modified activities controlling tasks involve the alteration of actual activities to meet mission needs. Controlling ensures the internal operations of the organization support the mission of the organization in the external operational environment. Address the following information in this paragraph.

- **Identify** the controlling tasks performed by managers to support the mission need.
- **Describe** current MIS deficiency(ies) to perform effectively controlling tasks.
- **Define** MIS requirement(s) to perform effectively controlling assigned tasks.

c. **Joint Potential and Multinational Applicability**. Joint potential and multinational applicability pertains to AIS capability for joint Service or multinational use. Address the following information in this paragraph.

- **Identify** any joint Service or multinational applications for the AIS.

2. **Threat**. The threat pertains to identifying and describing the encountered threat environment for achieving mission needs to support the national strategy. In the threat environment, MIS not only can engage threats, but also are vulnerable to threats.

a. **Threat Engagement**. Threat engagement pertains to the threat environment that must be countered to achieve mission needs. Most MIS are not designed to encounter threats. Normally, this subparagraph would be addressed as "*not applicable*." If an MIS counters threats, address the following information in this paragraph

- **State** the Defense Intelligence Agency (DIA)-validated threat encountered by MIS.
- **Discuss** the projected threat environment to be countered by MIS.
- **Discuss** the shortfalls of existing MIS capabilities or systems in meeting these threats.

b. **Threat Vulnerability**. Threat vulnerability pertains to the threat environment that endangers an MIS survivability. Most MIS are susceptible to a variety of threats which cover accidental and deliberate threats. These threats engage information of the MIS. Address the following information in this paragraph.

- **Discuss** the threat of accidental modification, destruction, or disclosure of information used for decision making by managers.
- **Discuss** the threat of deliberate modification, destruction, or disclosure of information used for decision making by managers.

3. **Shortcomings of Existing Systems**. Shortcomings of existing systems pertain to the shortfalls of the status quo system and considered non material alternatives for achieving mission needs to support the national strategy. These options include the status quo, modifying doctrine, modifying operational concepts, modifying tactics, modifying the organization, and modifying training.

a. **Status Quo**. The status quo pertains to the MIS (automated, semi-automated, or manual information system) currently employed by the organization to manage information to support mission needs. Address the following information in this paragraph.

- **Discuss** the results of the mission need considerations.
- **Identify** the status quo with regard to MIS.
- **Describe** why the status quo with regard to MIS was judged to be inadequate.

b. **Modify Doctrine Option**. The modify doctrine option pertains to changing current US or Allied doctrine to achieve mission needs. Address the following information in this paragraph.

- **Identify** any changes in US or allied doctrine considered as a non material alternative solution to meet the mission need.

✪ **Describe** why such changes in doctrine were judged as inadequate to meet the mission need.

c. **Modify Operational Concept Option**. The modify operational concept option pertains to changing the current MIS operational concepts or maintenance concepts to achieve mission needs. Address the following information in this paragraph.

✪ **Identify** any changes in operational concepts considered as a non material alternative solution to meet the mission need.

✪ **Describe** why such changes in operational concepts were judged as inadequate to meet the mission need.

d. **Modify Tactics Option**. The modify tactics option pertains to changing current tactics to achieve mission needs. Address the following information in this paragraph.

✪ **Identify** any changes in tactics considered as a non material alternative solution to meet the mission need.

✪ **Describe** why changes in tactics were judged inadequate to meet the mission need.

e. **Modify Organization Option**. The modify organization option pertains to changing current organizational structure to achieve mission needs. Address the following information in this paragraph.

✪ **Identify** any changes in organizational structure considered as a non material alternative solution to meet the mission need.

✪ **Describe** why such changes in organizational structure were judged as inadequate to meet the mission need.

f. **Modify Training Option**. The modify training option pertains to changing current training methods to achieve mission needs. Address the following information in this paragraph.

✪ **Identify** any changes in training considered as a non material alternative solution to meet the mission need.

✪ **Describe** why such changes in training methods were judged as inadequate to meet the mission need.

4. **Capabilities Required**. Capabilities required pertain to the operational effectiveness and operational suitability performance of the AIS to achieve mission needs. AIS required capabilities address the critical operational issues (COIs), measures of effectiveness (MOEs), measures of performance (MOPs), and thresholds and/or objectives. COIs represent key operational effectiveness and suitability issues — as deemed by the user — that must be examined during operational test to determine the system's capability to meet mission needs. MOEs represent quantitative measurements of a system's degree of performance for specific operational/ assigned tasks. MOPs represent quantitative and qualitative measurements of system capabilities and characteristics to perform assigned tasks to achieve mission needs. Threshold values represent the minimum acceptable operational requirements. Objective values represent the desired, beneficial increase operational requirement. Address the following information in this paragraph, replacing terms underlined with appropriate system terminology.

✪ **State the following**: *"These capabilities address the required operational effectiveness and operational suitability performance capabilities and characteristics of the AIS to meet mission need(s)."*

a. **System Performance**. System performance pertains to the operational effectiveness COIs, MOEs, MOPs, and thresholds/objectives with regard to assigned task scenarios. Address the following information in this paragraph, replacing terms underlined with appropriate system terminology.

✪ **State the following**: *"Figure X diagrams the macro-managerial tasks both sequentially and concurrently. System performance addresses the operational effectiveness critical operational issues (COIs), measures of effectiveness (MOEs), measures of performance (MOPs), and associated thresholds/ objectives with regard to assigned task scenarios."*

✪ **Diagram** sequentially and concurrently in a PERT flow chart those macro-managerial tasks that the AIS will support.

APPENDIX N Recommendations for AIS ORDs

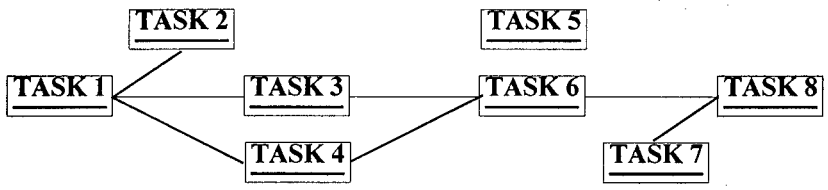


Figure N-1 Macro-Managerial Task Flow Chart

(1) **Effectiveness COIs**. Operational effectiveness represents the overall degree of mission accomplishment of a system when used by representative personnel in the environment planned for employment of the system considering organization, doctrine, tactics, survivability, vulnerability, and threat. The term "*mission*" refers to the task, together with the purpose, that clearly indicates the action to be taken and the reason therefore. In common usage, especially when applied to lower military units, the mission is a duty assigned to an individual or unit; a task. Within this context, MIS operational effectiveness evaluation focuses on examining the operational concept (assigned tasks) performed by users using the MIS to meet mission needs that accomplish operational task(s). As the OT&E test article, the AIS represents a system designed to provide managers useful information in a necessary time frame to make decisions. As a tool for management, an AIS must enable users to perform assigned tasks (planning, organizing, directing, and controlling actions) to manage effectively the assets (personnel and resources) necessary to accomplish the operational tasks of the organization in support of some operational objective and strategy.

✪ **State the following**: "*The operational effectiveness COIs are derived from the type of task scenarios the user must perform with the AIS to meet mission needs.*"

As subparagraphs to this paragraph heading, effectiveness COIs are written in one of two ways. The first method focuses on the functional operation areas (i.e., logistics support, transportation, Intelligence, weather, etc..) to manage the missions. Semantically, this type of effectiveness COI specifies the AIS; the functional operation area of the organization employing the AIS; the operational locations for employing the AIS; the operational task (management); and the mission being managed. To address this type of COI, the functional area assigned tasks (plan, organize, direct, and control) become the focus of the evaluation. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

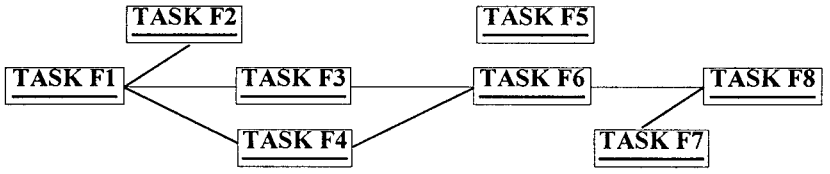


Figure N-2 Functional Area Task Flow Chart

✪ **State the following**: "*Figure X diagrams the functional area managerial tasks both sequentially and concurrently. The following subparagraphs address the functional operation area scenarios required to answer the COI: Do AIS capabilities support functional operational area decision makers at operating locations to manage the mission.*"

✪ **Diagram** sequentially and concurrently in a PERT flow chart those functional area tasks that the AIS will support.

✪ **Specify** the following for each functional operation area task scenario subparagraph:

- √ In "*bold*" face, the assigned task to be performed.
- √ Who performs the assigned task (user types, functional area, etc.).
- √ What do users do to perform the assigned task (major steps, subtasks, etc.).
- √ When do users begin/conclude the assigned task (input event, output event, etc.).
- √ Where do users perform the assigned task (location, managerial level, etc.).
- √ Why do users require an AIS to perform the assigned task effectively.
- √ How often do users perform the assigned task (frequency, workload, etc.).
- √ Effectiveness (task success rate) threshold and objective requirements.
- √ Performance (information value attributes) threshold and objective requirements.
- √ Associated rationale for quantified thresholds and objectives metrics.
- √ The AIS acquisition increment(s) whose capabilities address the assigned task.

The second method focuses on the assigned task areas (plan, organize, direct, and control) to manage the mission. Semantically, effectiveness COIs specify the AIS, the organization employing the AIS, the operational locations for employing the AIS, the assigned task (plan, organize, direct, and control), and the mission being managed. As described in the subsequent paragraphs, four assigned task area COIs exist and they collectively enable the accomplishment of the operational task (management).

(a) **Planning Task Scenarios**. Planning task scenarios pertain to those assigned tasks that are concerned with setting goals and defining policies, procedures, and programs to achieve mission needs. The planning task scenarios address the principle COI: "Do AIS capabilities support organization decision makers at operational locations to plan the mission." The "*planning the mission*" assigned tasks associated with this COI address the required functional users and their assigned tasks to plan for the use of their forces and resources to support the operational objectives of the organization. The planning task MOEs cover information value. Users specify whether planning tasks warrant a COI. Address the following information in this paragraph, replacing terms underlined with system terminology.

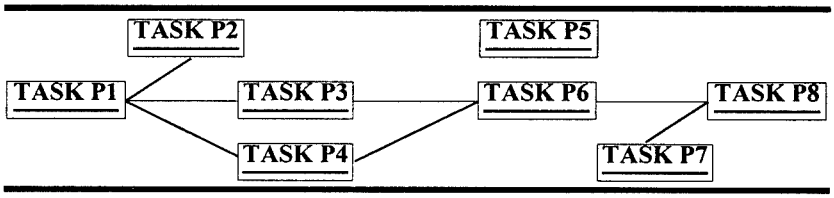


Figure N-3 Planning Task Flow Chart

✪ **State the following**: "*Figure X diagrams the planning managerial tasks both sequentially and concurrently. The following subparagraphs address the planning task scenarios required to answer the COI: Do AIS capabilities support organization decision makers at operating locations to plan the mission.*"

✪ **Diagram** sequentially and concurrently in a PERT flow chart those planning managerial tasks that the AIS will support.

✪ **Specify** the following for each functional operation area task scenario subparagraph:

- √ In "*bold*" face, the assigned task to be performed.
- √ Who performs the assigned task (user types, functional area, etc.).
- √ What do users do to perform the assigned task (major steps, subtasks, etc.).

- √ When do users begin/conclude the assigned task (input event, output event, etc.).
- √ Where do users perform the assigned task (location, managerial level, etc.).
- √ Why do users require an AIS to perform the assigned task effectively.
- √ How often do users perform the assigned task (frequency, workload, etc.).
- √ Effectiveness (task success rate) threshold and objective requirements.
- √ Performance (information value attributes) threshold and objective requirements.
- √ Associated rationale for quantified thresholds and objectives metrics.
- √ The AIS acquisition increment(s) whose capabilities address the assigned task.

(b) **Organizing Task Scenarios**. Organizing task scenarios pertain to those assigned tasks that are concerned with grouping activities to be performed as well as establishing organizational forms and relationships to meet mission needs. The following organizing tasks address the COI: "*Do AIS capabilities support organization decision makers at operational locations to organize the mission.*" The "*organizing the mission*" assigned tasks associated with this COI address the required functional users and their assigned tasks to organize for the use of their forces and resources to support the operational objectives of the organization. Its organizing task MOPs cover information value. Users specify whether organizing tasks warrant a COI. Address the following information in this paragraph, replacing terms underlined with system terminology.

✪ **State the following**: "*Figure X diagrams the organizing managerial tasks both sequentially and concurrently. The following subparagraphs address the organizing task scenarios required to answer the COI: Do AIS capabilities support organization decision makers at operating locations to organize the mission.*"

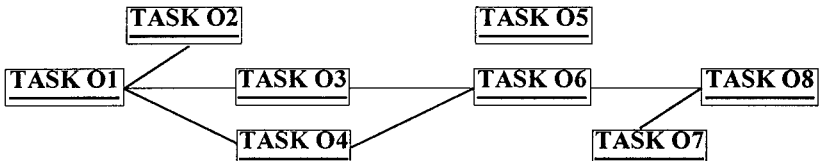


Figure N-4 Organizing Task Flow Chart

✪ **Diagram** sequentially and concurrently in a PERT flow chart those organizing managerial tasks that the AIS will support.

✪ **Specify** the following for each functional operation area task scenario subparagraph:

- √ In "*bold*" face, the assigned task to be performed.
- √ Who performs the assigned task (user types, functional area, etc.).
- √ What do users do to perform the assigned task (major steps, subtasks, etc.).
- √ When do users begin/conclude the assigned task (input event, output event, etc.).
- √ Where do users perform the assigned task (location, managerial level, etc.).
- √ Why do users require an AIS to perform the assigned task effectively.
- √ How often do users perform the assigned task (frequency, workload, etc.).

- √ Effectiveness (task success rate) threshold and objective requirements.
- √ Performance (information value attributes) threshold and objective requirements.
- √ Associated rationale for quantified thresholds and objectives metrics.
- √ The AIS acquisition increment(s) whose capabilities address the assigned task.

(c) **Directing Task Scenarios**. Directing task scenarios pertain to those assigned tasks that are concerned with leading, guiding, and motivating people in the organization through information facilitation and flow of knowledge to achieve mission needs. The directing task scenarios address the principle COI: "*Do AIS capabilities support organization decision makers at operational locations to direct the mission.*" The "*directing the mission*" assigned tasks associated with this COI address the required functional users and their assigned tasks to direct the use of their forces and resources to support the operational objectives of the organization. The directing task MOPs cover information value. Users specify whether directing tasks warrant a COI. Address the following information in this paragraph, replacing terms underlined with system terminology.

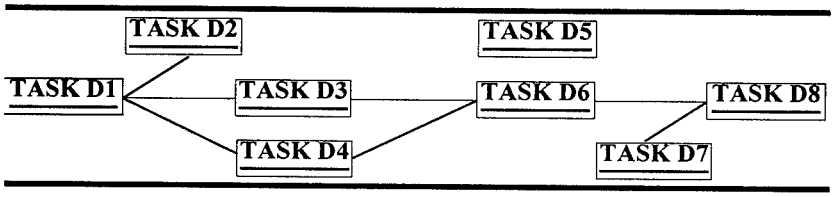


Figure N-5 Directing Task Flow Chart

- **State the following**: "*Figure X diagrams the directing managerial tasks both sequentially and concurrently. The following subparagraphs address the directing task scenarios required to answer the COI: Do AIS capabilities support organization decision makers at operating locations to direct the mission.*"
- **Diagram** sequentially and concurrently in a PERT flow chart those directing managerial tasks that the AIS will support.
- **Specify** the following for each functional operation area task scenario subparagraph:
  - √ In "*bold*" face, the assigned task to be performed.
  - √ Who performs the assigned task (user types, functional area, etc.).
  - √ What do users do to perform the assigned task (major steps, subtasks, etc.).
  - √ When do users begin/conclude the assigned task (input event, output event, etc.).
  - √ Where do users perform the assigned task (location, managerial level, etc.).
  - √ Why do users require an AIS to perform the assigned task effectively.
  - √ How often do users perform the assigned task (frequency, workload, etc.).
  - √ Effectiveness (task success rate) threshold and objective requirements.
  - √ Performance (information value attributes) threshold and objective requirements.
  - √ Associated rationale for quantified thresholds and objectives metrics.
  - √ The AIS acquisition increment(s) whose capabilities address the assigned task.

(d) **Controlling Task Scenarios**. Controlling task scenarios pertain to those assigned tasks that are concerned with monitoring, measuring, and modifying (when necessary) policy, procedures, and programs to meet mission needs. The following controlling tasks address the COI: *"Do AIS capabilities support organization decision makers at operational locations to organize the mission."* The *"controlling the mission"* assigned tasks associated with this COI address the required functional users and their assigned tasks to control the use of their forces and resources to support the operational objectives of the organization. Its controlling task MOPs cover information value. Users specify whether controlling tasks warrant a COI. Address the following information in this paragraph, replacing terms underlined with system terminology.

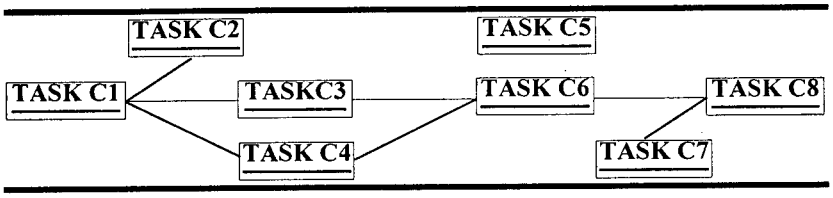


Figure N-6 Controlling Task Flow Chart

✪ **State the following**: *"Figure X diagrams the controlling managerial tasks both sequentially and concurrently. The following subparagraphs address the controlling task scenarios required to answer the COI: Do AIS capabilities support organization decision makers at operating locations to control the mission."*

✪ **Diagram** sequentially and concurrently in a PERT flow chart those controlling managerial tasks that the AIS will support.

✪ **Specify** the following for each functional operation area task scenario subparagraph:

- √ In *"bold"* face, the assigned task to be performed.
- √ Who performs the assigned task (user types, functional area, etc.).
- √ What do users do to perform the assigned task (major steps, subtasks, etc.).
- √ When do users begin/conclude the assigned task (input event, output event, etc.).
- √ Where do users perform the assigned task (location, managerial level, etc.).
- √ Why do users require an AIS to perform the assigned task effectively.
- √ How often do users perform the assigned task (frequency, workload, etc.).
- √ Effectiveness (task success rate) threshold and objective requirements.
- √ Performance (information value attributes) threshold and objective requirements.
- √ Associated rationale for quantified thresholds and objectives metrics.
- √ The AIS acquisition increment(s) whose capabilities address the assigned task.

(2) **Effectiveness Metrics — MOEs/MOPs**. For an MIS, the operational task concerns the management of the forces and resources of the organization to support operational objectives. Using an AIS, managers perform assigned tasks (planning, organizing, directing, and controlling) to manage their forces and resources effectively. The MOE evaluation criteria concentrates on the outcomes from executing the operational or assigned tasks. For an MIS, the evaluation criteria rests with the user-defined, assigned tasks effectiveness rates—the probability (percentage) that decision makers (managers) can perform assigned tasks effectively, based on

the value of the information provided. After all, if the MIS/AIS provides inferior, untimely, or unusable information, then the user capability to accomplish assigned tasks effectively to meet mission needs diminishes. The performance measures (MOPs) for evaluating assigned task information value address those capabilities (attributes) inherent to quality information. The MOPs required for evaluating assigned task information value include task timeliness as well as information accuracy, currency, completeness, relevancy, and format. These information value attributes represent the plausible AIS causes for ineffective performance of assigned tasks. These information value attributes apply to the effectiveness of any operational or assigned task for MIS. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"The operational effectiveness MOEs for the AIS address the information value provided to perform the assigned tasks of the AIS. Information value tasks measures the quality of information provided by the AIS to support organization decision makers at operational locations to achieve the mission. Its evaluation criteria represents the probability that decision makers using the AIS can perform the assigned tasks effectively. The required task effectiveness rate are specified for each assigned task. Effective assigned task performance is based on the following information value characteristics: task timeliness, information accuracy, information currency, information completeness, information relevancy, and information format."*

(a) **Task Timeliness**. Task timeliness pertains to the time required to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. To operationally quantify task timeliness requirements, AIS users must have specified up front the assigned task(s); the type of information required to perform the assigned task(s); and the amount of time (minimum) required to accomplish the assigned task effectively. Address the following information in this paragraph.

✪ **State the following**: *"Task timeliness measures the amount of time required to execute assigned tasks effectively. Evaluation criteria: The task performance time required to enable effective task performance. Task timeliness evaluation criteria thresholds/objectives are denoted for each assigned task either as a specific performance time requirement or as user satisfaction with actual information timeliness, where appropriate."*

(b) **Information Accuracy**. Information accuracy pertains to the correctness of information in reflecting reality to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. To operationally quantify information accuracy requirements, AIS users must have specified up front the assigned task(s); the information required to perform the assigned task(s); and the degree of information accuracy required to perform the assigned task effectively with regard to correctness and precision. Address the following information in this paragraph.

✪ **State the following**: *"Information accuracy measures the correctness of information in reflecting reality. Evaluation criteria: The percentage of presented information over a specified time interval deemed correct to enable effective task performance. Information accuracy evaluation criteria thresholds/objectives are denoted for each assigned task, where appropriate."*

(c) **Information Currency**. Information currency pertains to the degree to which information is up-to-date to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. To operationally quantify information currency requirements, AIS users must have specified up front the assigned task(s); the information required to perform the assigned task(s); and the degree of currency of the information required to accomplish the assigned task effectively with regard to response time and up-to-datedness. Address the following information in this paragraph.

✪ **State the following**: *"Information currency measures the degree to which the information is up-to-date. Evaluation criteria: The percentage of presented information over a specified time interval deemed up-to-date or current to enable effective task performance. Information currency evaluation criteria thresholds/objectives are denoted for each assigned task, where appropriate."*

(d) **Information Completeness**. Information completeness pertains to the thoroughness of sought information to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. To operationally quantify information completeness requirements, AIS users must have specified up front the assigned task(s); the type of information required to perform the assigned task(s); and the degree of completeness (minimum level) required to accomplish the assigned task effectively with regard to level of detail and exhaustiveness of information. Address the following information in this paragraph.

✪ **State the following**: *"Information completeness measures the thoroughness of sought information. Evaluation criteria: The percentage of presented information over a specified time interval deemed thorough enough to enable effective task performance. Information completeness evaluation criteria thresholds/objectives are denoted for each assigned task, where appropriate."*

(e) **Information Relevancy**. Information relevancy pertains to the essentialness of information provided to the user to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. To operationally quantify information relevancy requirements, AIS users must have specified up front the assigned task(s); the type of information required to perform the assigned task(s); and the degree of relevancy (minimum level) required to perform the assigned task effectively with regard to level of redundancy and appropriateness of information. Address the following information in this paragraph.

✪ **State the following**: *"Information relevancy measures the essentialness of information provided to the user. Evaluation criteria: The percentage of presented information over a specified time interval deemed germane or essential to enable effective task performance. Information relevancy evaluation criteria thresholds/objectives are denoted for each assigned task, where appropriate."*

(f) **Information Format**. Information format pertains to the composition or layout of the information to execute assigned tasks effectively. This operational performance characteristic applies to the effectiveness of any assigned task. Address the following information in this paragraph.

✪ **State the following**: *"Information format measures the adequacy of information presentation required to support decision makers to execute assigned tasks effectively. Evaluation criteria: The user satisfaction with information format required to enable effective assigned task performance."*

b. **Logistics and Readiness**. Logistics and readiness pertains to the operational suitability COIs, MOPs, and the thresholds/objectives with regard to suitability task scenarios. The operational suitability COIs are derived from the operational life cycle states of the AIS to support the achievement of mission needs. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Logistics and readiness addresses the operational suitability critical operational issues (COIs), measures of performance (MOPs), and associated thresholds/objectives for all task scenarios."*

(1) **Suitability COIs**. Operational suitability is the degree to which a system can be placed satisfactorily in field use with consideration given to availability, compatibility, transportability, interoperability, reliability, wartime usage rates, maintainability, safety, human factors, manpower supportability, logistics supportability, natural environmental effects and impacts, documentation, and training requirements. Within this context, AIS suitability

evaluation focuses on the life cycle of a system in its operational environment to support the performance of assigned tasks. As the OT&E test article, the AIS represents a system made up of fixed and possibly deployable nodes providing managers at various locations useful information in a necessary time frame to make decisions. As the tool for management, the AIS must have acceptable logistics support and readiness to support effective management of forces and resources to accomplish the operational tasks of the organization in support of some operational objective and strategy. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"The operational suitability COIs are derived from the operational life cycle states of the AIS for achieving mission needs."*

(a) **AIS Readiness**. Readiness is the ability of forces, units, systems, or equipment to deliver required timely output with finite deployable resources. AIS readiness involves the availability of AIS nodes (fixed or deployable) to support mission needs. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Readiness answers the COI: Does AIS readiness support mission requirements in the operational environment? It is addressed by the operational availability, operational dependability, mean time between maintenance (logistics reliability) and maintenance ratio performance measures."*

(b) **AIS Logistic Support**. Sustainability is the ability of forces, units, systems, or equipment to maintain the necessary level and duration of operational activity to achieve operational objectives. AIS sustainability involves the effective response of both fixed and deployable portions of the system to support mission needs. The effectiveness of achieving sustainable states with an AIS depends largely on system survivability, human supportability, infrastructure supportability, and software. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Logistics Support answers the COI: Does AIS logistics support sustain mission requirements in the operational environment? It is addressed by the system survivability, human systems supportability, infrastructure supportability, and software supportability performance measures."*

(2) **Suitability Metrics — Measures of Suitability (MOSs)/MOPs**. For MIS, two types of suitability COIs exist — readiness and logistics supportability. The AIS suitability MOSs for evaluating readiness concern operational availability, operational dependability, and preventative maintenance requirements and for evaluating logistics support concern system survivability, human supportability, infrastructure supportability, and software supportability.

✪ **State the following**: *"The AIS suitability MOSs address operational availability (Ao), operational dependability (Do), mean time between maintenance (MTBM), maintenance ratio (MR), system survivability, human systems supportability, infrastructure supportability, and software supportability."*

(a) **Operational Availability (Ao)**. This MOS pertains to the probability that a system can be placed in use for any specified assigned task, when required. This MOS answers the question: Does the AIS operational availability furnish operational users with information required to accomplish assigned tasks effectively? Ao includes both the inherent parameters and logistics support effectiveness of the AIS that relate to all time the system might be desired for use. Quantitative evaluation criteria (a ratio between 0 and 1) represents user satisfaction with the operational availability (Ao) of the AIS to support the performance of assigned tasks effectively. Qualitative evaluation criteria defines the inherent reliability parameters, maintainability parameters, and logistics support effectiveness issues that constitute an operational available AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Ao, where: Ao = (MTBDE)/(MTBDE + MDT). Evaluation criteria: percentage that the AIS Ao enables effective performance of the assigned tasks."*

✪ **Define** the inherent reliability parameters, maintainability parameters, and logistics support effectiveness issues that constitute an operationally available AIS and associated subsystems.

✪ **Define** the type of operational mission failures, preventive maintenance, training, maintenance and supply response, and actual on-equipment repairs that constitute downing events for the AIS and associated subsystems.

✪ **Define** the type of system repair time, administrative delays, and logistics delays that constitute downing time for the AIS and associated subsystems.

1. **Mean Time Between Downing Events (MTBDE)**. This MOP pertains to the average time between events which bring the system down. This MOP furnishes information required to calculate operational availability. MTBDE includes operational mission failures, preventative maintenance, training, maintenance and supply response, administrative delays, and actual on-equipment repair. Quantifiable objective evaluation criteria (average in hours) represents user satisfaction with the MTBDE of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of operational mission failures, preventative maintenance, training, maintenance and supply responses, administrative delays, and actual on-equipment repair events that constitute downing events for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean Time Between Downing Events (MTBDE), where: MTBDE = (Number of operating hours)/(Number of downing events). Evaluation criteria: MTBDE that the AIS enables effective performance of the assigned tasks."*

2. **Mean Downtime (MDT)**. This MOP pertains to the average elapsed time, as the result of a downing event, required to repair and restore the system to full operating status. This MOP furnishes information required to calculate operational availability. MDT results from system repairs, administrative delays, and logistics delays. Quantifiable objective evaluation criteria (average in hours) represents user satisfaction with the MDT of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of system repairs, administrative delays, and logistics delays that constitute downing events for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean Down Time (MDT), where: MDT = (Total down time in hours)/(Number of downing events). Evaluation criteria: MDT that the AIS enables effective performance of the assigned tasks."*

(b) **Operational Dependability (Do)**. This MOS pertains to the probability that a system can be continuously used to execute a specific assigned task. This MOS answers the question: Does the AIS dependability furnish users with information required to accomplish assigned tasks effectively? Do includes both the inherent reliability parameters, maintainability parameters, and logistics support effectiveness of the AIS that relate to all time the system might be desired for use. Quantifiable objective evaluation criteria (a ratio between 0 and 1) represents user satisfaction with the operational dependability of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines inherent reliability parameters, maintainability parameters, and logistics support effectiveness that constitute the type of downing events for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Operational Dependability (Do), where: Do = (MTBOMF)/(MTBOMF + MCMTOMF). Evaluation criteria: percentage that the AIS Do enables effective performance of the assigned tasks."*

✪ **Define** the inherent reliability parameters, maintainability parameters, and logistics support effectiveness issues that constitute an operationally dependable AIS and associated subsystems.

✪ **Define** the type of inherent hardware, software and firmware failures, induced user and maintainer failures, and could not duplicate failures that constitute operational mission failures for the AIS and associated subsystems.

✪ **Define** the type of system repair time that constitute corrective maintenance time for the AIS and associated subsystems.

1. **Mean Time Between Operational Mission failures (MTBOMF).** This MOP pertains to the average time between failures or unacceptable degradation of essential system functions. This MOP furnishes information required to calculate dependability. Operational mission failures do not necessarily occur during a mission; they merely must or could have mission impact. Quantifiable objective evaluation criteria (average in hours) represents user satisfaction with the MTBOMF of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of operational mission failures that constitute downing events for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean Time Between Operational Mission Failures (MTBOMF), where: MTBOMF = (Number of operating hours)/(Number of operational mission failures). Evaluation criteria: MTBOMF that the AIS enables effective performance of the assigned tasks."*

2. **Mean Corrective Maintenance Time for Operational Mission Failures (MCMTOMF).** This MOP pertains to the average total elapsed time, as the result of a critical failure, required to repair and restore a system to full operating status. This MOP furnishes information required to calculate operational dependability. Quantifiable objective evaluation criteria (average in hours) represents user satisfaction with the MCMTOMF of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of operational mission failures that constitute corrective maintenance time for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean Corrective Maintenance Time for Operational Mission Failures (MCMTOMF), where: MCMTOMF = (Total corrective maintenance time for operational mission failures)/(Number of operational mission failures). Evaluation criteria: MCMTOMF that the AIS enables effective performance of the assigned tasks."*

(c) **Mean Time Between Maintenance (MTBM) MOS.** This MOS pertains to logistics reliability, the average elapsed time between on-equipment maintenance events consisting of corrective maintenance actions (inherent, induced, and no-defect), and preventive maintenance actions. This MOS answers the question: Does the AIS maintenance assure that users can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria (average time in hours) represents user satisfaction with the time required for maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of maintenance required for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean Time Between Maintenance (MTBM), where: MTBM = (Total operating time)/(Total number of maintenance events). Evaluation criteria: MTBM that the AIS enables effective performance of the assigned tasks."*

✪ **Define** the type of test preparations, troubleshooting, remove and replacement of components, repairs, adjustments, and functional checks that constitute unscheduled maintenance events for the AIS and associated subsystems.

✪ **Define** the type of inspections, detections, or corrections on incipient failures before they occur or before they develop into major defects that constitute scheduled maintenance events for the AIS and associated subsystems.

APPENDIX N Recommendations for AIS ORDs

1. **Mean Time Between Unscheduled Maintenance (MTBCM) MOP.** This MOP pertains to the average elapsed time between on-equipment, corrective maintenance actions (inherent, induced, and no-defect). This MOP answers the question: Does the AIS corrective maintenance assure that users can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria (average time between unscheduled maintenance) represents user satisfaction with the time required for corrective maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of corrective maintenance required for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State**, if appropriate, **the following**: *"Mean time between unscheduled maintenance (MTBCM), where: MTBCM = (Total operating time)/(Number of corrective maintenance events). Evaluation criteria: MTBCM that the AIS enables effective performance of the assigned tasks."*

2. **Mean Time Between Scheduled Maintenance (MTBPM) MOP.** This MOP pertains to the average elapsed time between the performance of scheduled or preventive or scheduled maintenance events. This MOP answers the question: Does the AIS scheduled maintenance assure that users can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria (average time between scheduled maintenance) represents user satisfaction with the time required for preventative maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of maintenance deemed preventive or unscheduled for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State**, if appropriate, **the following**: *"Mean time between scheduled maintenance (MTBPM), where: MTBPM = (Total operating time)/(Number of preventative maintenance events). Evaluation criteria: MTBPM that the AIS enables effective performance of the assigned tasks."*

(d) **Maintenance Ratio (MR) MOPs.** This MOS pertains to the average maintenance work-hours expended over the operational life of the system, covering corrective maintenance actions (inherent, induced, and no-defect), and preventative maintenance actions. This MOS answers the question: Does the AIS maintenance assure that users can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria denoting (the ratio or percentage of time the system requires maintenance) represents user satisfaction with the time required for maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of maintenance (corrective and preventive) required for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State**, if appropriate, **the following**: *"Maintenance Ratio (MR), where: MR = (Total corrective and preventive maintenance hours expended)/(Total system possessed time). Evaluation criteria: MR that the AIS enables effective performance of the assigned tasks."*

- **Define** the type of test preparations, troubleshooting, remove and replacement of components, repairs, adjustments, and functional checks that constitute unscheduled maintenance events and time for the AIS and associated subsystems.

- **Define** the type of inspections, detections, or corrections on incipient failures before they occur or before they develop into major defects that constitute scheduled maintenance events and time for the AIS and associated subsystems.

1. **Mean Corrective Maintenance Time (MCMT) MOP.** This MOP pertains to the average elapsed time to correct malfunctions, including preparation for test, troubleshooting, removal and replacement of components, repair, adjustment, functional checks, et cetera. This MOP answers the question: Does the AIS corrective maintenance assure that users

can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria (average corrective maintenance time) represents user satisfaction with the time required for corrective maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of corrective maintenance required for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean corrective maintenance time (MCMT), where: MCMT = (Total down time for corrective maintenance)/(Number of corrective maintenance events). Evaluation criteria: MCMT that the AIS enables effective performance of the assigned tasks."*

2. **Mean Preventive Maintenance Time (MPMT) MOP**. This MOP pertains to the average elapsed time to prevent malfunctions, including inspections, detections, or corrections of incipient failures either before they occur or before they develop into major defects such as adjustments. This MOP answers the question: Does the AIS preventive maintenance assure that users can accomplish assigned tasks effectively? Quantifiable objective evaluation criteria (average preventive maintenance time) represents user satisfaction with the time required for preventative maintenance of the AIS to support the performance of assigned tasks effectively. Quantifiable subjective evaluation criteria defines the type of maintenance deemed preventive or unscheduled for the AIS and associated subsystems. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State**, if appropriate, **the following**: *"Mean preventive maintenance time (MPMT), where: MPMT = (Total down time for preventive maintenance)/(Number of corrective maintenance events). Evaluation criteria: MPMT that the AIS enables effective performance of the assigned tasks."*

(e) **System Survivability MOS**. System survivability MOS for an AIS address the administrative and physical controls, communication controls, data integrity, and post-processing controls capabilities of the AIS to support mission requirements. This MOS and associated MOPs, thresholds, and objectives apply to all tasks unless otherwise noted. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"This MOS measures the adequacy of administrative and physical controls, communication controls, data integrity, and post-processing controls capabilities of the AIS to support mission requirements. Its evaluation criteria represents the aggregate of such characteristics as administrative and physical controls, communication controls, data integrity, and post-processing controls."*

1. **Administrative and Physical Controls**. Administrative and physical controls adequacy pertains to the protection of data processing operations required to perform assigned tasks effectively, and covers the use of guards, locks, badges, and software access controls such as passwords and lockwords. This system survivability characteristic applies to the suitability of any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology:

✪ **State the following**: *"Administrative and physical controls measures the protection of data processing operations to meet mission needs. Evaluation criteria: User satisfaction of planned administrative and physical controls, based on requirements described in paragraph 6.e that address the AIS threat delineated in paragraphs 2, 2.a, and 2.b."*

2. **Communication Controls**. Communication controls adequacy pertains to complete data transmission and receipt by authorized personnel, terminal, or computer recipients required to perform assigned tasks effectively and covers the secure use of terminals, networks, and connections. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology:

✪ **State the following**: *"Communication controls measures complete data transmission and receipt by authorized personnel, terminal, or computer recipients to meet mission needs. Evaluation criteria: User satisfaction of planned communication controls, based on requirements described in paragraph 6.e that address the AIS threat delineated in paragraphs 2, 2.a, and 2.b."*

3. **Data Integrity**. Data integrity adequacy pertains to the successful processing of data required to perform assigned tasks effectively, and covers the prevention of security violations that inhibit effective data processing. This system survivability characteristic applies to the suitability of any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology:

✪ **State the following**: *"Data integrity measures to the successful processing of data to meet mission needs. Evaluation criteria: User satisfaction of planned data integrity, based on requirements described in paragraph 6.e that address the AIS threat delineated in paragraphs 2, 2.a, and 2.b."*

4. **Post-processing Controls**. Post-processing controls adequacy pertains to determining (a) all transactions are processed once and only once, (b) transactions and processing were complete, accurate, and authorized, (c) distribution of processing results was made to only authorized recipients, (d) data and the required use of system resources were recoverable, and (e) there is an ability to detect and isolate violations required to perform assigned tasks effectively. Post-processing controls covers the validation of compliance with predetermined systems requirements through post-operations analysis of input, processing, and output information. This system survivability characteristic applies to the suitability of any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology:

✪ **State the following**: *"Post-processing controls measures the authorization of transaction, processing, and recovery methods to meet mission needs. Evaluation criteria: User satisfaction of planned Post-processing controls, based on requirements described in paragraph 6.e that address the AIS threat delineated in paragraphs 2, 2.a, and 2.b."*

(f) **Human Supportability**. The human supportability for an AIS address the adequacy of manpower and personnel support, training and training support, technical data, human factors engineering, and safety and health hazard capabilities of the AIS to support mission requirements. This MOS and associated MOPs, thresholds, and objectives apply to all tasks unless otherwise noted. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"This MOS measures the adequacy of manpower and personnel support, training and training support, technical data, human factors engineering, and safety and health hazard capabilities of the AIS to support mission requirements. Its evaluation criteria represents the aggregate of such characteristics as manpower and personnel support, training and training support, technical data, human factors engineering, and safety and health hazard capabilities."*

1. **Manpower and Personnel Support Adequacy**. Manpower and personnel support adequacy pertains to the utility of operations and maintenance personnel required to execute assigned tasks effectively. This human supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Manpower and personnel support adequacy measures the utility of operations and maintenance personnel required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned manpower and personnel support requirements, based on requirements described in paragraph 5.C.(1)."*

2. **Training and Training Support Adequacy**. Training and training support adequacy pertains to the utility of training and training support required to execute assigned tasks effectively. This human supportability characteristic applies to the suitability of

executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"Training and training support adequacy measures the utility of planned training and training support required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned training and training support requirements, based on requirements described in paragraph 5.C.(2)."*

3. **Technical Data Adequacy**. Technical data adequacy pertains to the utility of technical data required to execute assigned tasks effectively. This human supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"Technical data adequacy measures the utility of planned technical data required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned technical data requirements, based on requirements described in paragraph 5.C.(3)."*

4. **Human Factors Engineering Adequacy**. Human factors engineering adequacy pertains to the utility of implemented human factors engineering required to execute assigned tasks effectively. This human supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"Human factors engineering adequacy measures the utility of planned human factors engineering required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned human factors engineering requirements, based on requirements described in paragraph 5.C.(4)."*

5. **Safety and Health Hazard Adequacy**. Safety and health hazard adequacy pertains to the utility of planned safety and health hazard considerations required to execute assigned tasks effectively. This human supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"Safety and health hazard adequacy measures the utility of planned safety and health hazard conditions required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned safety and health hazard condition requirements, based on requirements described in paragraph 5.C.(5)."*

(g) **Infrastructure Supportability**. This infrastructure supportability MOP for an AIS address the adequacy of transportation and basing, facility support, supply support, support equipment, and software support capabilities of the AIS to support mission requirements. This MOS and associated MOPs, thresholds, and objectives apply to all tasks unless otherwise noted. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"This MOS measures the adequacy of transportation and basing, facility support, supply support, support equipment, and software support capabilities of the AIS to support mission requirements. Its evaluation criteria represents the aggregate of such characteristics as transportation and basing, facility support, supply support, support equipment, and software support capabilities."*

1. **Transportability and Basing Adequacy**. Transportation and basing adequacy pertains to the utility of planned transportation and basing required to execute assigned tasks effectively. This infrastructure supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State the following**: *"Transportation and basing adequacy measures the utility of planned transportation and basing required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned transportation and basing requirements, based on requirements described in paragraph 6.B."*

2. **Facility Support Adequacy**. Facility support adequacy pertains to the utility of planned facility support required to execute assigned tasks effectively. This infrastructure supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Facility support adequacy measures the utility of planned facility support required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned facility support requirements, based on requirements described in paragraph 5.E.(2)."*

3. **Supply Support Adequacy**. Supply support adequacy pertains to the utility of planned supply support required to execute assigned tasks effectively. This infrastructure supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Supply support adequacy measures the utility of planned supply support required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned supply support requirements, based on requirements described in paragraph 5.E.(1)"*

4. **Support Equipment Adequacy**. Support equipment adequacy pertains to the utility of planned support equipment required to execute assigned tasks effectively. This infrastructure supportability characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Support equipment adequacy measures the utility of planned support equipment required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned support equipment requirements, based on requirements described in paragraph 5.B."*

(h) **Software Supportability**. The software supportability MOS for an AIS address the software maintainability, software maturity, and software support resource capabilities of the AIS to support mission requirements. This MOS and associated MOPs, thresholds, and objectives apply to all tasks unless otherwise noted. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"This MOS measures the adequacy of software maintainability, software maturity, and software resource supportability capabilities of the AIS to support mission requirements. Its evaluation criteria represents the aggregate of such characteristics as software maturity, software maintainability, and software resource supportability capabilities."*

1. **Software Maturity Adequacy**. Software maturity pertains to the progress of the software development in its evolution to meet mission needs; and is determined by the adequacy of software change rates, software change implementation rates, software change closure rates, and software change severity. This software support characteristic applies to the suitability of any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Software maturity measures the progress of the software development in its evolution to meet mission needs. Evaluation criteria: User satisfaction with software maturity rates, based on based on requirements described in paragraph 5.D.(1)."*

2. **Software Maintainability Adequacy**. Software maintainability pertains to the capability of the software to be maintained by operational users; and is determined by the adequacy of documentation, organization, descriptiveness, and traceability; and the adequacy of software source code modularity, consistency, simplicity, expandability, testability, and traceability. This software support characteristic applies to the suitability of any assigned task.

Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Software maintainability measures the capability of the software to be maintained by operational users. Evaluation criteria: User satisfaction with software maintainability, based on requirements described in paragraph 5.D.(2)."*

3. **Software Support Resources Adequacy**. Software support resources adequacy pertains to the utility of planned software support required to execute assigned tasks effectively. This software support characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Software support resource adequacy measures the utility of planned software support required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned software support resource requirements, based on requirements described in paragraph 5.D.(3)."*

4. **Software Life Cycle Support Adequacy**. Software support resource adequacy pertains to the utility of planned software life cycle support required to execute assigned tasks effectively. This software support characteristic applies to the suitability of executing any assigned task. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State the following**: *"Software life cycle support adequacy measures the utility of planned software life cycle support required to execute assigned tasks effectively. Evaluation criteria: User satisfaction of planned software life cycle support requirements, based on requirements described in paragraph 5.D.(4)."*

c. **Critical System Characteristics**. Critical system characteristics pertain to those design features that determine how well the proposed concept or system will perform in its intended environment. Software engineering is the critical system characteristic for AIS operations. Software engineering pertains to principles of software development intended to reduce development risk and improve development disciplines under the evolutionary, incremental software acquisition process. Address the following information in this paragraph.

✪ **Outline** the capability requirements by increments.

✪ **Insert** a table to matrix the capabilities of each software incremental development to the assigned tasks described in paragraph 4.a.1 and its subparagraphs.

5. **Integrated Logistics Support**. Integrated logistics support pertains to a disciplined, unified, and iterative approach to the management and technical activities necessary to integrate support considerations into system and equipment design; develop support requirements that are related consistently to readiness objectives, to design, and to each other; acquire the required support; and provide required support during operational phase at minimum cost.

a. **Maintenance Planning**. Maintenance planning pertains to the process conducted to evolve and establish maintenance concepts and requirements for the lifetime of the system. Address the following information in this paragraph.

✪ **Develop** maintenance concepts using Repair Level Analysis (RLA) trade studies.

✪ **Determine** repairable, commercial NDI maintenance strategy.

✪ **Describe** the planning approach for contract versus organic repair.

✪ **Describe** the software maintenance concept.

(1) **Organizational Maintenance Concept**. Organization maintenance concept pertains to the user organization as responsible for performing maintenance on its assigned equipment. Organizational maintenance activities cover inspecting, servicing, lubricating, adjusting, and replacing parts, minor assemblies, and subassemblies. Address the following information in this paragraph.

✪ **Identify** the maintenance functional requirements and maintenance concept.

✪ **Specify** needed organic and interim contractor support.

- **Outline** maintenance tasks, support, documentation, and inter-service organic and contractor mix workloads.

(2) **Depot Maintenance Concept**. Depot maintenance concept pertains to the organization — DoD or contractor — responsible for supporting lower level maintenance by providing technical assistance and performing that maintenance beyond their responsibility or capability, providing stocks of serviceable equipment, or using more extensive facilities for repair than are available in organizational-level maintenance activities. Depot maintenance activities cover major overhaul or a complete rebuild of parts, assemblies, subassemblies, and end items to include the manufacture of parts, modification, testing, and reclamation as required. Address the following information in this paragraph.

- **Identify** maintenance functional requirements and maintenance concept.
- **Specify** needed organic and interim contractor support.
- **Outline** baseline planning approach for contract, organic, inter-service repair mix, and time phasing requirements.

b. **Support Equipment**. Support equipment pertains to all mobile and fixed equipment required to support system operations and maintenance. Support equipment covers associated multi-use end items; ground handling and maintenance equipment; tools, metrology and calibration equipment; and test equipment. Address the following information in this paragraph.

- **Identify** needed standard, commercial NDI support equipment.
- **Specify** the desired test and fault isolation capabilities for automated test equipment in terms of affordable and realistic probabilities.
- **Describe** the depot-level support equipment requirement to support the system throughout the system life cycle.

c. **Human Systems Integration**. Human systems integration pertains to the consideration of manpower, personnel, training, human factors engineering, safety, and health hazards as factors towards readiness, force structure, affordability, and wartime operational objectives.

(1) **Manpower and Personnel**. Manpower and personnel pertain to the identification and acquisition of military and civilian personnel with the skills and grades required to operate and support the system over its lifetime at peacetime and wartime rates. Address the following information in this paragraph.

- **Specify** thresholds and objectives for manpower (authorizations, specialty codes, skill level, high drivers).
- **Specify** thresholds and objectives for personnel (aptitudes, knowledge, skills, specialty code structure, high drivers) requirements.

(2) **Training and Training Support**. Training and training support pertains to the processes, procedures, techniques, training devices, and equipment used to train civilian and active duty and reserve military personnel to operate and support the system. Training curriculum covers initial as well as continuation training for individuals and crews; new equipment training; initial, formal, and on-the-job training; and logistics support planning for training equipment and training device acquisitions and installations. Address the following information in this paragraph.

- **Specify** thresholds and objectives for training (methods, training system concept, high drivers) requirements.
- **Identify** operations and maintenance training concepts.
- **Describe** depot training requirements for maintenance, engineering, and software support personnel.
- **Do not document** specific equipment to be purchased.

(3) **Technical Data**. Technical data pertains to scientific or technical information recorded in any form or medium (e.g., hard copy, CD-ROM and video tapes) such as manuals, drawings, and documentation of computer programs or related software. Address the following information in this paragraph.

✪ **Specify** user-unique requirements for technical data (timeliness, validation and verification, user participation, special style and format, update medium and distribution, and technical orders) development.

(4) **Human Factors Engineering**. Human factors engineering pertain to the development of effective person-machine interfaces and preclude system characteristics that require extensive cognitive, physical, or sensory skills; require complex manpower or training intensive tasks; or result in frequent or critical errors. Address the following information in this paragraph.

✪ **Specify** thresholds and objectives for human factors engineering (methodologies, high drivers) requirements.

✪ **Highlight** the human performance and human-in-loop issues as outlined in the IMPACTS Program Plan.

✪ **Describe** the man-machine interface requirements for the AIS with regard to system operations and system maintenance.

(5) **Safety and Health Hazards**. Safety and health hazards pertain to the application of scientific and engineering principles towards identifying and reducing hazards associated with system operation and support with the objective of designing the safest possible system consistent with mission requirements and cost-effectiveness. Address the following information in this paragraph.

✪ **Specify** thresholds and objectives for safety (lessons learned) and health hazards analysis (lessons learned, high drivers) requirements.

d. **Computer Resources**. Computer resources pertain to the facilities, hardware, system software, software development and support tools, documentation, and people needed to operate and support computer systems. Address the following information in this paragraph.

✪ **Describe** the computer resource constraint (language, hardware, database, architecture, and interoperability) requirements.

✪ **Identify** spare memory, throughput, and computer memory growth requirements.

(1) **Software Maturity**. Software maturity pertains to the progress of the software development in its evolution to be reliable. Software reliability pertains to the probability that the software will contribute to failure-free system performance for a specified period of time under specific conditions. Address the following information in this paragraph.

✪ **State** software maturity requirements to include software change rates, software change implementation rates, software change closure rates, and software change severity for pre- and post-delivery of fielded software.

✪ **Establish** requirements with regard to patch-free software and number of mission critical problems acceptable before operational use.

(2) **Software Maintainability**. Software maintainability pertains to those software and computer support resource characteristics that affect the ability of software programmers/ analysts to change software. Software changes cover correcting errors, adding system capabilities, deleting features from programs, and modifying software to be compatible with hardware changes. Address the following information in this paragraph.

✪ **Identify** any automated tool requirements for software maintainability/trouble shooting.

✪ **Describe** software documentation requirements for software maintainability.

✪ **Describe** software source code requirements for software maintainability.

✪ **Describe** software implementation requirements for software maintainability.

✪ **Describe** software quality assurance standard requirements with regard to software design, development, and delivery to assure future re-competability of software support for the life of the software.

(3) **Software Support Resources**. Software support pertains to the utility of planned software support resources to perform assigned tasks effectively. Software support resources covers products, resources, and procedures that facilitate the support activities to

establish the operational baselines, to modify and install software, and to meet user requirements. Address the following information in this paragraph.

- **Describe** how products, resources, and procedures facilitate the support activities to establish the software operational baselines.
- **Describe** how products, resources, and procedures facilitate the support activities to modify and install software changes.
- **Identify** when the software support agency (SSA) must be functional (initial operational capability (IOC) declaration, full operational capability (FOC) declaration, et cetera) to provide for system updates, configuration control, and management of all computer programs and data.

(4) **Software Life Cycle Support**. Software life cycle support pertains to the adequacy of the software life cycle development processes as they affect the supportability of the developed software. Software life cycle support covers project and configuration management, management and technical personnel, support systems, and facilities support activities. Address the following information in this paragraph.

- **Describe** project and configuration management requirements to support the software life cycle.
- **Describe** managerial and technical personnel requirements to support the software life cycle.
- **Describe** support system and facility requirements to support the software life cycle.

e. **Other Logistics Considerations**. Other logistics support pertains to supplies, facilities, and land.

(1) **Supply Support**. Supply support pertains to all management actions, procedures, and techniques used to determine requirements to acquire, catalog, receive, store, transfer, issue, and dispose of secondary items. Supply support covers provisions for initial and replenishment supply support, and sustained logistics acquisition support for support and test equipment. Address the following information in this paragraph.

- **Identify** the contractual approach for provisioning initial supply support to support mission readiness.
- **Describe** the contractual and commercial-style inventory control management approach for acquiring, distributing, and replenishing inventory spares and repair parts to support mission sustainment.
- **Establish** the post production support (PPS) analysis requirement.

(2) **Facilities and Land**. facilities and land pertain to the permanent, semi-permanent, or temporary real property assets required to support the system, including conducting studies to define facilities or facility improvements, locations, space needs, utilities, environmental requirements, real estate requirements, and equipment. Address the following information in this paragraph.

- **Specify** facility and shelter requirements that are external and additional to the procured AIS.
- **Describe** facility-unique (e.g., hardening, electromagnetic pulse (EMP) protection, environmental effects, power sources, and life cycle cost) requirements.
- **Emphasize** environmental protection procedures.

6. **Infrastructure Support and Interoperability**. Infrastructure support and interoperability pertain to the compatibility of new system designs with the infrastructure that will support them, the identified unique infrastructure requirements to support the system, and the proper planning required to put the infrastructure support into place.

a. **Command, Control, Communications, and Intelligence (C3I).** C3I pertains to AIS-unique intelligence information requirements as well as AIS integration into the C3I architecture forecast to exist at the time the AIS is fielded. Address the following information in this paragraph.

- **Describe** the C3I constraints that may impact the mission needs.
- **Define** the desired C3I capability in the operational environment.

b. **Transportation and Basing**. Transportation and basing pertains to AIS deployability to/within theater as well as required basing and associated facility infrastructures. Address the following information in this paragraph.

- **Describe** the transportation and basing constraints that may impact satisfying the mission needs.
- **Define** the level of desired transportation and basing capability in the operational environment.
- **Define** for deployable facilities the setup and tear down time, manpower, and environmental conditions thresholds and objectives for field operations.

c. **Standardization, Interoperability, and Commonality**. Standardization, interoperability, and commonality pertain to the AIS joint use, procedural and technical interface, communications, protocols, and standards requirements to assure AIS interoperability with other Service, joint Service, and Allied systems. Address the following information in this paragraph.

- **Describe** the standardization, interoperability and commonality constraints that may impact satisfying the mission needs.
- **Define** the level of desired standards, interoperable, and commonality capability in the operational environment.

(1) **Standardization and Commonality**. Standardization pertains to the standard application program interfaces (API) for each common operating environment (COE) functional area used for information systems to operate effectively together. Commonality pertains to the common operating environment for information systems to operate effectively together. Address the following information in this paragraph.

- **Define** the COE.
- **Describe** the specific architecture for standard API to provide mission applications in the COE.
- **Describe** the specific architecture for standard API to provide support applications in the COE, to cover such function areas as:
  - √ **Administration Functions** (network administration, system administration, database administration, and security administration).
  - √ **Communication Functions** (message processing, communications, correlation, database management, and Mapping, Charting and Geodesy).
  - √ **Managerial Functions** (database management, file management, executive manager, alerts, and office automation,).
  - √ **Service Functions** (on-line support, multimedia support, data interchange services, network services, and distributed computing services).

(2) **Interoperability**. Interoperability pertains to the ability of systems, units, or forces to provide services to or accept services from other systems, units, or forces and to use the services exchanged so exchanged to operate effectively together. Address the following information in this paragraph.

- **Identify** the all systems, units, or forces that the AIS acquisition must maintain interoperability capabilities.
- **Describe** and correlate each identified AIS interoperability requirement with specific operational and assigned task to meet the mission need.

d. **Mapping, Charting, and Geodesy (MCG) Support**. MCG pertains to any cartographic materials, digital topographic data, and geodetic data needed for AIS employment. Address the following information in this paragraph.

- **Describe** the MCG constraints that may impact satisfying the mission needs.
- **Define** the level of desired MCG capability in the operational environment.

e. **Environment Support**. Environment support pertains to physical factors, operational locations, electronics, and advanced technologies, as well as behavioral factors, personnel perceptions, emotions, and cultural aspects that the mission of AIS is expected to be performed.

For managers, the MIS operational environment (whether automated or manual) concerns a communicative process where data are accumulated, processed, stored, and transmitted to appropriate personnel within the organization for the purpose of making decisions to support organizational objectives and needs. Address the following information in this paragraph.

- **Describe** the impact of the operational environment in which the mission needs are expected to be performed.
- **Define** the level of desired mission capability in the operational environment.
- **Describe** the AIS system survivability issues in the operational environment with regard to:
  - √ Administrative and physical controls.
  - √ Communication controls.
  - √ Data integrity.
  - √ Post-processing controls.

7. **Force Structure**. Force structure pertains to the number of AIS systems, subsystems, spares, and training units required to achieve mission needs. Address the following information in this paragraph.

- **Estimate** the number of AIS systems, subsystems (nodes), spares and training units required.
- **Identify** the type and number of hardware platforms that will employ the systems and subsystems under development and procured to meet mission needs.

8. **Schedule Considerations**. Schedule considerations pertain to the acquisition milestone timetable for procuring the AIS. Address the following information in this paragraph.

- **Define** the acquisition actions required for the AIS to attain initial operational capability (IOC) and full operational capability (FOC) declaration.
- **Highlight** the AIS operational capability (number of operational systems, operational and support personnel, facilities, and organization and depot maintenance support elements) necessary to declare IOC and FOC.
- **Highlight** the level of performance (operational effectiveness and operational suitability key parameters thresholds) necessary to declare IOC and FOC.
- **Specify** the projected AIS availability time frame objective and the impact of not meeting the window time frame for IOC declaration.
- **Define** the required action and desired suspense dates (e.g., RAA date, projected trial period dates, required support capability dates) for attaining IOC.

Requirements Correlation Matrix

A requirements correlation matrix (RCM) is a three-part summary attachment to the ORD, addressed in three parts: Part 1: The Requirements Correlation Matrix; Part 2: Supporting Rationale for System Characteristics and Capabilities Sheet; and Part 3: Rationale and Needs/ Requirements Change Sheet.

RCM Part 1: The Requirements Correlation Matrix (RCM). RCM Part 1 pertains to summarizing in matrix form those system capabilities, characteristics, objectives, thresholds, and key parameters germane to the operational effectiveness and operational suitability of the acquired automated information system (AIS). Assigned tasks are the ORD-derived measures of effectiveness (task effectiveness rate) needed for an MIS to accomplish its military objectives, missions, or tasks. System capabilities are ORD-derived measures of performance (such as information accuracy, currency, timeliness, etc.) needed for an AIS to accomplish military objectives, missions, or tasks. System characteristics are ORD-derived design features (weight, size, shape, etc.) needed for a system to accomplish approved military objectives, missions, or tasks. A threshold is a minimum acceptable operational value for a system capability or characteristic below which the utility of the AIS becomes questionable. An objective is an

optimal operational value equal to or greater than a corresponding threshold value. Key parameters are capabilities and characteristics so significant that failure to meet the their threshold is cause for the concept or system selection to be reevaluated or the program to be reassessed or terminated. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State** at the top of the page the following:

REQUIREMENTS CORRELATION MATRIX

PART I AS OF DATE: _____."

- **Construct** the RCM (see Tables N-1 and N-2) template.
- **State** the critical operational issue (COI) — effectiveness or suitability.
- **Denote** for each operational effectiveness COI the following:

√ **Specify** in the "*System Capabilities and Characteristics*" column each assigned task associated with the COI, and each information value performance metric associated with the assigned task (reference figure B-2).

√ **Specify** in the "*Thresholds*" column the minimum acceptable task effectiveness rate for each assigned task, and associated objective criteria for the assigned task information value performance metrics.

√ **Specify** in the "*Objectives*" column the optimal task effectiveness rate for each assigned task, and associated objective criteria for the assigned task information value performance metrics.

√ **Place** an "*," where appropriate, by those assigned tasks and associated information value performance metrics in whose threshold(s) denote key parameters to the AIS operational effectiveness.

- **Denote** for each operational effectiveness COI the following:

√ **Specify** in the "*System Capabilities and Characteristics*" column those standard suitability performance metrics (reference figure B-3).

√ **Specify** in the "*Thresholds*" column the minimum acceptable performance criteria for each suitability performance metric, where appropriate.

√ **Specify** in the "*Objectives*" column the optimal performance criteria for each suitability performance metric, where appropriate.

√ **Place** an "*," where appropriate, by those suitability performance metrics whose threshold(s) denote key parameters to the AIS operational suitability.

RCM Part 2: Supporting Rationale for System Characteristics and Capabilities Sheet. RCM Part 2 pertains to the reasoning for assigning thresholds values (minimum acceptable operational values) to specific system capabilities and characteristics. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

- **State** at the top of the page the following:

APPENDIX N Recommendations for AIS ORDs

PLANNING ASSIGNED TASK ILLUSTRATIONS	
Create training standards and requirements.	Forecast logistics support resource requirements.
Design training documentation and courseware.	Formulate recommended instructional sequences.
Determine aeromedical evacuation needs.	Plan capacity, orders and production requirements.
Develop CBI, CAI, IVD, and CDC training plans.	Project training resource cost estimates.
ORGANIZING ASSIGNED TASK ILLUSTRATIONS	
Allocate training resource availability and reserves.	Maintain time and attendance records.
Categorize aircraft aircrews, parts, and equipment.	Organize shop floor assignments.
Classify training instructors, resources, and courses.	Schedule theater and unit level airlift missions.
Inventory product requisitions and acquisitions.	Select maintenance crew shifts for wartime surge.
DIRECTING ASSIGNED TASK ILLUSTRATIONS	
Administer CBI, CAI, and IVD training.	Instruct personnel on ground safety procedures.
Communicate airlift mission status to theater CINC.	Lead en route air traffic to recovery destination.
Coordinate resources for generating airlift sorties.	Report budget and general ledger accounts.
Direct inbound and outbound airlift missions.	Train personnel on aerospace ground equipment.
CONTROLLING ASSIGNED TASK ILLUSTRATIONS	
Control cost receipts and expenditures.	Modify scheduled events for airlift missions.
Evaluate student performance.	Monitor launch, en route, and recovery missions.
Inspect inventories stock, parts, and equipment.	Regulate student performance, status, and awards.
Measure training course effectiveness.	Track inbound and outbound air traffic.
NOTE: AIS assigned tasks are performed by "people" using computer systems called AISs. To write an assigned task: (a) Begin by selecting an appropriate task-action verb. (b) Conclude by operationally stating the task that personnel must perform to meet mission needs. When writing the assigned task, assume that the assigned task can be performed (though ineffectively) without the use of an AIS.	

Table N-1 Illustrations of Assigned Tasks

APPENDIX N Recommendations for AIS ORDs

OPERATIONAL EFFECTIVENESS	ORD I		ORD II		ORD III	
System Capabilities & Characteristics	Thres-holds	Objec -tives	Thres-holds	Objec -tives	Thres-holds	Objec -tives
Critical Operational Issue X.						
1. Task (Effectiveness Rate)	TBD	XX%	XX%	XX%	XX%	XX%
a. Task Timeliness MOP.	TBD	XX	XX	XX	XX	XX
b. Accuracy MOP.	TBD	XX	XX	XX	XX	XX
c. Currency MOP.	TBD	XX	XX	XX	XX	XX
d. Completeness MOP.	TBD	XX	XX	XX	XX	XX
e. Relevancy MOP.	TBD	XX	XX	XX	XX	XX
f. Format MOP.	TBD	XX	XX	XX	XX	XX
2. Task (Effectiveness Rate)	TBD	XX%	XX%	XX%	XX%	XX%
a. Task Timeliness MOP.	TBD	XX	XX	XX	XX	XX
b. Accuracy MOP.	TBD	XX	XX	XX	XX	XX
c. Currency MOP.	TBD	XX	XX	XX	XX	XX
d. Completeness MOP.	TBD	XX	XX	XX	XX	XX
e. Relevancy MOP.	TBD	XX	XX	XX	XX	XX
f. Format MOP.	TBD	XX	XX	XX	XX	XX
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table N-2 AIS Operational Effectiveness RCM

REQUIREMENTS CORRELATION MATRIX

PART II AS OF DATE: _____ ."

✪ **Specify** the following for each system capability/characteristic having designated threshold values.

√ The parameter number (Parameter X) and the associated system capability/ characteristic in "*bold*" type.

√ The specific studies, analyses, threat assessments, modeling, or other reference sources including military judgment that justify and substantiate each system characteristic threshold.

RCM Part 3: Rationale and Needs/Requirements Change Sheet. RCM Part 3 pertains to the reasoning for changes in system characteristics, performance, and supporting parameters. Address the following information in this paragraph, replacing terms underlined with the appropriate system terminology.

✪ **State** at the top of the page the following:

REQUIREMENTS CORRELATION MATRIX

PART III AS OF DATE:_____."

✪ **Specify** the following for each system capability/characteristic changed in response to changes in needs/requirements.

√ The parameter number (Parameter X) and the associated system capability/ characteristic in "*bold*" type.

√ The report title, document number, get-well date, and schedule showing the rational for changes in system characteristics, performance, and supporting parameters.

APPENDIX N Recommendations for AIS ORDs

OPERATIONAL SUITABILITY	ORD I		ORD II		ORD III	
System Capabilities & Characteristics	**Thres-holds**	**Objec -tives**	**Thres-holds**	**Objec -tives**	**Thres-holds**	**Objec -tives**
Critical Operational Issue Y. Does AIS readiness support mission requirements in the operational environment?						
1. Operational Availability (A_O)	TBD	XX%	XX%	XX%	XX%	XX%
a. Mean Time Between Downing Events (MTBDE).	TBD	XX	XX	XX	XX	XX
b. Mean Downtime (MDT).	TBD	XX	XX	XX	XX	XX
2. Operational Dependability (D_O)	TBD	XX%	XX%	XX%	XX%	XX%
a. Mean Time Between Operational Mission Failures (MTBOMF).	TBD	XX	XX	XX	XX	XX
b. Mean Corrective MaintenanceTime for Operational Mission Failures (MCMTOMF).	TBD	XX	XX	XX	XX	XX
3. Mean Time Between Maintenance (MTBM).	TBD	XX	XX	XX	XX	XX
a. Mean Time Between Scheduled Maintenance (MTBUM).	TBD	XX	XX	XX	XX	XX
b. Mean Time Between Scheduled Maintenance (MTBSM).	TBD	XX	XX	XX	XX	XX
4. Maintenance Ratio (MR).	TBD	XX	XX	XX	XX	XX
a. Mean Corrective Maintenance Time (MCMT).	TBD	XX	XX	XX	XX	XX
b. Mean Preventive Maintenance Time (MPMT).	TBD	XX	XX	XX	XX	XX

Table N-3 AIS Operational Suitability RCM

OPERATIONAL SUITABILITY	ORD I		ORD II		ORD III	
System Capabilities & Characteristics	**Thres-holds**	**Objec -tives**	**Thres-holds**	**Objec -tives**	**Thres-holds**	**Objec -tives**
Critical Operational Issue Z. Does AIS logistics support sustain mission requirements in the operational environment?						
5. Systems Survivability.	TBD	XX	XX	XX	XX	XX
a. Administration and Physical Controls.	TBD	XX	XX	XX	XX	XX
b. Communication Controls.	TBD	XX	XX	XX	XX	XX
c. Data Integrity.	TBD	XX	XX	XX	XX	XX
d. Post-processing Controls.	TBD	XX	XX	XX	XX	XX
6. Human Support.	TBD	XX	XX	XX	XX	XX
a. Manpower & Personnel Support.	TBD	XX	XX	XX	XX	XX
b. Training & Training Support.	TBD	XX	XX	XX	XX	XX
c. Technical Data Support.	TBD	XX	XX	XX	XX	XX
d. Human Factors Engineering Support.	TBD	XX	XX	XX	XX	XX
e. Safety & Health Hazard Conditions.	TBD	XX	XX	XX	XX	XX
7. Infrastructure Support.	TBD	XX	XX	XX	XX	XX
a. Transportation & Basing Support.	TBD	XX	XX	XX	XX	XX
b. Facilities Support.	TBD	XX	XX	XX	XX	XX
c. Supply Support.	TBD	XX	XX	XX	XX	XX
d. Support Equipment.	TBD	XX	XX	XX	XX	XX
8. Software Support.	TBD	XX	TBD	XX	TBD	XX
a. Maturity.	TBD	XX	TBD	XX	TBD	XX
b. Maintainability.	TBD	XX	TBD	XX	TBD	XX
c. Resource Support.	TBD	XX	TBD	XX	TBD	XX
d. Life Cycle Support.	TBD	XX	TBD	XX	TBD	XX

Figure N-4 AIS Operational Suitability RCM (cont.)

Version 2.0

PART V

Additional Addenda

Blank page.

APPENDIX

O

Additional Volume 1 Addenda

Blank page.

APPENDIX

O

Additional Volume 1 Addenda

CONTENTS **PAGE**

CHAPTER 2, DoD Software Acquisition Environment Addendum
Addendum A, *MIL-STD-498: What's New and Some Real Lessons-Learned* ... O-3
Addendum B, *Adopting MIL-STD-498: The Steppingstone to the US Commercial Standard* ... O-12
CHAPTER 4, *Engineering Software-Intensive Systems*, Addendum
Addendum B, *Software Reliability: A New Software OT&E Methodology* ... O-21
CHAPTER 5, *Ada: The Enabling Technology*, Addenda
Addendum C, *The Ada 95 Philosophy.* ... O-37
Addendum D, *Ada Implementation Lessons-Learned from SSC and CSC* ... O-41
CHAPTER 7, *Software Development Maturity*, Addendum
Addendum B, *Lessons-Learned While Achieving A CMM℠ Level 3 Rating* ... O-43
CHAPTER 8, *Measurement and Metrics*, Addenda
Addendum B, *Software Complexity* (McCabe) ... O-50
Addendum C, *Metrics: The Measure of Success* (Hughes) ... O-58
Addendum D, *Making Metrics Work Miracles* (Ogden ALC) ... O-76
Addendum E, *Swords and Plowshares: The Rework Cycles of Defense & Commercial Software* ... O-82
CHAPTER 10, *Software Tools*, Addendum
Addendum B, *Rate Monotonic Analysis: Did You Fake It?* ... O-95
CHAPTER 11, *Software Support*, Addendum
Addendum B, *Electronic Combat Model Re-engineering* (INEL) ... O-99
CHAPTER 13, *Contracting for Success*, Addendum
Addendum B, *Contracting for Success* (Gabig) ... O-115
CHAPTER 15, *Managing Process Improvement*, Addenda
Addendum B, *Training — Your Competitive Edge in the '90s* ... O-129
Addendum C, *Lessons-Learned from BSY-2's Trenches* ... O-135

Blank page.

CHAPTER 2
Addendum A

MIL-STD-498: What's New and Some Real Lessons-Learned

Reprinted from *CrossTalk: The Journal of Defense Software Engineering,* March 1996

Paul A Szulewski
David S. Maibor

ABSTRACT

In his June 29, 1994 memo, Secretary of Defense William J. Perry challenged Department of Defense (DoD) agencies (and industry) to move to greater use of performance and commercial specifications and standards, and shelved a host of military standards, including those related to software. After an intense lobbying effort by the DoD and industry, the DoD approved the use of MIL-STD-498 for two years, assuming a nongovernment software standard would replace it in that time frame. The US Navy and the US Air Force have issued waivers permitting MIL-STD-498 to be invoked on contracts. The Institute of Electrical and Electronics Engineers (IEEE) and the Electronic Industries Association (EIA) are working together to create a nongovernment software standard. Because MIL-STD-498 is new and being applied on selected projects, there is no published information on its practical use. This article briefly:

- Highlights MIL-STD-498 as the new way to develop software.
- Examines MIL-STD-498's application on a government-sponsored real-time guidance, navigation, and control project underway at the Draper laboratory.
- Reviews the effort to create nongovernment software standards.

MIL-STD-498: What's New – What's Different

Software Development in the 1980s — DoD-STD-2167A

DoD-STD-2167A, Defense System Software Development, was the standard used for most DoD-sponsored software development from 1988 to 1994. DoD-STD-2167A represented the approach to developing software for the 1980s. What were some of the key elements in DoD-STD-2167A?

Appendix O Additional Volume 1 Addenda

Customer-Developer Relationship

DoD-STD2167A attempted to balance the customer's concerns with the developer's concerns, i.e., the customer must have sufficient project oversight and control while some flexibility is given to the developer. Often, the result was an "*us versus them*" mentality or adversarial relationship.

Single Pass Waterfall Model Bias

DoD-STD2167A defined an eight-step development process:

1. System requirements analysis and design.
2. Software requirements analysis.
3. Preliminary design.
4. Detailed design.
5. Coding and unit test.
6. Computer software component integration and test.
7. Computer software configuration item test.
8. System integration and test.

Many projects assumed the process:

- Must begin with requirements analysis, followed by design, then coding.
- Conclude each step with a formal review.
- Progress through the activities once as a single pass.
- Include configuration management, quality assurance, deliverable and nondeliverable documentation in each step.

Baseline Control

As discussed above, the focus of a waterfall model is requirements. System and software-level requirements were frequently frozen as customer-controlled baselines.

Formal Milestone Reviews

DoD-STD-2167A required the developer to conduct formal reviews (system design review, preliminary design review, etc.) in accordance with MIL-STD-1521B. Since review requirements were seldom tailored, the developer had to frequently conduct "*dog and pony shows*," presenting extensive engineering information on the product.

Hard Copy Deliverable Data

DoD-STD-2167A identified 17 separate deliverable data items, with associated Data Item Descriptions (DIDs). Many projects required the developer to prepare and deliver most or all of the documents in hard copy format 30 to 60 days prior to the milestone reviews.

Functional Bias

DoD-STD-2167A assumed that system and software products were developed following a functional decomposition approach. "*Requirements*" were specified as capabilities with associated performance and interface parameters and represented what the item must do. "*Design*" details were specified as hardware or software configuration items, computer software components (software preliminary design), computer software units (software detailed design), and represented how the item satisfied its requirements.

APPENDIX O Additional Volume 1 Addenda

Software Trends in the 1990s

Over the past three to four years, new trends have emerged for the 1990s that represent some fundamental shifts in software development. What are these new trends?

The Integrated Product Team Approach

There is a strong effort underway to replace the traditional adversarial relationship with the integrated product team (IPT) approach. An IPT approach means all project stakeholders (contracting agency, developer, end user, support activity, test agency, and independent verification and validation agent) meet frequently and play an active role in product development.

The Three Rules of Software Development

Just as there are three rules to buying real estate (location, location, location), there is growing pressure for software organizations to follow the three rules of software development — process, process, process.

Plan Your Work – Work Your Plan

Along with the pressure to focus on process, organizations must document how that process will be applied to the project. Detailed plans must be prepared (or reused if available as a corporate asset). These plans serve two purposes: convince the customer that the developer has a valid software development approach and serve as the "*project bible*" by which the software team operates. Since plans are useless if they are not followed, planning your work and working your plan are both key elements. (The ISO 9000 community similarly states, "*Document what you do. Do what you document.*")

Supportability

For government projects, the support question remains a key concern: "*Who will support the delivered software?*" There is still strong reluctance to be sole-source dependent on the original developer; the acquirer expects software support to transition to a government organization.

Ada and Object-Oriented Methodologies

In 1991, US law required all DoD software be written in Ada where cost effective and in the absence of a waiver. Most Ada projects also follow an object-oriented methodology instead of the traditional functional decomposition approach.

A New World Order For MIL-STDs

On June 29, 1994, Secretary of Defense William J. Perry issued a memorandum that defined "*A New Way Of Doing Business.*" Up to this point, DoD policy mandated the use of various MIL-STDs on all projects (a project office must obtain a waiver to avoid using MIL-STDs). This new memo "*threw out all nonperformance specifications and standards*" and required project offices to obtain a waiver to use a MIL-STD. (Note: Performance specifications and standards such as MIL-STD-1553 for bus communication were exempted.)

The US Air Force and the US Navy issued blanket waivers allowing the use of MIL-STD-498 in 1994. The waivers are in effect until December 1996.

MIL-STD-498 Highlights

Four cornerstones support MIL-STD-498, Software Development and Documentation. What are these cornerstones? The cornerstones are the first four trends discussed above for the 1990s: the IPT approach, process, process, process, plan your work and work your plan, and supportability. Some of the specific changes found in MIL-STD-498 are discussed below.

Development Model – What's a Waterfall?

Since many projects applied DoD-STD-2167A as a single-pass waterfall model, MIL-STD-498 "*goes out of its way*" to dispel any waterfall bias. It clearly states in §5.1, "*The order of the requirements [in Section 5] is not intended to specify the order in which they must be carried out.*" Essentially, MIL-STD-498 identifies separate activities the developer's process must address (requirements analysis, design, configuration management, quality assurance, joint reviews, etc.). The developer is free to select the order and timing of the activities — the build approach — essentially all aspects of the process.

Product Requirements Under MIL-STD-498

Under DoD-STD-2167A's baseline control, it was often problematic to determine where requirements ended and design detail began. What's different under MIL-STD-498?

- **No more baselines or customer control.** Under MIL-STD-498, the only data the customer explicitly approves (along with changes) are plans. MIL-STD-498 doesn't mention baseline control of requirements. Requirements information is treated as all other information generated by the process; the developer must review the information and place it under internal configuration control. If the acquirer wants to exercise baseline control of requirements, that must be added into the Statement of Work. (NOTE: Under an IPT approach, the acquirer could be part of a joint review board, approving all changes to internally controlled data.)
- **Requirement versus design detail.** Examination of the DIDs for requirements specifications and design descriptions reveals identical information in many places. How do you decide whether information is a requirement or design detail? According to Section 3 of the requirements specifications DIDs: "*The degree of detail shall be guided by the following rule: Include those characteristics that are conditions for acceptance; defer to design descriptions those characteristics that the acquirer is willing to leave up to the developer.*" Under MIL-STD-498, a requirement is any characteristic that serves as a precondition for acceptance. If the acquirer does not care about that characteristic, it can be considered a design detail and does not need to be demonstrated as part of qualification testing.
- **Requirements analysis can take place whenever.** Under MIL-STD-498, defining and recording requirements may take place prior to design and code, after design and code, after installation, or all of the above. The new standard essentially says, "*Here are all the activities your process should cover. One is system requirements analysis, another is software requirements analysis. Describe your process and include where and when requirements analysis occurs.*"

Joint Technical and Management Reviews

Instead of the formal milestone reviews imposed by DoD-STD-2167A, MIL-STD-498 tasks the developer to propose the schedule and location of all joint reviews. Joint technical reviews are intended to review evolving software products (software and associated information in their natural form), not force the developer to construct elaborate

presentations. Joint management reviews are intended to keep management informed on project status, surface problems that cannot be resolved at technical reviews, and receive management commitment.

Documentation — The New Approach

Under DoD-STD-2167A, many developers and acquirers complained that projects overemphasized hard copy deliverable documentation, and de-emphasized a well-engineered product. MIL-STD-498 assumes the developer has planned an appropriate process for the software work. That process consists of two elements: the performance of activities and the generation of information. MIL-STD-498 explicitly states, "*Defining and recording engineering and planning information is an intrinsic part of the software process.*" (§5.1.1) However the information (requirements, design, test, plans, etc.) need not be in hard copy form and need not comply with a DID. Instead, MIL-STD-498 assumes the developer's process naturally generates planning and engineering information. When no deliverable data is required, the developer uses the corresponding DIDs as a checklist of information the process might generate. MIL-STD-498 goes on to caution the acquirer:

- Do not think the developer will not develop key planning and engineering information if you do not order its delivery on a contract data requirements list (CDRL).
- Only order deliverable data when there is a genuine need for it.
- Recognize that the preparation of deliverable data is an extra task for the developer and diminishes the time and resources available for the end product.
- Provide as much flexibility as possible regarding the due dates for CDRL items.

Development Methodology — Whatever Works

DoD-STD-2167A assumed system and software development followed a traditional functional decomposition approach. Today, object-oriented and other approaches define different development paradigms. Consequently, MIL-STD-498 attempts to keep requirements and design language as neutral as possible. To describe software architectural design, MIL-STD-498 introduces a single term "*software unit,*" which is simply defined as "*an element in the design of a computer software configuration item.*" (§3.45)

Supportability

With supportability as a cornerstone, MIL-STD-498 has added "*Preparing for Software Transition*" as an activity (§5.13). The standard also requires the developer to demonstrate that the deliverable software can be regenerated and maintained by software and hardware designated in the contract or approved by the acquirer (§5.13.7b).

Management Metrics and Process Improvement

MIL-STD-498 tasks the developer to use management indicators to manage the process and communicate its status to the acquirer. The developer proposes which metrics will be used in the SDP (§5.19.2). With the emphasis on process in MIL-STD-498 and the popularity of process improvement, MIL-STD-498 requires the developer to periodically assess the processes used on a project for suitability and effectiveness and identify any necessary and beneficial improvements (§5.19.7).

Real Experiences Using MIL-STD-498

This section describes a real project's experience with application of MIL-STD-498 for the first time in an organization. The proposed development was complex and had budget and schedule constraints. Two software configuration items (CIs) were planned.

The first CI would be re-engineered from a legacy system, and the architecture and a major portion of the code would be reused. Also, new capabilities would be added to the legacy system. The system is expected to be supported from 10 to 30 years, and the acquirer will be responsible for maintenance and support. The acquirer was also interested in being involved technically. The first CI can be characterized as follows:

- **Size.** Estimated total source lines of code (SLOC) — 90K of C code with 80% reuse.
- **Application domain.** Tactical system, real-time guidance, navigation, and control with a sophisticated graphical user interface (GUI) for user interaction.
- **Complexity.** Realtime, user interactive, time-constrained computing, over 100 interfacing elements.
- **Criticality.** Life-critical safety application with built-in hardware fail-safe backups.

The second CI would be a user interface and display generation product using some commercial-off-the-shelf (COTS) software and reused components to evolve a user interface that uses rapid prototyping. The second CI can be characterized as follows:

- **Size.** Estimated total SLOC—12K of C code with 70 percent COTS and reused code.
- **Application domain.** A sophisticated GUI for user interaction, guidance, navigation, and control.
- **Complexity.** Realtime, user interactive, and time-constrained computing.
- **Criticality.** Primary user interface with built-in hardware fail-safe backups.

This is a perfect scenario for using MIL-STD-498: an integrated product team atmosphere — reused and COTS software, two non-waterfall life cycle development models for the CIs and, in the interest of reducing development costs and accelerating the schedule, a desire to minimize time wasted in traditional formal reviews and formal documentation, yet assure a maintainable system.

Integrated Product Team

The acquirer has encouraged a positive IPT atmosphere. An aggressive independent verification and validation (IV&V) agent was assigned to help review the deliverables and periodically visit the developers in "*working sessions*." This tactic has generally provided timely and constructive criticism. In addition, the IV&V team has helped refine project plans, especially those related to risk management and safety provisions. The downside of this close cooperation has been an increase in the effort expended in communication, resolving problems, and making the software development visible. Since the acquirer wanted more control and visibility, additional planning and oversight activities have been required.

Informal Reviews

The acquirer agreed to periodic joint technical reviews instead of formal milestone reviews. This review concept marries technical aspects with some aspects of management reviews. The parties do not usually bring contracts types that would be required if contract modifications are proposed. These reviews typically happen at six-week intervals and focus on snapshots of works-in-progress. The materials presented require less than traditional formality and preparation, but there is still some issue with expectations on both sides. The acquirer, familiar with traditional reviews, still looks for completed products. It is difficult to plan in advance what is to be presented at each review, yet this level of detail must be in the development plan. Operationally, these reviews still look like formal reviews. Materials must be submitted in advance, minutes are taken, and action items tracked. Schedule, risks, and software metrics are also presented. This adaptation, in practice, has consumed nearly the same preparation and presentation time as did traditional reviews, and there are more of them.

APPENDIX O Additional Volume 1 Addenda

Delivered Documentation

The developers have planned a comprehensive set of deliverable documents that are based on those prescribed by MIL-STD-498 DIDs. Had the developers chosen to provide documents in "*contractor format,*" each document would have to be described in the plan. Since plans are the only deliverables "*formally approved*" by the acquirer, they have been subjected to a large amount of scrutiny. A detailed draft software development plan was prepared and submitted along with the proposal. Several revisions of the plan have, to date, been submitted but not yet approved. This problem should not repeat with documents that are not approved. For supportability reasons, the acquirer is also interested in other development documentation generated as a record of design decision making. Additional configuration management provisions have been taken to assure that all software products, not just source code, are managed and controlled. Even developer's notes that support design decisions are carefully controlled.

Flexible Development Model

The development model being followed is analogous to the contrast between a five-course meal and a buffet-style presentation. In a five-course meal, a diner is presented with an orderly progression of food beginning with an appetizer and ending with dessert. The diner knows what to expect and when things will happen. With a buffet, the diner may begin with any or all selections and end the same way. The diner may also return to the buffet as many times as desired. With either meal process, the diner is ultimately satisfied. The buffet style is usually the less expensive option.

The benefits to the developer of a buffet-style development model are obvious. The developer has the flexibility to mix and match activities to get the job done. In this case, nonwaterfall activities like re-engineering and rapid prototyping were planned before or in parallel with requirements engineering. Also, the software quality assurance team works with the development team during the development of software products, rather than acting as a gate at the end. The developer still needs to plan in advance how to get from "*A to B,*" and it must be documented and conveyed in an intelligent and understandable form to the acquirer. The acquirer must then approve the plan. Once the plan is approved, the acquirer assumes that you are following the plan until formally notified to the contrary. Since there are no assumptions made, the plan provides the acquirer all information about exactly what comes next. When the activities and processes invoked are nonstandard, they must be documented in the plan. It saves significant time if these nonstandard practices can be pulled off the shelf and your organization has used them before.

Summary of Lessons-Learned

To successfully execute a development effort using MIL-STD-498, your organization should have standard processes, tools, and methods on the shelf and ready to use in addition to the tailoring experience it takes to apply them to a particular project. Back to the buffet analogy; it's better if you have done it before and have a plan of attack already worked out. If not, you may spend all of your time strategizing and describing your dining approach and never have time to eat.

NONGOVERNMENT STANDARDS IN PLACE

EIA/IEEE J-STD-016-1995: The Nongovernment Twin to MIL-STD-498

A joint effort was undertaken by two key standards issuing organizations: The Institute of Electrical and Electronics Engineers (IEEE) and the Electronic Industries Association (EIA). In January 1996, the IEEE and EIA issued, for trial use, J-STD-016-1995, Standard for Information Technology, Software Life Cycle Processes, Software Development, Acquirer-Supplier Agreement. This nongovernment standard is technically equivalent to MIL-STD-498. All of the key themes discussed in the section "*MIL-STD-498: What's New — What's Different*" of this article appear in J-STD-016. Now available, this new standard can replace MIL-STD-498 and be invoked by the DoD without waivers.

ISO 12207: The New Software Standard for the World

Another key phenomenon is the official release on Aug. 1, 1995 of ISO 12207, Information Technology and Software Life Cycle Processes. The International Organization for Standardization (ISO) sponsored this new standard for software development, and it was ratified by 24 out of 25 countries. Since ISO 12207 is released as an international standard, many countries will likely feel obligated to adopt it—similar to the ISO 9000 standards experience. Since some aspects of ISO 12207 are more restrictive than MIL-STD-498, a key issue will be how the United States adopts the international standard, ISO 12207, without overturning the flexible "*1990s style*" themes of J-STD-016. The same IEEE/EIA working group that converted MIL-STD-498 to J-STD-016 is currently attempting to "*Americanize" ISO 12207 and create "US 12207*" — the US implementation of ISO 12207 (including the "*technical goodness*" of J-STD-016).

About the Authors

Paul A. Szulewski is a principal member of the technical staff at the Draper Laboratory. He has over 20 years experience in managing and developing software and software technology. Szulewski is actively involved in software development and is a key person in Draper's software process improvement effort. He has recently been involved with both the National Software Council and the National Software Data and Information Repository task forces initiated by the Pentagon.

Paul A. Szulewski
The Charles Stark Draper Laboratory, Inc.
555 Technology Square, MS 15
Cambridge, MA 02139
Voice: 617-258-1832
Fax: 617-258-3939
E-mail: pas@qmlink.draper.com

David S. Maibor is the principal author of DoD-STD-2167 and a key participant in the development of DoD-STD-2167A, DoD-STD-2168, MIL-STD-498, J-STD-016, and US 12207. As a principal of David Maibor Associates, Inc., he provides professional consulting, public seminars, and on-site training to the government and industry. Maibor also provides software capability evaluations and training and provides related software process improvement consulting services.

APPENDIX O Additional Volume 1 Addenda

David S. Maibor
David Maibor Associates, Inc.
P.O. Box 846
Needham, MA 02194
Voice: 617-449-6554
Fax: 617-455-8928
E-mail: maibor@aol.com

CHAPTER 2
Addendum B

Adopting MIL-STD-498: The Steppingstone to the US Commercial Standard

Reed Sorensen
Software Technology Support Center

INTRODUCTION

This article discusses a road map for readers to use in the adoption of MIL-STD-498 and the forthcoming US commercial standard for software development, which will include the technical content of MIL-STD-498.

What is the US Commercial Standard?

To compete in the global software market, you need to play by the global rules. These are often established by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). With the August 1995 approval of ISO/IEC 12207, *Information Technology — Software Life Cycle Processes*, the rules were established. ISO/IEC 12207 provides the core software process architecture. Each country has the option of *adapting* (not to be confused with *adopting*) the standard for their unique requirements.

> *The Joint Industry Working Group (JIWG) has initiated the project for the US adaptation of the ISO/IEC 12207. ... Once implemented, the standard will establish the basis for international trade and a common reference model for software in the US The standard provides the process framework for the acquisition, supply, development, operation, and maintenance of software. [1]*

The JIWG is working toward a December 1996 implementation of the US commercial standard to be designated ANSI (American National Standards Institute) 016.

Why Not Skip MIL-STD-498 and Just Wait for the US Commercial Standard?

Waiting may make sense if your software project starts after December 1996, the date when MIL-STD-498 is to be replaced. Otherwise, using MIL-STD-498 gives the organization experience in implementing the intent of the US commercial standard. Using MIL-STD-498 now is an advantage because it addresses the following key issues:

APPENDIX O Additional Volume 1 Addenda

- The documentation challenge.
- Cooperative teaming between the acquirer and the developer.
- Flexibility to allow the use of the developer's proven process.
- Supportability of the software.

Guidelines for the Successful Acquisition and Management of Software Intensive Systems states the logic of using MIL-STD-498.

> *Remember, **MIL-STD-498 is the preferred standard for all software-intensive Air Force systems.** If your on-going contract still stipulates compliance with -2167A, consider modifying it to require compliance with -498. If your program is too far along to benefit from changing to -498, be sure -2167A has been appropriately tailored! [2]*

THE ROAD MAP

The road map is based in part on the steps for applying MIL-STD-498 found in *MIL-STD-498 Overview and Tailoring Guidebook,* Jan. 31, 1996 [3]. The road map is summarized in Figure O-1.

The road map is used to adopt MIL-STD-498 in a single project referred to as "*Project A*" in Figure O-1. The project is assumed to be just one of several software development projects in an organization. Using the experience from Project A, other projects in the organization may adopt MIL-STD-498 or the US commercial standard.

The road map may be used by software developers. It may also be used by acquirers [4] who impose MIL-STD-498 as a contract requirement on the developer. The tasks covered in the road map may be done by the acquirer in one situation, or may be done by a developer in another situation. For example, tailoring may be done initially by the acquirer. A developer may do tailoring as a suggested modification to the acquirer's tailoring.

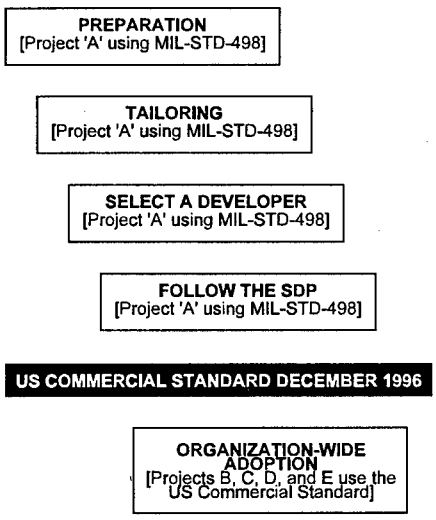


Figure O-1 Road Map Summary

PREPARE

The first tasks involve knowing where you are, i.e., understanding the standard, the software development process [5], general characteristics of the software being developed, the general approach to be used in developing the software and, of course, the requirements. Identifying and establishing communication with the parties that will constitute the software development or acquisition team is also critical.

Get MIL-STD-498 Training

CrossTalk articles, the standard itself, guidebooks, and workshops are all excellent sources to gain understanding. Tables O-1 and O-2 list pertinent articles and the sources of MIL-STD-498 and the guidebooks. The October 1995 issue of *CrossTalk* lists workshops available. (Also see the sidebar with this article.)

September 1994	DoD Policy on the Future of MILSPEC
February 1995	MIL-STD-498
April 1995	Changes from DoD-STD-2167A to MIL-STD-498
October 1995	How is MIL-STD-498 Being Used?*
November/ December 1995	MIL-STD-498 and the CMM: How Do They Relate?
February 1996	MIL-STD-498 and the CMM: The Mapping
March 1996	MIL-STD-498: What's New and Some Real Lessons Learned

*This article listed sources of training.

Table O-1 *CrossTalk* Articles Related to MIL-STD-498

FTP:	*diamond.spawar.navy.mil*
	MIL-STD-498
	Overview and Tailoring Guidebook
FTP:	*glider.logicon.com/pub/standards/498*
	MIL-STD-498
	Overview and Tailoring Guidebook
	Application and Reference Guidebook
Web:	*http://www.itsi.disa.mil/cfs/std498.html*
	MIL-STD-498
	Overview and Tailoring Guidebook
NOTE:	Each of these sites may be accessed through the STSC home page at *http://www.stsc.hill.af.mil.*

Table O-2 Sources for Downloading MIL-STD-498 and Related Documents

Consider the Context of the Project

The software being acquired or modified will either be a system in its own right, e.g., spares inventory system, or it will be embedded in a larger system such as an aircraft or submarine. If the software is embedded, consider the acquisition strategy being used for the larger system. The strategy being used is probably grand design (waterfall), incremental or evolutionary. Recognize the strategy for consideration with the software development process.

Use a Proven Software Development Process

The standard is written for the developer with a proven process. An acquirer needs to consider how the process will fit in the system acquisition strategy. For instance, if the system strategy is incremental, the acquirer considers how a prospective developer's

prototyping-based process might mesh with that strategy. Until a developer is selected, these considerations are done in a "*what-if*" frame of reference.

Use a Developer with a Track Record

With a proven process and a track record of success, an acquirer can be confident that the developer is likely to succeed. This confidence frees the acquirer from formal oversight via costly reviews and deliverable documents.

Have a Defined Set of Requirements

The acquirer must provide the requirements [6] including a description of the environment in which the software will be used. The developer must understand the requirements and demonstrate that understanding.

MIL-STD-498 provides — and the US commercial standard will provide — a framework on which the acquirer and developer can build an understanding of the acquirer's requirements and of the developer's process. Figure O-2 depicts understanding between the developer and acquirer.

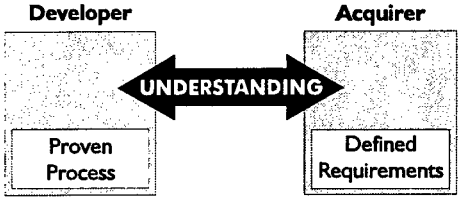


Figure O-2 Understanding Between Acquirer and Developer

Consider the Software Support Concept

The software will be supported by the developer or a separate maintenance organization. If the latter is true, the transition must be scheduled and planned, and support requirements for personnel and the support environment must be defined.

Identify the Types of Software on the Project

The focus of the effort is the software system itself. But other software will be developed or updated to support the effort. Examples are test scripts and simulation software. Some software will be deliverable while some may not. Some software may be of a type that is technically unprecedented for the prospective developer. Identifying these software characteristics is a preparatory step in using MIL-STD-498, a step that will also be needed for using the US commercial standard.

Define the Software Builds

Break the software development into pieces (builds[7]) based on technical risk, need to show quick success to management, requirements priority, architectural design, resources, or other rationale. Determine an order for the builds and consider how build schedules will overlap and dovetail.

TAILOR

Having established a starting point, requirements, and a general approach, detailed decisions can be made. MIL-STD-498 provides for all aspects of software development. Some aspects will be absent from any given project. The MIL-STD-498 provisions for those missing aspects need to be removed from the standard by tailoring. Other aspects of the project will need more specifics than the standard provides and must be "*tailored in.*"

Tailoring can be done by the acquirer and developer. In a request for proposal (RFP), the acquirer may suggest tailoring be included. The developer can respond with modifications to that tailoring in a proposal. This exchange of information to arrive at final tailoring helps establish the understanding depicted in Figure O-2. The acquirer's tailoring is influenced by how well the requirements are understood and to a lesser extent by an understanding of whatever process might be used to implement those requirements. The developer's tailoring is done based on an understanding of the developer's process and the extent to which the requirements are understood.

The maturity of the developer's process and the acquirer's requirements definition process is critical. If the developer is still trying to figure out how to develop software or if the acquirer is still trying to figure out how to define requirements, understanding between the acquirer and developer about either is unlikely.

Tailor for Each Build

Conceptually, tailoring is straightforward. You consider the objective(s) for the build and delete, modify, or augment each paragraph of the standard appropriately. In practice, tailoring requires a lot of thought. Tailoring is done for each build and for each type of software. A worksheet such as is provided in [3] can be used to capture the thoughts. These thoughts are the basis for the actual tailoring included in the statement of work.

Consider a trivial example. Suppose the software to be developed will be embedded in the specialized guidance hardware for a submarine. Paragraph 5.12.3.3 of the standard calls for software center operator manuals. Obviously, this paragraph is not applicable to the submarine guidance software, and such is noted on the worksheet. In the statement of work, a tailoring section includes the words "*delete paragraph 5.12.3.3.*"

Tailor the DIDs as Activity Checklists — A Radically New Concept?

The standard calls for the typical software development activities: *requirements analysis, design, code,* and *test* are examples. The data item descriptions (DIDs) may be used as a checklist identifying the information or "*data*"[8] that is expected to *naturally* result from an activity. Using the DID as a checklist is *not* the same as "*producing a document,*" which is often what springs to mind with the mention of DIDs. It is likely that not everything on the DID checklist will apply to the software being developed. Those items that are not applicable are "*tailored out.*" Note that while you may add, delete, or modify MIL-STD-498, you may only *delete* information when tailoring the DIDs.

Tailor DIDs as Deliverables, If Necessary

To minimize the need for deliverables, the standard supports close acquirer and developer teaming and the use of online access by the acquirer into the developer's data (nondeliverable data). The idea is that the acquirer does not have to possess the information or data as a deliverable to be able to access the data. Such is the case when

an acquirer uses online viewing technology to review requirements in the developer's database.

But some data will need to be deliverable and will include the costs associated with qualification, packaging, marking, copying, and distribution. If the data that must be a deliverable is a subset of the data identified on the DID checklist, a second tailoring of the DID is needed to specify which data is to be delivered.

For example, consider a project developing new software that will be supported by the developer for the life of the software. Suppose the Software Product Specification (SPS) DID was tailored as a checklist to require the following data be *captured* (not delivered):

1. Executable code.
2. Source code.
3. Design description.
4. Compile/build procedures

By tailoring the DID a second time to specify that only the executable code needs to be delivered, the costs of delivering the other data is saved while the software maintenance organization (the developer) retains items 2, 3, and 4, which are essential to doing maintenance.

Record Tailoring in the Statement of Work

The statement of work is to include two types of tailoring: the tailoring of MIL-STD-498 and the tailoring of the DIDs. Specific examples of each are found in paragraph 5.4.9 of [3].

Clarify *"Shell Requirements"*

Because MIL-STD-498 assumes that the developer has a proven process, the standard mentions some activities only briefly as a reminder. For example, paragraph 5.19.3 of the standard is a reminder to address security and privacy on the project. Requirements regarding security and privacy need to be provided in the contract. If the developer has a documented process for security and privacy, that documented process can be referenced in the contract.

SELECT A DEVELOPER

Tailoring involves give and take between the acquirer and the potential developer. But the decision as to who will actually develop the software is left to the acquirer who is paying the expenses.

Request a Draft Software Development Plan

In the RFP, the acquirer should ask for the inclusion of a draft software development plan (SDP) with the proposal. Under MIL-STD-498, the SDP is the key document. In fact, the SDP's structure mirrors that of the standard itself (see Figure O-3 below). The draft software development plan is used by the acquirer in selecting a developer.

MIL-STD-498		SDP DID	
4.	General Requirements	4.	Plans for performing general software development activities
4.1	Software development process	4.1	Software development process
	4.2.1		4.2.1
	4.2.2		4.2.2
	4.2.3		4.2.3
	4.2.4		4.2.4
	4.2.5		4.2.5
	4.2.6		4.2.6
	4.2.7		4.2.7
5.	Detailed Requirements development	5.	Plans for performing detailed software activities

Figure O-3 Software Development Plan Mirrors MIL-STD-498

Use Established Criteria for Selection

The *Guidelines for Successful Acquisition and Management of Software Intensive Systems*[9] also provide criteria for identifying developers. The work breakdown structure "*must be evaluated for completeness and reasonableness of approach.*"[9] Consider whether the proposed development methodology supports proposed audits, reviews, and peer inspections. Are the reviews part of the developer's normal process, and does the developer's staff have the experience to make the reviews, audits, and inspections useful? The acquirer must be convinced that the developer has a proven process (see Figure O-2).

FOLLOW THE SDP

Once the contract [10] is awarded and the project begins, follow the Software Development Plan.

Evaluate Results

MIL-STD-498 requires metrics but does not specify which kind. The developer defines the metrics, the data to be collected, and how the data will be interpreted. Review the metrics periodically.

MIL-STD-498 supports informal technical reviews attended by the acquirer's technical people and the developer's technical people. Management need not be involved in the technical reviews. If issues involving schedule and budget arise, they are taken to a management review for resolution. Collocating an acquirer's technical representative with the developer can enhance the confidence an acquirer needs to feel comfortable with informal reviews. This representative can also witness demos of prototypes or partial capabilities and can witness testing.

Make Improvements

At major milestones such as at the completion of a build, it is well to revisit the tailoring if permitted by the contract. Considering the case of our earlier example, the developer's process for project security and privacy may have come up short, or maybe things are going so well that you want to remove some of the formality from the next scheduled review. No project goes exactly as planned, so plan to make adjustments.

APPENDIX O Additional Volume 1 Addenda

Keep a Diary

Identify a mechanism to record lessons learned from using MIL-STD-498 on contract since this will help the next project in adopting MIL-STD-498 or in adopting the US commercial standard. Lessons learned should be recorded by acquirers and developers.

ORGANIZATION-WIDE ADOPTION

Migrate to Other Projects

Having applied the standard to one project, you are ready to use it on other software development projects.

Address resistance by having strong sponsorship. People have a reason to make it work if the boss is behind it. Generate enthusiasm with success stories. Determine who is going to have to give up something and identify what it is, then determine what you can do to help fill that void. People resist what they don't understand; help them understand through training.

Improve the Process

Share the lessons learned. Evaluate the project procedures to identify what works. Based on the metrics from the project, identify problem areas and develop solutions. Capture any strategies you used to mitigate risk [11].

Reed Sorensen
Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-774-7802 DSN 775-5555 ext. 3049
Fax: 801-774-7996
E-mail: sorenser@software.hill.af.mil

REFERENCES AND NOTES

1. DeWeese, Perry, cochairman EIA/IEEE Joint Industry Working Group, letter to R. Sorensen, Oct. 4, 1995.
2. *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Version 1.1, February 1995, pp. 7-19.
3. *MIL-STD-498 Overview and Tailoring Guidebook*, Jan. 31, 1996.
4. MIL-STD-498 defines *acquirer* as "*The organization that imposes this standard and the associated contract on a developer in order to procure software products for itself or another organization.*" *Developer* is defined as "*The organization required to carry out the requirements of this standard and the associated contract. The developer may be a contractor or a government agency.*"
5. This is the prospective developer's process if you are an acquirer. It is your process if you are a developer.
6. Salvucci, Anthony, "Vision for a New Acquisition Process," speech presented to the Nov. 19, 1993 Armed Forces Communications and Electronics Association Luncheon, *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Version 1.1, February 1995, p. J-3.

7. *Build* can have either of two meanings: (1) A version of software that meets a specified subset of the requirements that the completed software will meet. (2) The period during which such a version is developed.
8. Examples of such data are requirements, flowcharts, state models, data flow diagrams, test output, code, and programmer's notes.
9. *Guidelines for Successful Acquisition and Management of Software Intensive Systems,* Version 1.1, Vol. 1, February 1995.
10. When in-house development takes place, *contract* may be interpreted as a memorandum of agreement, a list of tasks to be performed, or some other formal or informal understanding of the work to be performed by one group for another. *Guidebook for MIL-STD-498: Software Development and Documentation, Application and Reference,* draft Oct. 3, 1995, para. 5.26.2.
11. Dart, Susan A., "Adopting An Automated Configuration Management Solution," paper presented April 12, 1994 at the Software Technology Conference.

CHAPTER 4
Addendum B

Software Reliability: A New Software OT&E Methodology

Captain Randy McCanne
Software Test Manager
HQ AFOTEC/SAS

ABSTRACT

Software reliability is a relatively new software quality metric gaining acceptance in industry and government. In evaluating software reliability, we are interested in determining how often software will cause a system to fail during operational use. In this paper we look at the measurement of software reliability from an Operational Test and Evaluation (OT&E) perspective and present an example of how we might evaluate the reliability of software to support acquisition decisions.

INTRODUCTION TO SOFTWARE RELIABILITY

Measuring and controlling the reliability of software is important because:

- DoD systems contain more software than ever before. Our aircraft, space, and C3I systems are becoming more dependent upon software (the C-17 aircraft contains more than 1.3 million lines of computer code), and the Air Force is investing heavily in Automated Information Systems that are nearly all software.
- Software now controls some of the most critical functions with DoD systems — from flight control to missile defense. A software fault was the most likely cause of a computer failure in the Patriot missile system that allowed an Iraqi Scud missile to hit the American barracks in Dhahran during Desert Storm. 28 US soldiers were killed and 98 injured in the attack.
- The reliability of many hardware components has improved to the point where software reliability is often the critical factor in overall system reliability.
- Software reliability is **not** adequately reflected in system reliability calculations by current Reliability, Maintainability, and Availability (RM&A) measurement practices.

Goal of Measuring Software Reliability

In OT&E the goal of evaluating software reliability is to support the evaluation of the operational effectiveness and suitability of software intensive systems. Specifically, the goal of the methodology presented in this paper is to measure software's contribution to system reliability in support of effectiveness and/or reliability objectives in Operational Test and Evaluation (OT&E) efforts. Another possible use of the methodology is for measuring changes in software reliability resulting from major system modifications in support of Qualification and Follow-on OT&E efforts.

Definition of Software Reliability

Software reliability is one measure of software quality. Software quality refers to the degree to which the attributes of software enable it to perform its specified end-item use [DoD-STD-2168]. The end-item use of software in Strategy-to-Task terms is the relationship of the software to the accomplishment of the system's mission task or mission task element.

SOFTWARE RELIABILITY is defined as the probability that software will work without failure for a specified period of time in a specified environment.

In a system context, software reliability is the probability that software will not cause failure of the system for a specified time under specified conditions. The environment of software is characterized by its associated hardware, support software, users, and frequency of use.

THE SOFTWARE FAILURE PROCESS

This section defines the terms used to describe the software failure process, compares the way in which software and hardware fail, and discusses the reason why demonstrated reliability, which is often used as a measure of system reliability, does not adequately account for the contribution of software.

Software Reliability Definitions

These definitions were developed jointly by HQ AFOTEC/SAS and PRC Inc. under Contract No. F29601-89-C-0071, Software Effectiveness Evaluation Methodology Study subtask (Subtask 028/00). The taxonomy presented here will be incorporated into AFOTEC Pamphlet 99-102, Management of Software Operational Test and Evaluation, to provide a standardization of the terms associated with the evaluation of software effectiveness and suitability.

A **software failure** is a software functional imperfection resulting from the occurrence(s) of defects or faults. Failures can occur when execution of a fault results in unacceptable system behavior. A failure can be one of conformance, in which the program does not produce the correct output, or one of performance, in which the program does not perform a required function in a timely or resource-efficient manner.

A SOFTWARE FAILURE is defined as the inability of a system's software component to perform a required function, as perceived by the user, within specific limits.

A software failure may also be the result of unimplemented but documented user requirements or undocumented user needs. This class of failure is manifested, not as the result of the operation of the code product, but as non-operation of code since code required to fulfill the task requirement or user need does not (yet) exist. Failures can be measured as the sum of

Software Trouble Reports or software-related Product Quality Deficiency Reports and unimplemented Software Change Proposals required to achieve the user's intended requirements and needs.

A **fault** is a manifestation of a code generation error or errors in logic, computer instructions, or data. They are introduced during the coding and maintenance phases of the life cycle. Faults exist in code whether the code is exercised or not; faults exist in the program's static operating state. A fault is discovered when a specific set of inputs combined with execution of the "*path*" through the code which contains the fault results in a failure. Software reliability provides a measure of the state of faults in a program.

A SOFTWARE FAULT is an unintentional software condition that causes a functional unit to fail to perform its required function (often referred to as a "*bug in the software*").

Defects represent deficiencies, either intentional or inadvertent (the manifestation of undetected process errors). A defect may be a designed-in deficiency or an omission in the implementation of a user requirement needed to support functionality for successful mission task or mission task element accomplishment. Defects are introduced primarily during the analysis and design phases of the life cycle or during the development of the respective documentation products. They often result from incomplete or incorrect understanding of the user's requirements or from poor documentation practices—including requirements traceability in the documentation development process. The occurrence of unintentional defects is often seen in software that has a history of volatility in its requirements, design or code.

A SOFTWARE DEFECT is the lack of something necessary or desirable, an imperfection. Defects can occur in any software deliverable: specifications, designs, source code, manuals.

The potential for defect errors is reflected in the completeness of the functions implemented to completely support mission task or mission task element operations and in the traceability and stability of the requirements and design. The potential for fault errors is reflected in the reliability measurement.

Software for Air Force systems is normally developed using some variation of the "*waterfall*" development process outlined in DoD-STD-2167A (Figure O-4 below). **Errors** can occur in any phase of the development life cycle and result in defects and faults in the software. Defects created in the early phases which cascade through later phases of the development process lead to faults that can be very difficult to correct. After system integration, the software contains a fixed number of undiscovered, or inherent, faults. The goal of testing is to discover and correct as many of these inherent faults as practical before releasing the software for operational use.

AN ERROR is defined as an act (mistake) that unintentionally deviates from what is correct, right, or true; human action that results in the creation of a defect or fault.

Comparison with Hardware Failures

The hardware failure process has been studied extensively through the years and differs in many aspects from software. Hardware usually has many "*burn-in*" failures early in its life, then exhibits random failures at a fairly constant rate during a time period referred to as the "*useful life.*" During the useful life of a hardware component, the design does not change

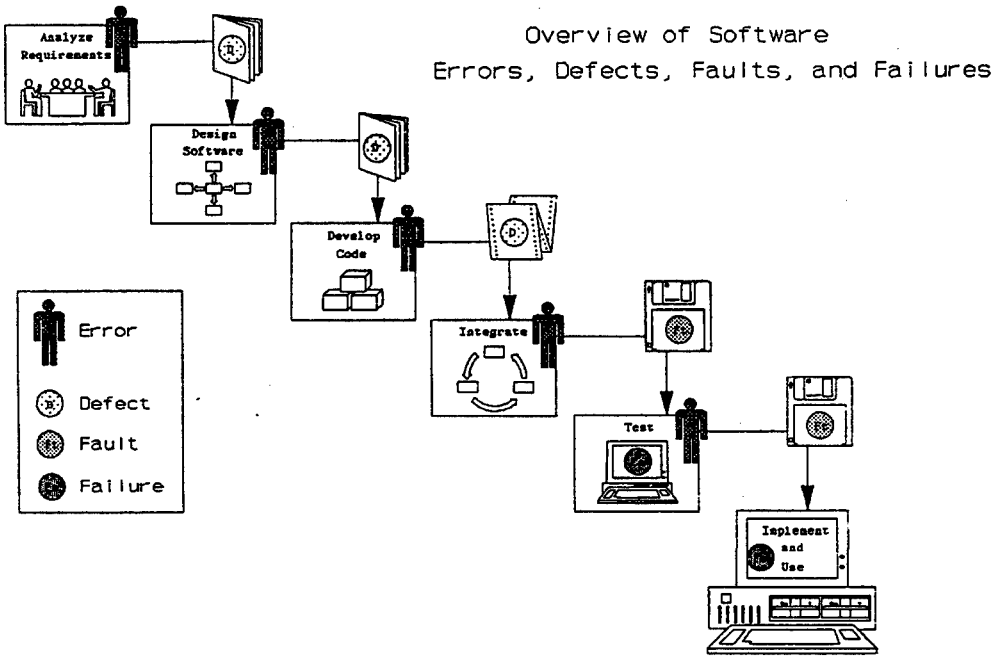


Figure O-4 Overview of Software Errors, Defects, Faults, and Failures

and the same types of failures are likely to occur over and over. Failures are corrected by either repairing in place (RIP), or by removing and replacing (R&R) the hardware component. Toward the end of useful life, the error rate increases due to "*wear-out*" failures. When failures are plotted over the life of a hardware component, the plot normally follows the "*bathtub*" curve of Figure O-5.

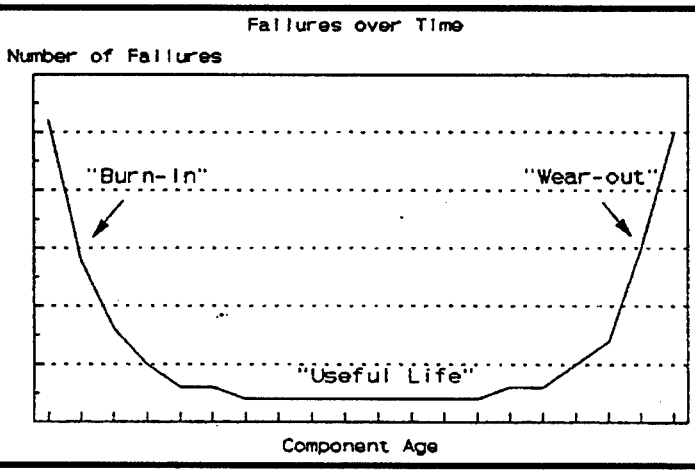


Figure O-5 Characteristic Hardware Failure Plot

Software does not exhibit this phenomenon. The design of the software may be modified many times during the life of the software to correct failure-causing faults. "*Repairing*" software can be difficult, expensive and new versions can take months to prepare and implement. Once a fault is removed from the software it is gone forever, although new faults may be introduced whenever the code is modified. When plotted against system age, software failures tend to follow the Rayleigh curve of Figure O- 6. [KEENE91] A large or poorly tested modification may result in a net increase in the number of inherent faults, but in general the number of faults remaining in software tends to decrease with time.

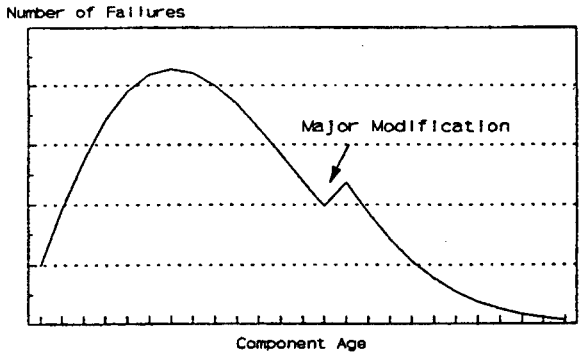


Figure O-6 Characteristic Software Failure Plot

The Problem with Demonstrated Reliability

The reliability of Air Force systems is often measured by counting the number of critical failures observed during a period of operational use *(called demonstrated reliability).* Critical failures are included in demonstrated reliability calculations only until the system developer can demonstrate the fault which caused the failures has been corrected. Once a fault has been resolved, then all recorded failures associated with that fault are removed from the data used to compute demonstrated reliability. For computing demonstrated reliability, only those failures associated with known, uncorrected faults are included.

For hardware components that have passed the "*burn-in*" stage, this method of measuring reliability may be valid. Hardware faults which cause failures are often difficult or impossible to correct, and continue to cause failures during the useful life of the component. On the other hand, software faults can almost always be resolved (eventually) and will usually cause only a single recorded failure. Once the input conditions that cause the failure are known, a work-around is usually developed to prevent further failures. Thus from a reliability point of view, the only interesting software failure is the next one encountered due to an as-yet-unknown fault. The effect of undiscovered software faults is not considered in the computation of demonstrated reliability. Consequently, the system reliability estimates are overly optimistic.

It is worth noting the impact of undiscovered faults in the *hardware* is also ignored in computing demonstrated reliability; but the effects of these faults are usually insignificant during the "*useful life*" of the hardware component. This is not the case with software.

Another way of presenting the problem with demonstrated reliability is shown in Figure 0-7 (below). The effect of computing demonstrated reliability is to fit a straight line on the failures-versus-time graphs. In the case of hardware, this procedure may be valid during the useful life of the hardware since the failure rate remains approximately constant during this period. However, the typical Rayleigh curve associated with software failures can not be accurately estimated with a straight line until late in the life cycle if at all.

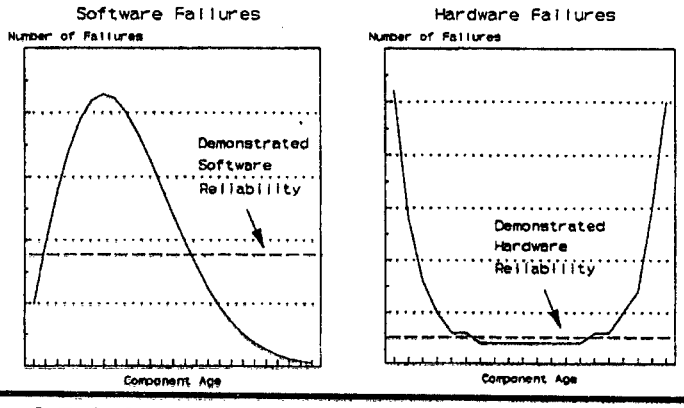


Figure O-7 Software versus Hardware Demonstrated Reliability

The effect of the pool of inherent faults that remain in the software at any point in time is measured by using a software reliability model. Thus both demonstrated (hardware) and inherent (software) reliability must be considered to obtain an accurate estimate of the system's true reliability. The collection of data to support the measurement of both demonstrated and inherent reliability could be accomplished through a slight modification of the current Joint Reliability and Maintainability Evaluation Team (JRMET) process as illustrated in Figure O-8.

EARLY EVALUATION OF SOFTWARE RELIABILITY

The reliability of software can be improved by either reducing the initial number of inherent faults, increasing the rate of discovery of faults, or both. This can be accomplished through four activities linked with the software life cycle: fault avoidance, fault elimination, fault tolerance, and structured maintenance. Software developers can be evaluated during program reviews and audits to ensure these four reliability improvement techniques are used to the extent appropriate for the system under development. Safety-critical systems, such as aircraft, may require more attention to reliability than other systems.

Fault Avoidance

Fault avoidance consists of applying sound software engineering practices, including comprehensive standards (for documentation, design and programming), rigorous quality assurance (formal reviews and audits), and independent verification and validation (IV&V). One way to measure a software development organization's ability to reduce the number of inherent faults through fault avoidance is through the Software Engineering Institute's (SEI) Capability Maturity Model. An organization which scores high (3 or greater) using this model is considered to have a well-defined development process and should consistently produce highly reliable software.

Fault Elimination

Fault elimination is accomplished through design and code inspections and effective testing. While it is possible through exhaustive testing or mathematical proof of correctness to develop fault-free code, it is usually impractical to do so in systems of significance size and complexity. Test coverage models are available to assess the effectiveness of the test strategies

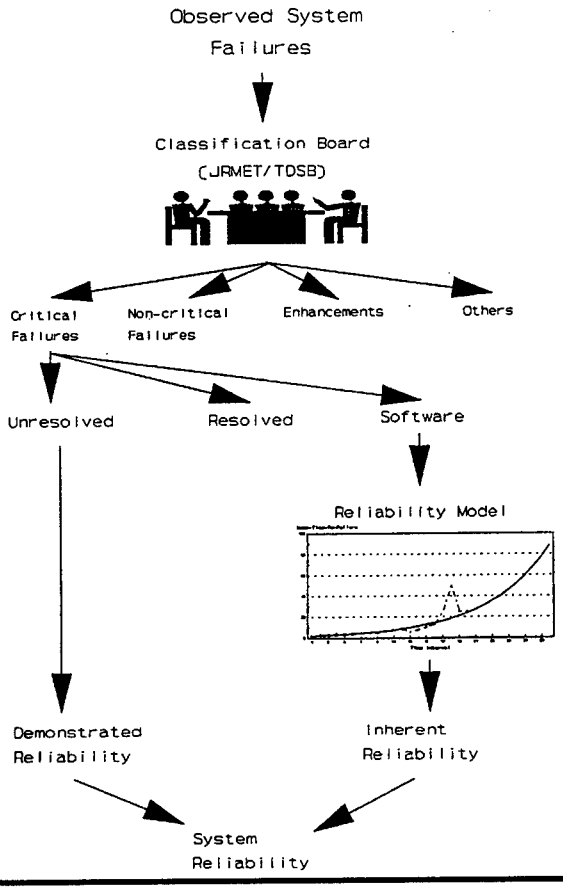


Figure O-8 Joint Reliability and Maintainability Evaluation Team Data Collection Process

employed by software developers. Common testing methods will identify many, but certainly not all, faults.

Fault Tolerance

Fault tolerance is achieved through special programming techniques that enable the software to detect and gracefully recover from error conditions. One method of programming fault tolerant software is the development of redundant software elements that provide alternative means of fulfilling the same function. The different versions must be programmed such that they will not all fail in response to the same input state (called a "*common failure mode*"). A more common but less effective example of fault tolerance is the use of good exception handling in Ada.

Structured Maintenance

Each software maintenance action should be performed as a microcosm of the full development life cycle. As such, the techniques of fault avoidance, elimination, and tolerance can be applied to modifications made during the maintenance of software as well. This is necessary to avoid introducing new faults as a result of code modifications made to correct known faults, add enhancements, or adapt the software to changes in the computing environment. Unfortunately, software will continue to contain faults and will occasionally fail even where sound fault avoidance, fault elimination, fault tolerance, and structured maintenance techniques are applied. For this reason, it is necessary to measure the reliability of software systems before releasing the system for operational use. A number of ways have been proposed to measure the inherent reliability of software.

SOFTWARE RELIABILITY MODELING

Software reliability models are abstract representations of the process of discovering faults through the observation of software failures. This process begins after the software, with all its inherent faults, has been integrated into a coherent system. In general, there are three main factors which determine the reliability of a software product:

1. The fault characteristics of the software product under study, such as the initial fault density (faults per lines of code), the interdependence between faults, and the fault tolerance of the software design. These fault characteristics often change with each new implementation of the software.
2. The effectiveness of the fault discovery process (the testing and operational use of the software). The discovery of hidden faults is highly dependent upon the testing strategies employed and, later, how the software is used in operation.
3. The age of the software. Any software product will eventually meet a given reliability goal if enough time and effort is expended.

To simplify the modeling process, assumptions must be made about these factors. The analyst must be familiar with the software development process and the software product to make valid modeling assumptions.

Types of Reliability Models

Three general approaches to measuring software reliability are available. These approaches are based mainly on the failure history of the software under study and can be classified as follows.

1. **Failure count models**. The process under study in this type of model is the number of failures encountered in specified time intervals or the successive time between failures during testing and operational use.
   - **Tagging models**. This group of models includes both fault seeding and dual test group models.
   - **Fault seeding models**. The approach in this model is to intentionally *"seed"* a number of faults into the software and characterize the inherent fault content as a function of the number of these *"seeded"* faults that are found during testing.
2. **Dual test group**. To use this model, testing must be performed simultaneously by two or more independent test groups. The inherent fault content of the software is computed as a function of the number of common faults found by the independent groups.

3. **Test coverage models**. This group of models is based on the assumption the reliability of software is directly related to the effectiveness of the testing performed. Several test coverage models are available including the IEEE Test Coverage Model [IEEE Standards 981 and 982].

Because of the difficulty and high-cost associated with collecting the data to use the tagging and test coverage models, the methodology presented here for use in OT&E is based upon the failure count models.

PROPOSED OT&E SOFTWARE RELIABILITY METHODOLOGY

The methodology developed by HQ AFOTEC/SAS to measure software reliability is quantitative in nature and based upon mathematical models used throughout industry and government. The methodology can be used to measure the current state of reliability of a software product against documented operational requirements. The following paragraphs provide a brief overview of the software reliability measurement methodology proposed for use during OT&E. The basic steps involved in building a software reliability measurement model are:

Step 1. Study Software Failure Data
Step 2. Choose a Reliability Model
Step 3. Obtain Estimates of Model Parameters
Step 4. Obtain the Fined Model
Step 5. Perform Goodness-of-Fit Test
Step 6. Obtain Estimates of Performance Measures
Step 7. Make Decisions [GOEL85]

Step 1: Software Failure Data

As a minimum, the following data must be collected during the OT&E for the purpose of building a software reliability model:

- **Software problem number**: a unique identifier used to distinguish one software failure from another.
- **CSCI**: the particular CSCI where the fault was discovered.
- **Severity level**: a numeric value (1 to 5) which describes the impact the software failure has on the system performance [Ref: TO 00-35D-54]
- **Date discovered**: the calendar date when the failure was recorded.
- **Operating time**: the total operating (or execution) time on the CSCI up to the point when the failure was recorded.

Other data items which can be collected to help in the analysis are:

- **Problem description**: a short narrative of the failure or fault.
- **CSC/CSU**: the computer software component (CSC) and unit (CSU) where the fault was discovered.
- **Phase**: the phase of the development life cycle where the fault was introduced (e.g. requirements analysis, design, coding, or testing).
- **Changes in testing**: any changes to the test strategies employed must be tracked since the models depend upon the satisfaction of assumptions regarding the type of testing performed.
- **Implementations**: the total operating time on the CSCI at each new implementation. This can be plotted on reliability graphs to show the net effect of each new implementation on the software's reliability.

Step 2: Choosing a Reliability Model

HQ AFOTEC/SAS has acquired an automated-tool for conducting software reliability modeling and analysis called the Computer-Aided Software Reliability Estimation (CASRE) tool. [LYU92] This package contains a number of failure count models and an integrated environment for preparing data, building models, choosing the best model for the data, and for graphing the results.

The type of failure history data available (time between failures or failure counts) will narrow the field of available models. From the remaining list of models the CASRE program will choose the model(s) that best fits the failure data provided. Often two or more models will be identified by CASRE as providing a "*good*" fit. The "*best*" model is chosen from this list based upon an understanding of the testing process and of the underlying model assumptions. In some cases a linear combination of models may be appropriate.

Step 3: Estimating the Model Parameters

Once a particular model or combination of models has been chosen, the model parameters are estimated. For most of the reliability models in CASRE three characteristics of the failure process are estimated in order to build the model:

1. The initial fault content or fault density of the software;
2. The initial failure rate; and
3. How the failure rate will change over time.

The CASRE package will estimate these parameters from the failure data by using either the maximum likelihood or least squares methods. Future research is directed at estimating these parameters based on expert opinion and historical trends using Bayesian statistical techniques.

Step 4: Obtaining the Fitted Model

At this point we have a fitted model with parameters estimated from the available failure data. The model form was chosen through a process of eliminating the inappropriate models, but it is necessary to ensure the chosen model provides an adequate fit of the data.

Step 5: Performing Goodness-of-Fit Tests

CASRE includes the following statistical functions to help analysts determine the applicability of a particular model to a specific set of failure data:

- Computation of the prequential likelihood function (the "*accuracy*" criterion).
- Determination of the probability integral transform u_i (the "*bias*" criterion).
- Computation of y_i to produce a y-plot (the "*trend*" criterion).
- Noisiness of model predictions (the "*noise*" criterion).

A weighted combination of the accuracy, bias, trend, and noise criterion are used to rank the models for *goodness-of-fit* in Step 2.

Step 6: Estimates of Performance Measures

From CASRE we can obtain a plot or table of the following reliability measures for the model developed:

1. **ROCOF**: the estimated rate of occurrence of critical software failures;
2. **MTTF**: the estimated mean (or median) time to the next new (unique) software failure;
3. $P\{T_i < t\}$: the estimated probability the system will operate without a critical software failure for t time units.
4. The estimated number of remaining software faults; and

5. The estimated fault density (number of remaining faults per thousand lines-of-code) of the software;

In addition, it may be possible to compute the following measures for some models:

6. T_m The projected time when a target software reliability requirement will be achieved (assuming the assumptions of the reliability model remain valid); and
7. T_0 The optimal release time for the software based upon the costs of testing, debugging, and encountering critical failures during operational use.

Step 7: Making Decisions

The results of a software reliability analysis must help the decision maker decide whether or not the system has met the user's requirements. In large and complex systems, reporting the results in a concise yet unambiguous way may be the most difficult task in the evaluation process.

REPORTING SOFTWARE RELIABILITY RESULTS

The primary reporting level is Computer Software Configuration Item (CSCI). In multi-CSCI systems, the CSCI reliability measures must be aggregated into a system level measure and combined with the demonstrated reliability results.

Aggregation of CSCI-Level Results

The aggregation of CSCI-level reliability measures into a system-level measure is necessary when the operational requirement is stated in terms of mission or system reliability. The size and complexity of the system will determine how this aggregation is performed.

When the system contains only a small software component, the overall system mean-time-between-critical-failure ($MTBCF_{sys}$) can be computed by considering the hardware and software factors as independent components using the following equation:

$$MTBCF_{SYS} = \frac{1}{\frac{1}{MTBCF_{HW}} + \frac{1}{MTTF_{SW}}}$$

where:

- $MTBCF_{HW}$ = the demonstrated *mean-time-between-critical-failure* for the system which primarily measures the effects, of known, unresolved hardware faults.
- $MTTF_{sw}$ = the *mean-time-to-failure* for software which measures the effects of inherent software faults. MTBCF is not used for software because this term implies a relatively constant rate of failure whereas the failure rate of software is assumed to steadily decrease.

Note the system MTBCF estimate is always lower than either the hardware or software estimates. The hardware and software components are assumed to effect system reliability independently, and the effect of each component is always to decrease overall system reliability. Consequently, if hardware (measured through demonstrated reliability) exactly meets the system reliability requirement, then we can be certain the system will **not** meet the requirement if a software reliability model indicates any software faults remain. When the system contains more than one CSCI, a reliability model must be developed for each CSCI independently. The reliability performance measures of each CSCI model must be combined with demonstrated reliability information into a system reliability model. This model, normally developed by the logistic analyst in HQ AFOTEC/SAL, is used to compute overall mission or system reliability.

SOFTWARE RELIABILITY EXAMPLE

The following example is contrived but shows how software reliability can be used to support system reliability measurement objectives and the determination of OT&E test effort.

USSTRATCOM has specified a system reliability requirement for the new B-3A *Bogus* bomber Mission Computer (MC) of 400 hours MTBCF at the end of Initial Operational Test and Evaluation (IOT&E). The IOT&E is conducted in two phases, a combined DT&E/OT&E phase, and a dedicated OT&E phase. A **System Maturity Matrix (SMM)** was specified to measure the progress of the developer in maturing the system as illustrated on Table O-3.

FLIGHT TEST HOURS SCHEDULED	DATA COLLECTION PERIOD	SYSTEM RELIABILITY REQUIREMENT (MTBCF)
1,000	Combined DT&E/OT&E	50 hours
500	Dedicated IOT&E	400 hours

Table O-3 Bogus B-3A Bomber System Maturity Matrix

Software failure data was collected during the combined DT&E/OT&E over a period of 12 months and 1,000 fleet flying hours (FFH). The cumulative number of flight hours on the fleet of aircraft was recorded for each failure encountered. The flight time was divided into 20 time periods of 50 FFH each and the number of software critical failures in each period was determined from data collected by the JRMET (Table O-4). Although all types of failures were recorded only critical failures are used in the reliability analysis, and only the first occurrence of each unique failure is counted.

TIME PERIOD	FLIGHT HOURS (FFH)	NUMBER OF CRITICAL SOFTWARE FAILURES	CUMULATIVE SOFTWARE FAILURES
1	1-50	31	31
2	51-100	20	51
3	101-150	18	69
4	151-200	16	85
5	201-250	16	101
6	251-300	15	116
7	301-350	11	127
8	351-400	10	137
9	401-450	10	147
10	451-500	8	155
11	501-550	7	162
12	551-600	6	168
13	601-650	9	177
14	651-700	7	184
15	701-750	5	189
16	751-800	4	193
17	801-850	2	195
18	851-900	1	196
19	901-950	2	198
20	951-1000	2	200

Table O-4 B-3A MC Failure Data

APPENDIX O Additional Volume 1 Addenda

Choosing the Model

For the failure count history of Table O-4, the CASRE package identified the Non-Homogeneous Poisson Process (NHPP) model for interval data as the "*best*" fit. This model has the following mathematical form:

$$m(t) = a \bullet (1 - e^{-b \bullet t})$$

a = the number of inherent faults in the software at the start of testing; and
b = the fault detection rate per remaining fault.

Examination of the assumptions of this model did not reveal any inconsistencies with the way the software was tested or the way failure data was collected.

Building the Model

The maximum likelihood estimators of the model parameters were computed using CASRE. The curve fit produces the following form of the NHPP model.

$$m(t) = 215.9972(1 - e^{-0.1301426t})$$

Thus, we estimate that approximately 216 faults were inherent in the software at the start of testing. Since we have found 200, we also estimate that approximately 16 remain to be found. Figure O-9 shows a plot of the failure data versus the model. The model appears to provide a good fit to the observed failure history.

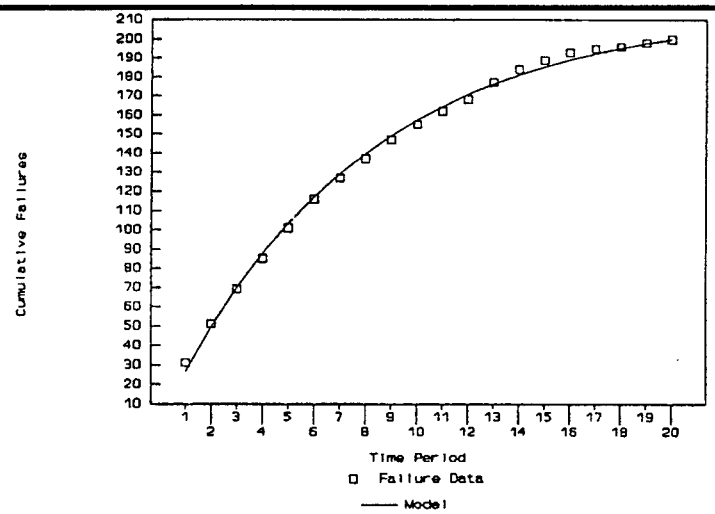


Figure O-9 B-3A Bogus Mission Computer Failure Data versus NHPP Model

Test Readiness Analysis

We wish to know if the B-3A *Bogus* Mission Computer meets the test readiness criteria specified by the system maturity matrix to begin dedicated IOT&E (50 hours MTBCF). While no specific *software* reliability requirements are given, we know that the reliability of the software must be at least as good, probably much better than, the system reliability requirement.

Software Mean-Time-To-Failure (MTTF)

The current MTTF for software is computed using the following equation for the NHPP model:

$$MTTF_{SW}(t) = \frac{1}{a \bullet b \bullet e^{-b \bullet t}}$$

or, for $t = 20$:

$$MTTF_{SW}(20) = \frac{1}{215.9972 \bullet 0.1301426 \bullet e^{-0.1301426 \bullet 20}} = 0.48033$$

Thus, the next software failure is expected to occur after approximately 0.48 time periods or:

$$0.48 \bullet 50 = 24.0 FFH$$

since each time period represents 50 FFH. The software achieved the system goal of 50 hours MTTF in only one time interval, and is generally well below the requirement.

System Mean-Time-Between-Critical-Failure (MTBCF)

The overall system MTBCF is computed by considering the demonstrated reliability of the mission computer hardware and the inherent reliability of the software as components in series. Given the mission computer has demonstrated a reliability (MTBCFHW) at the end of combined DT&E/OT&E of 3,000 hours, then the overall system MTBCF is:

$$MTBCF_{SYS} = \frac{1}{\frac{1}{3000} + \frac{1}{24}} = 23.8 \text{ hours}$$

Figure O-10 shows the results of combining the software and hardware reliability (system reliability) assuming the demonstrated reliability stays constant at 3,000 hours. This graph represents compelling evidence that the system can not consistently achieve the stated reliability requirement. In fact, the system will not achieve the requirement no matter how high the demonstrated reliability estimate is computed to be. To a decision maker this means the system does **not** meet the requirements to start dedicated IOT&E.

Calculating the Expected OT&E Test Effort

Suppose we choose to enter the dedicated IOT&E test phase anyway and want to know how much test effort (flight test hours) will be required for the system to reach the operational requirement of 400 hours MTBCF at the end of OT&E. Assuming the demonstrated reliability of the hardware remains at 3000, then the software requirement can be found by solving the following equation for $MTTF_{SW}$:

$$400 = \frac{1}{\frac{1}{3000} + \frac{1}{MTTF_{SW}}}$$

Thus the $MTTF_{SW}$ required to achieve a system reliability of 400 hours MTBCF is 461.5.

Assuming that the current software *test, analyze* and *fix* process continues through the rest of the testing, the expected test effort can be found by solving the following equation for t:

$$R(x,t) = e^{-a \bullet (e^{-b \bullet t} - e^{-b \bullet (t+x)})}$$

APPENDIX O Additional Volume 1 Addenda

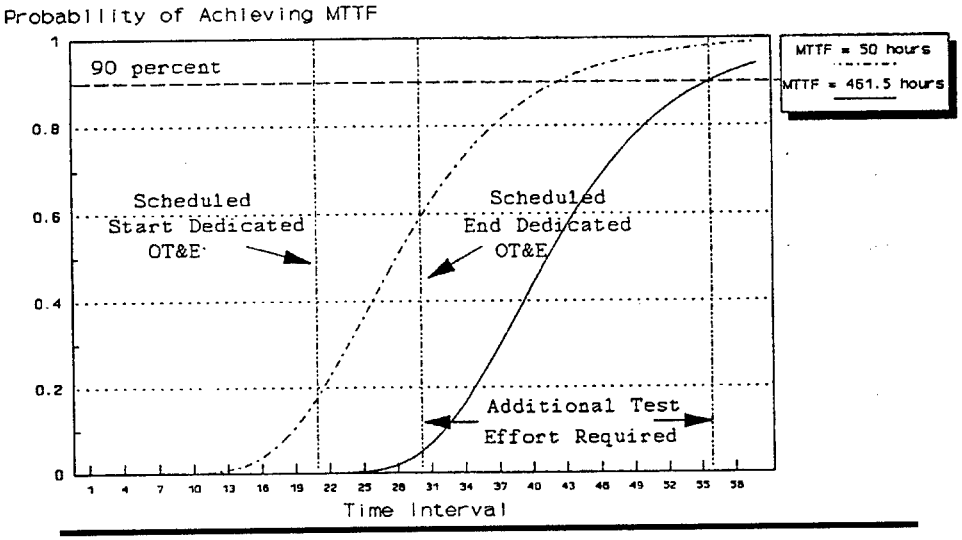


Figure O-10 B-3A Bogus Mission Computer NHPP Model Reliability Functions

where:
x = the desired MTTF expressed in terms of 50 hour time intervals;
t = the number of time intervals into the test; and
$R(x,t)$ = the desired probability that the software will operate for x time intervals.

For $R(x,t) = 0.90$ (90% probability), the flight test hours (t) required to reach a software MTTF rate of 461.5 hours ($x = 9.23$) is approximately 56 time periods or $56 \cdot 50 = 2{,}800$ fleet flying hours. We estimate that an additional 1,300 FFH will be required during dedicated OT&E to meet the operational requirement (beyond the 500 already scheduled). Figure O-7 shows a graph of the probability of meeting each SSM requirement based on the failure data available at the end of the combined test period.

CONCLUSION

Software has become an integral part of virtually every DoD system and controls evermore critical functions within those systems. Yet through our practice of using demonstrated reliability as system reliability we are largely ignoring the contribution of software to the reliability of our systems. The reliability of software can be controlled during the development life cycle through the application of reliability improvement techniques, such as fault avoidance, fault elimination, fault tolerance, and structured maintenance. We can and should assess the developer's use of these techniques during Operational Assessments.

Software reliability should also be measured during the OT&E of software intensive systems. Ideally, this measurement should be compared against *software reliability requirements* developed from an allocation of system reliability to software and hardware by the user. When specific reliability requirements for software are not provided, then software reliability measures should be combined with demonstrated reliability to more appropriately account for the impact of software on system reliability. Finally, the measurement of software reliability may become a significant factor in the determination of OT&E effort (test length) in the future.

ACKNOWLEDGMENTS

The author is grateful for the support and constructive criticism provided by Maj. John Keck on the Software Analysis Team (SAS) in developing this paper. Special thanks goes also to Capt. Paul "*Dag*" D'Agostino of SAL for his helpful *"hardware perspective"* review of the paper.

About the Author

Captain Randy McCanne is an aircraft software test manager in HQ AFOTEC/ SAS. He graduated in 1983 from the Air Force Academy with a bachelor degree in computer science and earned a master of science degree in operations research from the Air Force Institute of Technology in 1993. Capt. McCanne has more than 900 flight hours as a search and rescue pilot in H-3 helicopters.

REFERENCES

[GOEL85] Goel, Amrit L., "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering*, Volume SE-11, Number 12, December 1985

[LITTLEWOOD92] Littlewood, Bev and Lorenzo Strigini, "The Risks of Software," *Scientific American*, November 1992

[LYU92] Lyu, Michael R. and Allen P. Nikora, "CASRE -A Computer-Aided Software Reliability Estimation Tool," *Proceedings of The Fifth International Workshop on Case (CASE '92)*, Montreal, Canada, July 1992

[MUSA87] Musa, John D., Anthony Iannino, and Kazuhira Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987

[PFLEEGER92] Pfleeger, Shari L. "Measuring Software Reliability," *IEEE Spectrum*, August 1992

[SINGPURWALLA93] Singpurwalla, Nozer D. and Simon P Wilson, "Software Reliability Modeling," GWU/IRRA/Serial TR-93/3, Department of Operations Research, The George Washington University, January 1993

[STANKO91] Stanko, Capt Joseph J., "A Standardized Software Reliability Measurement Methodology," Master of Science Thesis AFIT/GCE/ENG/91D-09, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, December 1991

AFOTEC Pamphlet 800-2, Volume 1, *Management of Software Operational Test and Evaluation*, May 1, 1990

AFOTEC Pamphlet 800-2, Volume 6, Software Maturity Evaluation Guide, October 1, 1990

AFOTEC Pamphlet 99-102, Volume 7 (Draft), Software Reliability Evaluation Guide, December 15, 1993

IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Standard 982, New York: IEEE Publications, 1988.

IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Standard 981, IEEE Publications, New York, 1988

[KEENE91] Keene, Samuel, and Chris Lane. "Combined Hardware and Software Aspects of Reliability," *Quality and Reliability Engineering International*, Vol 8., December 1991

CHAPTER 5
Addendum C

The Ada 95 Philosophy

S. Tucker Taft

This article was first published in the *Journal of Object-Oriented Programming*, June 1995. Copyright © 1995 by SIGS Publications, Inc.

On February 15, 1995, Ada 55 became the first internationally standardized object-oriented programming language (OOPL) [ISO/IEC 8652:1995(E)]. Ada 95 is an upward compatible revision of the Ada 83 language, which was designed in the late '70s as part of a US DoD sponsored competition; Ada 83 became an ANSI standard in 1983. Ada 83 was designed by a French team located at Honeywell-Bull, led by Jean Achbiah. The syntax of Ada is Pascal-like, with a strong orientation toward readability. Here is a simple example Ada program:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
    Birthday : constant String := 
                    "February 15, 1995";
begin
    Put_Line("Happy Birthday, Ada 95, on " &
                    Birthday & '!');
    -- "&" is the concatenation operator
end Hello;
```

Ada 95 is the culmination of a revision process initiated in 1988 as part of the normal ANSI and ISO process of updating standards at regular intervals (typically every .5 to 10 years). The language revisions incorporated into Ada 95 were designed by a team located at Intermetrics, Inc. in Cambridge, Massachusetts under contract to the Ada 9X Project Office, headed by Ms Christine Anderson of the US Air Force. As technical head of this design team, I found myself in a unique position leading a revision of a language that was quite successful in certain applications areas, such as air traffic control, commercial avionics, satellite systems, medical systems, and so on, but that had never achieved the broad commercial use hoped for when it was initially designed. Although the existing users tended to be very loyal and somewhat reluctant to see major changes, it was clear that to broaden the applicability of the language some number of significant changes would be necessary.

In many ways, Ada 83 anticipated the object-oriented programming *"revolution."* Ada 83 provided the many of the features now associated with the most advanced of OOPLs, including encapsulation in modules; information hiding between modules; data abstraction via user-defined data types; strong type checking across separate compilation, with minimal implicit conversions between types; generic program units (also known as *"templates"*); exception handling with automatic runtime consistency checks; concurrency support in language; and so on. As it turned out, in the early '80s many system programmers were just beginning to be weaned off of assembly language, and the programming language C emerged as the popular *"portable"*

assembly language for systems implementation. The *"heaviness"* of Ada's strong typing and consistency checks, generics, concurrency in the language, and so on, were seen as overkill, when the primary goal was to escape from the target dependency of assembly language into a more portable, *"lightweight"* language like C. Nevertheless, in industries like commercial aviation, where one bug is too many, the additional safety and reliability inherent in Ada 83 more than justified any perceived heaviness of the language.

In the late '80s, many software development organizations began to hit a wall of complexity. Software promised for Q1 came out in Q4 of the following year. Revisions of a product were often buggier than the version they replaced. Software organizations, both large and small, began searching for methodologies, tools, languages, magical incantations to help manage the exploding complexity. It was in this context that the revision of Ada was initiated.

> ***Without a coherent and well-defined philosophy, there is a real danger in language design that one will attempt to be all things to all programmers...***

Our design team began serious work on *"Ada 9X"* in March 1990 (*X* was still an unknown at that point). As our design work proceeded, we began to evolve a clear design philosophy to help rationalize and guide the innumerable decisions we were making between the many possible alternative approaches to enhancing the language. Without a coherent and well-defined philosophy, there is a real danger that the language design will attempt to be all things to all programmers, and the language will become a complete mishmash of features with no underlying themes or principles to hold it together. By defining and sticking to a clear design philosophy, the language has a better chance of making intuitive sense to programmers as they learn more about it, so that ultimately, it becomes second nature. The reference manual, with all its picky rules, can become *"shelfware,"* since the programmer knows intuitively what does and does not represent a meaningful program. Without such a clear underlying philosophy the language rules never become intuitive, and the programmer spends half the day trying to remember the proper syntax or semantics for the various language features.

The first important philosophical principle that emerged from our design. efforts was to focus on providing *"building blocks"* for programmers, rather than ready-made solutions. During 1989, a large number of *"revision requests"* had been gathered from the Ada user community, and a synthesis of these requests in the form of *"revision requirements"* had been performed by the Ada 9X *"requirements team."* We recognized that if we were to add a new language feature to address each of these identified requirements we would end up creating a Chinese menu of features, ungainly and inflexible. Instead, we chose to focus on a small number of high-leverage enhancements to the language, which would enable programmers to solve problems more effectively, and efficiently, including those identified in the revision requirements, as well as those associated with projects yet to come.

> ***The overall mission of the Ada 9X project was to build upon the strong foundation established by Ada 83 and produce a .systems implementation language that would be even more productive and would further reduce the expense of producing high-quality, highly reliable, highly adaptable software systems.***

An equally important design philosophy was to agree that Ada was first and foremost a systems implementation language. It is not a *"do what I mean"* language; it is not a functional or logic programming language where the programmer specifies the desired result, but not necessarily the steps to achieve be result. Rather, Ada is a *"do what I say"* language — the programmer is in complete control. The *"primitives"* of the language should in fact be primitive — well-understood building blocks, with clear semantics and predictable time and space

performance. Abstractness is something that programmers define, using Ada's packages and primitive types. The primitives of the language should not have semantics that are so abstract that the programmer has no idea how the feature is implemented, or how long it will take to execute. To be able to build efficient production software, it is essential that the programmer can build an accurate *"intuition"* about the relative expense of language features, so as to combine them in the way that achieves the required time and space performance.

The third fundamental design philosophy that emerged was that the basic constructs of the language should be inherently safe. If necessary, the programmer should be able to bypass the consistency checks that were provided to ensure safety, but any such overrides should be clearly demarcated in the source code. The defaults should always be safe. This is in sharp contrast to C and similar lower-level, admittedly *"lighter weight"* languages. In C (and C++), the *"default"* language features provide no checks for null pointers, array overflow, numeric overflow, inappropriate type conversions, etc. It is possible in C++, and to a lesser extent in C, to construct relatively *"safe"* abstractions, but in both C and C++ the most basic building blocks of the language are unsafe. Creating safe abstractions using unsafe primitives is laborious, error-prone, and difficult to verify. By contrast, in Ada the building blocks are inherently safe, and an abstraction built using them automatically inherits at least the same level of safety. If it is decided to override the language-provided checks, testing and verification efforts can be focused on those places where the unchecked programming appears explicitly. In C, or C++, it is very difficult to identify all of the places in a program where potentially unsafe primitives are being used.

In addition to all of these serious-sounding, high-minded philosophical principles, there was an overriding goal of making the language flexible, powerful, and fun to use. Early users of Ada 95 have provided welcome confirmation that we accomplished this latter goal, while remaining faithful to our more high-minded principles.

The overall mission of the Ada 9X project was to build upon the strong foundation established by Ada 83 and produce a language that would be even more productive and would further reduce the expense of producing high-quality, highly reliable, highly adaptable software systems. We believe that part of the key to accomplishing our mission was the development of a clear language design philosophy to guide the process, rather than making each design decision independently and without fitting into an overall philosophy and mission for the language. You can see for yourself whether we accomplished our mission by acquiring GNAT, a free Ada 95 compiler developed at NYU, based on the GNU compiler back end and a new Ada 95 front end written in Ada itself. *[The compiler is available by anonymous ftp from cs.nyu.edu in the directory pub/gnat. Full sources are available, as are ready-to-run binaries for DOS (using D/GPP), Linux, Sun, HP, Alpha, and various other hosts and targets.]*

In addition to GNAT, all of the commercial Ada vendors are upgrading their Ada 83 compilers and other tools to support Ada 95. In the first quarter of 1996, an *"Academic Ada 95"* compiler will be appearing in university bookshelves and on CD-ROM bundled with an Ada textbook, with a student price of about $60. For more information on Ada 95, including information on compilers and tools, consult one of the various Ada World Wide Web servers:

- ***http://lgiwww.epfl.ch/Ada/:*** has the entire Ada 95 Reference Manual and Rationale online in hypertext, as well as introductions, tutorials, comparisons with other languages, etc.
- ***http://info.acm.org/sigada/:*** has information on the ACM's Special Interest Group on Ada, plus pointers to other Ada resources
- ***http://sw.eng-eng.falls-church.va.us/AdaIC/:*** has information on the Ada Information Clearinghouse, plus reports from commercial projects that have been using Ada source code, etc.

I encourage you to give Ada a try.

[The full Ada 95 Reference Manual and Rationale are also available by anonymous ftp from sw.eng-eng.falls-church.va.us in directory public/adaic/docs/standard/95lrm_rat/v6.0.]

Tucker Taft is Chief Scientist at Intermetrics, Inc., Cambridge, Massachusetts. He can be reached at:

Intermetrics
733 Concord Avenue
Cambridge, MA 02138
E-mail: stt@inmet.com.

CHAPTER 5
Addendum D

Ada Implementation Lessons-Learned from SSC and CSC

The **Standard Systems Center (SSC)** and the **Communications Systems Center (CSC/SD)**, in response to a request from SAF/AQK, compiled lessons-learned from their Ada and software engineering programs. The SSC programs upon which these recommendations are based include: the **Logistic Module-Base Level (LOGMOD-B)**, the **Air Force Operations Resource Management System (AFORMS)**, the **Manpower Data System (MDS)**, and **Air Force Command and Control System (AFC^{2}S)**. The CSC/SD lessons-learned were based on the **SARAH program**.

- Construction of Ada bindings to C products requires expertise in Ada, C, the product, and the internals of Ada and C for each platform.
- Automated tools reduce the amount of work required to construct and port bindings and improve reliability.
- Teams that design, develop, and maintain Ada bindings should be comprised of a stable, consistent group of people so less time is spent on familiarizing each new member with the tools, existing code, and special considerations.
- Building applications in Ada cannot compete with rapid prototyping techniques. There are few libraries to support rapid development with Ada. Until root analysis, design, and implementation of domain and utility-level objects are available, development using Ada will continue to be very time consuming.
- The Ada language is complex and can be hard to learn. Beginning programmers and those with a PASCAL background seem to have less difficulty in making the transition to Ada. Programmers with experience, especially long-term experience, in other languages like C, COBOL, and Assembler seem to have difficulty making the transition.
- Just using Ada does not provide automatic reuse. Developing in Ada is like using any other language. Reuse is not automatic, it requires careful planning and a vision of what is required by more than just the application currently being developed. Reuse in Ada means a lot more than merely building generic packages that support common data structures.
- Ada tools are just now becoming available and mature. Tools for using Ada within an XWindows environment and an object-oriented methodology are starting to mature. These are critical to the future usage of the language to support the DoD TRM.
- The AdaSAGE tool is SSC's most requested asset. AdaSAGE is a very good tool for Government programs being completed with limited Ada experience.
- Software converted to Ada from another language can be a maintenance problem. For example, a very large program was converted from C to Ada. The library metric tools produced scores that indicated the code would be difficult to maintain, primarily because the original code was not engineered to take advantage of the Ada language. The code is in Ada, but it is not modular, not reliable, and probably only supportable at a high cost. Code like this example

gives Ada an undeserved reputation. Systems should not be converted to Ada unless there is value added.

- There is a great difference in the speed and efficiency among Ada compilers. Always shop around, talk to current users, and "*test drive*" a compiler before you commit to buy.
- Needed capabilities in Ada compilers and development environments may differ. For example, some compiler RTEs are very efficient doing tasking and some are not. Make sure you talk to other customers of the proposed vendor who are doing work similar to what you will be doing.
- Ada is not a "*magic*" solution to all coding problems. Ada does a lot of checking of the code during the compile phase. This usually makes the testing and debugging phase much easier and much shorter. However, some areas remain difficult. Plan for a lot of time to debug applications with tasking and applications that interface with a lot of assembler.
- Ada works. Ada cannot be viewed as an "*immature language*." It certainly has some advantages over other languages, but also, it is not the "*Silver Bullet*" that solves all problems. You can still design and produce junk using Ada.
- Training and education are critical to a program's success. Getting the right education on the principles of software engineering and having a defined, mature software development process will help solve problems. Ada simply gives you some of the tools you need to do the work right.
- Ada is a means (not an end) to better design. Ada is best viewed as a tool. You still need trained engineers to effectively use any tool.
- Networked PCs with Alsys Ada is an acceptable environment for developing Ada systems. Tools like the **Remote Compilation System**, **Automated Problem Tracking System**, and online Repository distributed by the STSC (produced by HQ CSC/SD) can double productivity in this environment.
- A good development environment greatly enhances productivity. Be sure to allow enough money to properly capitalize the Ada development effort. In the long run, it is a lot less expensive to buy faster PCs than to employ more programmers.
- Key players and expertise do not always remain within a program. Plan for several people to gain knowledge in every area. A current backup for each job is good insurance.
- A LAN-based automated database for storing and retrieving information on development problems and their solutions is very valuable.
- Plan for the expansion of hardware. Upgrading technology makes the difference between night and day.
- If you use access types, be careful to release them when completed.
- Ada libraries can become very large. Transferring them between people can be speeded up significantly if a LAN is available.
- Some assembler is okay. In certain areas, it is beneficial to write some code in assembler and link it to Ada. This is true when speed is of importance. For instance, assembler can be used for calculating CRCs (standard mathematical tables) and I/O functions like printing, disk access, and communications.
- Training lead time is high. It takes an average programmer almost a year to understand the Ada language enough to program effectively in it. Even then, they still do not get to see everything Ada can do and no one wants to sit down and read MIL-STD-1815A.
- Design is the most important. From a management point-of-view, a good design makes coding easier. The effort involved can also now be accurately estimated.

CHAPTER 7
Addendum B

Lessons-Learned While Achieving A CMM[SM] Level 3 Rating

Tom Westaway
Sacramento Air Logistics Center (SM-ALC)

INTRODUCTION

The following article is an attempt to answer the question, "*How did you achieve a level 3 rating?*" I was the Software Engineering Process Group (SEPG) Team Leader from 1992 until the present time. This article is written from my perspective. When you finish reading this article, I hope you realize that achieving Level 3 status is dependent on people. It is not a matter of following a technical formula to achieve success.

Background

From the early 1970s through 1989, all software effort was performed in either the Maintenance Directorate or the Material Management Directorate. In the late 80's and early 90's, we and all other Air Logistics Centers experienced several reorganizations. The effect of these reorganizations was to destroy the infrastructure that supported the software maintenance effort.

When we conducted our first software process assessment in September 1991, we were rated as a Level 1 organization. However, the data showed a surprising amount of strength in both Level 2 and Level 3 key process areas in some parts of the organization. It was obvious from the results of the assessment that software organization fragmentation was a significant weakness. However, the concern of the SM-ALC Commander was that we not optimize software at the expense of our overall weapon system support. For this reason, we were directed to not consider consolidation as one of our recommendations. The concept of consolidation was not dead though, and within a couple of months, we were directed to participate in a study to determine how software maintenance should be organized at SM-ALC. The group conducting that study eventually recommended that the fragmented software groups needed to be consolidated. As a result of that study, the SM-ALC Commander in November of 1992 approved the consolidation of all maintenance software organizations into one Division, TIS. In January 1993, SM-ALC/TIS was created. In March 1993, people were transferred into the Division from their previous organizations.

SETTING THE GOAL

Shortly before official creation of the new software division, the TI Director took the TIS Division Chief, and the Deputy Division Chief with him on a visit to the TIS software division at Ogden Air Logistics Center, Hill AFB, Utah. On the way back home, the TI Director shared the following with the TIS leaders: "*My vision for you is that your division will become the Software Engineering Process Center of Excellence for the Department of Defense!*" He told them to consider how the organization at Hill AFB was progressing and do it better.

Shortly after TIS was officially formed and people were transferred into the organization, the Division Chief directed that all TIS supervisors and SEPG members participate in a 3-day team building session. I think that the Division Chief felt that his team building session had been successful. Many of us, including the SEPG, walked away from that exercise feeling very abused.

Through the SEPG, the Division Chief met with a consultant from SEI. He was advised to have the organization's leaders develop a written Strategic Plan that would include a statement of their vision, mission, principles, values, goals, objectives, strategies, and targets. The Division Chief then convened the division leadership once each week to establish the Software Engineering Division's Strategic Plan. While establishing the written vision and mission statements was not too hard, establishing the principles, values, goals, objectives, strategies, and targets was much more difficult. As more detail was added to the goals, it became obvious that people were not going to continue doing business the same old way. As this realization took place, discussions became more heated, and resistance increased. After a lot of discussion, the Division's goals were captured in print.

Objective 3 of goal Number 1 stated that the division would achieve a CMM[SM] Level 2 maturity by October 1993. Objective 4 of goal Number 1 indicated that the division would achieve a CMM[SM] Level 3 maturity by October 1994. The entire TIS management team established reaching Level 2 maturity as the highest priority objective for the division.

It became apparent to the Division Chief as the summer of 1993 wore on that nothing was really happening. Level 2 was an objective, but there was no evidence that any of the supervisors were taking any steps to incorporate Level 2 practices in their day-to-day activities. The common excuse was that they had too many fires they were fighting and were unable to do any "*CMM[SM] things.*"

During the August to November 1993 time frame, the Division Chief informed all supervisors that their personal performance plans had been rewritten by him. In those plans, they were given the opportunity to establish CMM[SM] Level 2 practices in their areas of responsibility. To exceed the standards of performance, they were given calendar dates to have certain practices in place. To be fully successful, they were given a second set of dates. June 1994 was the cut off date to be fully successful on all of the performance standards.

As the Division Chief continued to work with the Division leadership, he found a common theme among all of them. I characterize this common theme as follows: "*What does it mean to be level 2? What does it look like? What do supervisors do? What does it mean to 'organize, train, and equip?' Do I want to be a supervisor?*" Supervisors literally did not know what to do. They were all working hard, fighting a constant barrage of "*fires.*" On top of all of that, they were being told they had to operate at something called Level 2. Many began to question if they really wanted to continue being supervisors.

TRAINING

In late August 1993, the Division Chief asked the SEPG to obtain training on all the CMM[SM] Level 2 key process areas for the Division. At this point in time, the SEPG consisted of 2 people. I was not happy to be given this task as I had what I thought were more important issues upon which to work. This tasking absorbed most of my time for the next 4 months. We did a fly-off between several vendors and settled on Fastrak Training, Inc. as the source of our training.

APPENDIX O Additional Volume 1 Addenda

From September 1993 through July 1994, we trained approximately 100 people in the Division in the areas of requirements management, project management, configuration management, and quality assurance. In retrospect, one group that should have received this training was the first and second level supervisors — but, they did not. As a result, when we involved the supervisors in management reviews later on, they were not prepared to talk the same language as all of those who had been trained in the formal courses. They did not understand what their project managers had been trained to do, or what their own role was.

We did try to train the management by providing one course in project management for supervisors. Every supervisor was required to attend this week long course. Most supervisors did not want to be present. A year later, most supervisors did not remember ever having been in this class. They were not prepared to participate in such a class. Most felt that they had been to all the management courses they needed and they knew everything they needed to know about being a supervisor. A year later, they would be claiming that they needed to have the same training as their project managers had received.

MORE SETTING THE GOAL

By January 1994, the Division Chief was not sleeping very well. He realized that as a Division we were not progressing fast enough. He was well aware that McClellan AFB was being considered as a candidate for closure once again. Our Commander had clearly stated that we all needed to take extraordinary measures to change the way we did business if we were going to survive. So, the Division Chief established a project and project leader to help the organization speed up its metamorphosis. The Division Chief also indicated that we were to arrange to have a Software Process Assessment performed in October 1994. He again stated, we are going to become a Level 2 organization and then we are going to become a Level 3 organization. He said that by October 1994, he wanted the organization to be Level 2, a one year slip from the objective stated in the TIS Strategic Plan.

PMIP

Thus was born a project that we would later name the Process Maturity Implementation Project (or PMIP). The project leader rapidly put together a plan for how he was going to approach this project. The Division Chief had indicated that the project leader was to work with the SEPG in accomplishing his project. As the leader of the SEPG, I was very irritated that the Division Chief had established a separate project and named a project leader that did not have any training in process improvement work. I really did not want to work with the project leader. It seemed to me that the Division Chief was showing utter disregard for the SEPG. I had tried many times to obtain additional members for the SEPG so we could more effectively help the organization, but every time, the Division Chief had failed to provide the requested help. By creating the PMIP project he had put additional resources on the task and then taken the process improvement effort away from the SEPG. It was also distressing to observe the project leader and realize that both he and the Division Chief were continuing to act in a Level 1 way while they were telling the organization that they must become Level 2.

At this point I had to make some personal decisions about how I was going to act. I began to realize that maybe I could turn this situation into an advantage in order to achieve what we had originally been commissioned to do. I recognized that I was a flaming introvert. I watched the project leader and realized that he was acting much more like an extrovert. So I decided to work with the project leader. I supported him in going out and doing all of the interfacing with masses of people in the organization. On the other hand, I knew some things we needed to do in order to get to Level 2. We eventually worked things out between us that I would do much of the behind the scenes work and he would do much of the visible effort. He also provided another benefit.

He acted as a buffer between the Division Chief and the SEPG. I could discuss ideas with the project leader and when he understood them, he could then introduce them to the Division Chief. Through this process and over a period of time, the credibility of the SEPG increased.

As time went on, the project leader and I established a good working relationship. I put aside my irritation and tried to use the situation to the best advantage for the organization. The project leader had an impossible task given to him, but he, and the organization were successful in achieving even more than they had set out to accomplish. After this, several things began to happen in parallel. I will discuss them one at a time.

Practitioner Involvement

The project leader established project leader councils. Each council consisted of representatives from specific sub-organizations and they were tasked to produce a set of strawman or template work products that could be used as starting points for each project. These strawman work products consisted of things like the software development plan (SDP). Strawman work products were produced to satisfy every key process in each of the Level 2 key process areas. All of the project leaders and practitioners in TIS are to be commended for the efforts they made that contributed to the overall effort to achieve Level 2. Every one of them took these tasks on themselves on top of heavy project work loads and pressures from their customers. In addition to the effort required to develop the strawman work products, they also then turned around and instantiated those work products for their own projects. This process of involving practitioners in these councils resulted in the following benefits for the organization:

1. Cross pollination and sharing of ideas was fostered.
2. Communication between practitioners in different Flights was fostered.
3. Documented strawman work products were obtained.
4. The documented strawman work products became a starting point for project leaders to develop final work products. This resulted in more commonality in similar work products and reduced total time spent across the Division in developing the work products.
5. An archive of strawman work products was created.

Supervisor Responsibilities

In the April 1994 time frame, the Division Chief indicated to me that after a number of meetings throughout the Division, he was convinced that the supervisors did not know what they should be doing under the CMM[SM] practices we were telling everyone they had to follow. He asked if I could prepare something that would describe what the supervisors needed to do to satisfy the CMM.[SM] In retrospect, this was a turning point in our efforts to start a cultural change in our organization.

I extracted all of the supervisor responsibilities from the CMM[SM] and put them in a separate document. I produced a matrix that indicated each of their responsibilities and how often they needed to perform that responsibility. In this document, we began to lay the foundation for one of the key supervisor responsibilities: having regular formalized reviews of all of their projects.

Establishing Organizational Policy

One of the responsibilities that appeared on the supervisor's list required them to establish organizational policy. After several weeks, it became apparent that they were never going to have the time to do this and they also did not know what they needed to put into the policy. At this point I had to give up another principle I thought I had learned from the SEI in our early SEPG training. My understanding was that the SEPG was supposed to encourage others to do things. All we were supposed to do was stand by and provide consulting services when asked. I got fed up with that mode of operation — it didn't work in our environment. I knew I could write the necessary policy, and I finally volunteered to write the first policy.

I wrote the policy for project tracking and oversight very cautiously, very much aware of the resistance in the organization. When the policy was provided to the supervisors, we agreed that they would have one week to review it. After one week, I was to take all comments received and make any necessary revisions. The revised policy was to be given to the Division Chief to review and provide me his requested changes. I would then incorporate his changes, give him a final copy to sign, and then provide copies to the Flight Chiefs in their next staff meeting.

During the development of these policies, the supervisors were very concerned about how changes could be proposed and made to the policies. It became apparent that we needed a formal way to control the configuration of these documents. We therefore established a Division Software Configuration Control Board, with part of its responsibility to control changes to Division policy.

It was during this period that stress levels began to increase tremendously. People were very obviously on edge and, as a Division, I am sure we helped support a number of doctors with stress related illnesses and symptoms. Others decided that they had enough for one lifetime and prepared to retire as soon as they could.

Management Reviews

As supervisors were told their role was to organize, train, and equip, they became increasingly restless. They wondered what their role really was. They felt that they were being told that they could no longer do that which they had the most experience doing: working on technical projects.

One identified supervisor responsibility included in our policies required them to hold regular management reviews of all projects in their organization. Many were unclear about why that was important and did not know what to review. We produced a simple graphic that showed three boxes arranged in a triangle. One of the boxes was labeled "*Project Leader Activities*." Another was labeled "*SQA Audits*" and the last one was labeled "*Management Reviews*." We explained that there were three key elements that had to be functioning for the organization to be able to achieve Level 2 status. Each of the three, as represented on the diagram, had to function: they were checks and balances on each other. We explained that the supervisors had a key role to play in helping the organization mature.

Once the supervisors began to accept the idea that they had an important role to play, they were open to suggestions about what they should review during a management review. The Division Chief provided them an outline of what he wanted to see when a project came to him with a project review briefing. With that outline, the first and second level supervisors had a starting point to tailor their review requirements with project leaders.

To make sure that reviews were taking place, the PMIP team collected metrics data on management reviews and provided this data to the Division Chief at his weekly staff meetings. The constant review of this metric made it obvious when supervisors were not holding reviews with their project leaders. This visibility soon resulted in all managers holding reviews.

Another way to ensure that adequate reviews were taking place, required that each review be documented with minutes and that those minutes be archived in the project folder. During SQA process audits, the auditors looked for review minutes. If they were not present, the project received a deficiency write-up that had to be resolved.

Two important things resulted from the requirement that supervisors hold project reviews. First, supervisors felt a renewed sense of their importance and involvement in what was going on in their organization. Instead of having to always fight fires, this process began to give supervisors the tools to prevent fires from happening. Second, project leaders began to feel like someone really cared and was listening to what was going on in their project. They no longer felt they were struggling with their problems alone.

SQA Process Auditing

We started advocating SQA process auditing as early as February 1994, but it took time for the concept to mature. In March 1994 we began developing the criteria the SQA auditors would be using to actually perform their audits.

Several problems plagued us. First, we had only one person designated for SQA. Second, we had very little support from supervisors or project leaders to actually perform process audits. The Division Chief generally supported the concept, but he wanted it to start immediately. No one was prepared to start doing any audits immediately. One individual who was providing on-the-job training to our division had experience in the SQA area and he helped develop a set of detailed process audit criteria for Level 2.

The responsibility "*triangle*" that we used to show supervisors the importance of their function also pointed out the importance of SQA Process Auditing to our goal of achieving level 2. Once supervisors began to understand that concept, we began to obtain their support for performing the audits. With emerging support for the audit concept, the Division Chief asked each of the software Flight Chiefs to provide one individual to augment existing SQA resources to perform the process audits. This solved the resource problem.

The audit team started by just reviewing the criteria for doing project tracking and oversight. It took 3-4 weeks to complete all of the audits. In the process, we found things did not work well and improved the process as necessary. We also were confronted with lots of data, little of it consistent between audit teams, so we held training sessions with the audit teams to improve consistency.

Following this first round of audits, the auditors started a second round in which they looked at the requirements management and project planning key process areas. After that round was completed, a third round was started in which they audited all 6 key process areas at Level 2. As the second round of audits came to an end, the Division Chief requested that all deficiencies for each project be tracked daily to see that these deficiencies were being corrected. This daily reporting soon encouraged all audited projects to clean up their act. The Division Chief received daily status reports on the deficiencies. On the 17th of October, the day the formal process assessment started, the Division Chief had in his possession a set of charts that showed that on the average, the projects being examined by the assessment (23 projects total) were about 96% fully compliant with the CMM[SM] practices for each key process area at level 2. This data was not made available to the assessment team.

SUMMARY

A key factor in achieving a Level 3 rating was the commitment at the top of the organization to do whatever was necessary to accomplish it. While focusing our energies on establishing the infrastructure to support Level 2 practices, the Level 3 efforts already ongoing were not neglected. Without the drive of the Division Chief and the PMIP project leader, and the support of the TI Director along with the SM-ALC Vice Commander and Commander, the efforts described above would have been futile. All of these individuals took considerable personal risk in pushing the process improvement effort.

The SQA process audits played a crucial role in helping the organization to progress. Without the audits and the deficiency tracking process, we would have had no insight into whether TIS policies were being applied and followed throughout the Division.

The Supervisors role was also crucial. By holding regular project reviews, they indicated that they were interested in what was going on in individual projects and that they would do their part to improve our processes.

The SEPG played a crucial role in providing knowledgeable guidance where needed. This guidance was often behind the scenes. The SEPG was instrumental in writing TIS policy and in stimulating process auditing and project reviews.

APPENDIX O Additional Volume 1 Addenda

The most important part of this complex process of maturing an organization is the individual people within the organization. If all of the people that make up this Division had not done their best and worked well beyond normal requirements, we could not have achieved a Level 2 or 3 status. So each and every member of TIS deserves credit for what has been accomplished.

Finally, it should be understood that what we have been through is extremely stressful. That stress level continued to rise right up to the assessment. Some people could not sleep well. Others got ulcers. Some probably gained or lost weight. The improvement effort almost ground to a halt. At that point, the pressure to change was relaxed and we began to tell the organization how much they had accomplished. That helped everyone's morale to improve.

About The Author

Tom Westaway is a member of the Engineering Test Branch of the Software Engineering Division at the Sacramento Air Logistics Center (SM-ALC/TIST). Tom is currently the team leader for the Software Engineering Process Group (SEPG) and has been part of the SEPG since its creation in March 1991. He was on the assessment team during the first assessment at SM-ALC in September 1991. Tom was also part of the team that created a documented software maintenance process known at SM-ALC as the Post-Deployment Software Support (PDSS) Process. For the first few years after he came to SM-ALC, he worked as a system engineer helping to prepare the Logistics Center to support MILSTAR and other satellite systems.

Prior to joining the SM-ALC team in 1981, Tom spent about 17 years working at what was known as the Naval Weapons Center (now Naval Air Warfare Center, Weapons Division), China Lake, California. While at China Lake, Tom helped develop several radar systems and signal processing systems. During this effort, he was awarded 3 patent holding awards for some of his work. It was during these years at China Lake that he learned the value of planning, understanding processes, and project management. These are principles that he has been advocating since coming to work for the Air Force.

Tom Westaway
SM-ALC/TIST
McClellan AFB, CA 95652-
Voice: 916-643-2920 DSN 633-2920
FAX: 916-643-6292 DSN 633-6292
Internet: westaway.thomas@sma1.mcclellan.af.mil

CHAPTER 8
Addendum B

Software Complexity

Thomas J. McCabe and
Arthur H. Watson,
McCabe and Associates, Inc.

INTRODUCTION

Software complexity is one branch of software metrics that is focused on direct measurement of software attributes, as opposed to indirect software measures such as project milestone status and reported system failures. Current military metrics programs emphasize non-complexity metrics that track project management information about schedules, costs, and defects. While such project tracking measures are necessary to any substantial software engineering effort, they lack predictive power and are thus inadequate for risk management. Complexity measures can be used to predict critical information about reliability and maintainability of software systems from automatic analysis of the source code. Complexity measures also provide continuous feedback during a software project to help control the development process. During testing and maintenance, they provide detailed information about software modules to help pinpoint areas of potential instability. Figure O-11 shows the control flow graph of a simple, low-risk software module. Figure O-12 (below) shows a complex, moderate-risk software module. Figure O-13 (below) shows an extremely complex, high-risk module. Complexity metrics quantify that difference for use in software management. Measurement of software complexity provides substantial value to a software metrics program.

Open Re-engineering

There are hundreds of software complexity measures, ranging from the simple, such as source lines-of-code, to the esoteric, such as number of variable definition/usage associations. It is important to select a good subset of these measures for implementation. An important criterion for metrics selection is uniformity of application. The key idea here is "*open re-engineering*." The reason "*open systems*" are so popular for commercial software applications is that the user is guaranteed a certain level of interoperability — the applications work together in a common framework, and applications can be ported across hardware platforms with minimal impact. The open re-engineering concept is similar, in that the abstract models used to represent our software systems should be as independent as possible of implementation characteristics such as source code formatting and programming language. Complexity measurement is a fundamental application, but open re-engineering extends to other modeling techniques such as flow graphs, structure charts, and structure-based testing.

We want to be able to set complexity standards and interpret the resultant numbers uniformly across projects and languages. A particular complexity value should mean the same thing whether it was calculated from Ada source code or from Jovial. Otherwise, to get predictive benefits from the complexity measures we would have to calibrate the results based on "*similar*"

APPENDIX O Additional Volume 1 Addenda

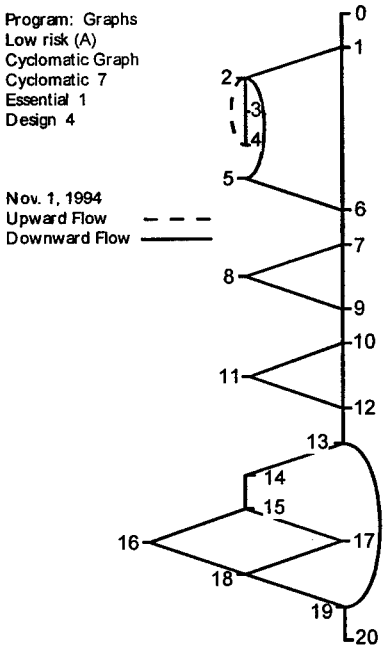


Figure O-11 Simple, Low-Risk Software Module

projects with known outcomes, and the process becomes too subjective for effective management. The most basic complexity measure, the number of lines-of-code, does not meet the open re-engineering criterion, since it is extremely sensitive to programming language, coding style, and textual formatting of the source code. The "*cyclomatic complexity*" measure, which measures the amount of decision logic in a source code function, meets the open re-engineering criterion. It is completely independent of text formatting and is nearly independent of programming language since the same fundamental decision structures tend to be available and uniformly used in all common programming languages. The software functions represented in Figures O-11, O-12, and O-13 have cyclomatic complexity measures of 7, 16, and 22 respectively.

Certainly, there are valuable complexity measures that are not "*open*." For example, the amount of access to global data elements is very useful in managing C projects, even though that measure is useless for COBOL in which all data is global. However, as a foundation for a complexity measurement program, it is best to concentrate on measures that can be applied consistently across projects and languages. That way, the same interpretations and methodology can be used without having to perform applicability assessments for each project.

Common Complexity Measures

We've already discussed lines-of-code, which is about the weakest complexity measure in common use. A refinement is to count the lines of executable code, data declarations, comments, and so on individually, then look at derived measures such as the percentage of comment lines. These all suffer from the weakness that most of what is being measured is source text format, which is not an intrinsic attribute of the software implementation. Most languages have "*pretty printers*" available that reformat code to a desired set of standards, and the "*indent*" program for C has about 50 switches that configure behavior. This leads us to a related set of measures, that of coding standards conformance. If code is supposed to have a comment at the

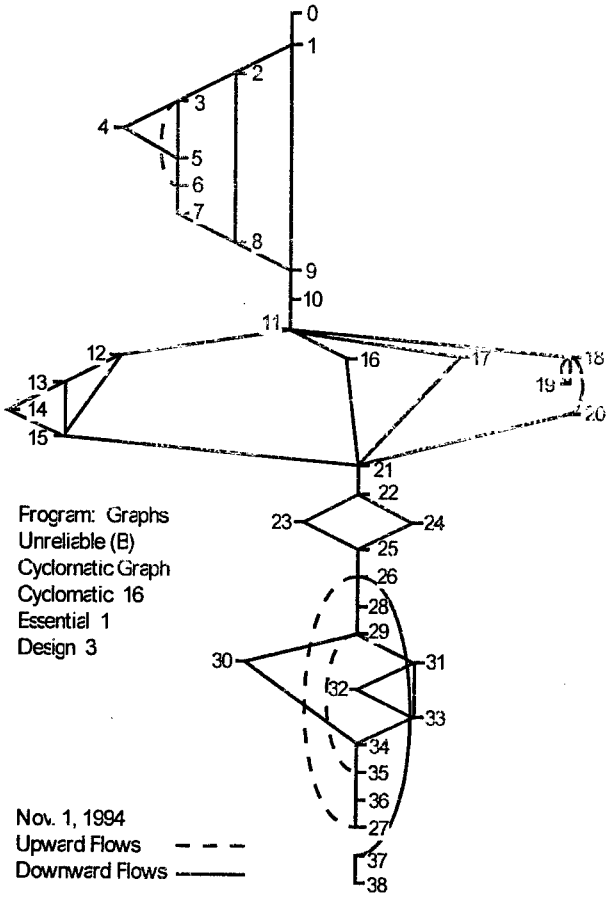


Figure O-12 Complex, Moderate-Risk Software Module

beginning of every procedure, the percentage of procedures that actually have the comment can be measured. While these source format measures give useful information for project management, they are not uniformly applicable. Their extreme sensitivity to cosmetic attributes of the source code makes them unsuitable as core complexity measures.

The Halstead Software Science metrics are a significant step up in value. [HALSTEAD77] By counting the number of total and unique operators and operands in the program, measures are derived for program size, programming effort, and estimated number of defects. Halstead metrics are independent of source code format, so they measure intrinsic attributes of the software. Since different languages have different sets of operators, it isn't immediately obvious that these measures can be applied across languages, but there's a "*language level*" measure that can help with conversion. Halstead metrics are a bit controversial, especially in terms of the psychological theory behind them, but they have been used productively on many projects. The main drawback is that the mathematical formulas of the major Halstead metrics are significantly removed from the code, so there isn't a strong prescriptive component. You can identify code as potentially unreliable, but the Halstead theory doesn't say much about

APPENDIX O Additional Volume 1 Addenda

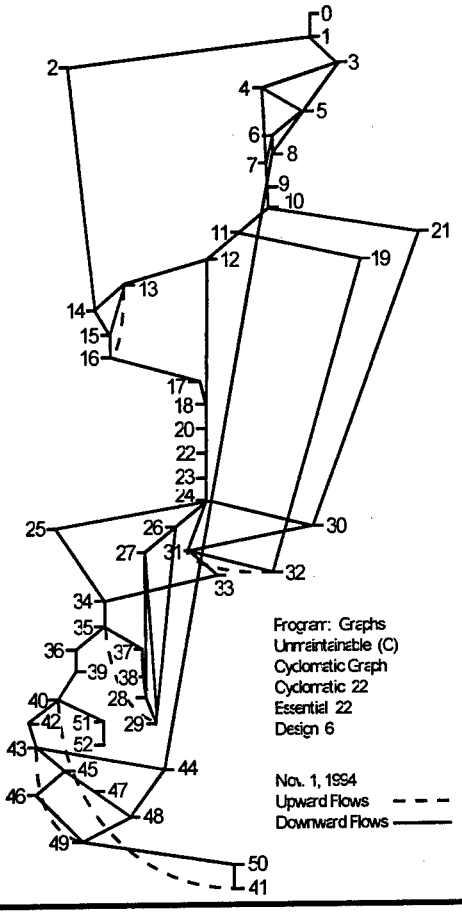


Figure O-13. Extremely High-Risk, Complex Software Module

how to test it or how to improve it. Also, and this gets back to uniformity of application, there aren't any established threshold values for what constitutes dangerous software; you're pretty much on your own when deciding what values constitute unacceptable risk. Despite these drawbacks, Halstead metrics are very useful for identifying computationally-intensive code with many dense formulas, which represent potential sources of error that other complexity measures are likely to miss.

The McCabe cyclomatic complexity measure is so versatile and widely used that it is often referred to simply as "*complexity*," and we recommend it as the foundation of any software complexity program. [McCABE76] Since it is based purely on the decision structure of the code, it is uniformly applicable across projects and languages and is completely insensitive to cosmetic changes in code. Many studies have established its correlation with errors, so it can be used to predict reliability. More significantly, studies have shown that the risk of errors jumps for functions with a cyclomatic complexity over 10, so there's a validated threshold for reliability screening. Also, this assessment can be performed incrementally during development and can even be estimated from a detailed design. For an individual software module, the programmer can easily calculate cyclomatic complexity manually by counting the decision constructs in the

code. This allows continuous control during a project, so that unreliable code is prevented at the unit development stage. Compliance can be verified at any stage of the project using automated tools. A final benefit of cyclomatic complexity, which we will discuss in more detail later on, is that it gives a precise verifiable testing prescription — the more complex and therefore error-prone a piece of software is, the more testing it requires.

There are several specialized McCabe metrics that are derived by calculating cyclomatic complexity after all control structures satisfying certain properties have been ignored. These metrics can thus be viewed as refinements of cyclomatic complexity for specific applications. The most widely used of these specialized metrics is "*essential complexity*," which measures the amount of unstructured decision logic in software. Unstructured code, typically caused by using "*goto*" statements or breaking out of loops, is harder to understand and maintain than well-structured code. This is because control structures that interact in unstructured ways cannot be decomposed, understood, and modified in isolation. Essential complexity is a widely used measure of maintenance risk, and a threshold value of four is typical for quality screening. Also, while cyclomatic complexity increases gradually when code is added during maintenance, essential complexity can increase dramatically by the addition of a single software patch. The patched code then becomes a source of risk for future maintenance. Using essential complexity to screen modules after each modification during maintenance can manage this risk.

As such, essential complexity is a good supplement to cyclomatic complexity as a cornerstone of a complexity measurement program. Although Figures O-11 and O-12 both have high cyclomatic complexity, Figure O-12 has high essential complexity and thus carries a significantly higher maintenance risk. Two other McCabe complexity variants, design complexity, which measures the amount of interaction between decision logic and subroutine calls, and data complexity, which measures the amount of interaction between decision logic and data references, are related to integration testing and design coupling. [McCABE89] These metrics are suitable for inclusion in a mature software complexity measurement program.

Complexity and Testing

The Structured Testing methodology is based on cyclomatic complexity, in the sense that the cyclomatic complexity is the number of tests required. [McCABE82] Given the correlation of complexity with errors, this is a desirable result since we want testing effort to be proportional to complexity. Many other coverage-based testing techniques, from the simple ones such as statement coverage to the complicated ones such as testing all data definition-usage associations, do not have this property. You could have arbitrarily complex software with lots of statements and data associations and still satisfy those other testing criteria with one or two tests, or you might require lots of tests. With cyclomatic complexity and Structured Testing, you know in advance exactly how many tests you'll need, so you can do detailed test planning and manage the schedules, costs, and risks associated with unit testing. Design complexity provides similar benefits for integration testing.

However, the connection between complexity and testing goes much deeper than the number of tests. From mathematical analysis, we know that the cyclomatic complexity gives the exact number of tests necessary to test each decision outcome in a function independently. The Structured Testing methodology says that we should run such a set of tests. Thus, we're not just testing statements or decisions individually; we're verifying the interactions between different parts of decision logic. In the underlying mathematical model, we can construct any path from a combination of the tests we are required to run during testing, so we're likely to detect any sources of potential error. There are techniques to calculate a set of test paths manually from the source code, and automated tools can verify that a satisfactory set of paths has actually been run during testing. The number of independent decision outcomes exercised then becomes a dynamic metric, and testing progress can be measured and managed as this number approaches the cyclomatic complexity.

APPENDIX O Additional Volume 1 Addenda

Complexity and Re-engineering

One of the most difficult tasks in software is maintaining a system without knowing the physical design of the code and how it relates to the original abstract design. For a large system, design documentation only takes you so far, then you have to work with the code. Not only does this entail risk in terms of introducing errors due to misunderstanding code, but in the absence of complexity analysis this is unmanageable risk. The scheduling and costing problems are almost as bad, since on the surface the code and documentation give very little indication of how big a particular maintenance task really is. Complexity analysis is a critical component of successful scheduling and risk management in a re-engineering environment.

Studies confirm that cyclomatic complexity is significantly correlated with debugging time, to a much greater extent than lines-of-code. [SHEPPARD81] Cyclomatic complexity has also been used successfully as the core metric of formal re-engineering cost models, and this is an area where a lot more work remains to be done. [DeFEE94] Although cyclomatic complexity is a good foundation and has been used in numerous case studies, for something like formal estimation we should work towards including a representative mix of complexity measures such as essential complexity and the Halstead metrics. Even the number of lines-of-code has a solid place in software management — complexity metrics don't replace your current system of software controls; they just add a new dimension of predictability, reliability, and risk management to your software process.

Complexity and Reuse

There's a lot of redundant code in software systems. This code duplicates the functionality and in many cases the actual implementation of other code in the system. The redundant functions tend to be maintained individually, so they diverge, and there's an enormous proliferation of errors. Redundant code is a particular risk on systems that are funded by the line-of-code, as we've seen when doing Independent Verification and Validation. It's definitely to our advantage to locate and eliminate redundant code, so that we can increase the amount of reuse and reduce the total complexity of our software. Complexity analysis can provide a lot of support. One important observation is that independent implementations of the same functionality tend to have similar control flow structure. Therefore, we can use complexity measures as a screen to identify sets of software that are potentially redundant. Using the cyclomatic and essential complexity measures to identify candidate redundant modules then proceeding to examine the full flow graph diagrams and source code, a significant amount of redundant code can be removed, with resultant benefits to system size and stability. [WILLIAMSON93]

So, complexity measurement can help us find redundant code during maintenance. But what about preventing it during development? There are many products that locate reusable code in databases, usually based on matching text in a functional description with requirements characteristics. These techniques are valuable, but are limited by the amount of effort put into documenting the code in the repository. A supplementary approach based on complexity measurement can be used with an arbitrary collection of code with no documentation or database indexing overhead. The key lies in estimating the complexity metrics of the desired component from the design or pseudo-code, and then searching the source code database for code with similar metrics. For this application, a wide variety of metrics are useful.

"*Open*" metrics are still important to find existing code in multiple languages, but if all you're looking for is Ada, you can get a lot of benefit out of measuring specific language constructs. The main requirement for using complexity measures to find reusable code is that the range of the complexity measure can be predicted from the design specification for the code. For example, you might know that a particular routine should have cyclomatic complexity between five and eight, have 20 to 50 lines-of-code, not contain any exception handlers, and contain exactly one loop. Then, just as with a text-oriented database search, you get the number of software functions that match your criteria, and you can refine or relax the criteria until you get

a reasonably sized list of candidates. At that point, you can look at the implementations and possibly save a lot of work with pretty much no extra overhead. This is just-in-time reuse, and complexity measurement provides the technology. The only organizational overhead is running a complexity measurement tool over the source code, which will be done anyway, and wasteful development of redundant code is avoided.

Implementing A Complexity Measurement Program

Complexity measurement is such a large and powerful area that it's tempting to assess hundreds of potential metrics, run pilot projects to assess potentially useful metrics, mandate data collection, correlate metrics with project performance, and eventually have a committee produce a complexity measurement policy. This doesn't work. It takes years to start getting value out of that kind of process, and we need to use complexity analysis to help manage projects right now.

The best way to implement a complexity measurement program is to start small. Collect data on a wide variety of metrics, but pick a small, validated, intuitive set of metrics to actually apply. Continue to use lines-of-code, and add cyclomatic complexity and essential complexity. Train the developers to calculate complexity by hand, and use tools to automate the process. Start using the complexity threshold of 10 immediately to improve software reliability. Start evaluating test plans in terms of complexity to make sure that error-prone code gets the testing attention that it needs. Then, once the operational benefits of complexity analysis have been widely experienced, risk management models can be refined with measures such as the Halstead metrics and data complexity.

CONCLUSIONS

Complexity analysis has an extremely high payoff for the investment. Moving from counting lines-of-code to calculating cyclomatic complexity has immediate, measurable benefits in terms of risk management, reliability prediction, cost containment, project scheduling, and improving overall software quality. Unlike the number of lines-of-code, a good measure like cyclomatic complexity can be used to give an objective assessment of software that is directly comparable across different projects, coding styles, and even programming languages. This enables organization-wide standards and procedures that can bring true repeatability and predictability to software. There are many valuable complexity metrics, and more are being developed every day, so it's important to start simple, not get overwhelmed, and build a solid complexity analysis program as a foundation for adding new metrics as their benefits are demonstrated.

Thomas J. McCabe
Voice: (410) 995-1075
Fax: (410) 995-1528
Internet: *tom@mccabe.com*

Arthur H. Watson
Voice: (410) 995-3770
Fax: (410) 720-0192
Internet: *arthur@mccabe.com*

McCabe & Associates, Inc.
5501 Twin Knolls Road, Suite 111
Columbia, MD 21045

REFERENCES

[HALSTEAD77] Halstead, Maurice H., Elements of Software Science, Elsevier North-Holland, New York, 1977

[McCABE76] McCabe, Thomas J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, SE-2 No. 4, pp. 308-320, December 1976

[McCABE89] McCabe, Thomas J., and Charles Butler, "Design Complexity Measurement and Testing," Communications of the ACM, 32, pp. 1415-1425, December 1989

[McCABE82] McCabe, Thomas J., *Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric*, National Bureau of Standards, Special Publication 500-99, December 1982

[SHEPPARD81] Sheppard, S., and E. Kruesi, *The Effects of the Symbology and Spatial Arrangement of Software Specifications in a Coding Task*, Tech Report TR-81-388200-3, General Electric Company, Arlington, Virginia., 1981

[DeFEE94] DeFee, Joseph M., "Integrating Analysis Complexity Tool Output with Formal Re-engineering Estimation Processes," *Proceedings of the Second Annual McCabe Users Group Conference*, Baltimore, Maryland, 1994

[WILLIAMSON93] Williamson, Eldonna S., "Determination of Redundancy using McCabe Complexity Metrics," *Proceedings of the First Annual McCabe Users Group Conference*, Baltimore, Maryland, 1993.

Editor's Note

This article, originally published in the December 1994 edition of *CrossTalk*, was reviewed by subject matter experts prior to publishing. One reviewer cautioned that any attempt to apply complexity measurements requires a thorough understanding of both the method and the software. When the decision is made to choose a method to measure software complexity, there is no single method that will meet every need and the use of hard and fast rules may actually increase complexity. Questions to the STSC, regarding software metrics, should be addressed to:

David R. Erickson
Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: (801) 777-8057 DSN 458-8057
Fax: (801) 777-8069 DSN 458-8069
Internet: *ericksda@hillwpos.hill.af.mil*

CHAPTER 8
Addendum C

Metrics: The Measure of Success

Editors Note

"Metrics: the Measure of Success" © 1994 was originally published as a brochure under the sponsorship of the Hughes Software Network Management Council. Reprinted with Hughes Aircraft Company's permission. All rights reserved.

FOREWORD

"Our highest priority operating commitment is to quality and continuous measurable improvement in everything we do."

These words from the statement of Hughes Guiding Values emphasize the importance of quality and continuous measurable improvement. Measurement (using metrics) serves as a powerful management tool for evaluating effectiveness and efficiency. Metrics enable us to manage on the basis of facts and data. Continuous measurable improvement cannot be achieved without measuring an existing process, changing some aspect of the process, and then measuring the result to verify it is an improvement. Measurement is an essential element in supporting Integrated Product Development (IPD). The IPD philosophy employs multi-disciplined teams to integrate and apply the best processes to effectively develop products that satisfy every customer's needs.

Common software processes based on best practices are being implemented throughout Hughes Aircraft Company to gain competitive advantage and reduce risk. These processes include standard reporting practices that define the metrics for monitoring project status and for communicating that information to functional and product management.

The metrics described in this brochure have evolved as best practices from more than 20 years of metrics data collection and reporting. They provide the means to measure cost, schedule, process, product quality, productivity, and technical parameters, each of which contribute to our fundamental measure of success — customer satisfaction. Our challenge is to continuously use data in optimizing our software development processes to ensure that Hughes remains a successful World Class performer in the global marketplace.

Terry R. Snyder
Manager, Software Engineering Division
Chairman, Software Network Management Council
Hughes Aircraft Company

APPENDIX O Additional Volume 1 Addenda

INTRODUCTION

Building a successful business often means building better software. In today's highly competitive markets, where software is increasingly a critical factor, a company's ability to plan and control its software development activities is essential. But, according to the Software Engineering Institute (SEI), most American companies lack a well defined and measurable process for managing software development. Typically, these companies face a high rate of failure because of unpredictable product quality, higher costs, and delivery schedules that are out of control.

There is a better way. Using the SEI's Capability Maturity Model as a framework, we find that implementing process maturity criteria, based on a controlled and measurable software development process, can reduce costly software errors, cut the risk factors, increase productivity and product quality, and shorten cycle time. Those organizations that achieve higher process maturity levels typically demonstrate significant control over their software development processes. One of the keys to achieving this control is the use of metrics. Basically, a metric is a standard of measurement. Just as a yardstick is used to measure height in inches or feet, a software metric allows us to use quantitative values to assess how well we're doing in relation to things such as budget or schedule.

Nowadays, a physician uses a digital thermometer to measure a patient's temperature. If the temperature exceeds the norm by several degrees, it's an indication that something may be wrong. Other tests are run to determine the cause of the increased temperature (e.g., blood pressure, respiration, and white blood count). Once the medical data (usually referred to as the patient's "*vital statistics*") has been collected and analyzed, the doctor can recommend treatment to restore the patient to health.

Metrics have a similar function. During the life of a software development project, metrics are the statistical tools that help the software management organization determine how well it is achieving its scheduled commitments. Basically, metrics provide the factual basis for effectively evaluating a project's performance over time (measured in relation to budget, schedule, and other key factors). Metrics are used by managers to assess the progress being made (the overall "*health*" of the project) or to detect unfavorable trends and do something about them before they become show stoppers. Organizations that wish to improve their software development processes can realize a significant return by establishing metrics programs that include the people, facilities, tools and training required for collecting and interpreting metrics data. Hughes Aircraft Company's expanding emphasis on metrics reflects the company's top-down commitment to quality and continuous measurable improvement. The story that follows provides insight into how metrics can be used to quantitatively — and successfully — manage software projects.

This document is not intended as an academic exercise nor does it cover all of Hughes' metrics activities. It provides a description of 12 metrics that have proven useful in the management of software development projects at Hughes. Our experience with software measurement over the years has shown us that metrics work — they help us do our jobs better. The objective in this brochure is not only to define the metrics themselves, but to describe their use (and ultimate value) in the real world by relating them to a hypothetical project.

OVERVIEW OF THE SAMPLE PROJECT

This brochure uses a hypothetical software development project to illustrate the uses of metrics. It provides a basis for understanding how metrics actually function in the life and health of a project. Although the data shown is entirely fictitious, the intent is to make this generic software project as "*real*" as possible, based on the sort of problems one would expect to encounter in a development effort of moderate size. The project has the following general characteristics:

- 30 month schedule,
- 82,000 source lines-of-code (SLOC) delivered,
- Staff peak of 33 people,
- Consists of two software builds, and
- At this point in time, the first build has been completed and we are two-thirds into the project schedule.

The story line has one main theme: during the Preliminary Design phase, project personnel tried to reduce the size and complexity of the job through automatically generated code and software reuse. But, this inadvertently created a problem: the new design required more memory than allocated. Without metrics, the problem might not have surfaced until very late in the project, during the final stages of integration and testing (around months 23-24), resulting in a major redesign effort with a severe impact on cost and schedule. With metrics, just as in medical science, the key to success is the early detection of problems.

Project Schedule

Project managers need to visualize how and when key activities are planned to occur over the life of the project. They also need a progress report that tells them where they are in relation to this plan. The Project Schedule is a tool that provides a "*quick picture*" of the project. It maps out the workflow, shows the relationships between activities over time, and illustrates progress. It can also be used to identify problems and as an aid to taking corrective action.

The Project Schedule is structured as a timeline plotting all major activities and milestones from project inception to scheduled completion. This schedule reflects all current officially approved activities, dates, and progress status (including any slippage from the officially approved plan). Also shown (at the bottom of the schedule) is a running total of planned and actual data deliveries for each scheduled timeline increment.

In our sample project, there were some early problems in accomplishing the software design. The need to address these design problems caused delays, which, in turn, had a ripple effect on the design reviews. However, early corrective actions allowed the overall project to recover and get back on schedule.

Milestone Reports

The Milestone Report is intended to identify major software tasks specified in the Project Schedule, intermediate checkpoints for those major tasks, more detailed software component and documentation deliveries, and management-oriented activities (for example, quarterly self-audits). These milestones are listed by description, plan date, latest estimate date, actual date, and reason for slippage.

The Milestone Summary Report is a summary of the data from the Milestone Report that can be merged with milestone data from other projects to measure the organization's overall effectiveness in meeting its customers' expectations for delivery. In order to merge later milestone data from many different projects, a set of standard milestones is included in all project milestone statusing and used as the basis for the data in the Milestone Summary Report.

Rate Chart Report

Software development activities must be planned in great detail. For each of the hundreds (and often thousands) of software units, milestones are established for design, code, unit test, and integration. As time passes, the actual completion dates for these milestones are recorded. This milestone data can be used to measure the software development effort. The reporting of this status is accomplished using Rate Charts.

APPENDIX O Additional Volume 1 Addenda

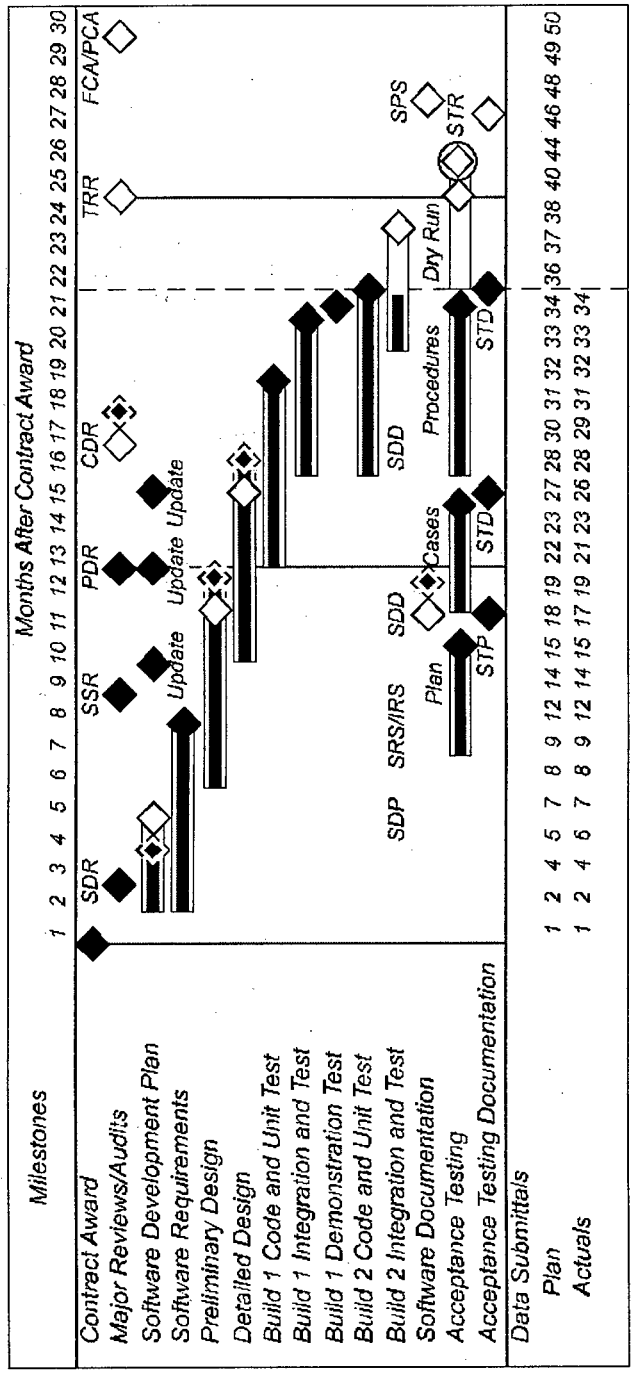


Figure O-14

A Rate Chart for an activity is a two-dimensional graph showing a plan line and an actual line versus calendar time. The plan line shows the percent of milestones planned to be completed versus time. The actual line shows the percent actually completed. With this effective management tool, you can quickly see where you are in relation to the plan at a given point in time, and you can visualize your rate of progress. By comparing the actual line with the planned line, management is alerted to trouble early in the development process and can consider corrective action to remedy the situation (for example, allocation of more resources).

A phase Rate Chart simultaneously provides rate charts for design, code, unit test, and integration. A composite Rate Chart depicts, at a higher level, a weighted summation of several parallel activities of planned and actual work accomplished versus time. It is designed to provide a means to assess the overall progress and status for a project and its individual development activities.

Most software development Rate Charts are defined and measured in terms of software units. However, any activity that can be planned as a detailed set of similar milestones can be depicted as a Rate Chart. Thus, Rate Charts can also be used to report the status of such tasks as documentation and formal test. In our sample project, the Rate Chart shows rapid early progress due, apparently, to the lower complexity of the job. But, once the design problem was caught and corrective action took effect, the rate dropped off and then gradually recovered.

Earned Value Report

The Earned Value Report compares work accomplished against work planned. In this report, the financial budget (Budgeted Cost of Work Scheduled or BCWS) is plotted along with the actual expenditures (Actual Cost of Work Performed or ACWP). A third plot, which represents earned value or the amount of the job that has been completed (Budgeted Cost of Work Performed or BCWP), is then added to the picture. This graphically illustrates the project schedule and financial status at a glance. What you get is an overview of the project's health. For example, if half the money has been spent (ACWP), then half the job should be completed (BCWP). If money is being expended (ACWP) faster than planned (BCWS), there should be a corresponding assessment of progress against the project's milestones (BCWP).

Based on the earned value plots, two additional key indicators can easily be derived and plotted: the Cost Performance Index (CPI) and the Schedule Performance Index (SPI). The CPI is the ratio of work completed in dollars (BCWP) divided by the actual cost of performing the completed work (ACWP). Values greater than "*1*" mean it is costing less than originally planned to perform the work. The SPI is the ratio of work accomplished (BCWP) divided by work planned (BCWS). Numbers greater than "*1*" signify that work is being achieved quicker than originally planned (i.e., the activity is ahead of schedule).

We see in our sample project that, around month 8, there was a critical dip in the key indicators: money was being spent, but very little (in terms of earned value) was being accomplished. We were faced with a slowdown in performance. Immediate corrective action was needed to offset the possibility of a cost overrun and/or late delivery. Both the SPI and the CPI suffered as a result, but the project gradually recovered once the design problem was solved. Easy to use and understand, these key indicators are needed to measure and assess a project's progress and health. They convey powerful messages. Are we meeting our schedule? Are we meeting costs? Is the project healthy?

Financial/Staffing Report

The Financial/Staffing Report provides a clear indication to management of a project's performance in relation to financial and staffing plans. The Financial graph shows the cumulative dollars spent to date compared with the projected budget. The Staffing profile reveals the number of personnel working on the project each month with respect to the planned staffing levels. The initial baseline plan is established at the beginning of the project. On a monthly basis, the actuals are compared to the plan. In the event an activity needs to be replanned, the original

APPENDIX O Additional Volume 1 Addenda

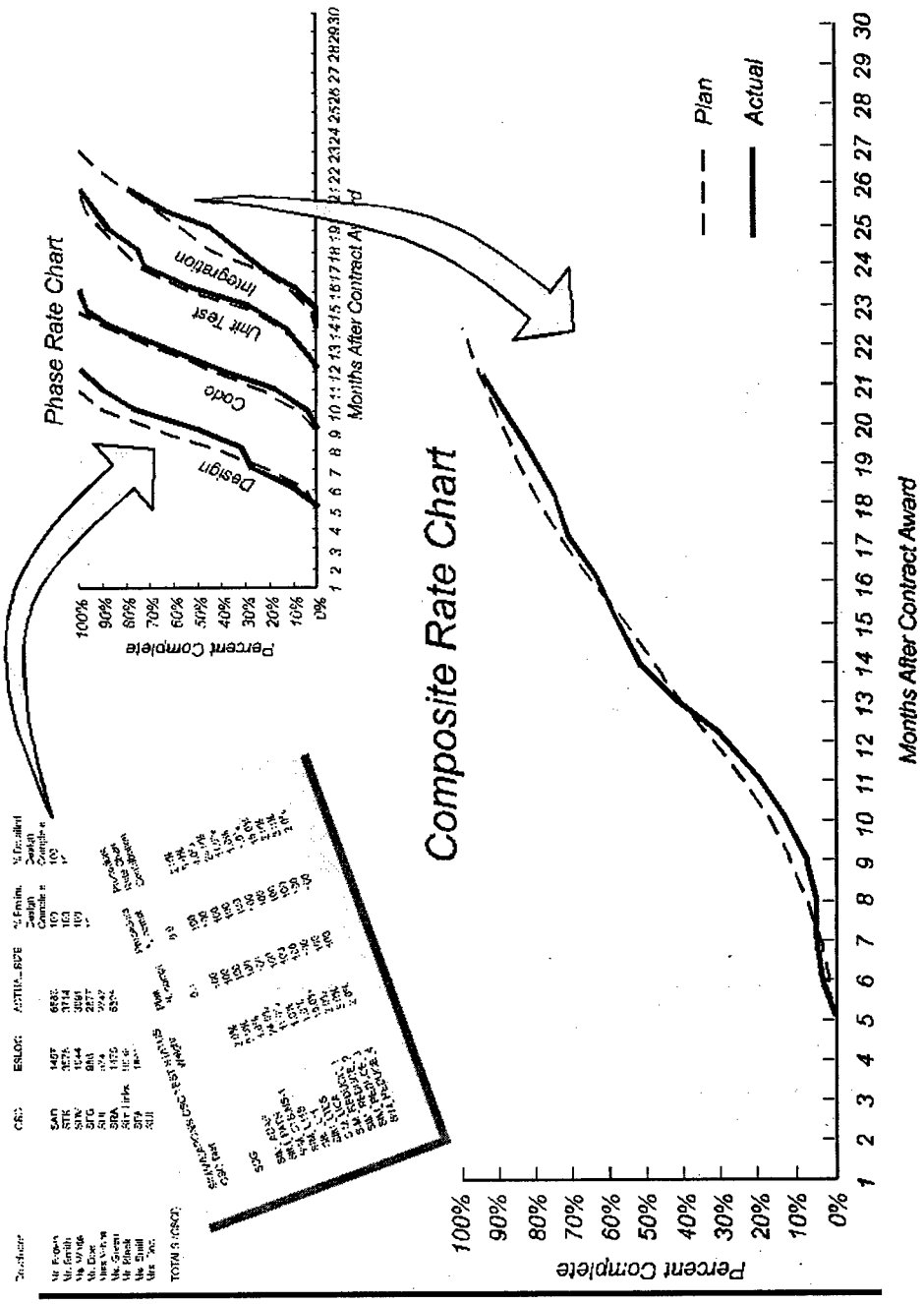


Figure O-15

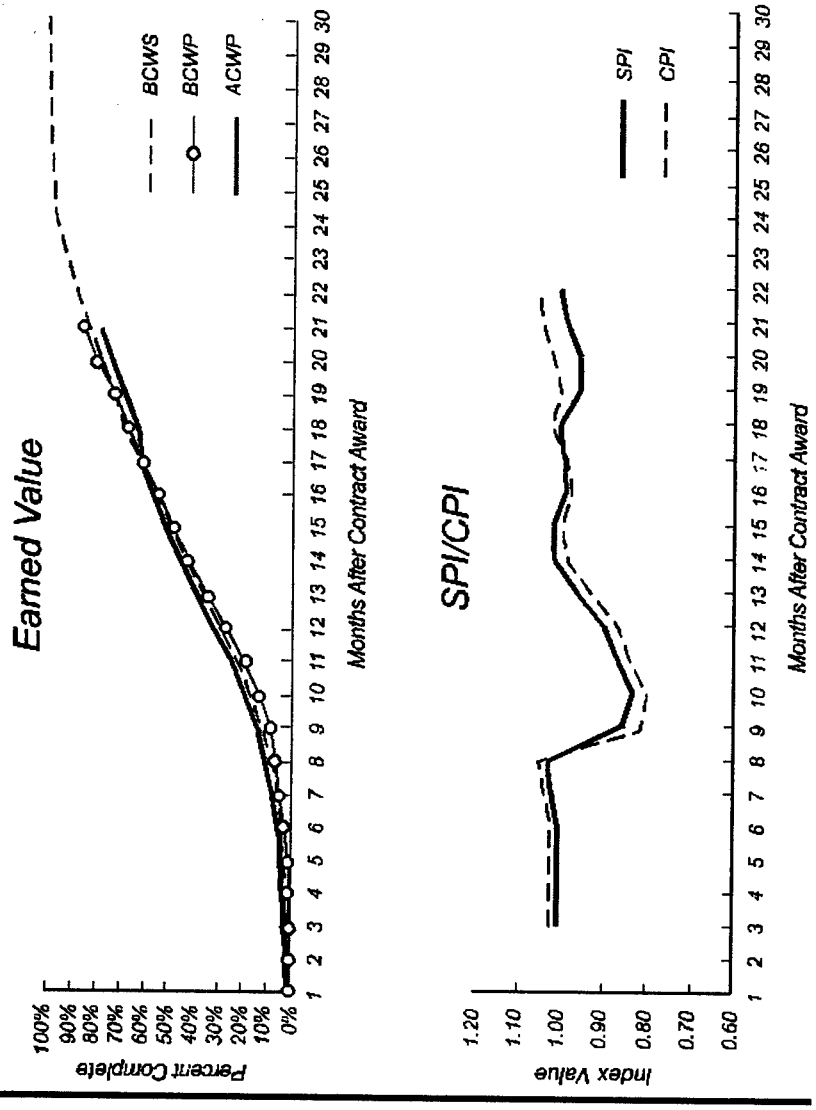


Figure O-16

plan is retained as a baseline and a Current Operating Plan (COP) is used to forecast the revised spending and staffing. In our sample project, we recognized the need to take corrective action to fix the design problem highlighted in the Target System Resource Usage Report. To accomplish this, more project personnel than originally planned were assigned early in the recovery effort (during months 8-9) to preserve the schedule. Also, during months 17-20, it appeared that project staff were being redirected to other efforts and, thus, were not available to this project. The Staffing metric pointed to this problem and allowed for corrective action.

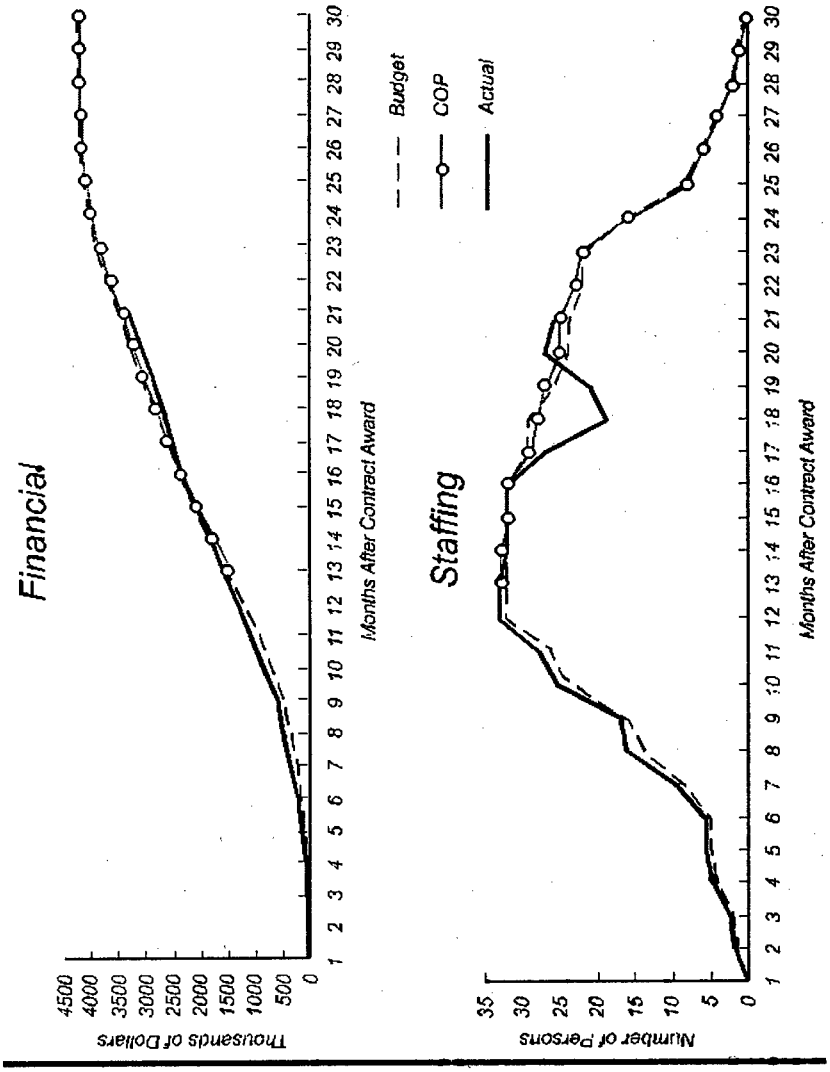


Figure O-17

Size Trend Report

The purpose of the Size Trend Report is to uncover potential development problems related to changes in code size before they become critical. This report is updated monthly to show the estimated size at completion. The size of each Computer Software Configuration Item (CSCI) is tracked via separate Size Trend Reports. The Size Trend Report is used in conjunction with the Productivity Measurement Report. If size is increasing (which is often the case in software development projects) and productivity is not above what was planned, a significant problem exists requiring immediate attention — namely, at a given productivity rate with an increasing job size, it will take longer to finish the job than what was scheduled.

In our sample project, the size of the job (measured in equivalent source lines-of-code or ESLOC, derived from actual source lines-of-code by taking into account the reduced effort for reused and modified code) was reduced because of an efficient design that introduced more functionality in auto code and less in new code. But, we see that the size of the delivered source lines-of-code (DSLOC) ballooned because of the growth in the less efficient auto code, which directly impacted memory use. What we were faced with was a potential "*fit*" problem requiring more memory than allocated. A new design was required, resulting in corrective action that had an immediate impact on productivity and cost performance.

Productivity Measurement Report

The Productivity Measurement Report is used to evaluate a project's performance by measuring software productivity, which is based on equivalent source lines-of-code (ESLOC) produced per staff month and is reported for each Computer Software Configuration Item (CSCI). What the report shows is the planned productivity (based on the estimated size of the CSCI, cumulative planned staffing and planned percent complete) versus actual productivity (based on current estimated or actual code counts of the CSCI, cumulative actual staffing and the actual percent complete). Not only does this support better project management, but it also establishes a baseline for measuring improvement and future cost estimates needed for bidding. In our sample project, we can see that productivity was running high early in the project due to the lower complexity of the job. However, once it was realized that a portion of the design would have to be redone to correct the memory use problem, the rate of productivity slowed down dramatically (while corrective action was taken) and then gradually recovered.

Software Problem Status Report

The purpose of the Software Problem Status Report is to track the maturity of deliverable products, measuring the status of known software problems that need to be fixed versus the rate of closure of problems. Software problems are tracked after software goes under configuration control, usually at the start of software integration.

The Problem Change Request is a form used to document software problems. Based on the number of opened Problem Change Requests, the Software Problem Status Report serves as an indicator of the maturity of the software product. Software usually is ready for delivery when the rate of finding new problems has decreased significantly and the gap between opened and closed problems is near zero. The following measurements are plotted on the graph:

- Opened shows the number of software problems that have been detected and reported by the designated project authority.
- Resolved shows the number of software problems for which a technical solution has been found and verified.
- Closed shows the number of problems that have had the correction verified and formally closed by incorporating the change in the product baseline.

Some projects also find it advantageous to show the number of software problems a project of this size can expect to encounter over time, based on historical data from similar projects. In our sample project, the problem reporting began in month 16 as integration started. Initially, problems were detected faster than they could be resolved. Without corrective action (for example, assigning more resources), a schedule slip was likely. This project was able to eventually close the gap between opened and closed problem reports and the rate of new problem reports tapered off, indicating that the software was ready for the Build 1 demonstration during month 21.

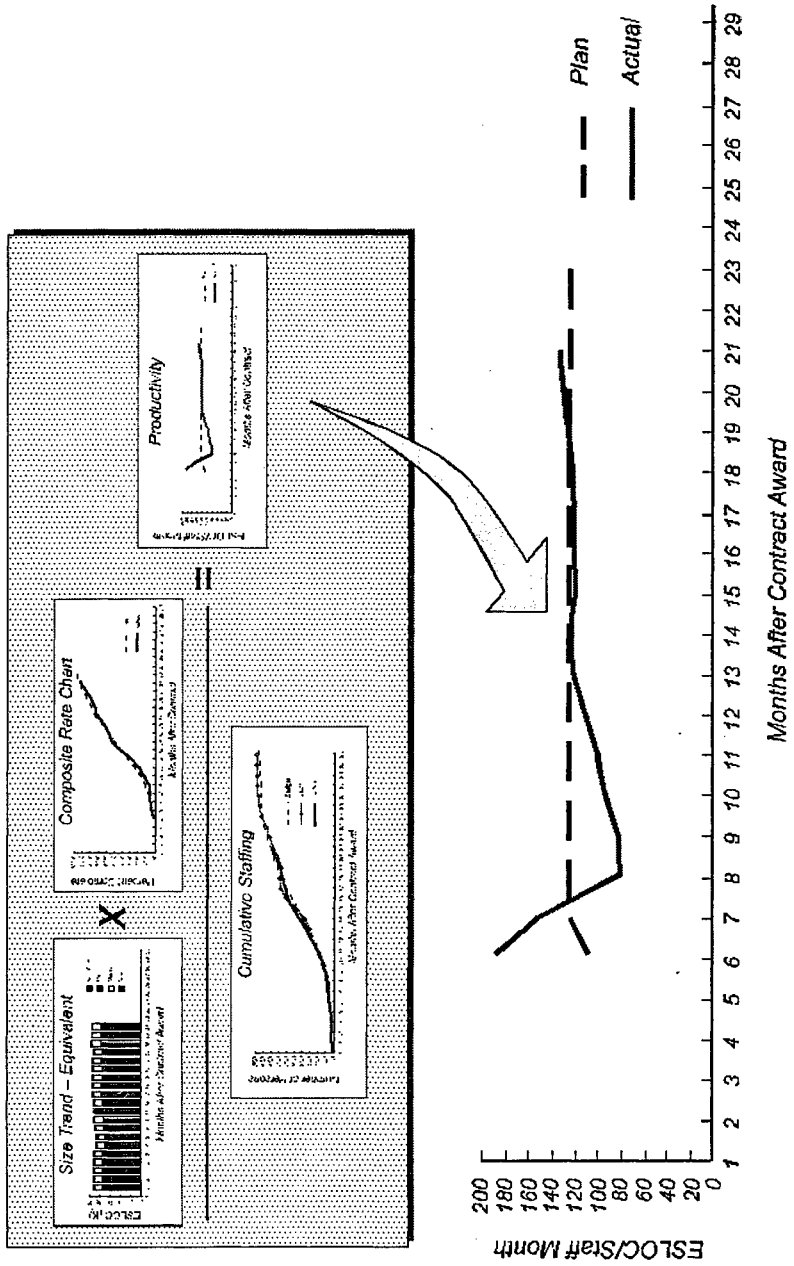

Size Trend – Equivalent
Composite Rate Chart
Productivity
Cumulative Staffing
Plan
Actual
ESLOC/Staff Month
Months After Contract Award
0
20
40
60
80
100
120
140
160
180
200
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

Figure O-18

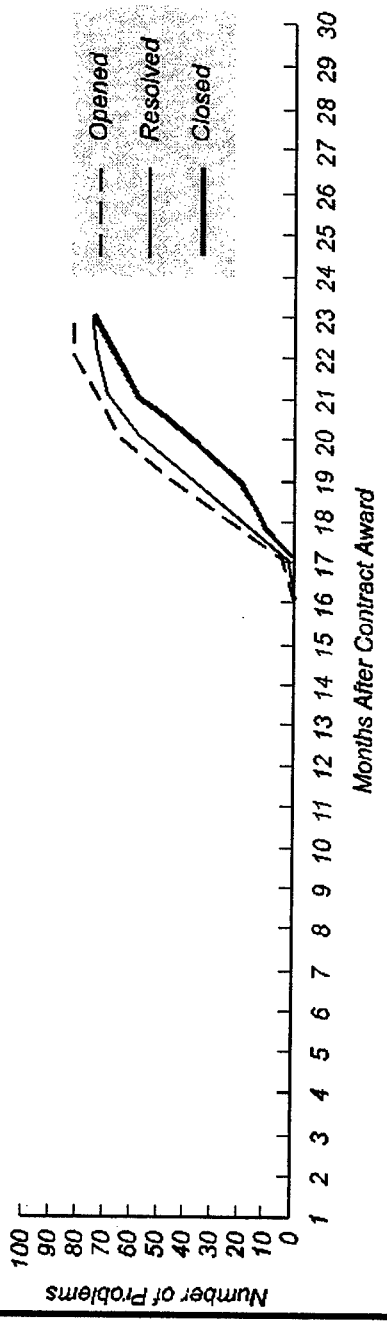

Opened
Resolved
Closed
Number of Problems
0
10
20
30
40
50
60
70
80
90
100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Months After Contract Award

Figure O-19

APPENDIX O Additional Volume 1 Addenda

Quality Indicator Reports

The Quality Indicator Reports are summaries of defect data that can be used to understand and improve the software process. Software quality indicators include both product and process defects.

Defects can be introduced into software during each phase of the software development life cycle. Procedures need to be established to prevent these defects, wherever possible, or discover their existence at the earliest possible moment. Data from Hughes projects and other industry sources shows that the earlier you catch the problem, the less costly it is to fix. The Quality Indicator Reports categorize the kinds of defects discovered and the life cycle phase in which the defects were detected. The Quality Indicator Reports consist of the following charts:

- A summary chart showing quality indicator status,
- Charts that focus on type of defect,
- Charts that focus on phase detected, and
- A quality indicator analysis report.

The primary goal of the various charts is to provide management with increased visibility into patterns and trends appearing from the statistical summary of defects discovered during the software life cycle. This data gives the manager and technical personnel the ability to home in on the specific causes of defects. Defect prevention teams can analyze these charts and outline a plan of action to eliminate the root cause of the most frequent and/or costly defects. The Quality Indicator Reports are an increasingly important part of the Hughes approach to continuous measurable improvement (cmi).

Defect Density Tracking Report

Defects are quite simply deficiencies in a product or process (flaws in the design or coding process, for example) that can lead to development of a product that fails to meet customer requirements. They are found during both internal and external reviews and testing of software products under development. Projects identify defects during reviews based on review procedures and checklists. Defects are then documented when identified, including information on the type of defect, the cost to repair, and the phase in which each defect was detected.

Defects cost time and money. Even with defect prevention techniques, experience shows that the best software processes today are not yet capable of "*defect free*" results. So, we can expect to find defects. Historical data from past Hughes projects indicates that it costs significantly less to find and fix defects in the development phase that caused them rather than later in the project. The early detection of problems is the key to maintaining high productivity at minimum costs.

The Defect Density Tracking Report is used to compare the number of defects being found in each phase of software development with historically derived control limits. The control limits determine a planned range of defects for each phase that a typical project should experience. The actual defect rates being found are then compared with the control limits to provide insight into the process being used on the project.

If the actual defect rates are within the control limits of the planned defect density, the process being used by the project is performing as expected. The fact that defect rates are lower than the lower control limit requires further analysis. It may indicate that insufficient reviews are being conducted, calling for immediate corrective action. Or, this may mean that the project has found a process improvement that has produced fewer defects. If so, the improved process should be documented for others to use. Similarly, defect rates that exceed the upper control limit indicate conditions that should be examined for possible improvements.

In the sample project, where Coding Defect Density was measured per thousand equivalent source lines-of-code (KESLOC), we saw that the actual data was within the control

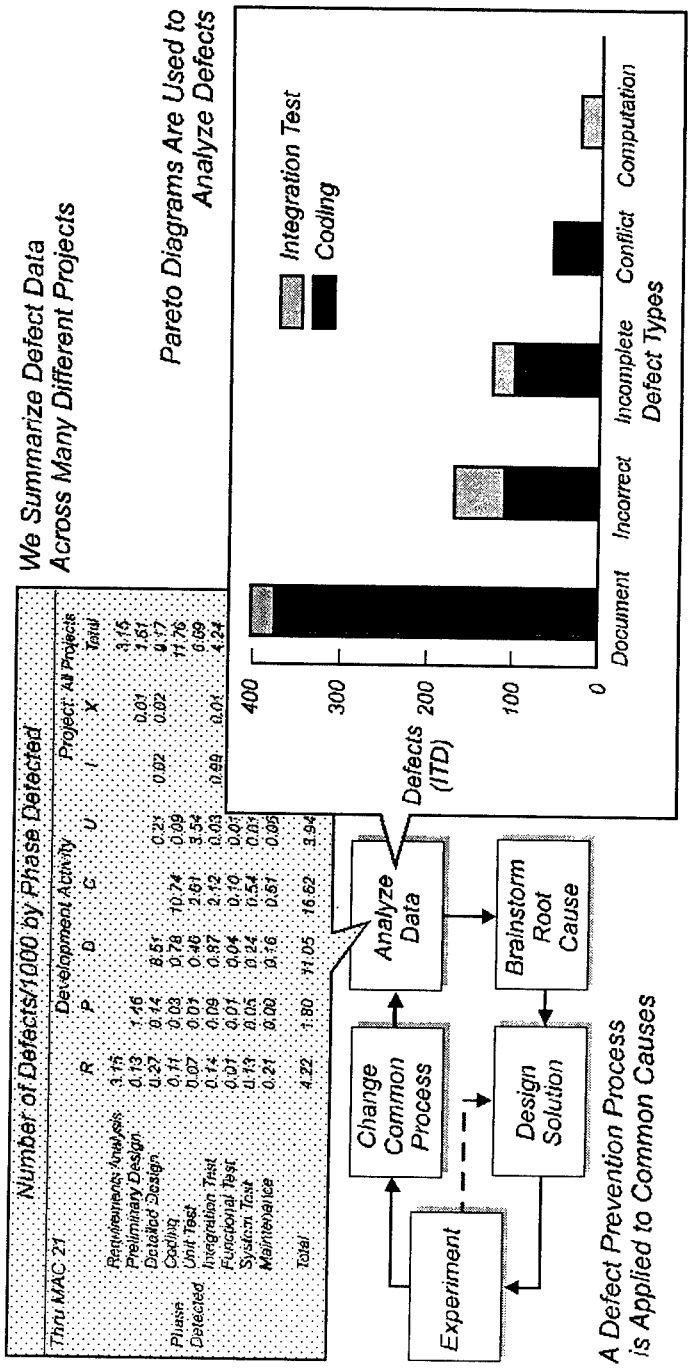

Number of Defects/1000 by Phase Detected
Thru MAC 21
Development Activity
Project: All Projects
Phase Detected
R
P
D
C
U
I
X
Total
Requirements Analysis 3.15 3.15
Preliminary Design 0.13 1.46 1.61
Detailed Design 0.27 0.14 8.51 0.01 0.02 9.17
Coding 0.11 0.03 0.78 10.74 0.21 0.02 11.76
Unit Test 0.07 0.01 0.46 2.61 3.54 0.09 6.09
Integration Test 0.14 0.09 0.87 2.12 0.03 0.69 0.01 4.24
Functional Test 0.01 0.01 0.04 0.10 0.01
System Test 0.13 0.05 0.24 0.54 0.01
Maintenance 0.21 0.00 0.16 0.61 0.05
Total 4.22 1.80 11.05 16.62 3.94
We Summarize Defect Data
Across Many Different Projects
Pareto Diagrams Are Used to
Analyze Defects
Integration Test
Coding
400
300
200
100
0
Defects (ITD)
Document
Incorrect
Incomplete
Conflict
Computation
Defect Types
Analyze Data
Brainstorm Root Cause
Change Common Process
Design Solution
Experiment
A Defect Prevention Process
is Applied to Common Causes

Figure O-20

limits showing that the project was performing as expected. Hughes is constantly improving its metrics program. The Defect Density Tracking Report is a recent addition and is currently being piloted on some projects.

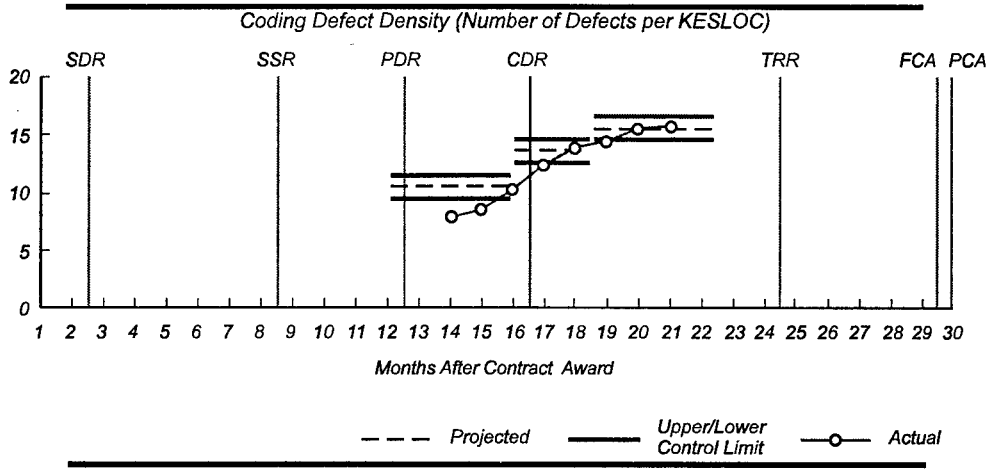


Figure O-21

Target System Resource Usage Report

Target System Resource Usage reporting is mainly concerned with the management of computer resources (e.g., main memory, processor time, mass storage, etc.), measured in terms of percentage utilized. Selection of which resources to monitor is done in the requirements analysis phase based on how critical the resource is or how risky it will be to meet the requirement.

Initially, control limits are established that represent the maximum resource utilization allowed under the contract, along with a management reserve that decreases in size as the ability to estimate/measure the utilization improves during the life of the project. Exceeding the management reserve figure indicates a "*risk*" calling for management action. During the life of the project, the actual resource utilization is estimated with more and more precision until actual measurements finally replace the estimates.

The key to success here (as in all the metrics) is to manage risk early in the project. In our sample project, the Target System Resource Usage Report provided an estimate that showed the new design would require too much memory. This estimate (long before actual memory usage could be measured) allowed us to focus on the problem early enough so that corrective action could be made with minimal effects on the project. Without the use of metrics, the problem would not have surfaced until very late in the project, during the final stages of integration and testing, resulting in a major redesign effort that would have had a severe impact on cost and schedule.

Scope Change Report

The Scope Change Report describes any addition or deletion in the scope of work. It forces management to "*size up*" a new task or requirement and assess the impact of these changes on cost and/or schedule. Essentially, it is a technique for managing (and forecasting) changes in the contract's scope of work, common to evolutionary product development, which gives project management a basis for understanding what is currently in the baseline and what is not. This planning tool is also used to look at related new business opportunities for a project (normally

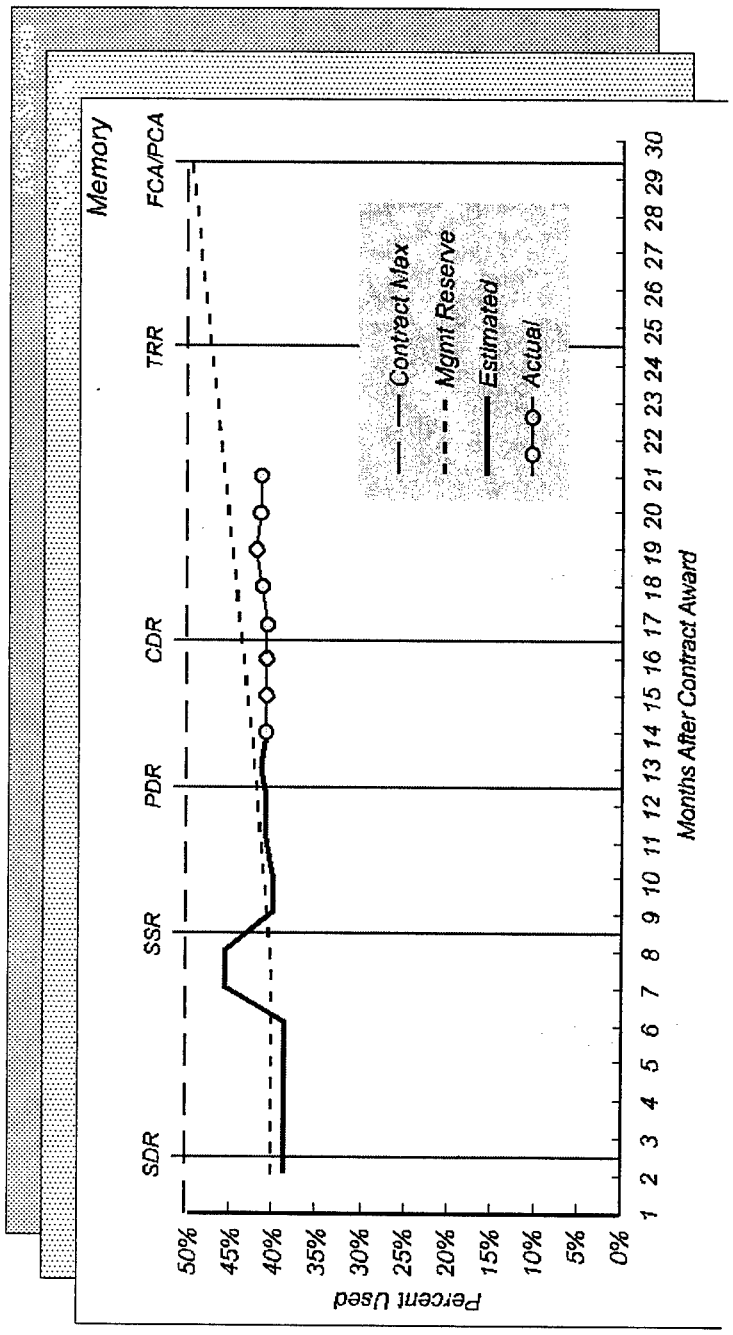
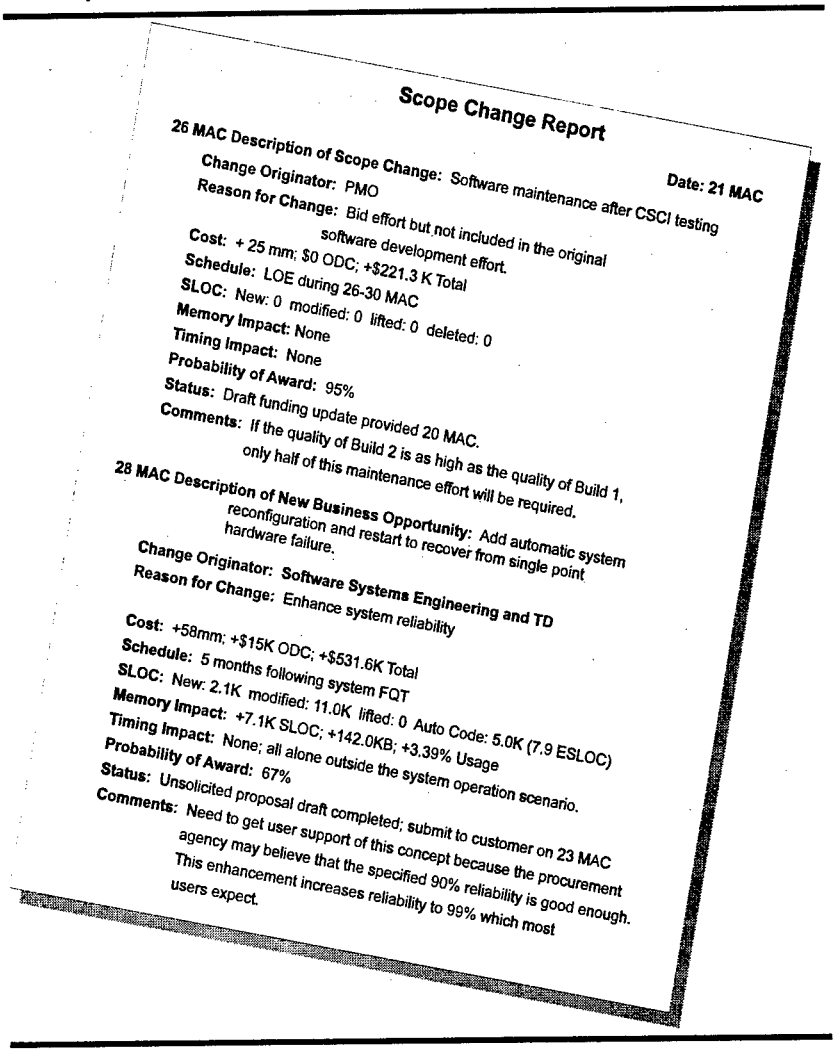

Memory
SDR
SSR
PDR
CDR
TRR
FCA/PCA
Contract Max
Mgmt Reserve
Estimated
Actual
Percent Used
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Months After Contract Award

Figure O-22

follow-on work from existing contracts) and to assess their impact, both technically and financially.



Scope Change Report

26 MAC Description of Scope Change: Software maintenance after CSCI testing **Date: 21 MAC**

Change Originator: PMO

Reason for Change: Bid effort but not included in the original software development effort.

Cost: + 25 mm; $0 ODC; +$221.3 K Total

Schedule: LOE during 26-30 MAC

SLOC: New: 0 modified: 0 lifted: 0 deleted: 0

Memory Impact: None

Timing Impact: None

Probability of Award: 95%

Status: Draft funding update provided 20 MAC.

Comments: If the quality of Build 2 is as high as the quality of Build 1, only half of this maintenance effort will be required.

28 MAC Description of New Business Opportunity: Add automatic system reconfiguration and restart to recover from single point hardware failure.

Change Originator: **Software Systems Engineering and TD**

Reason for Change: Enhance system reliability

Cost: +58mm; +$15K ODC; +$531.6K Total

Schedule: 5 months following system FQT

SLOC: New: 2.1K modified: 11.0K lifted: 0 Auto Code: 5.0K (7.9 ESLOC)

Memory Impact: +7.1K SLOC; +142.0KB; +3.39% Usage

Timing Impact: None; all alone outside the system operation scenario.

Probability of Award: 67%

Status: Unsolicited proposal draft completed; submit to customer on 23 MAC

Comments: Need to get user support of this concept because the procurement agency may believe that the specified 90% reliability is good enough. This enhancement increases reliability to 99% which most users expect.

Figure O-23

AFTERWORD

When flying a plane from Los Angeles to San Francisco, a pilot knows the destination and the direction in which to begin the flight. During the journey, however, winds may be encountered that cause the plane to go off course. The earlier the pilot detects the problem and corrects for the wind, the more likely the flight will arrive on schedule. Similarly, software metrics provide the software project manager with the information necessary to determine when a software development project is on course. As shown by the sample project in this brochure, metrics analysis enables early detection of problems and allows corrective actions to ensure delivery of quality products that are on time, within budget, and meet customer expectations.

Hughes has developed several tools for in-house use in collecting, analyzing, and reporting software metrics data. These include the Quality Indicator Program (QIP) and the Quantitative Process Management Information System (QPMIS). QIP has been used to collect data on nearly 100,000 defects during the past five years. Defect prevention teams analyze this data for root causes and select pilot projects to test proposed changes. Valid improvements are incorporated into the organization's practices and procedures.

QPMIS captures metrics data in a common database and automates the analysis and reporting for many of the reports described in this brochure. QPMIS allows software project managers to concentrate on the content and interpretation of the data and not concern themselves with formatting reports. A centralized historical database aggregates the information from each completed project and allows electronic access to the data for organization-wide trend analysis and for future reference.

Hughes has made a strong commitment to metrics over the years with significant support from the Software Network Management Council and the Company-wide Software Initiatives Program. Training courses are offered regularly and cover a wide range of topics related to metrics including Software Project Reporting, Introduction to Quantitative Process Management, Defects Collection, Analysis and Prevention, and Reaching for Higher Levels of Software Process Maturity.

Software development is a human intensive activity and, as such, is subject to human error. However, many errors and much costly rework can be avoided by continuously measuring and optimizing defined development processes. Improvements pay off in many ways. Lessons learned from past projects enable new projects to be more efficient. Analysis of historical data results in greater accuracy in predicting costs, schedules, and risks during proposal activities. Even small cost performance index improvements can translate into substantial savings when accumulated over time on multiple projects.

The successful companies of the future will be those that take advantage of metrics to institutionalize "*best in class*" practices and procedures for software development.

Glossary/Acronym List

ACWP	Actual Cost of Work Performed
BCWP	Budgeted Cost of Work Performed
BCWS	Budgeted Cost of Work Scheduled
CDR	Critical Design Review
cmi	Continuous Measurable Improvement
COP	Current Operating Plan
CPI	Cost Performance Index
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
DSLOC	Delivered Source Lines-of-code
ESLOC	Equivalent Source Lines-of-code
FCA	Functional Configuration Audit
FQT	Formal Qualification Test
IPD	Integrated Product Development
IRS	Interface Requirements Specification
KESLOC	Thousand Equivalent Source Lines-of-code
LOE	Level of Effort
MAC	Month After Contract
ODC	Other Direct Costs
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PMO	Project (or Program) Management Office
QIP	Quality Indicator Program

APPENDIX O Additional Volume 1 Addenda

QPMIS	Quantitative Process Management Information System
SDD	Software Design Document
SDR	System Design Review
SDP	Software Development Plan
SEI	Software Engineering Institute
SLOC	Source Lines-of-code
SPI	Schedule Performance Index
SRS	Software Requirements Specification
SSR	Software Specification Review
STD	Software Test Descriptions
STP	Software Test Plan
STR	Software Test Report
TD	Technical Director
TRR	Test Readiness Review

CHAPTER 8
Addendum D

Making Metrics Work Miracles

David A Haakenson and
David R Webb
Ogden Air Logistics Center

Let's face it. Measurements are boring. No matter how interesting the subject being measured might seem, measurements are just numbers, and a parade of numbers across any paper is bound to make most people stifle a yawn or two. A vast majority of people (the authors included) will simply keep turning pages, looking for pretty pictures. Measurements can be represented graphically, which gives the reader pretty pictures to look at while conveying the numbers, but it doesn't take long to realize that measurements, of themselves, tell the reader little — this many of such-and-such widget were produced. Great. Turn the page. Look for more pictures.

It is comparisons of these measurements with others taken at various points in time that make the numbers interesting and since most of you have already seen where we are going with this, we will just blurt it out: measurements are not interesting until they become *metrics*. The purpose of this article is not to teach what metrics are or how to use them. We will demonstrate how, in an F-16 software project at Hill Air Force Base, we turned measurements into metrics, how we used those metrics to improve the way we do business, and how this process altered our ways of thinking. We will also share some lessons we learned that might help anyone who attempts to use their metrics for similar improvements efforts.

In 1992, the Software Division of the Technology and Industrial Support Directorate (TIS) at Hill Air Force Base was working on the contact to upgrade the software in the suite of Block 30 F-16 (early model Fighting Falcon jet) core avionics computers in a project called System Capabilities Upgrade 2 (SCU-2). The division had been doing the F-16 airplane software for many years and the project, though somewhat arduous, was not unusual. Designers knew what to design. Coders knew what to code, and testers knew what to test. The problem was this was our first exposure to the Block 30 aircraft, and several of our old rules no longer applied. There also were other problems. There was a great deal of rework in SCU-2, much of it performed late in the project. This made life a little unpleasant for many of the workers who had to deal with long hours and short tempers. In the end, however, TIS delivered a fine product at a great price.

Still, the amount of rework performed in SCU-2 bothered us. We began to wonder just how much better and how much more cost-effective the product might have been with less rework. We had recorded all the problem reports and their solutions, but as raw measurements, the numbers told us little. So, with our next Block 30 project impending, we decided to turn those measurements into metrics, find out what was going on, and try to correct the problem in SCU-3.

APPENDIX O Additional Volume 1 Addenda

Since a metric is, in part, a collection of measurements taken over time, we began by taking our simplest rework measurements (numbers of problem reports) and putting them on a graph with time as the X-axis (see Figures O-24 and O-25). The total number of problems reported increased as the project progressed, and the total number of problems still being worked decreased over time. We divided these problem reports into categories to fell us what their status was at any point in time and saw that problem reports were being correctly analyzed, defined, assigned, and fixed.

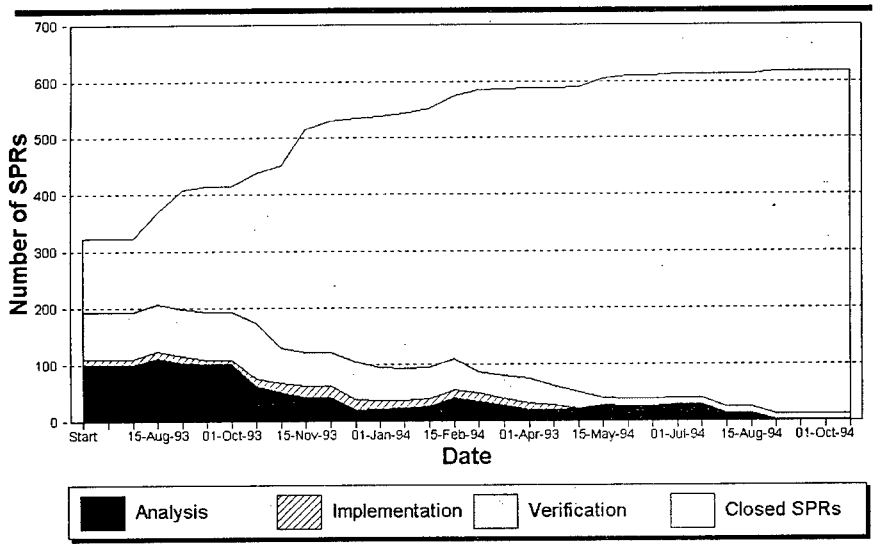


Figure O-24 Open System Problem Reports (SPRs) Highlighted[1]

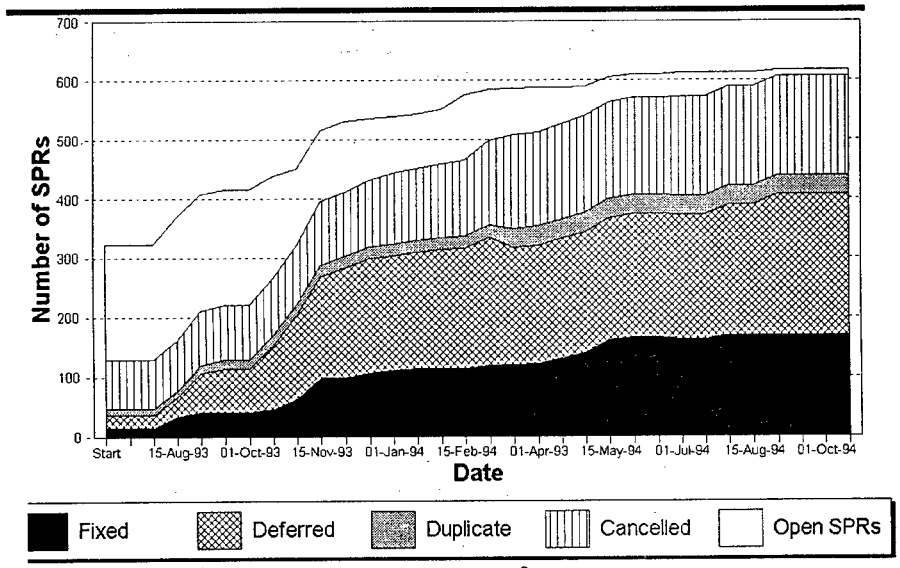


Figure O-25 Closed SPRs Highlighted[2]

So far, the metrics told us that we were finding problems and fixing them but did not tell us when and where those problems were found. Fixing a problem discovered early in the project is quicker, simpler, and infinitely more cost-effective than to repair a problem found near project completion. So, our paradigm shifted a bit, and we recalculated our metrics based on the new information we sought.

The results astonished us (see Figure O-26). Virtually all of the problems in SCU-2 were found in the test phase, late in production, which caused replanning, re-coding, retesting, and rework! Given the proper method of expression, our metrics stood there, stuck their tongues out at its, and wiggled their fingers under their noses. We had been doing extensive rework, but we had not realized why.

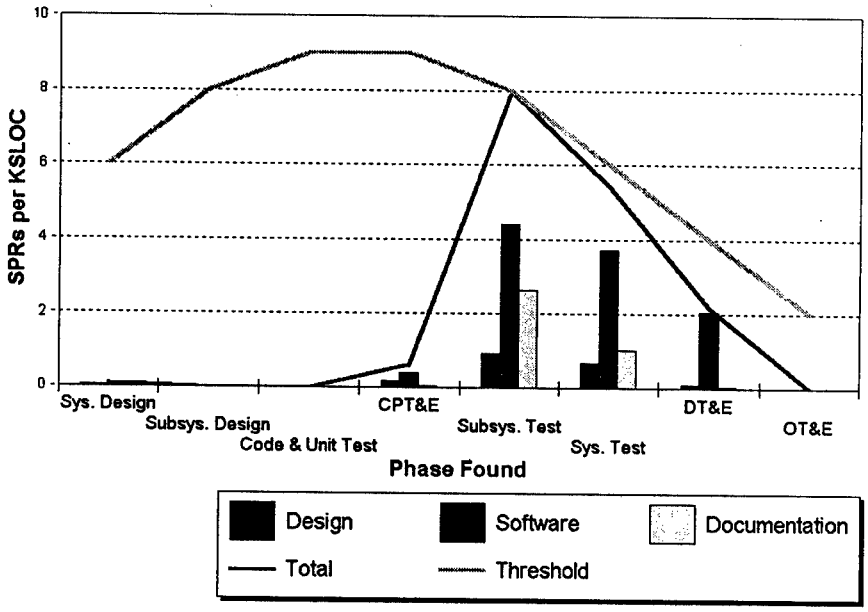


Figure O-26 Valid SPRs by Phase Found and Category [3]

The positive side to this chart was that it told us that most of the problems were being found in subsystem testing, which is a CSCI-level test and is the least expensive portion of the formal testing phase. System test was finding the next largest group of problems, and this was also a fairly inexpensive test. However, a significant number of problems were being found in Developmental Test and Evaluation (DT&E), which was a flight test and therefore, expensive. Since these tests took place after the operational flight program (OFP) was released to formal testing, replanning and rescheduling was required.

We were baffled. It was obvious we were doing something wrong. In the best of all worlds, the greatest number of problems should be found early in the project, and this number should decrease significantly with each phase until, at flight test, no problems are reported. But to discover where we needed to concentrate our efforts, we needed to find from where the problems were coming. Were our designs faulty or was it our coding? Was it a lack of proper unit testing before formal release? We didn't know for certain, so we looked at the information another way, concentrating on which phase of the project introduced the reported problems.

This time, we were "*hit*" with three more surprises (see Figure O-27). First, the majority of problems were not introduced by us (not visible on the graph shown). Second, our design problems were relatively few. Third, all of our own software problems (which accounted

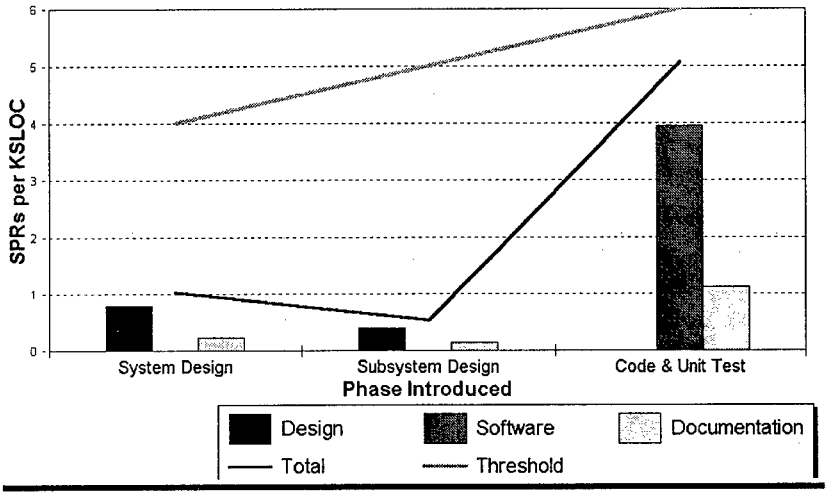


Figure O-27 Valid SPRs by Phase Introduced [3]

for nearly half of the total problems) were introduced in the implementation or coding phase of our project. The first bit of information made us feel a lot better about ourselves, and the second bit told us that our designs, for the most part, were good, but that last bit of business pointed an ugly finger right into the middle of our project. This was the area where design became implementation and where ideas turned into actions. We were not doing something right. The process was broken, and we had to discover how to fix it.

By looking at both the Detection and Introduction Summaries, it was obvious that we were not finding enough problems in our code and unit test phase, i.e., our Computer Program Test and Evaluation (CPT&E). For the most part, we are talking,; about simple problem — an incorrect constant or an improperly initialized variable — problems that, if found before formal release, would be quick and easy to fix. Obviously, CPT&E needed an overhaul.

While trying to figure out why we were not finding more problems during the design phase, we discovered that we *were* finding and correcting problems during the design phase, but this was not being recorded as rework. Suddenly, we realized that one reason we only saw rework during the test phase was that we had only allowed testers to write problem reports. Designers never wrote problem reports, so we never saw that they were finding and fixing problems; therefore, our process for reporting problems was changed.

A problem report was now required to be written before any design or code change could be accepted. Every change was now traced to a problem. We also created an online problem reporting system, which automatically logged the person who found the problem and the area in which the problem was found. This tool was able to produce a report of current problems and their status at any time. By the time SCU-3 began work in earnest, our metrics had almost begun to gather themselves.

It also was obvious that our CPT&E process had to be altered — too many simple problems were being found by formal test. So we extended the amount of time allotted to CPT&E and involved the formal test personnel. This allowed the coders to take the time they needed to thoroughly review their code. It also gave the test people the chance they needed to firm up their test procedures before formal testing began. Any problem found during this phase was properly logged so that we would be certain to find out how many problems were fixed as this level.

This approach had at least one major downfall. Since we were increasing the number of people reporting problems and widening the period during which problems were reported, we could significantly increase the number of problem reports found during an OFP update. We were

concerned that this would make us look bad to our customer. It might appear that instead of improving ourselves and our product, we were getting worse. To offset this concern, we prepared some "*customer education*" material and prepared to explain what we changed, why we changed it, and why there was a sudden large increase in the number of problems. As it turned out, we need not have worried.

When the metrics from SCU-3 started to come in, what we saw pleased us (see Figure O-28). The number of reported design problems (nearly trivial in SCU-2) were equal or greater than the number of software problems, and most design problems were being found and fixed in the earliest stages of development, when it was the cheapest. It was even more satisfying to learn that the greatest number of software problems were being found in the CPT&E stage, before the software went to formal test.

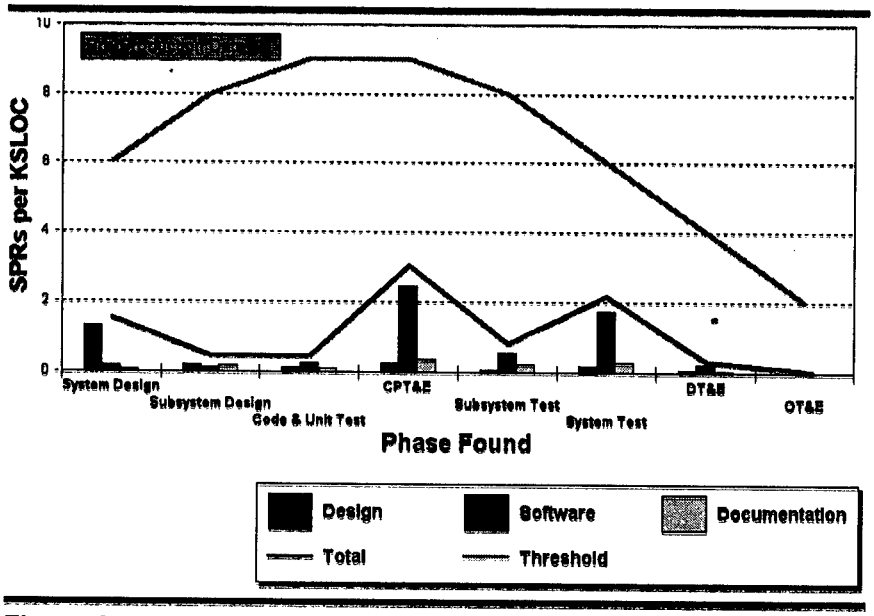


Figure O-28 Valid SPRs by Phase Found and Category [3, 4]

The number of problems reported did not increase significantly, even though we were collecting problem reports from many more sources. Was this because we were now requiring designers to report their problems and they were, therefore, more cautious? Was it because we had improved our CPT&E process? Or was it something else entirely? Our guess is it is all of the above.

By looking at our metrics, finding our problem areas, and improving our processes, we got a return on investment that we would never have imagined. SCU-3 has yet to be completed as this article is written; we might yet see metrics that show us other less desirable trends, but for now, things are looking good.

Not that we were satisfied; we were not. We hope to be able to find even more problems upfront, to involve the test group more in the design area, and the design group more in the test area. We were trying to make OFP development a true "*team*" effort, instead of the "*us versus them*" or "*design versus test*" undertaking it has been in the past. We have found that using metrics as we have changes the ways you perceive your metrics. It's recursive. Each new view of the data making you want to view it yet another way. For example, we now have

metrics not only on what problems were found and when, but also on when the problems were introduced and what the priorities of fixing those problems were. We are now looking at measuring the man-hours taken to fix each problem report so that we can better judge our rework effort.

CONCLUSION

So how are metrics made to work miracles? You watch, you record, you judge, you react, you alter the way you watch, and you keep doing that over and over. Sometimes, the miracles you find will be obvious; other times they will shock you, but from our own experience, we guarantee they will happen.

David A. Haakenson
00-ALC/TISFD
6137 Wardleigh Road
Hill AFB, UT 84056-5843
Voice: (801) 777-0326 DSN 777-0326
Fax: (801) 775-2541 DSN 775-2541
Internet: *haakensd@software.hill.af.mil*

David R. Webb
00-ALC/TIS-3/SEPG
7278 Fourth Street
Hill AFB. UT 84056-5205
Voice: (801) 775-5372 DSN 775-5372
Fax: (801) 777-8069 DSN 777-8069
Internet: *webbda@software.hill.af.mil*

Notes

1. **Metric goal**: to continuously reduce the number of open SPRS.
2. **Metric goal 1**: to continuously monitor and reduce the number of open SPRS, especially in analysis. **Metric goal 2**: to continuously reduce the number of canceled and duplicate SPRs through effective communication within the assigned integrated product development teams.
3. **Metric goal**: to find zero defects in configured work products. Obtaining credit for finding defects is not the metric goal. Includes valid defects found on configured work products: analysis, canceled, and duplicate SPRs are not included in this metric. Peer review results not included in this metric. Baseline defects are not included in this metric.
4. All members of the integrated product development team are responsible for the successful implementation of the software candidate assigned to them.

CHAPTER 8
Addendum E

Swords and Plowshares: The Rework Cycles of Defense & Commercial Software

Kenneth G. Cooper
Thomas W. Mullen

INTRODUCTION

A worldwide survey recently revealed that less than half of all development projects meet their targets for development time and cost.[1] Technological development projects dominated by software-based systems constitute increasingly significant portions of companies' new product and business plans. Traditionally defense-oriented firms in the post-Cold War period search, with varying degrees of aggressiveness and desperation, for how to make an effective transition toward commercial markets. Companies already firmly established in commercial software markets search, with varying degrees of frustration, for ways to bring new products to market faster and at lower cost. Indeed, the success of virtually all companies has never before been more dependent upon timely, low-cost development project execution (nor more threatened by its absence).

With the magnitude of the stakes involved, why does there seem to be so little progress in achieving better-managed software development projects? Why in such projects are cost and schedule problems so persistent and pervasive? What are the underlying sources of consistently "*surprising*" project overruns? Why are we so *bad* at estimating when developing products will be completed and ready for the market? How far must "*defense*" firms be prepared to go to become commercially competitive? What must both they and commercial firms alike do in order to achieve dramatic project performance improvement? Are there any fundamental lessons we can learn and transfer from one "*unique*" project to another?

Here we aim to provide some initial answers to these questions. To do so we draw upon over ten years of experience, shared with our colleagues, in developing and applying computer-based dynamic simulation models[2] of software system development projects. We have used such models to accurately recreate, forecast, diagnose, and improve the performance on dozens of major development programs and projects in aerospace, defense electronics, financial systems, construction, and telecommunications. Among these, many whole projects and significant segments thereof have been dedicated to software system development.

At the core of the structure of these models is a different but straightforward view of development project work — one which recognizes the *rework cycle*.[3] Indeed, what is most lacking in conventional methods for project planning and monitoring is any acknowledgment or

measurement of **rework**. For all their utility, most planning tools treat a development project as being composed of individual, discrete tasks which are "*to be done*," "*in process*," or "*done*." No account is taken of incomplete or imperfect task products, or the amount of rework needed. This is particularly inappropriate for naturally iterative development efforts; the dozens of analyses we've conducted show that *rework can account for the majority of project work content and cost!*

THE REWORK CYCLE

Using dynamic simulation models to analyze and aid the management of software development compelled us to **simulate** the performance of actual projects as they really did occur — not just how they are planned to go, or how they "*should*" go. Hence, we had to design and employ a structure which could accurately recreate such projects. To do so we **had** to treat their substantial rework explicitly, as well as its causes, detection, and execution. We developed a core structure which proved to be universally applicable to development projects and project stages. We term this structure *"the rework cycle."* See Figure O-29.

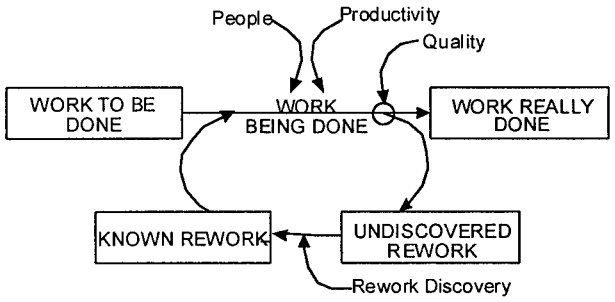


Figure O-29 The Rework Cycle

At the start of a project or project stage, all work resides in the pool of **Work To Be Done**. As the project begins and progresses, changing levels of **People** working at varying **Productivity** determine the pace of **Work Being Done**. But unlike all other program/project analysis tools and systems, the Rework Cycle portrays the real-world phenomenon that work is executed at varying, but usually less than perfect, ***"Quality."*** A fraction that potentially ranges from 0 to 1.0, the value of Quality (as well as that of Productivity) depends on many variable conditions in the project and company. The fractional value of Quality determines the portion of the work being done that will enter the pool of **Work Really Done**, which will never again need redoing. The *rest* will subsequently need some rework, but for a (sometimes substantial) period of time the rework remains in a pool of what we term **Undiscovered Rework** — work that contains as-yet-undetected errors, and is therefore *perceived* as being done. Errors are detected by "*downstream*" efforts or testing; this **Rework Discovery** may occur months or even years later, during which time dependent work has incorporated these errors, or technical derivations thereof. Once discovered, the **Known Rework** demands the application of People, beyond those needed for completing the original work. Executed rework enters the flow of Work Being Done, subject to similar Productivity and Quality variations. Even some of the reworked items may then flow through the rework cycle one or more subsequent times.

Undiscovered rework plays a pivotal role in the propagation of problems through a project. Lurking undetected — for example, as a software "*bug*," or design flaw — it causes productivity loss and work delays, and triggers rework cycles on downstream dependent tasks.

The more tightly-scheduled and parallel the project tasks, the more of a "*multiplier effect*" on subsequent rework cycles. Undiscovered rework is *the single most important source of project cost and schedule crises*. To control undiscovered rework on software development projects, we must:

- **Acknowledge** its existence,
- **Plan** so as to allow for it,
- **Measure** it (once it is recognized as known rework),
- **Prevent** it (i.e., improve "*quality*," as defined here) as much as possible, and
- **Seek** to find and identify it early, so as to reduce its propagation.

We offer the following observations in the hope of spurring companies and individual managers to do just that.

OBSERVATIONS

The full simulation models of these development projects employ thousands of equations. They explicitly portray the time-varying conditions which cause changes in productivity, quality, staffing levels, rework detection, and work execution, as well as the interdependencies among multiple project stages. All of the dynamic conditions at work in these projects and their models (e.g., staff experience levels, work sequence, supervisory adequacy, "*spec*" stability, worker morale, task feasibility, vendor timeliness, overtime, schedule pressure, hiring and attrition, progress monitoring, organization and process changes, prototyping, testing) cause changes — some more directly than others — in the performance of the rework cycle. Because our business clients require demonstrable accuracy in the models upon which they will base important decisions, we have needed to develop highly accurate measures of all these factors, especially those of the rework cycle itself. We do not, however, offer here a treatise on model design. Instead, we draw upon the many real business applications of our models to provide heretofore unavailable guidelines and benchmarks on the characteristics of the rework cycle.

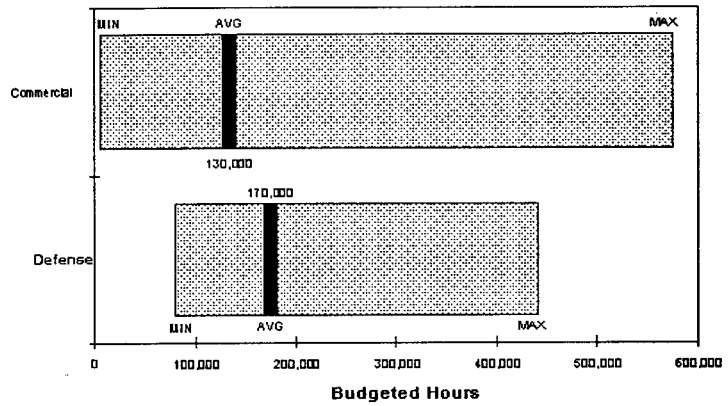
The Sample

The observations come from seven defense and fourteen commercial software development efforts. They range from modest upgrades of existing systems that involve about ten people for a year, up to major first-of-a-kind system development efforts involving many hundreds of people for several years. The defense projects as a group differ from the commercial projects in many ways (average size, duration, customer for the product, etc.), so we've examined the averages and the ranges of values within these groups as well as across all the projects. Figure O-30 shows the difference in planned size and duration, with the commercial projects averaging under 130,000 hours of planned work over the course of about a year's schedule, versus 170,000 hours and a two-year-plus average planned duration for the defense projects.

The Problems

The defense projects' performance against plans was substantially worse than those of commercial developments. See Figure O-31 (below). On *average*, the defense projects took about three times the planned hours, versus about 1.4 times the planned hours for the commercial projects. Schedule performance was even worse. The commercial projects were poor, taking nearly twice as long to complete as planned. On average, though, defense projects took over **four times** as long to complete as was originally planned.

APPENDIX O Additional Volume 1 Addenda



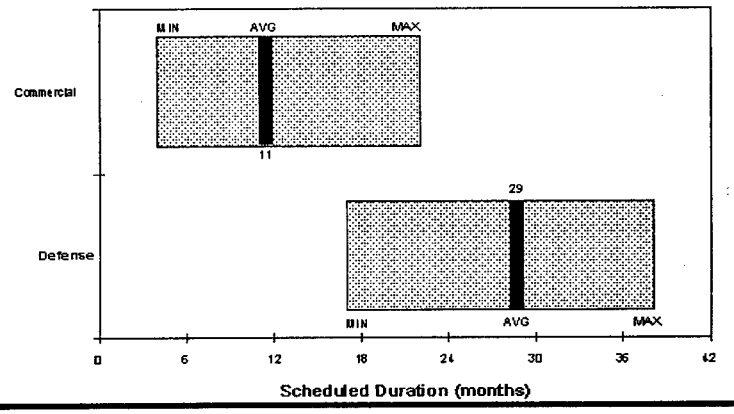
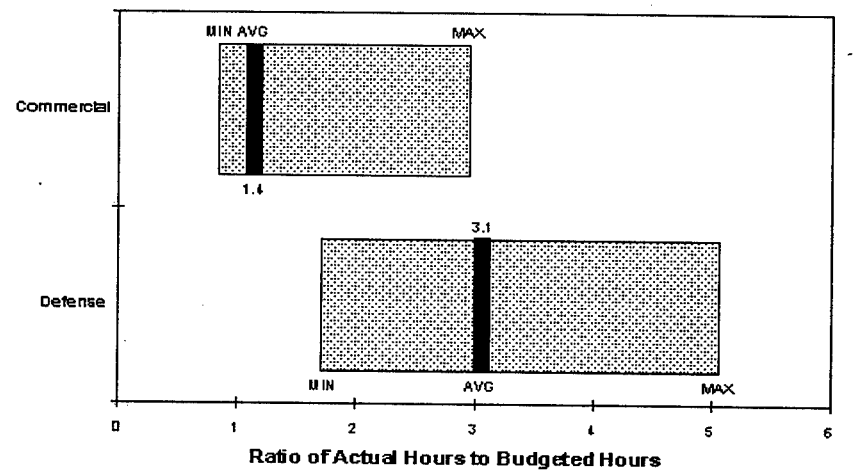


Figure O-30 Project Targets

How could this be? These projects involved some of the brightest, most experienced, and hardest-working people in the field. They were managed by seasoned and conscientious managers using the standard planning tools, and yet they **still** cost many times their budgets. We acknowledge the bias caused by the fact that easy, smoothly-running projects rarely command the attention of external consultants. Still, these projects are not anomalies; recall that *most* projects fail to meet their targets. We believe there are **systemic** reasons why software development projects perform so poorly.

To be fair, *some* of the problems — especially on the defense side — are due to midstream scope and specification changes by the "*customer*," be they internal or external to the company. These are not fully anticipated at the start of the project, and not reflected in the original budgets and schedules (although, given the history of large projects, such changes are to be expected). The difference in the typical magnitude of midstream changes is part of the reason for the differences between defense and commercial project performance. However, even adjusted for those changes, the same patterns remain. We believe that our use of simulation modeling has helped identify the systemic causes of those patterns, through the characteristics of the Rework Cycle — rework creation, identification, and execution.



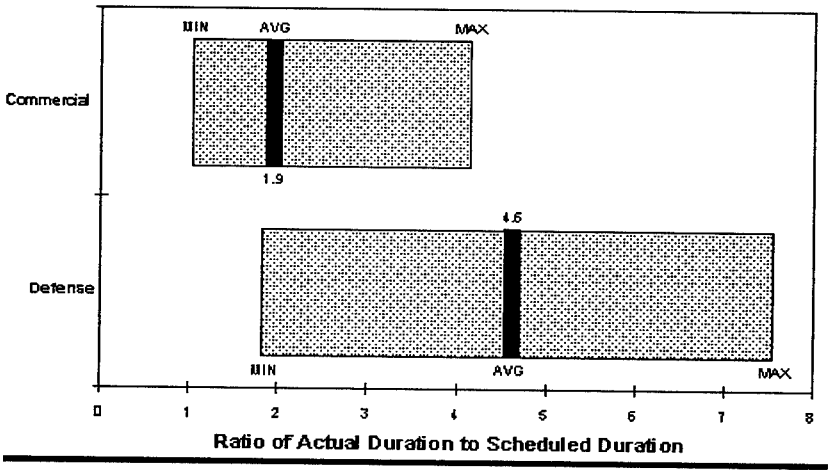


Figure O-31 Performance versus Targets

Rework Creation

In the rework cycle, "*quality*" is the fraction of work being done at any point in time which will **not** eventually need to be reworked; the lower the quality, the more rework. The average levels of quality on defense projects was half that observed on commercial projects, 0.34 versus 0.68. See Figure O-32. In other words, only about 1/3 of the work products being executed on a defense software development project will not need reworking (or *another* round of reworking), as opposed to 2/3 in commercial projects.

The rework cycle quality on the worst of commercial projects was nearly as low as that of the average defense software project, but the best was nearly "*perfect.*" The best among defense efforts exhibited a 0.55 quality, but the worst was near 0.10.

APPENDIX O Additional Volume 1 Addenda

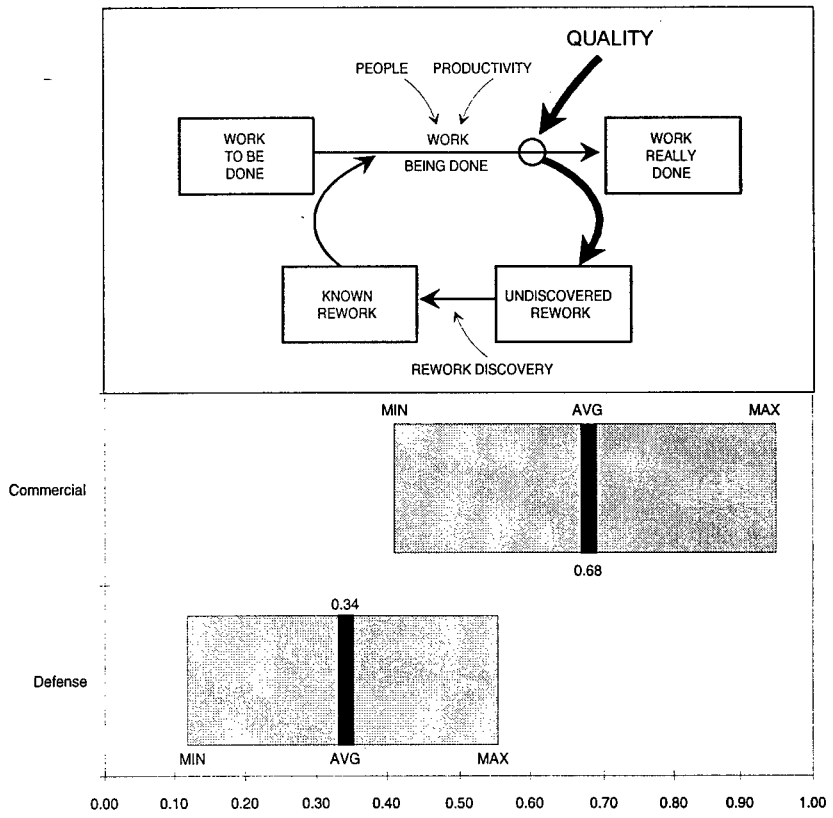


Figure O-32 Average "*Quality*" Rework Creation

Rework Discovery

We know of no company which routinely monitors or measures the amount of time elapsed between the commission of design or coding errors and their detection. And yet it is undoubtedly one of the most critical determinants of "*time to market*" **and** of the true quality (in the conventional sense) of the delivered product. Think for a moment. On projects of this nature, what would you say is the typical amount of time between making and finding an error? A week? Two? A month? The actual average across all these projects is about *nine months*. See Figure O-33 (below).

On this measure we find little absolute difference between defense and commercial projects, on average. In fact, the worst of commercial software projects exhibited **longer** rework discovery times than the worst defense projects. Perhaps this is due to the extraordinary amount of oversight, testing, and review endemic to defense projects. Regardless of the cause, it is troublesome to note that for commercial efforts the average scheduled duration of work, eleven months (versus 29 for defense), is *only three months longer* than the time to detect the need for a single round of rework.

Only the **relatively** high levels of quality in commercial software developments prevent consistent disasters. Even so, rework discovery times which are nearly the length of the planned work explain why firms so consistently overpromise on software product introduction dates (open any business or software journal for an example). Rework keeps being found near

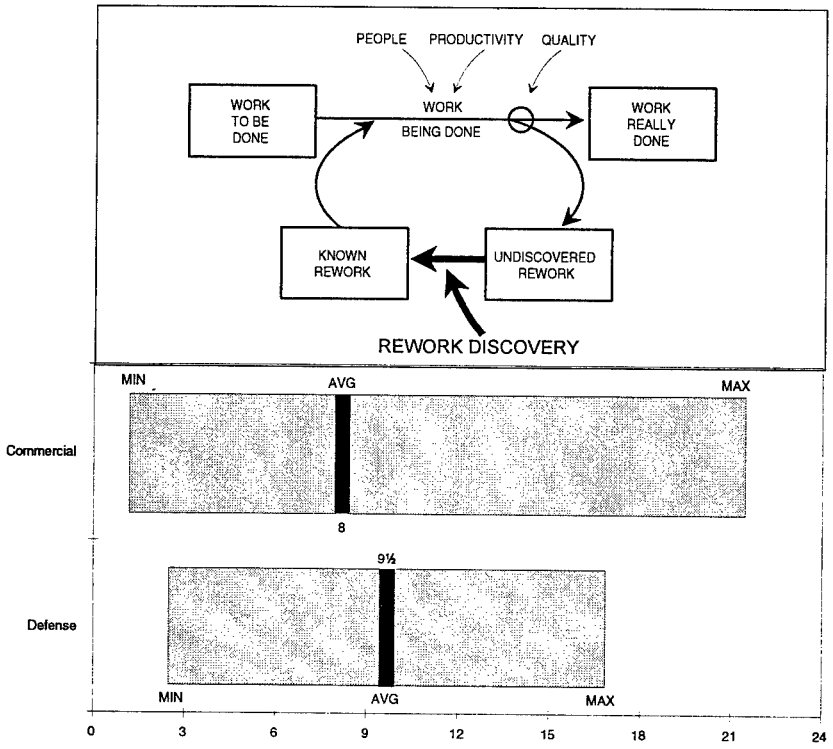


Figure O-33 Rework Discovery Time (months)

"*the last minute*," and revised announcements sheepishly admit to the delay. The more aggressively the projects are scheduled in response to competitive pressures to "*be first*," or to respond fast, the worse this tendency. No "*productivity*" gain will help — only improvements (reductions) in the rework discovery time will.

And when you consider that a project with a 0.34 average quality (the defense average) will require **seven** cycles of rework to surpass 95% real completion,[4] a rework discovery time of 9-10 months for each cycle would add *over five years* to an effort for which rework was not planned. Interestingly (and not coincidentally), that is near the amount by which real completion of defense software development projects in our sample exceeded their schedule targets.

Rework Execution

As the rework cycle structure indicates, some work may cycle through multiple times before it is complete and correct. Figure O-34 shows the average number of full revisions[5] of each task (e.g., specifications, modules). The average task product on defense projects was revised fully three times. On commercial projects, only 40% of the tasks were revised, on average. **This** is the major reason for the cost and schedule performance differences. Some commercial software projects saw over one full round of revisions, and defense projects as many as **seven**.

So where was all the time spent? On defense projects, *more time was spent on rework than was spent to do tasks for the first time*. See Figure O-35. An average of nearly one and a half hours were spent **fixing**, for every one hour spent to do it the first time. Even on commercial projects, over forty minutes were spent on rework for every hour spent on the first iteration. This

APPENDIX O Additional Volume 1 Addenda

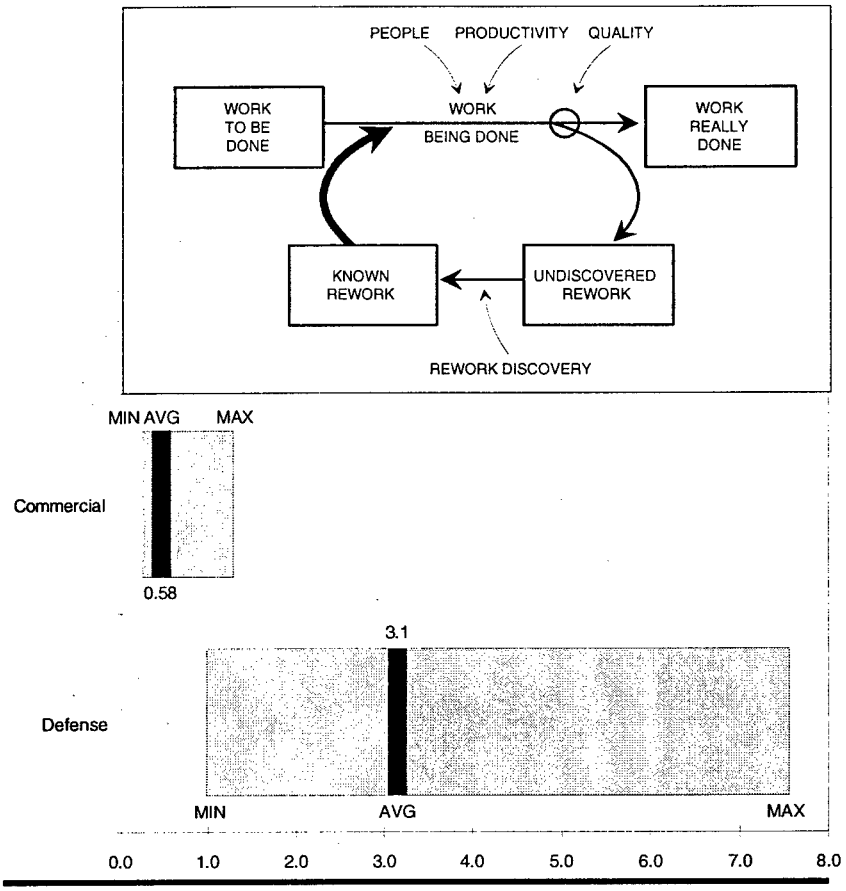


Figure O-34 Revisions per Initial Product

is even more astonishing when put in perspective: about half of the time spent on all these software development projects was for work not even tracked in most planning and monitoring systems!

No wonder managers of software projects have difficulty — they're using tools that only let them see half of the job. Even worse, in the later stages of a project, 70, 80, even 90 % and more of the work is "*invisible*" to their planning systems. Managers and staff on these projects aren't incompetent, they aren't (usually) deceitful, and they aren't lazy — *they are misled by their planning tools*.

Experienced managers are certainly aware of rework, and make allowances for it. However, this means that all of their "*sophisticated*" planning tools become nearly worthless at the end of the project, when the bulk of the work is rework. Instead that effort is being forecasted purely on the basis of instinct. For most project managers, a handful of projects constitutes a career. By the time some managers work up to a really difficult project, they may have had little experience with "*major rework*" on which to base their planning (and promises). Under tremendous pressure to deliver (from senior management, the customer or the marketplace, and dependent efforts), managers' well-intended plans and promises will be thwarted by the rework cycle.

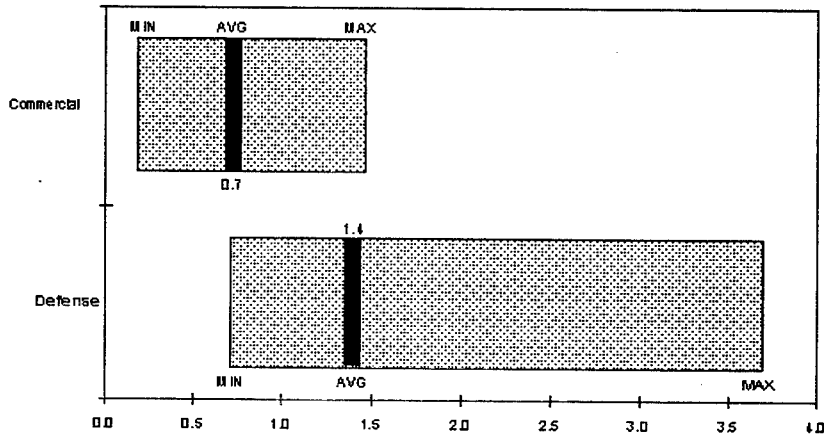


Figure O-35

Progress Monitoring

Just how thwarted those plans can be is illustrated in Figure O-36. This "*progress ramp*" displays the accuracy of progress monitoring in the sampled commercial software developments. For the range of commercial efforts modeled, the display charts the *perceived* progress, as it was reported at different points in time, against the *real* progress (which excludes undiscovered rework) at that time. Perfectly accurate project progress monitoring would yield a straight 45° diagonal (hence the triangular ramp shape): at a perceived/reported condition of 20% complete, the *actual* % complete would be 20%, and so on. Instead, real progress is typically less than reported progress.[6]

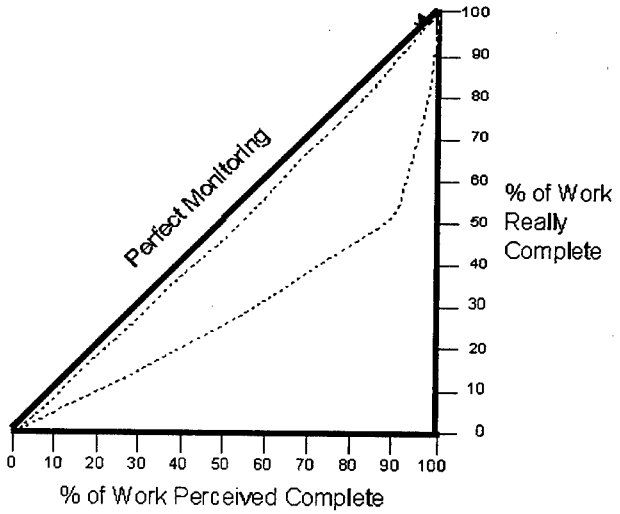


Figure O-36 Progress Ramps of Commercial Software Projects

Note that the "*best*" of the sample achieved nearly perfect monitoring. The lower the quality, and the longer the rework discovery time (hence the more undiscovered rework), the larger the gap between real and reported progress, and the longer that gap persists: the "*worst*" of the commercial efforts reported *90%* completion when as little as *60%* was really complete. With 40% of the effort really left, it is easy to see why that last "*10%*" can seem to take so (unexpectedly) long, with the attendant delays in product introduction dates. Indeed, the product is often delivered when perceived "*complete*," leaving it to customers to find the as-yet-undiscovered rework. Before moving on to a comparison with defense software projects, imagine the difference in ease of management and accuracy of projections just between these two commercial extremes. When considering best-practice/TQM/process re-engineering, consider the differences displayed here — all caused by variations in "*quality*" and rework discovery time.

Figure O-37 overlays the same kind of envelope for progress monitoring in the defense software efforts modeled. Generally longer rework discovery times, and notably lower levels of "*quality*," produce a much more bowed shape overall, reflecting less accurate progress monitoring. The best of the defense projects is near the typical commercial project in progress monitoring accuracy. The worst among the defense efforts, with quality levels near *0.10*, reports 75% completion *when less than 15% is really done*. After reaching 90% reported completion, the line goes nearly vertical — meaning a **long** time was spent thinking the end was near, only to discover more and more rework that extended the project and increased its cost far beyond the original **and** interim plans. By these measures again, the distance that defense firms must move in order to be viable commercial competitors is a long one. And, again, "*productivity*" improvement is not the answer. Instead, the answer lies in measures that are rarely monitored, let alone being the focus of control and improvement efforts — quality and rework discovery times.

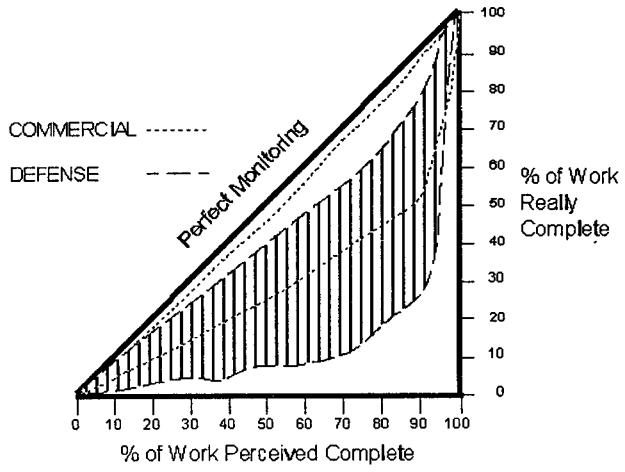


Figure O-37 Progress Ramps

IMPLICATIONS FOR IMPROVEMENT

The nature of the performance improvement that can be achieved if efforts are successfully focused on rework cycle quality is illustrated in Figure O-38. The individual data points from all the software development efforts chart (a) their average quality achieved versus (b) their costs incurred, as a ratio to their budgets. As an example, we've circled one data point for a project that achieved an average of 70% quality, and saw a cost/budget ratio of about 1.5 (a 50% overrun).

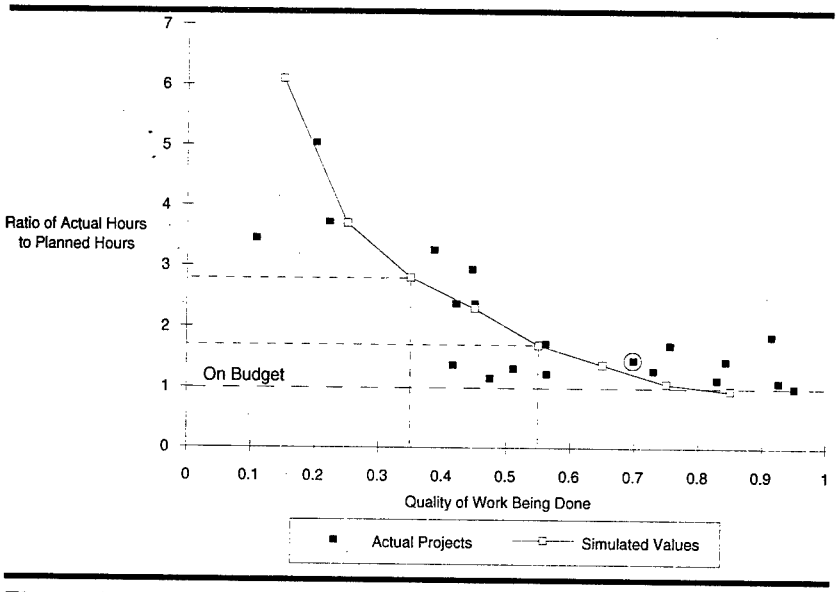


Figure O-38 Better Quality Yields Lower Cost

Of course, the accuracy of the original budgets causes some deviation among the plotted points. Nevertheless, the pattern is clear: higher quality, lower cost. Most projects with quality levels in excess of 0.70 achieve costs comparatively near their budgets. In contrast, projects with a quality less than 0.40 exceeded their budgets by factors of 3, 4, ... 5!

We overlaid on this chart a line that plots from several *simulations* the resulting costs versus budget for **one identical project** which varies among the simulations **only** in the average quality achieved. We do so to drive home the extent to which quality improvement translates to cost performance improvement. With no productivity change whatsoever, a change in rework cycle quality from 0.35 to 0.55, for example, would eliminate most of the project cost overrun, reducing costs by an amount equal to the original budget.

CONCLUSIONS

We need to improve our fundamental understanding of how development projects really work. In order to avoid the persistent cost and schedule performance problems so closely associated with software development efforts, we must take a more strategic and realistic view of project work content and processes. While the products and technical steps may indeed be unique, we need to recognize that there **are** common structures and processes, and common problem causes. Only then is it possible to extract lessons and to implement changes that achieve radical improvements in project performance and business success.

APPENDIX O Additional Volume 1 Addenda

Our experience in simulating software development projects indicates that conventional methods and systems are inadequate to support the management of such projects. Further, the improvement in project or product development performance sought by most companies is frustrated by the prevailing managerial mindset. It is a mindset encouraged by the use of systems which treat projects as the sum or sequence of purely discrete tasks.

We need to recognize the flows of work in software development projects, flows in which there are multiple *rework cycles*. Managerial systems which ignore rework and its cycles are deficient, misleading, and constitute a roadblock to achieving breakthrough improvements in project and product development performance.

Indeed, we see from the reported results a clear indication of just how dramatic a breakthrough is required in order for traditionally defense-oriented firms to transfer their substantial expertise to compete effectively in commercial markets. Their cost performance against targets must improve by at least *a factor of 2*. Schedule performance versus targets must improve by *a factor of 3*, lest faster and more nimble commercial competitors thwart the attempted transition to "*plowshares*." Rework content in their projects must be cut to *1/3 to 1/2* of currently prevailing levels. The "*quality*" of on-going work must *double*. Rework *detection* must be encouraged, to avoid the snowballing effects of undiscovered rework. Both better quality and rework detection are required to improve radically managers' ability to: (a) contain costs; (b) assess accurately the true state of ongoing projects' work progress; and (c) foretell dependably the time at which developing products will be completed and delivered to the market.

And all this presumes that commercial development efforts themselves do not improve — that they represent a standing target. This would be imprudently optimistic, for there is substantial room for improvement in most complex commercial software developments as well. Managerial, technical, and procedural improvements which could increase work quality by just 10 points would cut cost overruns in half. Most significant for commercial projects, reductions in the rework discovery time would yield a faster "*time to market*," and more dependable estimates thereof. But we cannot control by mandate the "*levers*" of quality and rework detection in the rework cycle. Instead we need to influence them through that which we **can** control, or more directly influence — interim schedule targets, staffing, monitoring systems, coding techniques, testing practices.

The role of a simulation model on a specific project is to portray accurately the project and to aid its managers in evaluating potential actions through "*What if*" analyses. However, employed on several development projects, the model also helps to illuminate the underlying structure of such projects. In the "*rework cycle*" we have developed a near-universal structure which facilitates both roles. The resulting improved understanding helps managers to identify transferable lessons. In turn, these **do** lead to significant performance improvements. Indeed, without some more complete and realistic "*model*" than is now the norm, the seemingly intractable cost and schedule problems of software development projects will continue to plague defense and commercial firms alike.

Notes

1. "Strategic Management of Technology: Global Benchmarking," Dr. Edward B. Roberts, December 10, 1992, Cambridge, MA.
2. These project models were built using the dynamic continuous simulation language DYNAMO; see DYNAMO User's Manual, Pugh-Roberts Associates, and Introduction to System Dynamics Modeling with DYNAMO, Richardson, George P. and Pugh III, Alexander L.
3. The concepts and workings of the rework cycle model were explained in "The Rework Cycle: Benchmarks for the Project Manager", Cooper, K. G., from which some introductory description here is excerpted, and first published in Project Management Journal, March 1993.
4. Do the math: Really complete after 1st release= .34; after 1st rework cycle= prior complete + (quality • remainder)= .34 + (.34 x .66)=.56; after 2nd cycle= .56 + (.34 x .44)=.71; 3rd= .71 + (.34 x .29)=.81... after 7th rework cycle= .96

5. Revisions reported here are normalized to be equal in effort to a full re-execution of the first release of the work product, so as to correct for significant variations in the effort content of different revisions.
6. With the benefit of data on a completed project or project stage, one may construct one's own *"progress ramp"* chart by plotting for the completed project: (1) the historically reported "% complete," versus (2) a retrospective computation of the % really complete then (you should compute the % really complete based on hours spent to that point, relative to the total hours eventually spent).

Acknowledgments

Many years of work by our colleagues and clients have gone into making possible these observations and findings. Without naming the dozens of individuals and companies with whom we have worked, we wish to acknowledge here their invaluable contribution. We wish to thank explicitly our colleagues, Dr. Thomas G. Kelly and Alexander L. Pugh, for their substantial and timely help in preparing the material on which this article is based. The interpretation of the assembled information remains the authors' responsibility.

About the Authors

Kenneth G. Cooper is Director of the Management Simulation Group and Senior Vice President of Pugh-Roberts Associates, a division of PA Consulting Group. Mr. Cooper's management consulting career spans twenty years, specializing in the development and application of computer simulation models to a variety of strategic business issues. His clients include AT&T, Aetna, Arizona Public Service, Hughes Aircraft, IBM, Litton, MasterCard, McDonnell-Douglas, Northrop, Rockwell, and several law firms. Mr. Cooper has directed over a hundred consulting engagements, among them analyses of sixty major commercial and defense development projects. Mr. Cooper is an original author of the program management model introduced in this article. His group's offices are in Cambridge, Massachusetts and Oxford, England. Mr. Cooper received his bachelor's and master's degrees from M.I.T. and Boston University, respectively.

Thomas W. Mullen is a Senior Manager in the Management Simulation Group of Pugh-Roberts Associates, a division of PA Consulting Group. Mr. Mullen has managed and participated in over thirty consulting projects in his eight years with the firm. He has concentrated on the use of simulation models to analyze and aid major commercial and defense development programs. Mr. Mullen has worked with many clients in the aerospace, software, and financial services industries. He has also managed the development of several simulation software products. Mr. Mullen received both his bachelor's and master's degrees from M.I.T.'s Sloan School of Management.

CHAPTER 10
Addendum B

Rate Monotonic Analysis: Did You Fake It?

Reprinted from the Software Program Manager's Network newsletter, NetFocus: Technology Update, Number 210, June 1994

Rate Monotonic Analysis

> *In the past, developers have had few tools to help them ensure on-time performance of their real-time systems...A real-time system generally has several activities (or tasks), each of which must be completed by a specified deadline. Some of these deadlines may be hard (or critical) and some may be soft (such as those based on average performance). Missing a hard deadline can result in catastrophic loss of system performance or even loss of life.* [OBENZA94]

RMA: Did You Fake It?

Did you fake it? This is the question concerned program and software managers should ask their development teams when it comes to the design and implementation of their real-time systems. Not only has "*faking it*" been an option when designing real-time systems, it has become the process. Unfortunately, there is immense corporate ignorance for the need of a proven analytical process to address real-time requirements at the design stage of a real-time system. With the exponential growth of the size of real-time systems, the decrease of development dollars, and the liability issues that are emerging due to defective software, faking it is no longer a viable option.

Schedulable, "*on time*" software is essential for real-time systems. Without such a capability, task timing conflicts cannot be identified early enough in development to avoid timing overruns, which cause system crashes and the subsequent need for budget-busting corrective measures. Simulation alone isn't the answer, for it doesn't guarantee timing, besides being extremely costly in time and money.

An alternative to faking it is **Rate Monotonic Analysis (RMA)**. RMA is a scientific and mathematically sound way of guaranteeing the timing requirements of time-sensitive systems and is one of the most well known and often used scheduling algorithms for realtime applications. Boeing, General Electric, General Dynamics, Honeywell, IBM, McDonnell Douglas, Magnavox, Mitre, NASA, Naval Air Warfare Center, and Paramax are just some of the growing number of organizations beginning to use RMA on actual systems. RMA will increase your project's productivity and reduce integration and testing costs, risk and complexity.

RMA's system scheduling heuristic is shortest-task first, so the ready task with the shortest period always runs. The analysis considers worst-case time and combinations of system load, phasing, and resource consumption, ensuring real-time performance and stability under heavily loaded conditions. RMA provides insight into the hardware and software design that

effects system timing performance and helps to identify possible bottlenecks and errors that degrade schedulability. RMA has the ability to predict timeliness — processing of an event during worst-case time — and guarantees that events meet their deadlines.

When you begin to integrate RMA into your process, you will become aware that the scheduling of tasks for a real-time system is significantly different from the traditional forms of scheduling. It will become apparent that tasks compete for resources; whether it's a CPU, backplane bus, or network. This requires that more important tasks be given priority to execute over all other tasks. A preemption occurs when a higher priority task replaces a currently running task on a resource. Preemption by way of priorities, to increase the responsiveness of a system, brings up several questions:

- How do you ensure beyond a reasonable doubt that the lower-priority tasks will ever get access to the resource?
- If there are simultaneous activities, how do you guarantee that all of them are completed on time?
- If they all cannot finish in a timely manner, which one succeed and which ones fail?
- Is it possible for all the tasks to run in a way that guarantees timely access to the resource?
- And most importantly, will the tasks meet their execution deadlines?

All of these questions can be answered through the use of RMA.

Adoption of RMA has been met with reluctance in the past. Questions arise about the costs of training, materials, and access to the appropriate tools that support RMA. Today, however, affordable CASE tools that support RMA are available, and provide a proven process as part of good engineering discipline in real-time systems. Unfortunately, management cannot promote good discipline if it doesn't thoroughly understand real-time system development issues. System engineering practitioners need the ability to analyze the run-time performance of a real-time system at all phases of the software life cycle:

- During the proposal phase there is a demand for support in system prototyping, iterative development techniques, and trade studies.
- Guaranteed timing deadlines and adequate system hardware are a necessity during system requirements definition — before writing a single line-of-code.
- At the time of implementation, the intent is to painlessly detect scheduling errors, calculate overall CPU usage, and diagnose corrective actions.
- The objective during test is to reduce testing, verification, and demonstration validation with the confidence that the algorithmic proof used at the design phase guarantees the system performance.

Achieving the objectives of real-time systems is possible if a system is designed with the support of a RMA CASE tool. CASE tools provide a practical, highly cost-effective, and easy way to automate use of RMA. Emerging RMA CASE tools take the guesswork out of identifying significant performance issues and can help develop an overall solution strategy as opposed to spending dollars on new hardware or using a less effective approach to modify a real-time system.

Adopting RMA methods as part of an organization's general engineering discipline and standard software development process will provide real cost savings and quality process improvements.

APPENDIX O Additional Volume 1 Addenda

The "*Did You Fake It?*" Quiz

1. When it comes to real-time software, do you practice the method of first making the software work, and then trying to make it meet timing requirements?
2. Do you have unrepeatable system failures?
3. Have you ever bought a larger CPU to solve timing overruns?
4. Has rewriting your software in a different language been proposed as the answer to a timing error?
5. Do you wish you had a process on your current real-time program?
6. Do you think real-time means real fast?
7. Do you wish you had an accurate account of your real-time system's processing limitations?
8. Are you tempted to push more processing down to an interrupt level to get your system to run faster?

An affirmative answer to one or more of these questions indicates that your real-time systems are at high risk. Just say no to faking it by saying yes to the benefits of effectively using RMA.

RMA IN PRACTICE

A contractor had been directed by their customer to learn more about RMA and perform real-time analysis on the system they were building. They indicated that they did not need to perform any analysis because their simulation proved the system could meet timing requirements. By gathering information to understand the simulation, an RMA expert was able to extract the system design, analyze it, and identify a queue that was approaching overflow and would eventually bring the system down. The RMA expert asked the contractor to run the simulation an additional 10 minutes, which did cause the system to fail.

A Navy contractor had been building a submarine sonar trainer. At integration testing, the system was experiencing severe timing overruns, which were causing the system to crash. RMA showed only 300 lines-of-code had to be modified to fix the timing problems. This was considerably cheaper than recoding 17,000 lines of Ada to C, which was the original plan. The real-time analysis also showed that recoding to C would not have solved the problem.

> *Through the use of rate monotonic scheduling, we now have a system that will allow [Space Station] Freedom's computer's to budget their time, to choose [among] a variety of tasks, and decide not only which one to do first, but how much time to spend in the process.*
>
> — Aaron Cohen, then Acting Deputy Administrator of NASA in 1992

RMA is derived from a paper presented by C.L. Liu and James Layland in 1973. In it they state that a set of n independent tasks will always meet its deadlines, for all task phasings, if:

$$\frac{C_1}{T_1}+\ldots+\frac{C_n}{T_n} \le U(n) = n(2^{1/n}-1)$$

C_1 = worst-case task execution time of task

T_1 = period of task

$U(n)$ = utilization bound for n tasks

Appendix O Additional Volume 1 Addenda

NOTE: For more information on RMA, see SEI's A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.

REFERENCES

[OBENZA94] Obenza, Ray, "Guaranteeing Real-Time Performance Using RMA," *Embedded Systems Programming*, May 1994

CHAPTER 11
Addendum B

Electronic Combat Model Re-engineering

Idaho National Engineering Laboratory
March 1995

EXECUTIVE SUMMARY

As modern software systems become increasingly complex and critical, executives, managers, and technical team leaders are faced with ever more difficult choices. Regardless of your position in a government organization or a commercial business operation, the insights contained in this monograph can be of benefit, and can help you achieve and keep a competitive advantage. Decisions regarding the disposition of existing legacy software systems can have an enormous effect on the operations—even survival—of government and commercial organizations. Many organizations are grappling with trade-offs of retiring older software systems and moving into more modern and efficient architectures, while trying to find ways to leverage some remaining capabilities of legacy systems to reduce cost and risk of software modernization.

The United States Air Force (USAF) is facing these difficult decisions. In a cooperative effort with the United States Department of Energy's (DOE) Idaho National Engineering Laboratory (INEL), the Air Force successfully addressed many of the hard issues facing commercial executives and managers today. The lessons learned by USAF/INEL provide valuable insights and models for action for both government and commercial decision makers. Although the project involved the re-engineering of a military software system, the applicability of the experience is appropriate across many commercial domains. The critical necessity for migrating from older legacy software systems to modern software architectures crosses boundaries of virtually every application domain in the internationally competitive software marketplace.

Re-engineering Legacy Systems

The Air Force was faced with maintaining a legacy system which provided important capabilities to its users. While the users were satisfied with the system, the Air Force was finding it increasingly expensive and difficult to maintain software written for a proprietary, "*non-open*" hardware platform. As maintainability of the code decreased, support costs increased, and reliability of the system deteriorated.

The Air Force initially directed the INEL to re-host the application, and move it to an open systems architecture. Basically, this effort involved translating the existing code from Fortran into C. Although the resulting code functioned properly, and the application was moved to an open systems architecture, the code itself was even less maintainable than the original Fortran software. As more functionality was added to the original application, maintainability of the software continued to decrease. INEL was then asked to conduct a study of the application, to include an analysis, evaluation, and recommendations to the Air Force as to future directions

for the program. The recommendations for the future were to ensure the continuing satisfaction of the Air Force user community. The study showed the system needed to be re-engineered to provide for future user needs, and INEL recommended a blend of technologies and methods, including a layered, object-oriented software architecture, implemented in the Ada programming language. The resulting re-engineered system produced a threefold improvement in maintainability of software. Additionally, the re-engineered system has enabled new functionality to be added in a fraction of the time required with the original and re-hosted versions.

Maintainability Index and Metrics

The decision to re-engineer a legacy software system is difficult to justify without appropriate metrics. As part of the USAF/ INEL justification, specific metrics were collected and analyzed to provide indicators of system maintainability and complexity. These metrics, in combination with independently developed polynomial equations, have been used by some organizations to calculate a "*maintainability index*" which provides an indication of the maintainability of a software system. Based on independent studies and separate work performed by the University of Idaho and verified in the field by Hewlett-Packard, the maintainability index confirmed INEL's recommendation to re-engineer the application. This experience with the maintainability index, and the subsequent verification of its applicability, offer powerful insights for decision makers who are contemplating re-engineering of legacy systems. As the USAF/ INEL project has shown, a maintainability index can help in providing sound economic justifications for the re-engineering of legacy software.

The Role of Software Architecture

Many legacy software systems have been developed using a functional decomposition methodology (if any formal methodology was used at all). Early generations of software systems typically make extensive use of proprietary platform features, operating system calls, and language-specific constructs which severely hamper maintenance, reuse, and portability. These fundamental differences in the underlying software architecture contribute significantly to the lack of benefits to be derived from rehosting or translating efforts, as the Air Force learned on this project.

INEL used an object-oriented layered software architecture to achieve many of the benefits of their re-engineering activity. By using layers for the application, interface, graphics, operating system, and hardware, the development team was able to deliver superb benefits to the Air Force. The layered architecture also enabled the use of a hybrid of technologies, language, and methods, in a well-engineered development effort.

COTS (Commercial-Off-the-Shelf) Software and Ada

For government software developers and decision makers, particularly those in the US Department of Defense (DoD), current policy requires the use of COTS software products wherever those commercial products can satisfy DoD needs. In those cases where there are no adequate COTS products, new software must be developed in Ada, unless a waiver is obtained to use another programming language. On the surface, the DoD policy on COTS and Ada seems to be fairly straightforward, but as the Air Force and INEL discovered, a superficial implementation of the policy can be extremely costly — in terms of support and funding. INEL's experience in evaluating the "real" costs and facets of using COTS software can provide government decision makers with substantial savings in choosing between COTS and new Ada code. Decision makers and software managers are deluged with new software product offerings which claim to offer powerful capabilities and benefits. Filtering through the claims and discerning the real benefits, then comparing those benefits with the cost/ benefit of a "*from scratch*" application is a daunting task — regardless of your application domain.

APPENDIX O Additional Volume 1 Addenda

Dual-Use Opportunities

As DoD and other government budgets continue to shrink, it is becoming more important for all government agencies to work together to exploit technologies and programs of common interest. It is also critical for government agencies to exploit technologies which have applicability in major commercial markets — what is known as "*dual-use*" technologies. The USAF/INEL effort is a superb example of interagency cooperation, with benefits accruing to the users, the agencies, the government, and the taxpayer. Much of the code is available as "*public domain*," government-funded software, with excellent applicability in major commercial domains. The combination of object-oriented methods, COTS software, and Ada makes this project an ideal candidate for technology transfer to the private sector.

Summary

This monograph is designed to be a valuable information resource for decision-makers, software developers, and users. Most of the issues addressed are generic in nature, and span a broad cross-section of software domains. This document is part of an ongoing series of monographs which will investigate the major software challenges and solutions required for viable modern complex software systems.

Project Background

The Idaho National Engineering Laboratory (INEL) is a Department of Energy (DOE) National Laboratory, located in Idaho Falls, Idaho. INEL is engaged in a wide variety of projects, ranging from computer and software systems development, to environmental programs, to energy generation .technologies, and national and international technology transfer projects. The organization has a track record for delivering complex software solutions for a broad range of applications, in both government and commercial environments.

This project, the Electronic Combat System Integration (ECSI), was initiated by the INEL in support of die US Air Force Information Warfare Center (AFIWC) at Kelly Air Force Base, in San Antonio Texas. AFIWC's mission includes detailed electronic combat modeling support for a variety of Air Force organizations. One of the models developed by AFIWC is the IMOM (Improved Many-On-Many) electronic combat model. IMOM is an electronic combat modeling system, which supports air operations combat mission planning. Basically, the system helps combat pilots plan their missions in an environment of hostile electronic combat systems. For example, pilots planning combat missions would be very interested in knowing the range at which a hostile radar system would detect their aircraft, so that they could avoid detection during the mission. Furthermore, using IMOM, pilots can run different scenarios showing the difference in detection ranges that would occur if they changed the altitude of the mission profile (i.e., flying in at 500 feet instead of 1,000 feet of altitude can make a huge difference in detection ranges).

IMOM has also been incorporated into the Air Force CTAPS (Contingency Theater Automated Planning System). CTAPS is a command and control system developed by a joint Air Force/INEL team for managing complex air/land battle operations anywhere in the world. Various models of different detection devices and technologies can be depicted in the IMOM system, showing the range and capabilities of a variety of radars. Also, the effect of height above the ground of the sensor, and the effect of electronic countermeasures (ECM) can be determined. In this modern era of increasingly sophisticated detection systems and antiaircraft technologies, the success and survival of US military pilots are heavily dependent on an accurate depiction of the expected coverage of electronic combat systems.

The benefits that IMOM provides to pilots of modern military aircraft were proven during the Persian Gulf War, where the system was used extensively by US pilots, under the auspices of the Air Force Sentinel Byte program. Sentinel Byte disseminates and displays intelligence and key early warning data for use by Air Force combat mission planners and pilots. At the time of the Persian Gulf War, IMOM was a Sentinel Byte application, having been moved

from the AFIWC and CTAPS programs into the Sentinel Byte environment. The fact that IMOM could be moved from one major functional environment (CTAPS) to another (Sentinel Byte) is significant. Basically, IMOM delivered critically important capabilities to Sentinel Byte, virtually immediately, without additional development time and resources being required. Substantial additional functionality was then added to the core IMOM application, enhancing the value of the system to Allied pilots in the Persian Gulf War. This ability to move critical capabilities from one major functional environment to another has tremendous applicability in non-Air Force organizations. Similar benefits can be achieved by large government and commercial operations that are able to share and leverage common operational requirements and solutions.

The IMOM system was originally an AFIWC in-house program used in response to tasking and requests from other Air Force components. Due to its excellent performance and capabilities, IMOM became very popular with Air Force users from other commands. As a result of this superb performance and popularity, IMOM was distributed to a broad range of Air Force users, including pilots and mission planning personnel. The graphical representation of the various IMOM family of models allows users to take advantage of the capabilities of the system in a familiar context and manner — just as they would if they were working with a manual system of maps, charts, and markers. The color graphical interface enhances the fundamental capabilities of IMOM, with corresponding benefits to its users.

Project Evolution — Translating from Fortran to C

The original IMOM capability was developed in 1984. Written in Fortran, the original version was hosted on a proprietary VAX/VMS platform, and used Tektronix PLOT-10/STI graphics. The software was designed using a top-down functional decomposition approach, with a high degree of machine dependency in the code. The IMOM system was a success from the perspective of the users, and the system established a track record of successful usage. Due to the long-term successful track record, IMOM users requested numerous enhancements to the IMOM model(s). As electronic combat threats and technologies evolved, IMOM users required additional functionality in the software. As a result of user needs and advances in electronic combat technologies, the original IMOM expanded quite rapidly, with substantial additions to the initial software system.

In terms of current open systems architectures, the original IMOM was most definitely a "*closed*" system. As the Air Force and the Department of Defense migrated toward open systems technologies in the late 1980s, it became clear that the proprietary IMOM architecture needed to be modified. In 1989, the Air Force tasked the INEL with modifying the IMOM system to enable it to operate in a UNIX/Windows environment, using standard GKS graphics. To accomplish the required modifications, INEL translated the original IMOM Fortran code to C, and moved the software to a Digital Equipment Corporation (DEC) workstation. In keeping with the Air Force's tasking, the original software architecture was retained, and the majority of the translation from Fortran to C was accomplished using an automated translation tool. The end result of this re-hosting task was an open systems implementation of IMOM which ran on UNIX, incorporated GKS (Graphical Kernel System) graphics, and used a "*point and click*" user interface. As was the case with the Fortran version of IMOM, the new C implementation was well-received by users of the system. Some additional functional enhancements were made to the C version of IMOM through 1991.

The original Fortran IMOM evolved as a baseline for the additional capabilities. In 1990, INEL added the functional capabilities required by the Air Force's Sentinel Byte program, eventually evolving the Fortran IMOM baseline through version 5.0. In addition to the Fortran enhancements, the C version of the IMOM baseline was upgraded to include the Sentinel Byte requirements. The additional models are different from IMOM, and have their own community of users, as well as a separate Air Force office. Those models, the COMJAM (communications

jamming), PASSIVE DETECTION, and RECCE (reconnaissance) models, were all originally implemented in Fortran. Unlike the original Fortran versions of IMOM, the other models were not translated into C. The Ada versions were a result of the re-engineering effort conducted by INEL.

Although the various IMOM implementations were quite successful from the users perspective, the different language and platform implementations suffered from configuration management (CM) problems. In addition to the CM challenges, the software was becoming more difficult to maintain and modify. In 1991, the Air Force realized that the existing implementations of IMOM would not be adequate to support user needs into the future. The cost of maintaining and modifying the software was becoming unacceptable, and enhancements were exacerbating the complexity of the system. As a result of these concerns, and in anticipation of expected user requirements in the future, the Air Force tasked INEL with conducting a research study to ascertain the future directions of IMOM.

Research Study — The Future of IMOM

INEL's research study included four primary objectives:

1. Determine the objectives and future goals for IMOM.
2. Identify current industry and DoD standards which could apply to IMOM.
3. Analyze the current IMOM implementations from a software engineering perspective.
4. Offer recommendations to the Air Force as to how to achieve the IMOM objectives.

Among the objectives and future goals identified for IMOM was the need to provide software which would be more maintainable and modifiable than the existing Fortran and C code. More easily maintainable code would allow the Air Force to minimize support costs without sacrificing functionality and reliability. More easily modifiable software would enable the AFIWC to keep pace with changing user needs as well as new and emerging technologies. The ability to migrate the software to more powerful computer platforms was also a sound objective for IMOM. The current industry trends and standards which could apply to IMOM included a wide variety of technologies and tools. Clearly, the need for an open systems architecture was an ongoing requirement for the future. In addition, the use of modern software development methods, such as object-oriented techniques, offered substantial promise for long-term IMOM usage and support. Other DoD and industry standards, such as the Ada programming language, were included in the INEL research study.

As the study progressed, it became clear that there were two major factors which had a direct impact on the design and structure of the IMOM software: the software architecture; and the programming language used for the implementation. The deficiencies of the original software architecture, with its reliance on hardware-specific features and operating system calls, were perpetuated in the translated C version of the code. INEL concluded that a continuation of the original software architecture would effectively preclude any major improvements in the quality and maintainability of the IMOM software. Since INEL's study was conducted from a software engineering perspective, the role of programming language selection in support of a well-engineered software implementation was included in their evaluation of IMOM. In this context, INEL compared Fortran and C with Ada. As INEL applied and evaluated various software engineering principles (i.e., abstraction, information hiding, encapsulation, modularity, etc.), it became clear that the Ada programming language offered superior support for a re-engineered version of IMOM.

Ada also offered benefits in the application of an object-oriented design for IMOM. As part of their evaluation of object-oriented technologies and techniques, INEL noted the importance of applying object-oriented methods in a disciplined software engineering context, as opposed to focusing on overrated and highly abused object-oriented programming features, such as inheritance and polymorphism. The evolution of the various versions of IMOM proved

the importance of a sound, flexible design as a key factor in obtaining the benefits sought by the Air Force in the areas of maintainability and modifiability. From a design perspective, as well as a software engineering perspective, Ada was a superior choice for the future of the IMOM application. INEL's bottom-line recommendations to the Air Force were as follows:

- Re-engineer and redesign the system,
- Use object-oriented technologies, and
- Use the Ada programming language.

Re-engineering IMOM

INEL produced a Software Development Plan which set forth and documented the process by which IMOM would be re-engineered. The Plan emphasized the use of sound software standards, such as Ada, as well as the disciplined application of software engineering principles. The development team followed an iterative life cycle approach, to ensure flexibility and fast response to changing user requirements. The redesign of the software used object-oriented (OO) analysis and design techniques with an implementation in Ada. A hybrid OO methodology was used for the analysis and design, drawing from a variety of well-known OO methodologies offered by Rumbaugh, Booch, and Coad/Yourdon. Basically, the INEL development team used the best features of these various methodologies and combined and adapted them to fit the requirements of the IMOM software redesign. The result of the object-oriented redesign of IMOM was a reusable layered software architecture. The layered nature of the new software architecture enabled INEL to clearly define the interfaces between the layers, and implement the various pieces and subsystems of the application in a highly modular manner. The clear delineation between layers and between modules within layers was an explicit design goal to enable ease of maintenance and modifiability of the IMOM code.

One of the major benefits of the layered reusable software architecture was the mitigation of the risk of using a mix of software technologies and methods. While INEL had recommended the use of solid technologies, such as Ada, object-oriented design, UNIX, GKS, and Motif, those technologies and methods had typically not been combined together all in the same system. The underlying software architecture allowed the use of a hybrid solution consisting of a mix of language, architecture, methodology, and technology.

Because of INEL's focus on a well-engineered layered software architecture, they were able to apply a wide variety of powerful technologies and methods in a disciplined and cost-effective manner. The variety of techniques, methods, and technologies were applied in a controlled and well-engineered process to produce the layered software architecture. Due to the solid success of the IMOM re-engineering effort, the Air Force directed the INEL to proceed with the re-engineering of the other electronic combat models. These other programs included Ada versions of the COMJAM, PASSIVE DETECTION, and RECCE electronic combat models. All of these models were successfully re-engineered during the 1991 - 1993 time period, using the same layered, object-oriented architecture and Ada.

Measures of Success — Speed of Development

Once the re-engineering effort was completed, and all of the IMOM "*family*" of models had been implemented in well-engineered Ada code, an analysis was conducted to measure the success of the program. Although IMOM users were quite satisfied, die Air Force needed to ascertain whether the fundamental IMOM goals of improved maintainability and modifiability of the software had been achieved. Over time, software systems which have been developed using top-down, functional decomposition approaches have experienced significant deterioration in terms of maintainability. This is due, in part, to the effects of adding additional capabilities and functionality, which result in substantial increases in the complexity of legacy

systems. In software systems which have not been well-engineered, new levels of complexity are introduced as defects are corrected, leading to further deterioration of code maintainability.

The introduction of new complexity, combined with the demand for new functionality far beyond the scope of the original design, has an extremely detrimental impact on the ease with which the code can be modified. The Air Force recognized that the limitations of the original IMOM software architecture precluded the implementation of well-engineered, modular upgrades to the software. Although IMOM was meeting the current needs of its user community, the Air Force was anticipating problems which would limit responsiveness to future user requirements.

The translation of the baseline IMOM system from Fortran to C required 54 manmonths of effort. While most of the translation was accomplished using an automated translation tool, the final C implementation required a significant amount of "*clean up*" by the development team. By comparison, the re-engineering effort of the baseline IMOM capability from Fortran and C to Ada required 72 manmonths of effort. Both phases of the project (translation and re-engineering) used a 4-person staff of developers. The re-engineering of the additional derivative models in Ada required a total of 20 manmonths for all three models. These models included the COMJAM, PASSIVE DETECTION, and RECCE models cited earlier.

At first glance, the Ada IMOM re-engineering effort appears to have required an additional 18 manmonths of effort to deliver the same basic capability as the Fortran and C IMOM implementations. This is definitely not the case. The re-engineering project involved the design of an entirely new software architecture, as well as the development of highly modular, reusable Ada code. Basically, the re-engineering effort was an investment in the future, with an expectation of leveraging off that investment to accommodate user needs in a more cost-effective and reliable manner.

As evidence of the value and validity of that investment, the real payoff for the Air Force and INEL began to accrue with the implementation of the Ada versions of the additional models. The actual time required to implement each of the models in Ada was on the order of 5-6 manmonths per model. By comparison, if each model was re-engineered from scratch, they each would have required a level of effort comparable to the baseline IMOM system — the order of 70+ manmonths per model. This is a clear validation of the payoffs to be achieved with a well-engineered software development effort, with a deliberate commitment to design or maximum reuse. Although no specific metrics were collected for re-engineering, the additional models in any language other than Ada, INEL believes that it would have required significantly more time to implement them using Fortran or C. The end result is that, without the application of sound software engineering methods, including a layered, object-oriented software architecture, and Ada, each of the additional models would likely have required the same number of manmonths as the original re-engineered IMOM implementation.

From a user perspective, the reduction in manmonths for each new model has important ramifications: the "*time to market*" or fielding of these critical capabilities can make an enormous difference in terms of lives and equipment or combat pilots and military mission planners. This factor is critical for most organizations, in both military/government and commercial markets. Actual tracking of IMOM project files showed that an interim release of the re-engineered Ada implementation contained 20% fewer defects than the C baselines. Members of the development team attributed the use of an object-oriented design and Ada as major factors in the reduced number of defects. The reduced defect rate was achieved even with the shorter development time for the Ada implementations.

Measures of Success — The Maintainability Index

INEL's research study indicated that the use of well-engineered software architectures, combined with object-oriented analysis and design, and an implementation written in Ada, would result in software that was more maintainable than code which was developed using top-down methods written in other languages. INEL set out to verify that the expected results and benefits

had been obtained. One of the most impressive and critical effects of the Ada re-engineering effort was the impact on the measured maintainability of the code. The maintainability index of the IMOM Ada code was more than three times greater than the index for the equivalent functions written in Fortran or C. This is based on several software metrics which were collected and analyzed by the INEL development team, leading to the calculation of a *"maintainability index"* for the fielded software.

INEL conducted a static analysis of the source code of the various Fortran, C, and Ada implementations. Using PC-Metric, a source code analysis program, the development team calculated the value of two widely known software metrics: Halstead's effort/module; and McCabe's cyclomatic complexity/module. These metrics were just two factors which INEL used to assess the maintainability of the IMOM family of software. Using the values from the Halstead and McCabe metrics, INEL then applied a set of field-proven polynomial metrics to calculate maintainability indices for each of the IMOM baselines. The polynomial metrics were developed at the University of Idaho and have been validated in the field by Hewlett-Packard (HP). Hewlett-Packard has set a *"maintainability cutoff"* of 65 on the maintainability index. Based on HP's experience with software in the field, a software package with a maintainability index of less than 65 is considered to be *"difficult to maintain."* HP's separate evaluation and independent use of the maintainability index verified that it was applicable for HP's software systems. INEL applied the maintainability index to the IMOM family of software systems to provide a comparison and perspective as to the maintainability of the Ada, Fortran, and C versions of the IMOM baseline. The Fortran and C versions showed an accelerated decline on the maintainability index, while the Ada implementations stayed constant. The Fortran and C versions of the IMOM software were becoming *"more maintainable"* as additional functionality was added. In contrast, Ada versions of IMOM exceeded the HP cutoff value, and stayed virtually constant, even with additional functionality being added.

Measures of Success — Software Complexity

Tracking the trends indicated by the metrics, INEL documented a dramatic increase in the complexity of the Fortran-based models of IMOM. The trends in the Fortran models showed a nonlinear increase in software complexity as IMOM evolved from version 4.0 to 5.0, with a corresponding projection for continually worsening maintainability over the life of the application. For the C versions of the IMOM baseline, the complexity was roughly the same as the Fortran implementations. This rough equivalency is due in part to the fact that the C code was derived from the Fortran software, using the same basic software architecture, indicating a functional equivalency between the various language implementations of IMOM. For example, the Fortran version 4.0 is functionally equivalent to the C version 1.0, and the Ada version 1.0.

The average cyclomatic complexity (a measure of the number of paths through source code) for the Ada IMOM baseline was slightly larger than 3, while it was more than 9 for the Fortran implementation. For the C version, the complexity metric was greater than 13. The several-fold increase in complexity for the Fortran and C versions in comparison to the functionally equivalent Ada code has a direct effect on the maintainability of the respective language implementations. The maintainability of the Fortran and C code markedly decreased over time, while the maintainability of the Ada code stayed virtually constant. The consistency of the maintainability of the Ada code indicated a significant reduction in the complexity of each individual module. Each IMOM baseline module written in Ada was smaller, simpler, and easier to read and understand than the equivalent programs written in Fortran and C. The bottom-line benefit for personnel engaged in software maintenance and modifications was that less effort was required to understand the function and execution of each subprogram.

The Ada IMOM version 2.0 has many more functional capabilities than version 1.0. Contrary to the trends of the Fortran and C implementations, the increased functionality in the enhanced Ada code has not increased the complexity of the code. The cyclomatic complexity

of the Ada IMOM version 2.0 had the same relative magnitude as the Ada version 1.0, indicating that the modifications and maintenance of the code had little impact on the complexity of the Ada software. The number of unique operators and operands in the Ada versions of IMOM greatly increased in comparison to the Fortran implementations. The number of unique operators were 1.8 times higher in the Ada IMOM version 1.0 than in the Fortran version 4.0, and there were 2.5 times as many unique operands in the same Ada version. The total number of operators and operands also nearly doubled.

One of the primary reasons for the increase in unique operators and operands in the Ada code is that the Ada software has many more local variables, less globally accessible data, and passes many more formal parameters between modules. These facets, in turn, are "*by-products*" of the object-oriented design, and the layered reusable software architecture, where well-defined interfaces between objects, layers, and components are required. Basically, these characteristics of the Ada code reflect the application of the proven software engineering principles of modularity and information hiding.

Even with the additional volume of code in the Ada version of the IMOM baseline, the Halstead-estimated effort, as a complexity indicator, was 53% less for the Ada code than it was for the equivalent Fortran implementation. This is in spite of the fact that the Ada version has 63% more executable statements that the Fortran version, and more than four times as many subroutines. The fact that Ada source code, unlike many languages, is not terse, cryptic, or obscure, contributed significantly to the expanded number of statements and subroutines. The size of the Ada modules was much smaller than the modules written in Fortran and C. The Fortran IMOM baseline was comprised of about 334,000 lines-of-code, with the typical use of global data access and Fortran subroutines that is common with functionally decomposed software designs. By comparison, the Ada implementation was comprised of 213,000 lines-of-code, which means that the Ada modules, although more numerous, were also much smaller than the equivalent Fortran components. The reasons for the reduced lines-of-code in the overall systems include the following:

- A significant amount of code was shared between modules,
- More streamlined implementation of required functions, and
- Changes in the functionality of the models.

The smaller code modules in Ada offer additional benefits to the Air Force in terms of maintainability and modifiability. Smaller, easier-to-understand modules enable the use and exploitation of the reusable layered software architecture, with explicit support for the object-oriented design employed by the INEL development team. The Ada modules are easier to work with, test, and modify, thus facilitating the iterative software life cycle approach chosen by the INEL team. The discrete and well-defined interfaces between modules prevents major surprises when the software is integrated and tested — an area that is historically a major source of delays and problems and cost increases.

Code modules of low complexity and size also have an impact on the effectiveness of all support personnel. Additionally, the number of people required to provide support, as well as the skill and experience of the support staff, are directly affected by the low complexity and size of the IMOM baselines. With small and easy-to-understand modules, new maintenance personnel can be brought up to speed very quickly, with minimal impact on the quality and responsiveness of support. Furthermore, with easy-to-understand code packages, highly-skilled senior software engineers who are expensive and in short supply, can be much more productive and efficient. Less senior personnel can be used to conduct routine support, error fixes, and modifications to the code, freeing the senior staff to address more complex support requirements. Finally, fewer people are required to provide on-going maintenance and support, with a corresponding reduction in resource allocation and funding.

By comparison, the Fortran and C versions of IMOM, due to a significant degree to their much greater levels of complexity, are very challenging and difficult to maintain and modify. Safe and reliable maintenance and modification of the Fortran and C code requires the allocation of very experienced, very costly, and very scarce software engineering talent. Even senior and experienced personnel will require much more time to assimilate and understand the intricacies and complexity of the non-Ada implementations of IMOM. For both government/ military and commercial organizations, the ability to allocate top software talent to areas with a higher return on investment (i.e., the design of new systems) can have a profound effect on the efficiency of operations. In military commands, where deployment of systems like IMOM to remote areas of the globe is common, the capability to maintain and modify critical software systems is often measured in terms of lives and equipment. Making software easier to maintain and modify with fewer people and with people who are less experienced provides an enormous return on investment.

From an operational perspective, the combination of the layered software architecture, object-oriented design, and Ada has shown conclusively the effect of software as a "*force multiplier*" for the US military. Without that working combination of technologies and methods, the benefits of the USAF/INEL re-engineering effort would have been substantially reduced. INEL also studied the three additional electronic combat models which were implemented (RECCF,, PASSRVE DETECTION, and COMJAM), to compare results on those derivative models. The trends discovered in the IMOM baseline also showed up in all three of the other models, including the following:

- The Ada version contained many more modules than the Fortran implementation. This is, to a significant degree, the result of a completely different software architecture being used for the well-engineered Ada versions of the models.
- The Ada code modules were smaller in size than the Fortran versions. The smaller modules greatly facilitate the understanding and maintainability of the software. The modules are easier to comprehend, modify, and reuse.
- Each of the Ada modules is much simpler than the Fortran modules, as measured by the cyclomatic complexity values.

Measures of Success — Module Maintainability

To provide a more detailed examination of the complexity and relative maintainability of the various systems, INEL conducted another analysis using the same polynomial model cited earlier. For this analysis, however, instead of evaluating an entire model (i.e., all of the IMOM baseline), INEL calculated the maintainability index for each subroutine of each software baseline. In other words, each subroutine in the Fortran, C, and Ada IMOM baselines was evaluated, and a maintainability index calculated. Once each subroutine in each of the various IMOM baselines was evaluated and a maintainability index calculated, the modules were then categorized according to their respective maintainability values. Modules with a maintainability index of less than 65 (deemed unmaintainable by Hewlett-Packard) were categorized as "*low*." Modules with a maintainability index between 65 and 85 were ranked as "*medium*." Modules with a maintainability index higher than 85 were rated as "*high*." The Ada modules in the implementations of the various IMOM models were substantially more maintainable than the modules written in either Fortran or C. Generally, nearly two-thirds of all of the Ada modules rank in the "*high maintainability*" category. Conversely, around half of all of the modules written in Fortran and C are rated as "*low maintainability*."

There was also a downward trend in maintainability as the Fortran and C versions were modified and additional functionality was added. In both the Fortran and C implementations, the number of modules which fell into the "*low maintainability*" category increased over time. Conversely, the number of Ada modules which were rated for high maintainability remained

relatively constant over time and with additional functionality. The INEL development team attributed the benefits of the Ada versions of IMOM to the emphasis on sound software engineering, the use of the layered software architecture, the application of object-oriented design, and the stability, clarity, and software engineering support of Ada.

Software Reuse

Software reuse does not enjoy a commonly accepted definition. Basically, "*reuse*" means many different things to different people, depending on their area of expertise, technical background, etc. The premise and promise of software reuse has been analogous to a technical "*holy grail*" for many years. The DoD and other government agencies, as well as major commercial enterprises, have pursued high degrees of software reuse as a means of lowering software costs, increasing productivity, improving reliability, and leveraging investments in software. Unfortunately, the promise of software reuse has not been widely attained, due to a variety of factors and impediments.

The INEL development team was able to amply demonstrate that software reuse is not a myth, and that significant levels of reuse can be obtained, with corresponding benefits to both developers and users. As a foundation for obtaining substantive reuse, the underlying software architecture and software design are critical. The INEL team designed the Ada IMOM baseline with reuse in mind, with the expectation of reusing significant amounts of code as new models were added to the IMOM family. The layered software architecture was a key element in the attainment of high degrees of reuse in the re-engineered Ada implementations of the IMOM family of models. The INEL development team, knowing that additional models and functionality would be required beyond the initial IMOM baseline, planned their software architecture accordingly. They structured the layers to enable high levels of reuse within the domain-specific portions of the applications, an used layered "*application frameworks*" to leverage their investment in common interfaces.

For example, by planning for and designing the system for reuse, INEL was able to develop common user interfaces (man-machine interfaces, or MMI) which could be reused for all of the IMOM models. Because of the structured, object-oriented layers of their design, the development team was able to simply concentrate on the engineering algorithms of each new model or function (RECCE, PASSIVE DETECTION, and COMJAM), without concern for the underlying software architecture and MMI.

The layers also provide insight into the "*type*" of reuse attained. Basically, reuse can be separated into two general categories: domain-specific, and general purpose. Using the "*Application Layer*" as an example, there was significant reuse across that layer which was domain-specific to the IMOM models. INEL achieved high levels of reuse in both the domain-specific and general purpose layers of the ECSI effort. The middle layers were analogous to "*application generators*" and were reused throughout every layer. For example, the "*Interface Layer*" provided reusable code which could be applied throughout the vertical range of layers. These generic reuse capabilities can provide powerful advantages for users and development teams, for requirements such as mapping tools. By reusing the common constructs for maps (i.e., bearing, heading, coordinates, etc.) the INEL team has established a generic mapping tool with a general applicability for a wide variety of domains. By using this tool, a developer would be able to quickly produce a capability equivalent to a software program of 50,000 to 100,000 lines-of-code. The tool has the additional advantage of being comprised of code which has already been tested and fielded.

Generally, INEL realized a "*reuse rate*" 65% which means that 65% of the Ada code modules were reused in one or more applications. The savings which resulted were impressive: without that level of reuse, each of the three IMOM models (other than the baseline) would have required at least another 100,000 lines of new code. Stated another way, a less structured and disciplined approach would have necessitated a minimum of 300,000 lines of additional new code development. The impact on productivity and "*time to market*" is the most obvious benefit

of the reuse rates achieved by the development team. There was also a substantial benefit which accrued due to the reduction in testing and integration, since the reused modules had already been through that process. The reduction in risk, due to the virtual elimination of the new errors that would have occurred with the development of new code, was also noteworthy. The bottom line for the reuse of Ada code in the various IMOM models can be summed up with this observation by a representative of the INEL development team:

> *Due to the insight and foresight of the Air Force, we were encouraged to apply sound software engineering rigor and discipline. We were allowed to apply the appropriate methods in the up-front engineering of the system, to ensure a payoff to users in the future. If we had not been able to design for reuse, and actually take advantage of reused Ada code on this project, each of the derivative models would have taken significantly more time to develop. The results achieved on ECSI are a validation of the benefits and return on investment which can be realized by applying a soundly engineered mix of technologies and methods.*

If a poor process had been employed for ECSI, and sound software engineering had not been used, each of the three additional models (COMJAM, RECCE, and PASSIVE DETECTION) would have required 72 manmonths of development time, for a total of 216 manmonths. By taking advantage of Ada code reuse (in support of the engineering process used by INEL), all three of those models were implemented in a total of 20 manmonths — less than one-tenth of the time without reuse.

As most software engineers and managers know, reuse can be achieved with software designs, as well as actual code. One of the highest payoffs can be achieved by reusing are designs, including object-oriented designs. INEL recognized during the analysis phase of their effort that each of the four IMOM models had a wide variety of components which could be shared among the models. Each of the applications contained common or similar user interfaces, modeling algorithms, graphics, and electronic components. During the design phase, the development team focused on determining the appropriate abstraction for each object class and its respective role within each IMOM model. The level of reuse for each object was directly linked to its role within the various models. Some object classes had the potential for reuse outside the electronic combat application domain, others could be shared by two or more of the IMOM models, and others were highly specialized and specific to a particular role within a single model. By evaluating the specificity of the various object classes, the INEL development team was able to make decisions as to when to design for broad reuse (i.e., outside the electronic combat domain) and when to limit the design to a specific IMOM model. The development team discovered that most object classes in the IMOM domain were conceptually the same although their characteristics and roles within each of the models were different. When those differences were relatively minor, an object class was designed which would meet the multiple requirements. Even with the broader requirements, this approach added little additional complexity to the object classes.

The results of this approach were impressive. Interim releases of the four IMOM models indicate the benefits to be derived — the four models in those releases had approximately 112 object classes, of which 73 were shared between two or more software systems. Each class had an average of 1,200 lines of commented source code. There were 11 algorithmic or interface libraries which are shared between the models. INEL discovered that reuse is not the general panacea which is often touted as a goal for software development. For certain types of problems, and within specific application domains, reuse can improve the quality of software and enhance the development process. The greatest reuse is possible when sharing objects within the same specific application domain by a single software development team. Beyond those parameters, object reuse is still quite difficult to achieve.

APPENDIX O Additional Volume 1 Addenda

The Benefits of Ada

Throughout this monograph, the benefits of Ada in terms of its explicit support for software engineering discipline and object-oriented design have been cited as major factors in the success of the USAF/INEL project. The Ada language features provided excellent support for the implementation of the layered software architecture, as well as the attainment of significant levels of code reuse. Furthermore, Ada's strong typing, parameter checking in subroutine calls, and array constraints contributed to the benefits achieved in the ECSI program. In addition to these powerful attributes and benefits, the clarity of Ada source code and the benefits of that characteristic must be specifically cited. The names of variables used in the Ada IMOM baseline are more descriptive of the actual items or phenomena being represented than comparable representations in the Fortran version. The Fortran baseline used primarily six character identifiers for variables and subroutine names, a limitation which was carried over to the C-based translation. The descriptors in the Ada version allowed the "*real world*" to be more accurately represented in the actual source code. The C version of the IMOM baseline was just as terse as the Fortran version, due to the fact that the C code was translated from the Fortran and retained the same brevity for variable names. C has earned a reputation for being more terse than other programming languages, so it can reasonably be expected that a new version of IMOM written in C would still include terse names for variables.

The additional clarity of the Ada source code pays substantial dividends in the areas of maintenance and modification of the software. Maintenance personnel are able to relate or "*tie*" Ada source code to design documents and change requests more effectively and efficiently than with comparable Fortran and C implementations. The clear and descriptive nature of variable names in Ada help significantly in reducing error rates and the introduction of new defects during code maintenance and modification operations. Ada's package construct also supports the use of layered, object-oriented, flexible software architectures. The Ada package basically consists of two parts: the package specification and the package body. The specification part of the package enables a sound engineering implementation of the software requirement, through its support of encapsulation, information hiding, abstraction, etc. Interfaces between packages are specified and verified before the package body is written. The inputs, outputs, and assumptions for each package are clearly delineated and established.

The package specification provides a great deal of information to maintenance personnel, akin to a high-level summary of the characteristics of the package. In addition to defining module interfaces, package specifications also offer essential insights to the humans who must be able to understand the inner workings of the software modules to be able to maintain and modify the code in a reliable, cost-effective manner. Neither Fortran nor C provide any comparable capabilities to the Ada package specification. The clarity of Ada source code (Ada code is written in English, as opposed to obscure symbols and other representations, and resembles a structured English outline), combined with the structured nature of its package specifications, has a profound effect on documentation and software reuse. By minimizing the complexity and size of Ada code modules, Ada software implementations become virtually "*self documenting*" due to the explicit representation of the function and interfaces of the respective modules. The extensive information included as part of the package specification enables developers to ascertain the potential for the reuse of existing modules in other applications, without having to delve into the source code of the package body itself. The improvements in efficiency, accuracy, and productivity are substantial.

The INEL development team discovered that the learning curve for Ada is far less than perceived. The team was able to assimilate fundamental Ada constructs within a minimal period of time. Furthermore, the team was able to use and apply engineering discipline and methods using specific features of the Ada language. As the INEL team noted, "*You do not have to use all of the features of Ada to make effective use of the language.*"

Ada and COTS

As mentioned in the Executive Summery policy requires the use of commercial-off-the-shelf(COTS) software whenever they will satisfy requirements. For those applications where COTS is inadequate to meet DoD needs, new software must be developed in Ada. The USAF/ INEL re-engineering project had some valuable experiences in effectively applying that policy. Many of the "*COTS or Ada*" decisions for the IMOM models were obvious. For example, INEL did nor want to write its own operating system, or its own version of UNIX or Motif/X, Similarly, existing software interfaces or drivers to peripheral devices did not warrant the development of new Ada code. The flexibility of the layered object-oriented software architecture enabled the development team to mix and match the appropriate technologies and tools without sacrificing maintainability and the ability to add new functionality. To meet the requirements of IMOM users, the INEL team had to develop two *"spin-off"* products: the Data Stream Analyzer and FormBuilder. The Data Stream Analyzer was basically a generic parser which translated data from one format to another. This is a valuable and essential capability for IMOM users, since different sources provide data in different formats. The Data Stream Analyzer takes care of converting differing formats into one which can be accepted by the various IMOM models.

In the case of the FormBuilder, the Air Force and INEL both knew that there were several COTS products on the market which provided the capability which was needed. The basic requirement was for an application programmers interface for developing Motif-based graphical user interfaces (GUIs). At first glance, it appeared that simply buying a COTS form-building tool would be the best choice for the Air Force, and in keeping with DoD policy. Fortunately for the Air Force (and the taxpayer), INEL went beyond a "*first glance*" evaluation. The INEL team used GKS (Graphical Kernel System) for a significant length of time on the ECSI project. Although GKS is a standard, the implementations of the standard by commercial vendors are not consistent. By using commercial product implementations of GKS, INEL and the Air Force were dependent on product changes and support from vendors, who were subject to commercial market influences and different business goals. This risk was realized by the development team when the vendor of the GKS product being used for the ECSI project suddenly decided to discontinue marketing and supporting their implementation. INEL also did not want the Air Force to be subjected to the additional cost of GKS runtime licenses, so the development team built their own graphics package, called FBGraphics. For the Motif-specific requirement, INEL discovered that many COTS form-building tools worked effectively, and provided the capability to develop GUIs for applications. However, the tools all generated substantial amounts of code which was Motif-specific. This Motif-specific code, while performing the GUI tasks required for an application, was not itself readable or maintainable. INEL was thus able to justify the development of a FormBuilder written in Ada, which did not have the overhead of licensing fees and the generation of unmaintainable code. The development team required only two months to develop the Ada-based capability.

While the policy of "*COTS first, Ada second*" makes sense or many applications, there are significant hidden costs and other factors which need to be considered. For example, configuration management (CM) of an application can become a nightmare for program managers and maintenance personnel, if multiple COTS products are incorporated into a system like IMOM. As COTS vendors make changes and offer new releases, the impact of those changes can have serious and unforeseen effects on the rest of the system. If a fielded system uses several COTS products, the CM problems can be rapidly compounded over time. When evaluating a choice between COTS capabilities and new Ada software, decision makers must take into account much more than just the perceived "*up-front*" costs. In the case of the IMOM effort, INEL only required two months to produce a capability which could have been superficially satisfied by a COTS product, but the COTS solution would have been a serious detriment to the maintainability and modifiability of the IMOM models over time.

The IMOM family of models also required a database management system (DBMS) capability to meet user needs. In keeping with DoD's "*COTS first*" policy, it has become popular and common for program managers to pursue the use of commercial database management

products, especially relational DBMS capabilities. For the ECSI effort, the INEL team evaluated the efficacy of using a COTS DBMS versus writing the appropriate capability from scratch. Their solution for DBMS for ECSI was straightforward and easy to maintain, and consisted of ASCII flat files. This solution met user needs, and saved the Air Force substantial time and development resources.

Dual-Use Potential

Since the software developed for the IMOM models has been paid for by the US Government, some of it is "*in the public domain*" and available for use by other government agencies and private enterprises. Some of the code has obvious commercial applicability and reuse potential, while other pieces of the IMOM applications are limited to specific domains. Clearly, the Data Stream Analyzer and FormBuilder tools are ideally suited for commercialization and exploitation by private companies. Similarly, other government organizations can derive substantial benefits from the use of these "*free*" tools. The user interfaces and graphical capabilities of the IMOM implementations have widespread generic applicability. The point and click nature of the user interface, along with the color capabilities of the various representations, are common features in most modern commercial systems. As an example of possible dual-use applicability, the generic mapping "*application generator*" cited earlier could be used by some commercial developers. Commercial developers who require basic mapping functions, as well as state and local governments engaged in economic development programs, are examples of organizations who could benefit from mapping software which is well-engineered and implemented in an international standard programming language. Other industries, such as aviation, marine, oil exploration, and land management/industrial planning operations could also make use of this generic capability.

By virtue of the partnership between the Air Force and the Department of Energy's INEL, the process of dual use has already begun. The INEL has, as part of its mission, the transfer of technology to the commercial sector. Many significant software capabilities embodied in the IMOM models offer valuable features and benefits for commercial exploitation and use.

SUMMARY

The IMOM re-engineering project provides valuable lessons for both government and commercial software developers and decision makers. As modern systems continue to grow in size and complexity, the critical nature of well-engineered software in the success and survival of virtually all organizations is becoming more pronounced. The intelligent selection and application of solid methods and technologies are essential facets for progress. Key decisions related to the migration from legacy systems to more powerful distributed client/server architecture require sound justifications. The cost of maintaining and modifying legacy software is a critical factor in justifying a re-engineering effort. The "*hidden*" costs of providing adequate capabilities to the user base are often substantially higher than the explicitly measured costs of support.

The IMOM re-engineering project provided conclusive lessons as to the importance of software architecture in achieving significant software-related benefits. The program also showed that the application of object-oriented methods, in the context of a disciplined software engineering process, can deliver major gains in reuse, productivity, and maintainability. Finally, the IMOM re-engineering effort showed the value of Ada in obtaining maximum payoffs from an investment in software engineering practices. The benefits achieved in applying sound software engineering principles were multiplied by the explicit support that Ada provides for that disciplined approach. Simply stated, the magnitude of the benefits obtained would have been substantially lower if another programming language had been used. The IMOM formula for success was: Good people, with knowledgeable managers applying sound software engineering methods, implementing in Ada, with the needs of the user as the objective. That formula works for all segments of the global software community.

BIBLIOGRAPHY

Coleman, D., "Assessing Maintainability," 1992 Software Engineering Productivity Conference Proceedings, Hewlett-Packard, 1992, pp. 525-532

Idaho National Engineering Laboratory, "Improved Many-On-Many (IMOM) Model Research Study," EGG-EE-9555, Rev. 0, May 1991

Oman, P. and J. Hagemeister, "Construction and Validation of Polynomials for Predicting Software Maintainability," Software Engineering Tet Lab, Report #92-06 TR, University of Idaho, July 1992

Welker, K., M. Snyder, and J. Goetsch, "Ada Electronic Combat Modeling Experience Report," Presented at OOPSLA '93

Welker, K., "Application of Software Metrics to Object-Based, Re-engineered Code Implemented in Ada," Masters Thesis, University of Idaho, April 1994

Welker. K. and M. Snyder, "Electronic Combat Model Re-engineering," ECSI Project briefing slides, 1994

CHAPTER 13
Addendum B

Contracting for Success

Jerome S. Gabig, Jr.

ABSTRACT

Senator Cohen's scathing report, *"COMPUTER CHAOS: Billions Wasted Buying Federal Systems,"* alludes to "*inevitable problems with software development*" that cause cost overruns and schedule slippages. Cost overruns and schedule slippages need not be "*inevitable*." This presentation focuses on two critical success factors that enable the government to greatly increase the probability of a successful software development contract. The first critical success factor is equitably allocating the risks between the parties. The second critical success factor is structuring the evaluation criteria to maximize the probability of selecting the best qualified offeror.

OVERVIEW

Where an agency must use a vendor to perform a software project, there are two critical success factors regarding the contracting process that greatly increase the probability of a successful software development effort. The first critical success factor is to structure the contract to allocate equitably the various risks between the parties based on which party is best able to manage the risk. The second critical success factor is to structure the evaluation criteria to maximize the probability of selecting the best qualified offeror.

The Importance of "*The Written Word*"

A congenial relationship between the contractor and the government is almost indispensable to the successful completion of a software development effort. One might think that a congenial relationship would diminish the importance of the "*written word*." Instead, by minimizing the probability of misunderstandings, a well-written contract is a major contributor to a congenial relationship between the parties. As recognized by the Software Technology Support Center, "*once the initial contractor enthusiasm is over, the written word...has the most influence on contractor actions*." Experience has shown that when "*the written word*" is unambiguous as to the duties and responsibilities of the parties, the animosity that arises from quibbling over performance obligations usually can be avoided.

Structuring The Contract to Best Allocate Risks

DoD has consistently recognized the need to structure contracts to allocate risks in an equitable and sensible manner:

> *The contacting approach selected for each acquisition phase must permit an equitable and sensible allocation of risk between Government and industry.* (DoD Directive. 5000.1, Feb. 23, 1991, at C.3.)

Risks essentially fall within three categories: cost, schedule, and performance. Each of these three categories of risk deserves a separate discussion.

Cost Risks

The foremost way of allocating cost risks is through the selection of the type of contract. For instance, FAR § 16.103(b) states that a firm-fixed price contract "*shall be used when the risk involved is minimal or can be predicted with an acceptable degree of certainty.*" Although rarely followed by contracting officers, the FAR also admonishes that the contract type generally should be negotiated with the offerors:

> *Selecting the contract type is generally a matter for negotiation and requires the exercise of sound judgment. * * * The objective is to negotiate a contract type and price (or estimated cost and fee) that will result in reasonable contractor risk.* [FAR §16.103(a)]

The reasonableness of the cost risk to the contractor is a factor of how accurately the contractor can estimate the cost to perform the work. The highly recognized work of Barry Boehm, as shown in Figure O-39, reveals the increasing degree of accuracy for estimating costs as a software development project proceeds through the phases of the waterfall model. Superimposed under the x-axis of Figure O-39 is the linear progression of contract types in the sequence in which they represent decreasing risk to the contractor. The superimposed x-axis should not categorically dictate the contract type for any particular phase of a software development project. Nevertheless, Figure O-39 correctly suggests that as the relative accuracy of the cost estimate increases, it is appropriate to select a contract type that correspondingly places increased cost risks on the contractor.

The Premature Use Of A Fixed Price Contract Invites Failure

Within the government, there has been a long-standing aversion to cost-reimbursement contracts because of a lingering suspicion that contractors are not motivated to work efficiency. Another suspicion has been that a contractor is less likely to assign its best software engineers to a cost-reimbursement contract. As a norm, fixed price contracts for complex software development contracts are not conducive to the iterative nature of the process. The following excerpt expresses a user's perception of the anticipated "*give and take*" necessary to refine the software requirements specification:

> *Actually, the software specification review is an iterative process with the iterations consisting of the contractor submitting a draft of the spec, the technical monitor reviewing and recommending changes to the draft, the contractor making some changes and resubmitting a revised draft. The iterations continue until the program manager feels*

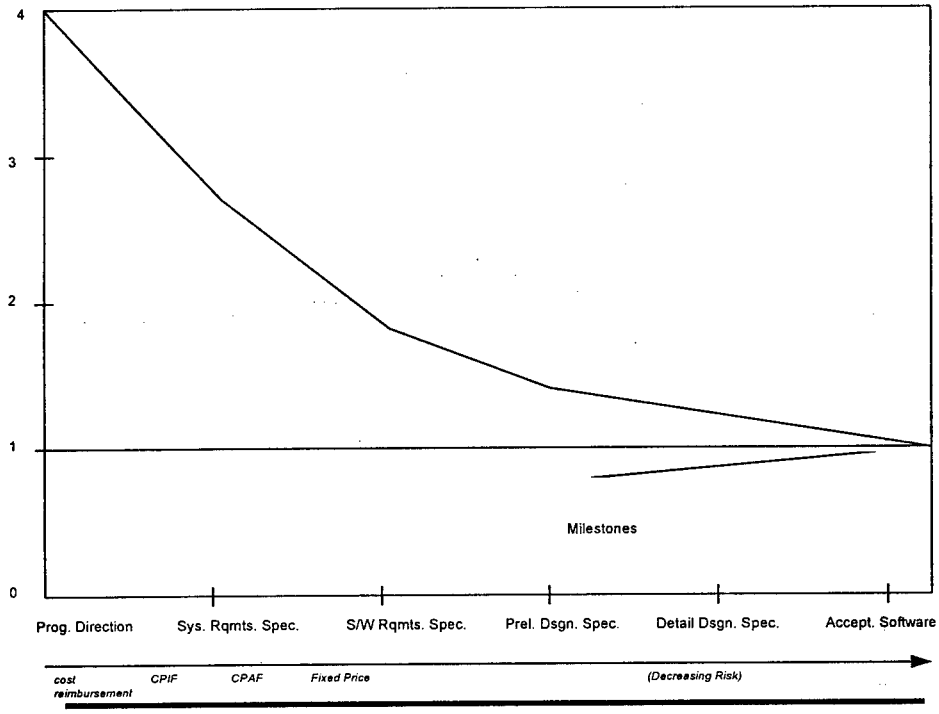


Figure O-39

that the software requirements specification establishes the allocated baseline for its CSCI.

Yet, on a fixed price contract, the contractor is apt to regard anything beyond the second iteration as unwelcomed meddling by the user's technical staff. Such behavior by the government's technical representatives may result in the contractor submitting claims. A fixed price contract that is inundated with valid claims typically does a poor job of shifting cost risks to the contractor.

The Premature Use Of Fixed Price Contracts Favors Vendors With Immature Processes

A major difference between a vendor with mature processes and a vendor with immature processes is the consistent capability to accurately estimate the costs to perform a software development project. Figure O-40 (below) graphically depicts the relative accuracy of a cost estimate by a SEI Level 1 vendor in contrast to a cost estimate by a SCE Level 3 vendor.

As shown in Figure O-40, the SEI Level 1 offeror is much more likely to be quantumly incorrect in its estimate than the SEI Level 2 offeror. Moreover, the probability is that the SEI Level 1 offeror will underestimate the cost of the project.

Routinely, where the SEI Level 1 vendor overestimates a project, its proposal is at a competitive disadvantage and rarely receives the award. Conversely, where the SEI Level 1 vendor underestimates a project, its proposal gains a competitive advantage and frequently wins the award. The net result is that, over a period of time, the Level 1 vendor's portfolio of contracts

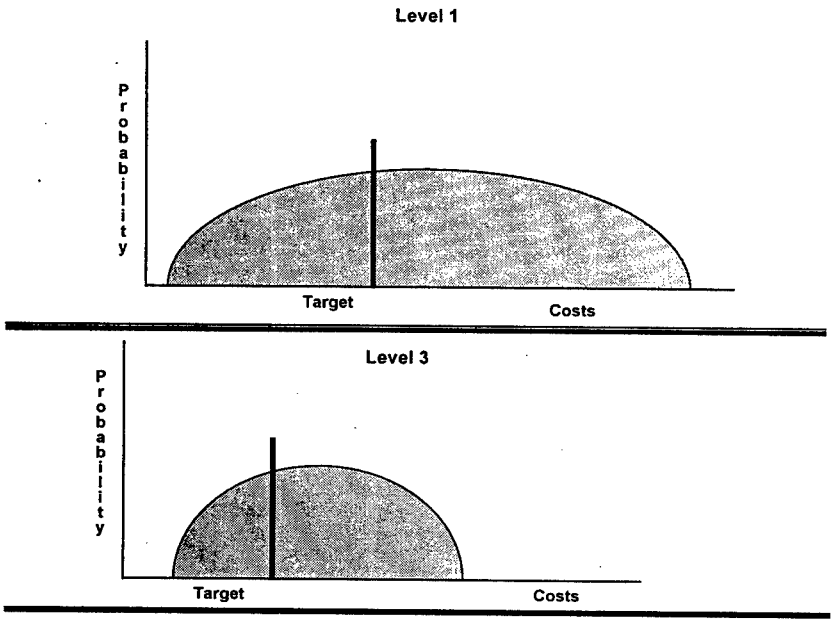


Figure O-40

predominately consists of fixed price contracts that are experiencing significant overruns. Anecdotal evidence strongly supports that once a Level 1 offeror begins to lose money on a fixed price contract, the likelihood that the software will be completed to the satisfaction of the agency is greatly diminished.

Unlike most SEI Level 1 vendors, SEI Level 3 vendors have invested heavily into process improvements. Consequently, in terms of reduced overhead, the Level 1 offeror enjoys a conspicuous price advantage. In light of this price advantage, vendors with Level 3 and higher rated processes are reluctant to spend their bid and proposal money on fixed price acquisitions unless the evaluation criteria is structured to favorably consider their superior capabilities and processes. Unwittingly, the government sometimes deters highly competent vendors from submitting a proposal by improvidently selecting the contract type or not astutely drafting the evaluation criteria. (See generally, the discussion below which essentially states that selecting a highly competent vendor is one of the two critical success factors to a software development project meeting the cost, schedule and performance requirements.) Plain and simple, discouraging the best qualified offerors to submit proposals greatly decreases the probability of a successful software development effort.

Performance Risks

Cost risks, performance risks, and schedule risks are generally interdependent notwithstanding that they are usually addressed separately in risk mitigation plans. The interrelationship between cost and performance risks is exemplified by the following observation of the Court of Claims:

> *[C]ontractors are businessmen, and in the business of bidding on Government contracts they are usually pressed for time and are consciously seeking to underbid a number of competitors. Consequently, they estimate only on those costs which they feel the*

contract terms will permit the Government to insist upon in the way of performance.

This observation is particularly applicable to software development contracts. It is a common attribute of the software development process that the preponderance of the requirement must be decomposed before a comprehensive specification can be drafted. Without a thorough specification to uniformly bind all offerors to a common baseline of performance, the competitive pressure to underbid competitors motivates offerors to only bid what the government *can "insist upon by way of performance."* Once the contract is awarded, the tact the vendor took to win the contract usually necessitates that the contractor contest the allocation of performance risks in each instance where the specification is not abundantly clear. Hence, it is not until late in the software development cycle that the government can effectively shift the risk of nonperformance onto the contractor.

An environment where a vendor frequently contests what the government perceives as a contractual obligation is not only disruptive to the smooth progression of work, but also it can be inimical to the much needed congenial relationship between the parties. The following is an extract of a decision of the Armed Services Board of Contracts Appeal which exemplifies the debilitating bickering that can arise where a fixed price contract is used without a detailed specification:

The lesson-learned is abundantly clear but frequently overlooked. Before the government selects a fixed price contract ostensibly to place the cost risks on a vendor, the government should scrupulously examine the specification to assure that the performance risks are unequivocally passed to the vendor.

A fixed price contract only encourages a contractor to perform the bare minimum since anything more must be paid from potential profits. The incentive to perform only the bare minimum is especially strong where the contractor begins to lose money on the venture. Accordingly, a fixed-price contract does not necessarily motivate the contractor to make the refinements that exceed the bare minimum. An example would be where the contractor is obligated to prepare the software requirements specification. Any money saved using a fixed-price contract might be a false savings since a substandard specification can be ruinously expensive to correct later in the software development process where, for lack of diligence, previously undiscerned requirements are discovered. Another example would be late in the software development cycle where the software satisfies the functionality of the software requirements specification but the software needs some minor enhancements to be user-friendly. Such enhancements are less likely to be made voluntarily where there is a fixed price contract.

Changes Can Impact The Previous Allocation Of Performance Risk

An endemic problem with large software development projects has been excessive changes. Generally, the causes of excessive changes are either a substandard requirements analysis or requirements that are too dynamic to be effectively "*frozen*" into a specification. A landmark GAO report documented why changes frequently are not recognized as contributing to performance risk:

Changes requested after projects have started, which seem trivial to the customers, have often required major rework and have resulted in delays and increased costs.

- *Changes are not usually as thoroughly researched as original design concepts and sometimes have unforeseen effects on other parts of the system.*
- *Effective use of contract phasing can be destroyed by constantly making changes to work that was competed and approved in earlier phases.*

Additionally, because software is generally perceived as pliable, the users frequently do not appreciate the cost impact of seemingly minor changes. The National Research Council has observed:

> *Late discovery that some required functions intended to be implemented in hardware cannot practically be so implemented and are shifted to software. This shift might not occur save for the prevalent optimistic view of the pliability of software. In truth, software is not pliable in large, complex systems; a small change in software function can ripple through many interfaces amounting to a major redesign effort, particularly if the added function was not anticipated during the decomposition of CPCIs and modules.*

With regard to the cost risks, it is a fundament rule of government contracts that the contractor is entitled to compensation for the "*unanticipated and extra out-of-pocket expenses it incurred in performing the contract as a result of the changes.*" It is not so widely recognized that changes can sometimes impact the previous allocation of performance risks. Specifically, where the government has crafted the contract to place the performance risks on the contractor, changes that require a contractor to perform in a manner different from what the contractor originally intended can transfer the performance risks from the changes to the government.

Performance Risk Regarding Architecture

Within the past several years, there has been increased recognition that fundamentally flawed architectures are one of the leading causes of fatality among large software development contracts. Where the architecture is fundamentally flawed, the consequences are catastrophic — often the entire project is either abandoned or restarted. A technique to reduce the risk of a flawed architecture is to require offerors to submit a preliminary software architecture in their technical proposals.

The risk of a fundamentally flawed architecture is particularly high for unprecedented projects. In those instances, a proven technique for the government to reduce performance risks is to award parallel development contracts with two different vendors who propose dissimilar architectures. Typically, sometime between the preliminary design review and the critical design review, the agency exercises a "*down-select*" decision to proceed with only one of the two contracts. When the decision is made is usually a factor of how apprehensive the government is about the design, the criticality of the software in terms of the agency's mission, and the availability of money to continue funding two contractors. The down-select decision is normally exercised in the form of an option to the contract of the selectee.

In addition to the obvious advantage of not having to select an architecture until it has been analyzed in detail, the use of parallel development contracts offers another benefit to the government. Experience has shown that when contractors recognize that they are in competition for the privilege of retaining the project for its life cycle, the vendors are significantly more conscientious about the quality of their work. The competing vendors are also more likely to assign their best software engineers to the project. Ironically, despite the substantial advantages

to parallel development contacts, they are rarely used within DoD for large software development projects. The most frequent reason for not using parallel development contracts is failure of the agency to budget adequate money. In retrospect, the failure to make the investment in parallel development contracts for the software architecture has often been regretted.

The More Participatory The Government Is In The Design, The More Difficult It Is to Shift Performance Risks to The Contractor

The general rule is that the party that has responsibility for the design of a system is accountable if the design results in the failure of the system. Not surprisingly, this rule causes the government to have a preference for performance specifications. A problem arises where the government uses a performance specification but insists upon a highly participatory role in the design of the system. This situation can place the contractor in a dilemma because the government can thwart a contractor from proceeding by failing to approve a review. Although the government can use reviews to "*hold the design in hostage*," the government has cleverly defined the term "*approval*" to distance itself from sanctioning the design.

Research has not disclosed any cases involving software development contracts where the government's participation in the design has caused the government to assume some of the responsibility for the performance risk. There are other decisions, however, which establish this principle of law. For example, the NASA Board of Contracts Appeal rejected an attempt to hold a contractor completely liable for design flaws that hampered the construction of a scientific facility. Specifically, the NASA Board stated:

> *In our opinion, this theory completely ignores the elaborate Government organizational structure for both design and construction of facilities, the review and approval requirements built into the contract, the pervasive role of the JSC project engineer who also served as the Contracting Officer's representative.*

Similarly, the Armed Services Board of Contracts Appeal was unwilling to hold Boeing responsible for the cost of redesigning a fuel-drainage system for the KC-135A aircraft where the government expressed safety concerns after the critical design review.

> *[T]he Government was anything but passive in monitoring and approving appellants Preliminary Design as it pertained to the drainage. *** The highly structured dialogue between the Government and appellant generated by the Critical Design Review defined the more detailed Part II Development Specification.*

In light of the above precedent, when planning its acquisition strategy, the government should first ascertain if it intends to play a pro-active role in the design before the government agrees to pay a premium ostensibly to place the performance risk on the contractor. For example, the government should first ascertain if it intends to rely heavily on a Federally Funded Research and Development Center (FFRDC) as an assertive systems engineer or if it intends to rely heavily on a pervasive IV&V contractor. In essence, the highly participatory role of a FFRDC or an IV&V may impede the government from placing the performance risks on the contractor. Stated differently, if the government wishes to shift the performance risk to the contractor, the contracting officer should assure that the government's technical representatives are merely reviewers of the contractor's work rather than participating in the design of the software.

Schedule Risks

According to Doctor Fredrick Brooks, "*more software projects have gone awry for lack of calendar time than for all other causes combined.*" Before probing into the reasons why schedules often doom software development projects, it is important to appreciate the interrelationship between cost, performance and schedule risks. With regard to schedule impacting costs, there is a clear correlation between the number of people on a project and their productivity. In essence, there are efficiencies to be gained when a dedicated but small workforce methodically develop software over a lengthy period of time. The schedule can be expedited, to some degree, by adding additional software engineers. The following table of a hypothetical project is indicative of the dependent relationship between cost and schedule:

SCHEDULE (Months)	SOFTWARE ENGINEERS	STAFF MONTHS	COSTS
9	30	270	$4,500,000
12	20	240	$4,500,000
15	14	210	$4,000,000
18	10	180	$3,750,000

Table O-5

The correlation between schedule and performance risks is that, in attempting to meet an unrealistic schedule, contractors often expedite the process in a matter that is harmful to quality. Frequently, the resulting product is too defective to perform as required. That GAO has repeatedly observed that unrealistic schedules increase performance risk:

> *Technical problems result from the need to meet deadlines—programs are often designed and written hastily, and are tested and documented inadequately or not at all. Thus quality is sacrificed to urgency. Documentation — material prepared to explain a computer program — is often deferred until after the program is running and sometimes is never completed. When programs are later modified or converted, the work is usually done by someone other than the originator. If documentation is missing, incomplete, or obsolete, a great deal of the original development work often must be repeated.*

As shown above, the schedule can force quality to be sacrificed for urgency. When quality is sacrificed, often the software is degraded or rendered unusable.

In the past, many DoD software projects succumbed to unrealistic schedule that were generated under the euphemism of being "*success oriented.*" In 1994, the Air Force published an excellent handbook which acknowledged that unrealistic schedules had a debilitating effect on software development projects. The handbook explains why "*success-oriented schedules are seldom successfully achieved.*" The handbook also provides some useful guidance on what it calls "*schedule-plus contracts.*" In essence, these contracts are structured to use award fees or incentive fees to motivate a contractor to be realistic in bidding schedules. Equally as important, "*schedule-plus contracts*" are not as likely to cause the contractor to "*sacrifice quality to urgency.*"

THE EVALUATION CRITERIA SHOULD BE STRUCTURED TO MAXIMIZE THE PROBABILITY OF SELECTING A HIGHLY COMPETENT VENDOR

One expert has observed that "*the competency of the contractor is the single most important ingredient in the recipe for successful contract performance.*" For software development contracts, it is axiomatic that the greater the competence of the software development contractor, the greater the probability that the software development project will be successful. Consequently, it behooves the government to structure the evaluation criteria to maximize the probability of selecting a highly competent vendor.

In his book, The Decline And Fall Of The American Programmer, Edward Yourdon summarizes the startling results of some careful studies which reveal that there can be an enormous variation between the capabilities of software engineers. Equally as surprising, there is no simple means to readily distinguish between the top quartile and the bottom quartile of software engineers:

When a programmer is good,
He is very, very good,
But when he is bad,
He is horrid.

This conclusion was based on the results of a programming exercise given to a group of 12 experienced programmers. Careful records were kept to see how long the programmers took to finish various phases of the programming job, and what results they produced. The outcome was staggering: the best person in the group finished coding and debugging the exercise 28 times faster than the worst person, and the best program was approximately 10 times more efficient (in terms of memory and CPU cycles) that the worst. Equally important was the discovery that the actual performance of the programmers had no significant correlation with years of programming experience or scores on standard programming aptitude tests.

In the same way that there are an enormous variations among the capabilities of software engineers, so too there are enormous variations among the capabilities of software development vendors. Moreover, just as there is no simple way to readily discern which software engineers are in the top quartile, so too there is no simple way to readily discern which software development vendors are in the top quartile. The following discussion is to provide guidance on how the government can distinguish the relative competence of software development vendors.

Software Engineering Institute's Software Capability Evaluations

The SEI's Software Capability Evaluations (SCEs) enable a contracting activity to appraise the maturity level of the offerors. Since ample guidance can be obtained from the SEI on SCEs, this paper will not explain the intricacies of how SCEs are conducted. Suffice it to say that SCEs are expensive and burdensome for both the offerors and the government. As a rule of thumb, a SCE is appropriate where the cost of the software development is expected to exceed ten million dollars or where more than 50,000 lines of code are expected. An exception to the rule might be appropriate where there is a critical need for the software or where human life would be in jeopardy if the software failed.

There are various ways in which a SCE could be considered by the source selection authority. One technique is to make the SCE an affirmative responsibility criteria. For example, the evaluation criteria could state: "*To be eligible for award, an offeror must attain, through*

a Software Capability Evaluation, Maturity Level 3 or higher." Before using such an evaluation criteria, the procuring activity should recognize that a high standard such as Level 3 might be challenged by a protest. Under the Competition In Contracting Act, agencies must *achieve "full and open competition."* The Federal Acquisition Regulation defines this term to mean *that "all responsible sources are permitted to compete."* To have the protest denied, the agency must show that its minimum requirement is for a Level 3 offeror. Since Level 3 represents less than 10% of the vendors in the software development industry, the agency should have a convincing reason why a Level 2 vendor is not a responsible source. Rather than take the risk that the GAO or GSBCA would disagree with the agency's justification to exclude lower level offerors, contracting activities should consider other approaches which are easier to reconcile with the Competition in Contracting Act.

Another technique is to establish the rating which the offeror obtains during the SCE as a separate evaluation criteria. Hence, using the above scenario, a Level 2 offeror is not automatically excluded form consideration. Instead, with regard to that evaluation factor, a Level 2 offeror is placed at a competitive disadvantage in relationship to a Level 3 offeror. To have a significant impact on the source selection decision, the evaluation criteria involving the SCEs should be placed relatively high in the relative order of importance among the other evaluation criteria.

A third possibility is to make the SCE a "*general consideration.*" A general consideration is factor that permeates the other evaluation criteria. The most widely used general consideration is past performance. There is a logical correlation between an SCE and past performance. Each is a reliable indicator of future success. This logical correlation evinces that an SCE is highly suitable to be a general consideration. The disadvantage of using an SCE as a general consideration is that it does not afford the SCE the dignity it deserves in playing a pivotal role in the selection of the awardee. Stated differently, a SCE deserves more visibility since the maturity of a vendor's processes is perhaps the paramount indicator of whether the vendor will be able to successfully develop the software. Moreover, if a source selection authority inadvertently relies directly on the SCE in making a selection rather than factoring the SCE into his assessment of the established evaluation criteria, a disappointed offeror might be able to protest successfully.

As stated in the prior paragraphs, there are various ways in which an agency can consider an SCE in making its source selection decision. The method that the agency intends to use should be identified in Section M of the RFP. The Comptroller General will sustain a protest where the offerors have not been advised of the relative order of importance of the evaluation criteria:

> *It is fundamental that offerors must be advised of the basis upon which their proposals will be evaluated. A solicitation that does not set forth a common basis for evaluating offers, which ensures that all firms are on notice of the factors for award and can compete on an equal basis, is materially defective.*

In summary, it is imperative that Section M of the solicitation clearly state how the agency will use the SCE as part of the evaluation process.

Past Performance

A vendor who has had a consistent history of successfully performing software development efforts is more likely to successfully perform on future software development efforts than a rival vendor who has had a checked history. Unlike a SCE, which only validates that a vendor has the necessary processes, past performance is a strong indicator of whether the vendor has the fortitude to "*make it happen.*"

In recent years, the federal government has become more adamant about relying on past performance in awarding contracts. In a policy letter dated January 11, 1993, the Office of Federal Procurement Policy (OFPP) mandated that past performance be an evaluation factor for all competitively negotiated contracts that were expected to exceed $100,000. The letter also directed the larger agencies to create databases on vendors' past performance. Additionally, in 1994, the OFPP Director obtained pledges from fourteen agencies that they would weigh past performance equally with the other nonmonetary evaluation criteria.

Agencies are afforded considerable discretion in making judgments on past performance. For example, an Air Force procurement of training devices for the F-15 and F-16 aircraft rated an offeror as high risk. The offeror had received more poor performance evaluations than favorable evaluations on previous contracts. The offeror protested to the GAO. In denying the protest, the GAO concluded that the Air Force's risk assessment was reasonable. Additionally, sometimes an offeror seeks to contest a poor rating that it received on another contract. A disappointed offeror has little recourse when it wishes to dispute an unfavorable evaluation. The fact that a disappointed offeror disagrees with an agency's evaluation of its past performance does not invalidate the agency's conclusion.

Previous Experience

Like past performance, previous experience is a credible indicator of the likelihood that an offeror can successfully perform. For instance, if a Level 1 vendor struggled to complete a previous software project, that vendor still might be preferable to other Level 1 offerors for a comparable project merely because the vendor probably learned many lessons which will benefit it on a subsequent project.

Ideally, the government desires a vendor with both previous experience and an excellent record of past performance. The source selection official faces a more difficult decision where the first offeror has had comparable previous experience but also has had a checked record of past performance. A second offeror lacks previous experience in the domain involving the specification but does have a laudatory past performance record. There is no textbook answer as to which offeror should be selected. The government should, however, anticipate such a quandary and draft its evaluation criteria to accurate forewarn offerors which evaluation criteria is more important.

Previous experience is sometimes expressed as a definitive responsibility criteria. Although definitive responsibility criteria are apt to be protested by vendors who are excluded from competing, the GAO will uphold the agency's decision if it is reasonable. For example, the GAO upheld the decision of the Air Force to exclude an offeror who lacked personnel that were experienced in maintaining a land mobile radio system. The following extract from the decision reveals the GAO's willingness to defer to the agency on definitive responsibility criteria involving previous experience:

> *Given the agency's explanation . . . that the equipment involved here is used by those units at the base responsible for human safety and the safeguarding of information relating to national security, we have no basis for objecting to the imposition of the experience requirements. Specifically, we see nothing improper in the agency's taking steps to insure that the personnel maintaining the specialized equipment are particularly well-qualified to do so, and the experience requirements seem to us to be reasonably aimed at achieving this end.*

Although the GAO is deferential to an agency's determination of the requisite experience an offeror should have, the GAO will sustain a protest if the agency's position is unreasonable. For instance, in acquiring software support for a shipboard command and control system, the Navy required "*detailed knowledge of the JOTS II Plus program.*" An offeror who

had a detailed knowledge of an earlier version of the software was excluded from the competition. The GAO sustained the protest of the excluded offeror. Sometimes an agency errs on the side of caution when establishing its minimum experience requirements. The agency later determines that, although a proposal that does not comply fully with the solicitation, the proposal is capable of meeting the agency's requirements. If the agency seeks to award to an offeror who does not meet the minimum experience requirements of the solicitation, the award can be successfully protested. In those circumstances, the correct action for the agency is to reduce the minimum experience requirements by amending the solicitation and then seeking a new BAFO.

Sample Problems

According to Gabig's Premise, *"best brochuremanship — not best value — frequently wins government contracts."* Not surprisingly, many sophisticated vendors use professional proposal writers to respond to RFPs. The net result is that the quality of the contractor's proposal may not be indicative of the quality of the technical staffs that will ultimately perform the work. A useful technique to minimize the impact of Gabig's Premise is to use sample problems. One possibility is to identify as sample problems a few modules that are suitable candidates for rapid prototyping. Another possibility would be to draft the sample problem to address some difficult interfaces that are anticipated during the performance of the contract.

It is not uncommon for a disappointed offeror to protest that its poor score on the sample problems should have been brought to its attention and that it should have given an opportunity to revise its answer. To protect itself again such protests, the agency should draft the solicitation to emphasize that the purpose of the sample problem is to test the offeror's understanding of the problem as well to test the offeror's technical competency. Under those circumstances, as shown by the following quote from a GAO decision, the protest is likely to be denied:

> *It is also apparent that the Air Force wanted to gauge the offerors' independent management and technical abilities and expertise to propose, on their own, solutions to a variety of complex engineering tasks. While the pointing out of deficiencies in the proposed solutions might well have produced improvements in the offerors' approaches, what was to be evaluated here was not how well an offeror could improve the problem areas, but rather how well an offeror could independently size up a problem and come up with a viable, efficient solution.*

Consistent with the theme that sample problems are a test, they should be given letter grades — A,B,C,D, or F. A pass/fail grading scheme is vulnerable to being struck down as being inconsistent with the nature of negotiated procurements.

Avoiding "*Buying-In*"

FAR § 3.501-1 defines buying-in as submitting an offer below anticipated costs, expecting to: (a) increase the contract amount after award (e.g., through unnecessary or excessively priced change orders); or (b) receive follow-on contracts at artificially high prices to recover losses incurred on the buy-in contract. Software development contracts are especially vulnerable to "*buying-in.*" In addition to having a high incidence of changes, the opportunity for follow-on maintenance contracts at artificially high prices is particularly great.

Many vendors are unaware that the FAR does not prohibit "*buy-in.*" Instead, the FAR only admonishes the contracting officer *to "take appropriate action to ensure buying-in losses are not recovered by the contractor."* For a cost-reimbursement contract, the best way to

protect against buy-in is to use a vigorous cost realism analysis. The FAR recognizes that for *"cost-reimbursement contracts the cost proposal should not be controlling, since advance estimates of cost may not be valid indicators of final costs."* Consequently, an agency is granted considerable leeway to reach an independent evaluation of what it will cost the offeror to complete the project.

An excellent example of an agency asserting its prerogative to adjust a cost proposal for cost realism occurred during an Air Force procurement for software development to support the Joint Space Intelligence Center in Cheyenne Mountain. The Air Force made a $29 million cost realism adjustment to a $69.7 million proposal from McDonnell Douglas Electronics Systems Company. A subsequent protest by McDonnell Douglas was denied by the GAO.

If the contract is fixed price, the best way for an agency to protect against a buy-in is to use a best value procurement. A best value procurement provides the agency with greater flexibility to make tradeoff decisions. Moreover, the cases involving best value have upheld an agency's decision to spend considerably higher sums of money to achieve superior technical performance.

For a complex software development project, it is not enough that the contracting officer merely comply with his obligation under the FAR to guard against the contractor being able to recover its losses. Buying-in usually results in an antagonistic relationship between the parties since the contractor typically is looking for a way to *"get well."* Conversely, the government obligated to resist the contractor's attempts to *"get well."* In light of the fact that a congenial relationship is almost indispensable to the successful completion of the project, it is in the government's interest to avoid awarding to an offeror who is buying-in if it is at all possible.

Notwithstanding the techniques mentioned above, often the evaluation criteria greatly constrain a source selection official's flexibility to avoid awarding the contract to an offeror who is buying-in. Although rarely explained to the source selection official, there are two other factors which may be considered despite the factors not being expressed in Section M. The first factor is that an agency may infer that the risk of poor performance increases where a contractor is forced to perform a contract with little or no profit. This observation is consistent with the FAR. FAR § 15.901(b) recognizes that profits *"stimulate efficient contract performance."* Inferentially, a lack of profit suggests that the contractor is not stimulated to perform the contract efficiently. The second factor is to assume that the contractor will be forced to pay its workforce low compensation. The source selection official may assume that the low compensation will impacting the offeror's ability to recruit and retain highly quality software engineers. The anticipated difficulty for the offeror to recruit and retain a high quality workforce is justification to increase the agency's assessment of the risk of nonperformance.

About the Author

Jerome Gabig, Jr. is Of Counsel in the Washington, D.C. office of Venable, Baetjer, Howard & Civiletti where he specializes in government software development contracts. Through a variety of assignments as an Air Force officer, he gained considerable experience in contracting for software development:

1990-92 Deputy Staff Judge Advocate, Electronic Systems Center (Air Force Materiel Command), Hanscom AFB, MA
1990-87 Director of Contract Law, Armament Division (Air Force Systems Command), Eglin AFB, FL
1987-83 Staff Judge Advocate, Air Force Computer Acquisition Center (Air Force Communications Command), Hanscom AFB, MA
1983-80 Director of Telecommunications & Acquisition Law, Air Force Communications Command, Scott AFB, IL

Mr. Gabig is the course director for Federal Publications' program on software development contracts. Additionally, he is the author of Federal Publications' text entitled Government Contracting For Software Development. Mr. Gabig has graduated from West Point (engineering), Harvard University (management & administration), and the University of California (law). He has instructed at George Washington University, the Army JAG School, the Air Force JAG School, the Defense Systems Management College, the DoD Computer Institute, and the Naval Post Graduate Institute. He performed extensive research and analysis to support the Section 800 Panel. Mr. Gabig founded the Information Systems Committee of the American Bar Association's Public Contract Law Section which he currently serves as Vice-Chair. He is a National Contract Management Fellow and recipient of the Delaney Award for 1993. His numerous publications have appeared in The Harvard Journal of Law & Public Policy, The Public Contract Law Journal, The National Contract Management Journal, Program Manager, Contract Management, and The Computer Lawyer.

Jerome S. Gabig, Jr.
Venable, Baetjer, Howard & Civiletti
Suite 1000
1201 New York Avenue, N.W.
Washington, D.C. 20005
Voice: 202-962-4953
Fax: 202-962-8300
Internet: JGabig@Venable.com

CHAPTER 15
Addendum B

Training — Your Competitive Edge in the '90s

Eileen Steets Quann,
President, Fastrak Training, Inc.

How many of you had a car in the shop in the last year — yours, your spouse's, or a son's or daughter's? Could I see a show of hands please? How many of you thought the mechanic who worked on that car was overtrained? Cars have changed a lot in the last 10 years and mechanics are constantly going to classes to certify themselves on new technology. Last week, when I drove my husband to pick up his car from the dealer's service department, I asked the manager how much training their mechanics receive. About two months in the first two years and then about two weeks a year after that to stay current on the new models. That's about 8-10% of their time in the first two years in formal training classes, plus OJT and 4% annually after that to maintain their skills. Is your auto mechanic getting more training than your software engineer?

If you've been to see a doctor in the last year, did you think that the doctor was overqualified to deal with your problem? Probably not. We expect doctors to stay current on the latest medical findings. We expect them to continually read medical journals, and attend medical conferences. We expect them to be knowledgeable about the latest cure for whatever ails us. We expect the people who service our needs to stay current in their field.

The 1980s saw an explosion of technology unprecedented in the history of mankind. Computers became better, faster and cheaper. The growth in raw computing power was exponential. Dr. Fred Brooks, in his well-know book *"The Mythical Man-Month"* was describing system development in the 1960s, and stated that memory on a Model 165 computer rented for about $12 per kilobyte per month. Last week I received a flyer which advertised a 540 MB hard drive for under $400. To rent that much memory for a single year back then would have been over $77 million. Better, faster, cheaper, no question about it. Hardware has become a disposable commodity. New generations of machines are available every 18 months, and computers become virtually obsolete in less than 5 years.

New generations of hardware are available every 18 months, and computers become virtually obsolete in less that 5 years ... Software has changed equally as much.

Software has changed equally as much. Let me make an analogy. How many of you have teenage children? Can I see a show of hands? Do they ever need money? I have two teenage boys and I know for a fact that they do. Suppose I make a deal with my 17 year old son Sean to build a dog house in our back yard. He and I agree on a price for his effort and we sit down at our kitchen table and design it. He picks up the wood at our local lumber yard, and builds it over the weekend, with some oversight and positive words of encouragement from me. Let's assume he builds a beautiful dog house, the best in the neighborhood.

Now it's time to build your retirement home. How many folks here would be willing to have Sean build it? By the way, if anyone is interested, please see me after this session — I'm his agent. However, I suspect that I won't find any takers. The fact that he can build that doghouse does not qualify him to build your home. You have no reason to believe that he knows anything about reading a blueprint, plumbing, wiring, roofing, flooring, insulation, inspection requirements, or any of the other things you need to know to build a house.

Let's assume that you have found the builder for your retirement home. An RFP is published to build a skyscraper in downtown Salt Lake City. What are the odds that your home builder would bid on it? Not likely. It's a pretty different business. Walls of glass, steel beams and concrete, elevators — not the stuff our homes are typically made of.

While it's pretty obvious that the skills required to build a dog house, a single family home and a skyscraper are quite different, I would suggest that the differences for software are just as real, but not as obvious. We would never describe the primary differences between these structures as size — 2 square feet, 2000 square feet, 2 million square feet, but we often describe software in terms of its size — 200 lines of code, 20,000 or 2 million lines of code. And in software, we often mistakenly assume that anyone who can write 200 lines of code can write 2,000 or 20,000 or 2 million, given enough time.

What makes these structures different? Tools, for one thing. To build a skyscraper, I need a crane. Suppose I give Sean a crane. Now can he build a skyscraper? Of course not, yet that is exactly what we often do in software. We expect tools alone to solve the problem. Don't misunderstand, we need those tools, it's just that tools alone are not the answer. Another difference is that it takes a lot of people to build a skyscraper. Suppose I bring Sean's entire senior class to Salt Lake City and give them all cranes. Now can they build a skyscraper? No way. We do that in software too. We staff up with people who have built doghouses, give them cranes and expect them to build skyscrapers. And then we get angry when they fail. The real problem is that the process is different.

A huge gap exists today between our formal education and what we need to know to stay competitive, and that gap is widening.

In today's DoD environment, we are finding that our requirements more often that not are for skyscrapers, yet our staff, experienced in building doghouses, and sometimes single family homes, are not given the training they need to produce those skyscrapers. If I want these people to build skyscrapers, they need to be trained in new processes, new standards and procedures, new methods, and new tools. If I am their employer, I would expect that it would be my responsibility to see that they got the training that they needed. Why is it so hard to accept that in the software industry?

Today's educational system was custom made to fit the industrial age, developed for a society in which it made sense to treat everyone the same. They needed to educate factory workers and the managers of those workers. The mass production oriented society relied on uniformity to produce results. Students learned almost everything they needed to know in school and education, for the most part, ended at graduation. Unfortunately, that formula no longer equates to success in business. Many of us have been educated by industrial age standards and then thrown into the information age to compete. A huge gap exists today between our formal education and what we need to know to stay competitive, and that gap is widening.

Our management structure is also a product of the industrial age. Uniformity, control and centralization in the factory were the ideals of industrial society. Responsibility rested with the bosses, who made the decisions that were implemented by the workers who followed orders. Training was limited to teaching new employees the specific job skills for their positions. That paradigm worked well in the industrial age, but it falls apart when we are competing in a global marketplace for which the only constant is constant change and survival demands continually

improving our products, our services and our ability to respond quickly. The notion that workers are supposed to be already trained may have made sense 50 years ago — it simply doesn't today.

Technology concepts are ushered in faster than our ability to master, or even understand them, unless we are perpetually in a learning mode. We are perceived as the leaders of technology, yet all too often we are woefully ignorant of technological progress. I attended a one day technical exhibit two weeks ago. The themes were multimedia, client server networks and information engineering. Two thousand people attended. Ten years ago, that conference could have been held in an elevator. We cannot allow our software engineers to become technologically obsolete.

Where will the software of the future come from? In this month's IEEE Spectrum magazine, there is an article about the growth of the software export industry in India. In 1985, India exported $24 million worth of software. They expect to export $350 million this year, with projections of $1 billion by the year 2000. About 60% of India's exported software goes to clients in the United States.

Recognizing the export potential for this industry, India's largest software houses invest heavily in skills training and continuing education programs to keep abreast of developing technology. Some Indian companies spend as much as 5% of their annual revenues on training, many times what we currently spend in the United States on our software resources. The Wall Street Journal stated that Japanese and European firms spend 4 to 6% of their operating budget on employee education, while US firms spend only about 1.5%. Even American companies noted for their training programs, Motorola, IBM and others, are way below the percentage for India.

What about the quality of their software? Many of these Indian companies are working toward certification under ISO 9001, which corresponds roughly to the Level 3 on the Software Engineering Institute's Capability Maturity Model. And they're looking beyond that standard to the Malcolm Baldrige National Quality requirements. Other developing nations are also moving into software as a viable, profitable export. These will be our competitors in the global marketplace, and unless our attitude towards continuous training and learning changes, we may not be much competition for them.

Every person in this room will be technologically obsolete by the year 2000, if we don't do something about it. So what can we do? Peter Drucker, the management guru, in an article in INC magazine stated simply, "*Assume that it is the responsibility of the organization to train.*" With the continuous changes in technology, our software engineering work force cannot stay competent and current without support from the organization.

I would like to add, "*Assume that it is the responsibility of the individual to learn.*" We no longer live in an age where education can stop when we graduate from college. The explosion in technology is both exciting and frightening. We can envision a future with wonderful new ideas, yet we may be afraid that we will become obsolete. And perhaps some of us will. But that is our choice — not our mandate.

The most successful people in the next 10 to 20 years may not necessarily be the smartest ones today, or the ones who know the most right now. The technology they know now will be obsolete by then. Success in the information age will be defined by your ability to learn. Knowledge will become obsolete so quickly that the only survivors will be the life long learners.

In the world in which change is the constant, the critical skills we need are the ability to think, to analyze and to learn...the best learners are becoming the most valuable people in our organizations.

In the world in which change is the constant, the critical skills we need are the ability to think, to analyze and to learn. These are not necessarily skills we were born with, and unfortunately, in many cases, they are not skills that we have been taught. They are however skills that we must acquire because the best learners are becoming the most valuable people in our organizations. And the responsibility for learning begins with the individual. Look to the people to the right and left of you that you have just met. One of you three is in a learning mode right now and may already be a life long learner, one of you could become one, and one of you will be technologically obsolete by the year 2000. Which one are you?

What is a life long learner and how do you become one? First of all, a life long learner reads. I went to the newsstand near my office three weeks ago and counted 47 computer related magazines. How many do you subscribe to? How many do you read? How many business newspapers and magazines do you read? This is a software conference. How many books have you read on software engineering this year? Ever? What were the last five books that you read that related in any way to your professional growth? When was the last time you were in a library?

Life long learners don't stop their formal education when they graduate from college. When was the last time you were in a classroom to improve your professional skills? Interestingly, very few people take advantage of educational opportunities offered to them by their employers. The March 1994 issue of Benefits and Compensation Solutions reported that less than 7% of employees take advantage of educational assistance programs at their place of work.

A life long learner never thinks that he or she is too smart to learn. How many of you have small children? A child is born wanting to learn. Among their first words is certainly the question "*Why*?" They must ask that question and others a hundred times a day. Perhaps we all need to ask why more often. Every time we are afraid to ask questions, afraid it will make us look stupid, we lose an opportunity to learn. The life long learner has not lost the ability to ask why, or what, or how. They have confidence in their ability to learn because they have done it over and over again. They know that the learning stretch — when we are pushed to perform beyond what we already know — can be painful, frightening for fear of failure, humbling for we have to admit that we don't know, but also exciting, exhilarating and rewarding when we succeed. By the way, if you've been in a job for two or three years and feel like you're not learning, you're probably due for a stretch assignment.

Even while we recognize that it is the responsibility of the individual to learn, individual learning alone cannot save an organization. To be competitive as an organization, we must institutionalize the learning process. Organizations must provide the climate and opportunity for learning. We must foster the learning organization.

In a learning organization, emphasis is on creative thinking. People are encouraged to ask the right questions, rather than learn the right answers to predictable questions. Mistakes are recognized as an essential part of learning. I enjoyed the analogy of Robert Rosen in his book, The Healthy Company in which he says, "*Mistakes are as natural a part of learning as sore muscles are of athletic training.*" With spring finally arriving, many of us will start up new exercise programs and rediscover muscles we forgot we had. If we quit with the first sore muscle, we'll never get in shape. Likewise, if we quit after our first mistake, we'll never learn.

A learning organization provides training that is consistent with the goals and business plans of the organization and follows up to make sure that the goals are being met. Every employee has his or her own personal training plan, evaluated regularly, that is consistent with the goals of the organization. The SEI Capability Maturity Model talks about software process improvement. An organization's ability to improve is in direct proportion to its ability to learn. An organization that cannot learn cannot improve.

I believe that for most software organizations, moving from Level 1 to Level 2 is the most difficult step because it demands a culture change. Individuals must commit to learning and organizations must commit to training, or there won't be any change. If you can get over those two hurdles, movement up to the higher levels is much easier. You cannot institutionalize process improvement without institutionalizing individual learning and organizational training.

We do training in Level 2 Key Process Areas. I have discovered that I can now tell within a couple of days of working with an organization, whether it will ever become a Level 2. You just have to listen to the people to find out. Let me give you two examples. In one organization, where we give classes in Project Management, Requirements Management, QA, CM and a variety of other courses, the classes are always filled by the technical staff. ***On the evaluation forms they frequently say, "My boss needs to hear this." Yet when we have scheduled classes for the managers, short overviews to tell them what their employees were learning, not a single manager attended. Rarely do students get to apply what they have learned in class to their job. The employees are discouraged.***

APPENDIX O Additional Volume 1 Addenda

Assume that it is the responsibility of the organization to train ... Assume that it is the responsibility of the individual to learn.

In another organization, at the start of every new class, the director comes in, tells everyone why they are there, outlines for them the goals of the organization and ties the class objectives to the organization's goals. He then encourages them to learn everything they can and to come back ready to apply what they have learned on the job. The students are enthusiastic, many of them clearly in a learning stretch. The contrast between the two groups is striking. One of them will succeed. ***An organization's ability to improve is directly proportional to its leaders' commitment to create a corporate culture that not only invites but actually demands continuous learning.***

The learning organization must be led by a learning leader—someone who is actively committed to his or her own learning. Learning leaders attend training with or before their people. They define the goals for their organization, and then commit the resources to make it happen. They don't set arbitrary dates for "*Becoming a Level 2*" without understanding what that means, without putting in place the mechanisms and committing the resources that can make it possible.

In the leading edge companies, continual learning is recognized as an essential ingredient to competitiveness and survival. They understand that the only competitive advantage an organization in the information age will have is its ability to learn faster than its competitors. They make the commitment to spend the money required for training, but only after they make sure that the training is linked to their business strategy. They don't waste training dollars.

Unfortunately, I think the DoD supported software industry has fallen very far behind in the learning process. We put off training with every budget cut and think somehow that we have saved money. We expect contractors to always provide trained workers and don't acknowledge the continuous nature of learning. Much of the money we do spend on training is wasted because it is not directed at the goals of the organization. Our model for training must be dramatically overhauled. First, we must link training dollars to business strategy. Training dollars must be budgeted first, not last. Our greatest resource is our people and the maintenance of that resource requires continuous investment in training just to stay current. For every dollar spent on hardware or software, another dollar must now be budgeted for training. And because of downsizing and reduced budgets, it may be necessary to get that dollar from the hardware and software budgets.

As an individual, you must make the conscious decision to become a life long learner. The bad news is that even if you know everything there is to know technically in your field today, you will be obsolete in five years if you don't learn anything new. The good news is that if you commit to life long learning today, you can be the technological leader in five years.

If you are a manager, your success depends upon your ability to inspire the best in your employees, to expand their competence and capacity and to create the right conditions so that they can learn. It is your responsibility to ensure that training is adequate, appropriate, directed at the goals of the organization and not wasted.

If you are the leader of an organization, you must recognize that the competitive edge in the 90's will go to the organizations that train their people. You must define the goals for your organization, communicate those goals to your people and allocate the resources to support those goals. Your organization can lead or it can follow, but in the fee for service environment, being competitive may mean the difference between surviving and thriving or going out of business. As you drive onto the information highway of the future, check your rear view mirror. The competition from around the world is gaining on you.

Our success, indeed our survival in this rapidly changing world, will depend upon our ability to respond to change. To do that, we must be able to think, to analyze and most importantly to learn. Training is your best competitive strategy in the 90s. Don't pass up that opportunity.

About the Author

Eileen Steets Quann is the founder and president of Fastrak Training Inc. Incorporated in 1987, Fastrak is a recognized resource for training and mentoring in software engineering process, object-oriented methods, and the Ada programming language within DoD, NASA, the FAA, and many of the major aerospace and defense contractors. In both 1993 and 1994, *Washington Technology* identified Fastrak as one of the 50 fastest growing high technology companies in the Washington DC area. In 1993, INC. Magazine recognized Fastrak as one of the 500 fastest growing privately owned companies in America.

Ms. Quann has over 20 years of experience in the software industry and is an instructor on management, software process improvement, and Ada topics. Active in the Ada community, Ms. Quann was the 1989 Program Chair and 1990 Conference Chair for the Washington Ada Symposium (WAdaS). She represented industry training on the education panel for the DISA-sponsored Ada Dual Use Workshop in 1993. Ms. Quann was a distinguished reviewer for the *Ada Quality and Style: Guidelines for the Professional Programmer (SPC-91061-CMC)*. She is a member of the AFCEA International Steering Committee on Women and Minorities. Ms. Quann has published articles and papers on management, software engineering, and training issues.

Eileen Steets Quann
Fastrak Training Inc.
9175 Guilford Road, Suite 300
Columbia, Maryland 21046
(301) 924-0050.

CHAPTER 15
Addendum C

Lessons-Learned from BSY-2's Trenches

Robert F. Sullivan Jr.

INTRODUCTION

You've read the headlines. You've watched "*60 minutes.*" Large government software development programs ending up in huge cost overruns, way over schedule, and no clear indication of either the quality or the functionality that the government is receiving for the enormous price tag. Everyone wonders what went wrong. So when a subset of the seven sensor AN/BSY-2 Combat System (the single sensor AN/BQG-5) was deployed on the USS Augusta in 1994, and initialized properly the first time it was powered up on the submarine, and generally performed outstanding in its first sea trials, you have to start asking what went right. The following areas are arguably the most succinct summary of what went right. However, there are a great many lessons learned. These 4 key areas are not a *silver bullet.* The areas are:

- **Emotional Mission Statement**. There was a strong, emotional purpose attached to the project and everyone **involved bought into it;**
- **Process improvement culture**. Over the course of the project, there were vast, and essential process improvements;
- **Strong configuration management**. Reliable, effective, automated baseline control and problem reporting;
- **Use of Ada**. Strong typing, information hiding, and other factors lead to a solid, reliable product.

To get an appreciation of BSY-2 and the magnitude of its success, you have to appreciate the immense size of it. It is over 3 million lines of tactical code, with several million lines of support software. The tactical code is broken up into over 100 CSCIs. Most CSCIs were assigned to an individual team. Several of the CSCIs are so large and complex, no one person knows all the details the CSCI. Needless to say, no one person knows all the details of the entire system.

For sure, there is some breakthrough technology involved. But the challenge of creating that technology is dwarfed in comparison with the challenge of creating the countless pieces of "*trivial*" functionality that must all play together. Taken individually, most of the pieces are relatively easily understood. But that's the danger and the ultimate challenge. Few of the pieces can be taken individually. They all must play together with other pieces. The inter-team interaction required to pull this off is unlike any other program. Its difficult enough to balance the dynamics within one team. Imagine trying to balance the dynamics of over 100 interacting teams? Imagine if the teams are spread over several geographic locations? Imagine if not all the people on a given team report to the same functional management? Imagine if the

support groups like configuration management, test support software, simulation/stimulation software, and CSCI integration report to different management than the majority of organizations they serve? Sounds like a formula for disaster? Well it could have been. The successful techniques can be broken up into management of 3 categories: people, process, and technology.

PEOPLE

There is little breakthrough technology involved, so we aren't talking about finding a few superstars, giving them everything they want, and watching them do good things. We are talking about finding scores of solid, team players. This challenge is more difficult than any technology problems faced. You need to find good people, properly reward and motivate them, ensure a dynamic organization that responds well to change, and balance inter-team dynamics.

Motivation

Create an emotional mission statement that achieves buy-in from the entire team and their families.

This project required a tremendous, sustained commitment for many years. With anything this size, there will be built in bureaucracy, to the point where it's hard to do the right thing; much easier to conform to inefficient rules and regulations. But conformation leads to stagnation and over the 7 years, there are bound to be profoundly better ways of doing business.

How do you get several hundred people motivated to sustain long hours away from their families and friends, work crazy hours and keep bucking the system to make it better, no matter how hard it is? Just like you'll see in many "*peak performance*" books, the trick to superhuman effort, be it individual or team (although much more vital to a team), is commitment to an important mission. Everyone needs to feel important and adequate. What better way than to contribute to something vital to our nation's security? The Seawolf computer system!!! The Soviet Union had caught us sleeping, had advanced their submarine technology beyond ours and we needed to catch up in a hurry.

What a motivator! Everyone knew the importance of their contribution. Everyone wanted nothing but the best in quality. No one would settle for anything less. The customer didn't need to worry about getting shortchanged. There was a passion for quality, a passion to build the best that America had to offer, the best in the world. One bad apple couldn't spoil the show, a dozen others would expose and correct the problem and preserve the integrity of the project. A tremendous emotional motivator produced superhuman results.

Any motivator such as this must be emotional. It has to touch the families of the engineers, not just the engineers. There is a sacrifice on everyone's part, and everyone must know what they are getting for that sacrifice. Peace of mind, the Seawolf sub will protect us from the bad guys. It will make the world safe for democracy and freedom. And we would prosper as a company with it. We have found that financial rewards are a side effect of a good emotional motivator. It is much more effective than a financial motivator, alone.

Print up T-shirts, get coffee mugs and hats with exciting, emotional logos and pictures. Outfit the family with the paraphernalia. Have family events and tours of the facilities. One of the most difficult aspects of this job was the security. We were sequestered off in a closed room. No family members allowed. It was 6 years into the program before I thought to bring home the office seating diagram. My wife was elated because she could relate better to my world. A simple little picture gave her a whole new understanding and appreciation for my work. And it increased her buy-in.

Watch the results. If the family buys into the mission statement, you've got it made. Tremendous things will happen. If the family doesn't buy in, you've got an uphill battle.

So how did we keep going when the Soviet Union collapsed and removed the threat, thus removing the main point of the mission? It's never been the same. But that doesn't mean it demoralized everyone. Sure, there were a few rough months. Sure, many good people left. Sure, some people retired on the job. Then throw on top of that the congressional budget cuts, eliminating production of several systems per year to only 2 systems overall!!! Not only is the mission removed, but the economic stability is damaged. Family security is at risk. How can you recover from something like this?

We did the only thing we could. We found another mission. We would strive to make the Seawolf technology be the foundation for the New Attack Submarine (NAS). The NAS is vitally needed for trouble areas like the Persian Gulf, where our current subs are too big or not sensitive enough. It wasn't a great substitute, but it was the only option available. It revitalized many and gave the promise of economic stability and growth.

We were back on track. Until the NAS funding got delayed, pending the success of the Seawolf. Now the pressure was on. Our economic stability was tied to the completion of this program, and even then, there was no guarantee. There might be a delay between the end of the Seawolf and any NAS funding. The family security was damaged. The mission was weakened. Luckily, there was enough momentum and enough process safeguards that the project will complete no matter what. Had this happened a few years ago, we would have been another disaster on 60 minutes.

In summary, if you don't have an emotional mission, get one.

Change Is Good

Create a culture where constant change is encouraged, embraced, and rewarded.

People tend to resist change. It makes them uncomfortable. It's harder because you have to learn a new way of doing business. It worked this way before, why can't we stick with it?

With a program this size, most of the old rules are history. With the global competition, most of the old rules are history. One management expert describes the only true, sustainable, competitive edge, as the ability to learn faster than the competition.[1] Learning new things implies changing the way you do business. On a program that stretches over 7 years, there are going to be vast changes in industrial best practices. If you don't prepare for change, it will hurt. Either you won't change and become obsolete, or the changes will not be embraced properly and you won't get the payoff.

How do you prepare for change? Well, it's got to start with upper management, and end with everyone involved. It's got to be embraced by everyone, including (and especially) the customer. The typical DoD project gets its share of standards imposed on it (including the contract itself). However, these standards are made to be tailored to industry and the contractor's best practices. It is in everyone's best interest to do this. Blindly adhering to a standard is no excuse for thinking, adapting, and changing. You and your customer should approach each standard as if its first requirement read, "*You shall continuously refine, document, and improve your interpretation and implementation of this standard*". Associating change with a "*shall*" makes it a requirement that needs to be met.

What happens too often is the customer and the contractor's standards enforcers, commonly known as gatekeepers or speed bumps, enforce the standards for the sake of enforcing the standards. There is no room for interpretation or change. There is often no written interpretation either. This mindset is deadly. It leads to demoralization and resentment. It stifles

creativity. It creates bottlenecks with no apparent value added. We'll talk more about the process specific aspects of this later but let's concentrate on the people aspects now.

You've heard of manufacturing success stories where direct discussions with people doing the work, senior management, and the customer produced miracles. Software development is no different. If change and improvement are part of the contract, part of the requirements, and embraced by everyone involved (especially the people in the trenches), people will create better solutions than any standard could have dictated or even hoped for. There is no standard process or guideline that should be imposed blindly on any software project, let alone imposed across the board on a huge project. Once a standard is in place, naturally it should be documented and enforced to the extent practical. But if the culture is not in place to constantly refine and improve standards and their implementation (yielding more efficiency without a compromise in quality), you will waste big bucks, and build frustration and resentment.

With the refinement must come the rewards. Now there are several lessons here. First, the struggle to implement change is often made unnecessarily difficult due to the bureaucracy, or inertia built up. Without the culture described above, you often end up with these brick walls on both sides — contractor and customer are both afraid to change. The walls are protected by the standard's enforcers (gatekeepers). Interpretation and implementation is often dictated or heavily influenced by the gatekeepers. The engineers implementing the standard are often unequally represented, both internally and with discussions with the customer. The fear is that the engineers will take the shortest path, thus compromising quality and product integrity. But in reality, rigid adherence to any standard cannot, by itself, ensure a quality product. Yet for any standard or process to be effective, the people doing the work need to buy into the spirit of the standard.

This apparent deadlock can be broken by tapping into the natural tendency of many (some argue all) engineers — they like to complain. They are constantly striving to avoid the hard stuff. But that is the irony. That is what they are trained to do, to create products and services that make tasks convenient, easy, or efficient. Without problems to complain about, they have no mission (there's that word again). Encourage and develop their complaining skills, and demand that they do something about it (active complaining). This is a reward in itself. By having more control of their own destiny, just about anyone is bound to have an increase in morale.

In this new style organization, the gatekeepers (for both contractor and customer) are still a vital piece of the puzzle. After all, a standard or process is nothing if it is not followed. The typical gatekeeper is very knowledgeable in the vast myriad of standards. As a change is proposed, the new role of the gatekeeper should be to ensure continued compliance with the contract (including standards imposed by the contract). Embrace the change as something that will make the standard, and consequently the product, better, not as something that threatens it. When a change causes a requirement (or interpretation of a requirement) in the contract to change, they should participate in the modification of the contract.

A very effective vehicle to accomplish this is a "*Memorandum of Agreement*" or *a "Memorandum of Understanding.*" These are nothing more than a change to the contract or an interpretation or clarification of the contract. However, it is drafted with the help of the people implementing it. Since it is drafted by the people charged with implementing it, by default it carries a much higher probability of success than one drafted by some authority. This is not meant to be a dissertation on how to coddle an engineer. Complaining must be measured against productive change. Changes must be weighed against programmatic (cost and schedule), contractual, and technical objectives. But there is often tremendous room for flexibility within these constraints, especially over a long development cycle. With this flexibility will come improved morale.

Now all pieces described here are vital to a creating successful culture of change. The people implementing a standard must be equally represented on all decisions relating to that standard. The people enforcing the standard must know and embrace their role. The customer must encourage change by the contractor. And management must reward change when it pays off. One final note relates to an observation and suggestion on an ideal organization to implement

change. Most organizations, once they establish a process, establish some form of process improvement team. This can range from informal to formal. In either case, there are often strong individual advocates for change that get involved one way or another. But even the most staunch advocates for change are susceptible to building walls and impeding change. Advocates for change often fight passionately about a few key issues that are close to their heart, often after they have spent many frustrating hours struggling with the current system. They typically can not sustain the energy necessary to constantly improve and question other aspects of the system.

To solve this, I suggest an organization where the improvement team is constantly changing also. This can be "*chaired*" by the same person but the people doing the leg work need to have fresh legs. By making the team's accomplishments recognized, involvement on this team becomes a reward, an opportunity. You will get plenty of fresh legs asking to get a chance to play. The veterans are still required to make sure the integrity of the product is maintained. But by constantly getting fresh legs, a fresh look at existing problems, you foster more innovation. And the competition can instill renewed vigor in the veterans.

In summary, change is not only good, it is required.

Diversity

You need to motivate as many people as possible, to be as productive and happy as possible.

With any large program, you will have a melting pot of people. From self starters, to superstars, to lazy but effective (if properly motivated) people, you'll see them all. The trick is to get as many people as possible, as productive and happy as possible. This involves quite the juggling act when there are scores of engineers, each with their own aspirations and needs. The corporate ladder becomes harder and harder to climb. Much of the management and senior technical people are entrenched in their positions, and necessarily so for consistency and continuity to the program. After all, we are talking about many years of development.

This is an eternity for a project. What's in it for the average engineer? Where's the career path, the upward mobility? Sure, there will be some attrition in the management and senior technical chain but there are dozens waiting to fill those shoes. The key becomes making each contributor feel like a vital piece in the big picture. Software development is a team sport. On a large program, it becomes a multiple team sport. You can't rely on a few superstars to carry the whole load. Upper management needs to balance the importance of each team. Middle management or the team leader needs to balance the importance of individuals on each team. As a project matures, different teams play different roles. At one time in the program, a certain team may be facing the number 1 high risk issue on the program. This team will get more than its share of attention. If the team performs well, it will get some rewards. The balance of criticality, performance, and rewards, as perceived by other teams, is the most important factor in keeping a diverse project productive.

This balance requires a positive environment. On a project where it is so difficult and long, it is easy to get caught up in the practice of catching people doing something wrong. After all, we all need to look for problems to solve, we all need to constantly improve to stay competitive. But the masses will thrive in a culture that catches people doing something right, and rewards those positive actions. The typical rewards just don't cut it on a large program. Most organizations will have some form of management awards, some will also have peer awards. Certainly, these and salary actions are the most tangible rewards. But this is only the tip of the iceberg. The rewards must be as diverse as the people involved. Each person has attached their own emotional reasons for their commitment to the mission. Understand that reason, each person's motivation, and you have the key to the proper rewards.

For example, some people are motivated by a purely technical challenge. This person would be thrilled to receive a gift certificate to a local computer bookstore, or a personal Internet account, or attendance to an industry conference. Others are motivated by family security. What about a family oriented reward? Like a dinner certificate for the entire family? The late hours wouldn't hurt so bad, the spouse and kids would think the company really appreciated the whole family's commitment. Another example is someone with many leisure activities. Some unexpected paid time off after a few long months would make it easier to work the overtime when it is called for again (and you know it will be called for again, and again, and again). The rewards don't need to cost money to be effective. A visit by someone from upper management to a team meeting before, during, or after a big deadline to say thanks for the effort goes a long, long way towards instilling a feeling of appreciation.

Earlier we talked about perception. This is nine-tenths of the law. The perception of balance is more important than the reality of balance. Not all jobs are glorious. Not all teams get the exciting work. Not all teams are equal (nor should they be). But all jobs and teams are vital. All teams must feel vital. It may take some creative advertising to increase the perceived criticality of some tasks, and the team that performs those tasks. Equally important, if a team doesn't have the reputation for performance yet gets recognized for an achievement, resentment will grow. And obviously no one team should get all the glorious work. Balance the perception between teams and you are half way there.

If you don't achieve this type of balance, you will fail in other inter-team aspects. All teams will need to interrelate to other teams in some way. The products produced by one team may need to be used by another team (e.g., a document produced by the Systems Engineering Team needs to be used as the requirements for building the product by the Software Engineering Team). Personnel may need to be reallocated from one team to another. If the teams aren't balanced, you will see negative side affects in the team members, and in the team leaders (e.g., middle management). This will produce an impossible situation to deal with for upper management. Now we aren't saying make everything equal. Balance does not imply equality. Some jobs are more difficult and will need better players. Some jobs are less difficult and can get by with lesser skilled players. The point is the differences in skill level needs to be accepted, and respected, not flaunted. Below are some examples.

One of the main reasons for poor proliferation of reusable software components is the "*not invented here*" or NIH syndrome. Inter-team conflicts reinforce NIH. We have seen reuse fail and we have seen it work. In all cases, it can be largely attributed to poor or good team dynamics. In one of the failures, the people doing the work did not have a strong reputation for technical abilities. The result was the other teams had an inherent resistance to adopting anything produced by this team. Contrary, in one of the best success stories, some of the best technical people of two teams were pooled together to create a reusable application architecture. Because the right people were chosen and their abilities were respected (not flaunted), adoption of the architecture was not questioned.

On any project, there will be changes in the needs of (people) resources for all teams. On a huge project, any given person will likely work on several different teams over the life of the project. If there are inter-team conflicts, it will be reflected in the middle management. They will start to protect "*their*" best players to keep their team strong, rather than to let the best person fill a particular need on another team. The result is the whole project suffers. You can work around weak links in the system but eventually these weak links will reflect on the entire project. You've got to be able to "*repair*" or help the weaker links by improving that team. Sometimes this demands drafting someone from another team. If there is any conflict between teams, human nature will often keep management from sacrificing on his or her own team rather than doing the right thing for the good of the entire project.

Ironically, when the manager has been willing to sacrifice a key player for the good of the entire project, that player (and most others on the team) feel deeply committed to that manager. They tend to be willing to go out on loan, as long as they can stay functionally assigned to that manager. So the unselfish manager gets the best of both worlds he or she helps the program

and also earns tremendous loyalty and respect from the players.

Another classic example is the typical "*advisor*" organization, or staff engineers. This is supposed to be the group of experts and hot-shots. Their charter is to provide advice and guidance, and to set policy and direction. Think about the resentment this organization will face if the people in this group are not perceived as experts? Or if the people in this group are arrogant, or condescending. Who would go to them for help and advice? This type of group is perceived as a service group. As with any service group, the customer (in this case, the other teams) must come first. Service must be with a smile. All actions by this group must show tangible benefits to the teams they serve. What you are trying to do is get other teams to learn from this group. Respect, integrity, and benefits are the key ingredients. Get your service teams to exhibit these characteristics and watch the results.

In summary, balance the perception (of criticality and rewards) between teams, then balance the perception within each team.

Software Architects

Like a building, a software system needs a solid architecture.

In analyzing an individual team, or more importantly, a collection of tightly coupled teams, it is helpful to use Dr. Covey's jungle warfare model.[2] There are the people wielding the machetes (coders), people sharpening the machetes and motivating the wielders (managers), and the lookout up in the tallest tree directing the energy of the machete wielders, guiding the path, making sure they are in the correct jungle (software architect). To this model, we would like to add the policy makers, the ones that put the team in the jungle in the first place with some objective (systems engineers). Without a balance in all parts, a team will lose effectiveness. If one team within the entire project loses effectiveness, other teams suffer and need to compensate.

Our experience has been that the most difficult role to fill is the lookout, the team leader, what we call a software architect. A software architect is a good communicator, an effective system engineer, an interface expert, and a proficient software designer. Often the ideal software architect can pound out excellent code and a typical reaction is to keep that person producing. This is a mistake. Many people can develop into proficient or adequate coders. Not everyone can become a proficient designer. Fewer still can become interface experts. Even less can understand requirements, the big picture. Rare is the individual with all the other attributes plus the ability to communicate.

These software architects must be carefully detected, and removed from a heavy production role. They are much more effective in this new role. On a large project, software architects are needed to provide the glue to hold the entire system together, especially in the early phases. It is difficult to document and articulate precisely a software architecture. It is more of a concept than a collection of rules. Until the foundation of the system is matured, the software architects must provide guidance and direction to ensure that all the pieces will fit together.

This is where interface definition and communication skills are most vital. Interfaces are the biggest (and sometimes the only) inter-team communication method. The combination of the CSCI architecture and the interface definition becomes the architect's leverage and communication vehicle. In designing interfaces, data format and content is important, but message sequencing in all phases of a CSCI's life are more tightly coupled to a CSCI's architecture. The sequence of messages must take into account initialization, reconfiguration, failures, and restoration. The CSCI's tasking interaction model, input and output mechanisms and sequencing, and major state sequencing (e.g., initialization, reconfiguration, etc.) all make up its architecture. Understanding the true requirements is essential to this task. As you can see, this is a rare individual.

Software architects are not always easy to spot, although there are trends. There seems to be an inbred attitude to be unselfish, to sense the greater need, to work on what's really important, not what's urgent. They are compelled to climb the tree to get a look (at the big picture), rather than to attack every clump of brush (code) they see. However, since they can produce, if pressured, they will produce, and often with star or superstar results.

Here is the danger. It is very difficult to comprehend that your best coder, who also happens to be your best (and quite possibly your only) architect, should be taken off coding and put on requirements analysis, interface design, software tasking interaction, and design guidelines. Why can't he or she just review other peoples design or code to make sure it is close enough, while he or she keeps swinging the machete? Why have the lookout direct the energy of the machetes before its expended? Energy spent is not recoverable. Development dollars spent are not recoverable. This is your most precious resource. And a good software architect can give you the most effective balance.

How? By making sure the vital tasks are done with consistency and accuracy. Requirements analysis is accepted as vital. But different pieces within a CSCI are often reviewed by different people. Often designers or coders are given a piece of the CSCI and allowed to run with the entire piece, from requirements analysis to tasking structure and software design, to coding and integration. The results can range from excessive use of tasking, to incompatible interfaces, to correct implementation of the requirements but discovery that the requirements had some fundamental flaw. This cost is compounded in a large system. The results often affect another team. Expand the problem from a stand-alone functional piece of one CSCI to a more complex scenario, the requirements for a thread of functionality that spans several CSCIs. Without a software architect ensuring that the thread will work from a structural and architectural point of view, its not worth getting the other players in the game yet.

Good team dynamics are essential to effectively utilizing a software architect. The architect doesn't typically write requirements, but guides their writing, ensures that each requirement fits in with the vision for the architecture. This can cause problems if the architect and systems engineers don't share mutual respect. It is even more difficult if the systems engineer is part of a different functional organization. On the positive side, establishing the position of software architect frees up a lead design spot for an individual CSCI. This gives other engineers a growth position and a chance to learn from the architect.

Some CSCIs need a software architect all to themselves, while in other cases one architect best serves a collection of interrelated CSCIs. Interaction between CSCIs is the most important aspect to manage early, and the most chain reactive. With a large program and the typical matrix organization, it is inevitable that the ideal coupling of CSCIs to architect will cross some organizational boundaries. Tough. If you don't do it, you will pay many times over. Give up your architect for the good of the project. Give the architect the authority to influence your CSCI's structure, to be consistent with the rest of the group.

In summary, identify and support Software Architects. It is a key ingredient to success.

PROCESS

Effective use of a process was one of the most important lessons learned. We define a process as the set of procedures used to produce some result. Thus, process affects every aspect of the development. The key areas presented here are the initial establishment of a process, several life cycle specific issues, the waterfall versus spiral (or evolutionary) development model, and configuration management.

APPENDIX O Additional Volume 1 Addenda

Initial Establishment of a Process

Lets figure out how we want to do business, before we do business.

Initial establishment and continuous improvement of a software development process is an investment and an attitude. You need to assign some of your best people to defining it (investment). One of the biggest dangers is letting who ever is available work on process. The group defining and improving the process is viewed as a service group. Remember the lessons about team dynamics. The service group must command respect and exhibit integrity. The group must provide demonstrable benefits. By putting some well respected individuals on this team, you start off on the right foot.

Initial establishment is often one of the most difficult hurdles in any organization. If there has been no formal process, it is difficult to comprehend the need for one (attitude). This is compounded when tackling a project that dwarfs any other project performed by that organization. It is difficult to anticipate just how different this large project will be. The need for a solid process that is embraced by all grows exponentially with project size. Obviously you need to change the attitude if you want the process to pay dividends. If the leaders have bought into the need for a process, and participate in its creation and maintenance, the rest will follow and watch for results (which must come soon).

Here are some tips on how to start. First, get complete buy in from your leaders. This must eventually include all groups that will be touched by the process, from the program office and systems engineering, to software design, test, integration, quality assurance, configuration management, and last, but not least, the customer. Start with a small group of representative people that can work together. It doesn't matter that you might be leaving someone important out yet, getting the ball rolling is most important at this stage. The people that will eventually use this process need to see momentum, not promises.

Start out simple and flexible. Don't take the whole life cycle at once. Grow the process. As you tackle different pieces of the life cycle, add the appropriate organizations to the team. One size will never fit all so don't try to spell out all possible combinations. Rather, strive for the architecture of the process. Let each team or subsystem tailor the process for their specific applications. Here is where the software architects can play a key role. You don't want each team to do their own thing, because that cancels out the consistency across the board, which is important for efficiency in the support organizations and to the customer. Try to keep each team or subsystem's tailoring definition to one or two pages.

Strive for readability first, completeness second. It doesn't matter if the process is 100% complete if no one reads it, follows it, or checks for compliance. Challenge each step to calculate the added value. Value is a difficult quantity to measure or anticipate. Perception plays a strong role here. If everyone perceives a certain step adds value, they will attempt to follow it rigorously. As discussed previously, rotate people on the process improvement teams to keep fresh ideas coming.

One final note on participation. Similar to the team dynamics lessons, all groups that touch the product in some way, shape, or form should be represented in the process and on the process definition / improvement team. If one group is excluded (by their or someone else's choice), problems will arise. Resentment will build. Quality will suffer. You need complete participation.

In summary, process is an investment and an attitude.

Interfaces

A large system lives or dies by its interfaces.

Interface definition is not part of the classical life cycle. Yet interface definition on a large, distributed system is vital. This can not be stressed enough. The problem is compounded when the development team is geographically separated. Concurrent development and the typical waterfall mentality add further complications.

Typical implementations of MIL-STD-2167A call for an IRS (Interface Requirements Document) and an IDD (Interface Design Document). The way I look at the interfaces is like a contract. If I sign my CSCI up to an interface, I am signing a contract that I will hold up my end of the deal. I am assuming the other side of the interface will do the same. It is an excellent vehicle to manage parallel development. However, it takes quite an effort to completely define one CSCI's interfaces. It is not a waterfall type activity. It can not be completely specified up front, any more than the requirements for a CSCI can be completely specified before designing or coding anything. The amount of detail required to unambiguously specify an interface is tremendous. Here are some tips for success. As discussed previously, the software architects should play the lead role in managing interfaces.

Utilize Code in the IDD

The use of code in the IDD, to the extent possible, provides an unambiguous definition. The use of Ada allows complete message formats to be specified. But this is the easy part of interface definition. The format of a message doesn't impact the CSCI architecture as much as the communications address of the message, sequencing, frequency of transmission, initiating conditions, and expected responses. These latter interface requirements can have a direct effect on a CSCI's tasking structure, its architecture. Some of these items may be well represented in code. For example, the frequency of transmission can be a constant. The initiating conditions and expected responses could often be specified as the identifier of another message (e.g., this message is initiated upon receipt of message X). As we'll discuss later in the section about reusable CSCI architectures, this type of information can go a long way toward removing any ambiguity from interfaces. The more information that is unambiguous, the more effective parallel development will be, and the easier integration will be.

For Ada to Ada interfaces, as long as both sides utilize the same compiler, the definition is sound. If the interface is from an Ada to non-Ada CSCI, it isn't obvious how to proceed. The Ada CSCI needs the definition to be in Ada anyway, so as a minimum, that work needs to be done. Here is where I would recommend the generation or purchase of a tool. This tool would take the Ada interface definition and generate the non-Ada definition. In the case of an assembler based CSCI, the complete definition could be specified using "*equate*" statements. This type of convention would need to be specified at the start of a program.

Put Inter-Process Interfaces in the IDD

The larger the CSCI, the larger the need for some type of internal interface definitions. This does not need to be formal, as in an IDD. But we have seen large programs with few CSCIs struggle due to a lack of internal interface control. In these programs, large CSCIs may physically be spread over several processors and/or processes. These CSCIs could be better managed by either splitting the CSCI (across physical boundaries, i.e., processor or process boundaries, or individual Ada programs), thereby causing more information to be put into the IDD, or by documenting internal CSCI interfaces (again, using physical boundaries) in the IDD. Putting these internal interfaces in the IDD causes some benefits. First, it raises the importance of the interface. It won't be carelessly changed. Second, it allows more parallel development by establishing a contract between the different pieces of the CSCI. Third, it benefits integration.

By putting these interfaces in the IDD, it allows test software, simulation / stimulation software, and data collection software to utilize the same interfaces as the CSCI under test.

Typical embedded systems often have one application process per processor, but as future systems move to a COTS (commercial-off-the-shelf) hardware environment, many processes may co-reside on the same workstation. This increases the need to document these interfaces in a formal manner.

Keep SRS Interface References to the Message Level

We've seen a lot of SRSs, and we argue that they could all adequately specify the requirements by referencing only the message level. The key things needed within an SRS are the processing and sequencing around the messages. What happens before sending this message, what should we do when we receive this message. The algorithmic details in the SRS may imply some data format content but refrain from putting that detail in the SRS.

Combine the IRS with the IDD

All SRS messages must be represented, one for one, in the IDD. Since the SRS doesn't mention message content, only message name and sequencing/processing requirements, the IDD is still free to implement that message using any style necessary. On a distributed system, each CSCI under test will need to be stimulated using the IDD anyway. The IRS interfaces are of no use. The requirements in the SRS for message sequencing are utilized to generate test cases for formal CSCI testing. The message content details from the IDD are used to create the test case messages.

Within the IDD, all sequencing details for SRS specified messages become requirements. All message content becomes implementation. Implementation only messages are still allowed in the IDD. These messages are typically utilized for low level handshaking that is not appropriate for an SRS. Since the IDD contains as much code as possible, including the complete message format, it gives you binding power and a built in management capability. The IRS is paper or an electronic model, and has no binding power. Therefore, it will inevitably drift out of date. If everyone is forced to use the same Configuration Managed IDD when building their code, interface problems will tend to settle themselves.

Configuration Management

Once an interface agreement is reached, get that agreement into the formal program baseline as soon as possible. Ideally, the interfaces would enter into the baseline before or during design of the components that will utilize the interface. This preserves the investment in the component and increases the importance to the contract. As discussed above, forcing the use of the formal program baseline IDD helps resolve conflicts.

Distributed Object Systems

One final futuristic point is about distributed objects. There is much work going on in this area and it could solve a lot of the interface problems. By localizing the data and its functions into an object that is shared between applications, it localizes all the processing associated with the data into one entity (e.g., a single Ada package). The single entity can be built by a single group. This will eliminate much of the interpretation problems of today's message based systems, where the source group builds some processing, and the destination group builds the rest of the processing.

Until there are standards and environmental support for distributed objects, intelligent linkers can give some benefit today. Many Ada environments provide linkers that only include subroutines that are referenced by the program. Thus, we could create packages that encapsulate all the processing related to a message, including creation, population, transmission, receipt, usage, and release. This naturally partitions into source and destination subroutines. The source CSCI only references the source subroutines, the destination CSCI only references the destination

subroutines. The linker is intelligent enough to only include the appropriate subroutines in each CSCI's image, thus optimizing image size. The package can be built by one individual or team, reducing or eliminating the possibility of ambiguity.

In summary, manage the interfaces, and you can manage the project.

Integrated Product Teams

Reduce inter-team conflicts by creating integrated product teams.

As discussed above, team dynamics are difficult to balance. Why not avoid the balancing act as much as possible? Integrated product teams are the answer. By locating all the people responsible for building and testing a product or component under the same functional management, and ideally in the same physical location, you tear down many artificial walls Within a CSCI, the key functional areas include systems engineering, and software test. Within a subsystem (a collection of similar or tightly coupled CSCIs), support software and CSCI-CSCI integration should be joined with the development team.

Here are some examples of the benefits of eliminating these walls. If the requirements generation and software design are tightly coupled, there will be more flexibility and support for the spiral development model. The requirements vital to CSCI architecture can be prioritized first. Complex requirements can be prototyped to see what makes sense, and what is unrealistic. Including software test on the same team allows them to be involved in more discussions, allowing them to achieve a better understanding of the CSCI under test. The manager must not allow the independent validation process to be compromised, but the benefits far outweigh the risks.

On a larger, subsystem scale, combining support software and CSCI-CSCI integration with the development teams removes several problematic walls. Support software is often a lower priority. By including them on the same team, there is more flexibility of moving people around, more sharing of knowledge, more assurance that the true requirements for support software will be built. Traditionally, one of the biggest walls is between the development team and CSCI-CSCI integration. Here is where the individual pieces of the product must come together into a system. Here is where early requirements, interface or design flaws will show up for the first time. Tension can easily build and finger pointing is a natural response. Finger pointing can build walls in a flash. If we are all part of one big team, the possibility of everyone being committed to an integrated product increases. There are less turf wars. You should never hear a developer saying to an integrator, "*I don't have a requirement to do that.*" An integrated team like this should all adopt this extra requirement — "*we shall build a product that works reliably, and meets the budget and contractual obligations.*" Adding this requirement eliminates a lot of finger pointing.

One specialty note on integrated product teams. Often there are organizations with specialized integrators, or integration support teams. These teams often are the wise old lab gurus. They have the ability to set up procedures and techniques that will help other integrators and developers. Often, there are also several CSCIs that form the services layer of functionality. the layer that other applications will utilize to perform their tasks. This includes the operating system, inter-program communications, display interfaces, resource management, and data management. These two groups have tightly coupled responsibilities, yet they are often chasing conflicting priorities and deadlines.

By combining the two teams, you have the ability to influence the design of the service layer with hooks and angles that will pay for themselves many times over in integration.

In summary, combine teams for better productivity and efficiency.

APPENDIX O Additional Volume 1 Addenda

Reuse

Reuse offers an often elusive payback.

Reuse, like process definition, is an investment and an attitude. It requires an investment by your best engineers to find what should be reusable, and build it so it is easy to use and reliable. This also helps with the attitude, having respected engineers sign off on a product makes its adoption easier. Sometimes, the attitude requires some legislation. Engineers are always trying to create. Trying to get them to reuse a component gives the perception of removing some of the creativity. But if the policy is established and the cost savings obvious, they will stop complaining, stop trying to show why they can't reuse a particular component (or why they could build a better component), and get on with it.

When dealing with reuse across teams, solid cost models must be established. These models must take into account the extra cost of building something for reuse as well as the payback when something gets reused. Middle management often gets measured on cost. Who would volunteer to spend extra money to build something reusable if there wasn't a mechanism to reflect the extra cost of construction and the payback when it is reused? How successful a reuse program is often comes down to deciding whether or not to make an investment in good people to manage and implement the program.

One overlooked possibility for reuse is a common CSCI software architecture. This helps in several ways — by using a proven architecture, each development team would not have to spend time integrating the basics — it would be a given. Integration of CSCIs would be simplified, and people would be more able to adapt to new assignments on other CSCIs. Understanding the basic CSCI architecture would need to be done once. Understanding each CSCI's unique part would be the only task on a reassignment.

Most CSCIs in a distributed system can initialize in the same basic way, elaborate, open their communications agents, request disk-based data, signal that they are ready to run, then run. And all but the hardest real time CSCIs could utilize the same type of input / output mechanisms and tasking structure. The use of common Ada package specs, and either unique package bodies or separate subprograms gives developers a compile-time binding to the common architecture.

In summary, reuse is an investment and an attitude.

Waterfall versus Spiral Model

The waterfall model has many shortcomings for large systems.

There is much enthusiasm in the literature for the spiral model of software development — design a little, code a little, test a little. On a project that must follow MIL-STD-2167A, and is too large to comprehend by one person, it may be difficult to decide where to start designing. Plus, the typical contract is structured with big, waterfall type events such as SSR (System Segment Review), PDR (Preliminary Design Review), and CDR (Critical Design Review). This waterfall contract encourages everything to proceed in parallel. In reality, some pieces (e.g., CSCIs) will progress more quickly than others, and some pieces are needed to mature earlier than others.

Why not structure the waterfall events in phases? Establish a layered approach to the system with the service layer first. Then add the other functionality into phases based on its criticality to the mission. Write the contract to have the SSR, PDR, and CDR for each phase. This allows the critical layers to mature when necessary. It also allows more resources to concentrate

on the biggest risk areas up front. Having the service layer built first gives many benefits. It allows all future development and integration to start with a solid system foundation. It allows tools and integration techniques to be matured as development and integration needs dictate. If the service layer were being built concurrently, there is often not enough resources to add the integration hooks and handles. But if the service layer is established, mature, and baselined, the layer experts could easily enhance it to grow with integration's needs.

Ideally, CSCIs will interface only with other CSCIs from the same phase. This sounds nice but will rarely happen in practice. In this case, we suggest a phased delivery for the CSCIs. Perform some top level design on the interfaces with CSCIs in later phases, but only build the portion that interfaces with CSCIs in this or earlier phases. This may sound radical but countless dollars are spent on rework because of immature interface contracts. By the time the later CSCI really evaluated the interface, it was found to be inadequate, thus wasting all the time spent by the earlier CSCI. Now for trivial interfaces, they can be specified early, but for any nontrivial interface, it pays to wait until both sides are really ready to sign a binding contract.

In keeping with the spirit of the spiral model, the contract must grow too. But it must grow together. This lock-step approach to the spiral model has some nice benefits. A functional system is ready much sooner, and much more often. Updates have less of a ripple effect since both sides of an interface have waited to take the next step together. This scheme causes some different tracking models to show progress. Functional threads, spanning more than one CSCI and the associated messages, become the ideal tracking mechanism. It is relatively easy to identify the messages and software components that support that thread, thus it is relatively easy to track their implementation progress. A good Configuration Management (CM) system will assist in this task by allowing related changes to be grouped or tracked together.

Using the spiral model also moves emphasis from documentation to functionality. The sooner this shift occurs, the better. The documentation is essential but it should not drive the program. Integrated product teams also greatly assist with implementing the spiral model.

In summary, structure the contract to map to the spiral model.

Configuration Management

Without a solid CM process, you will struggle.

Configuration management (CM) is important to any size project. It is absolutely required for a large project. All items generated on the project should be controlled in some manner. This includes all requirements, white papers, designs, code, test data and results, integration scenarios and results, everything. Now the degree to which each piece of information is controlled varies. Some items require customer approval to change. Milestones such as CDR trigger the transition from informal (or developmental) (DCM) to formal CM (FCM). Other items such as software development folders are always under DCM and never under FCM. One can think of the difference in the degree of formality in terms of who should approve changes or proposed changes. The less critical or smaller the chance of a ripple effect, the closer the approval should be to the person making the change (if there is no chance of a ripple effect, it may be appropriate to have no approval). The more critical or higher the chance of a ripple effect, the more people should be involved with approving the change. Ideally, the DCM and FCM environments should be one in the same. As the degree of formality increases, the approval simply increases to the appropriate level(s).

CM serves as a natural communication mechanism. Using only baselined items to perform all software builds, document generation, etc., provides a controlled mechanism to help mature the system (a nice complement to the spiral model). Once the community buys into utilizing CM properly, product integrity improves. This can be achieved for little cost. CM

doesn't have to be expensive, but not having CM is very expensive. Having DCM and FCM can provide some other benefits as side effects. Some obvious benefits are automated build support. Further, automated tools can be run on the baseline to calculate metrics, and perform compliance and consistency checks. The life cycle can be modeled in the automated CM systems. This modeling allows the developmental and maintenance process to be enforced, automated, and tracked.

For example, let's assume the process for a given CSCI is design, code, test, then baseline into FCM. As the design review is completed, the librarian could enter the review results into the system and the system would automatically register the items that completed the review. This could then be used to calculate some "*percentage complete.*" The same analogy could be used for code and test. Once baselined, the items should only change for additional functionality or rework. In either case, the change causes the items to revert back to some previous process phase (either design or code in this example). By indicating the items that must change, the process phase, and an estimate of the effort to implement the change, the system can keep track of how much additional functionality or rework is outstanding.

Some specific items that require extra attention are interfaces. These can be inter-CSCI interfaces or intra-CSCI interfaces. We've already discussed inter-CSCI interfaces and the importance of managing them, including configuring them. For intra-CSCI interfaces, utilizing Ada and the package spec, we can achieve similar control and benefits. Placing a more formal level of control on package specs elevates their importance and reduces the risk when developing in parallel. For example, after a design review the package specs may be placed into DCM and the CSCI team leader must approve all changes to them.

In summary, establish a complete CM process — don't start coding without it.

TECHNOLOGY

On a program this large, it may seem odd that we aren't talking much about technology. There were some high-tech developments in fault tolerance and sonar. But as far as the use of technology to manage the program, there wasn't too much to talk about. We had a variety of homegrown interface and integration tools that were utilized with much success. We had an enormous UNIX development and integration environment that gave us the horsepower needed to manage the vast amounts of data and code. There was always a search for *silver bullet tools* that would alleviate this or that problem. Most proved to be of arguable value. The only technology we want to single out is the Ada language itself.

Use of Ada

Ada was designed for large programs.

We feel no other language (except possibly C++) would have survived on a program of this scale. The benefits provided to us by Ada were tremendous. Sure, we stressed the Ada environment often, and broke it more than a few times. But the benefits far outweighed the problems. You've no doubt heard about the benefits of information hiding. We've already discussed the use of Ada for interfaces. Below are some additional benefits and techniques learned.

There's No Substitute for Experience

Ada is a rich language. It provides many features that need to be used in moderation

on a large, real-time program such as this. Experience with the run-time properties of some of Ada's more mysterious features paid off time and time again. In examples where a team did not have access to an experienced Ada person, often the results showed. Some of the dangerous features were over use of generics and tasks, use of variant records (caused many run-time problems), too much nesting of generics and packages, and too much information in the package spec (resulting in global objects).

Self Documenting Code

Maintenance on a large program such as this is actually performed ongoing. People leave or get reassigned and someone needs to pick up and finish where the originator left off. Reliance on external documentation often proves to be futile. When well designed, Ada code can be self documenting. Naturally, there were exceptions to the rule. But when done properly, the maintenance payoff was felt early.

Use Tasks Wisely

Tasks are not free. In a real-time system, improper use of tasks can wreak havoc. But when used properly, tasks provide a natural expression of the real concurrency problem being solved. When possible, utilize a common CSCI architecture. Here are a few suggestions on the use of tasks.

- Utilize the main program — it is in reality a task. We suggest using the main program to orchestrate the application's major state changes (e.g., initialization, shutdown, etc.). This allows a single reference to understanding each application's state transitions. By using common package specs for the common components of each application, the main program can even be common (yielding a common architecture). Even if it isn't, using this approach provides a consistent scheme among all CSCIs.
- Utilize an event management scheme within tasks, as appropriate. An event management scheme is a method where multiple, simultaneous events are managed within one task. This allows for more concurrency without more tasks. By constructing a common event management package early, you enable designers to reduce the number of tasks needed.

Integration Techniques

Several Ada environments provide excellent program debuggers. These tools are tremendous in isolating problems. Adding TEXT_IO statements to display intermediate states, events, and results also help isolate many problems. However, there will undoubtedly be problems that can't be found or even isolated with debuggers due to the timing or real-time nature of the problem. TEXT_IO may cause performance problems and therefore be impractical for all problems. We offer the following hybrid approach.

Sophisticated logic analyzers exist that can unobtrusively capture data from a system. By taking advantage of the analyzer's strengths, namely the ability to capture data from a specific address range, we can add instrumentation to key algorithms and functions and gain a tremendous insight into the underlying system. For example, we might create a block of 10 integers, each one corresponding to some key function. Whenever one of the key functions invoked, say a memory allocation routine, the function writes a code (in this case, the amount of memory allocated) to the corresponding integer. The code should be designed in such as way as to allow analysis of the operation through the logic analyzer. The analyzer simply captures all "*writes*" to these integers. Simple tools can decode the results. The instrumentation doesn't add any significant overhead, but the results can help find many tough problems.

APPENDIX O Additional Volume 1 Addenda

Design for Tuning Up Front

Integrating a large system will require many intermediate test environments. There won't always be the complete system to work with. Some components will need to be simulated, some will need to be turned off. Designing up front to support tuning is one of the most important factors for success. Utilizing the service layer and / or configuration files to turn on or off functionality, can provide the needed integration flexibility. With this mechanism in place, the TEXT_IO approach to logging events and data can also be useful.

Ada Is Not for Everything

Ada is not ideal for everything. Operating systems and low level functionality may be better served with C or assembler. Graphical User Interface functionality can benefit from automated tools that generate C or C++ code. Accepting this fact, you next need to deal with a mixed (Ada to non-Ada) environment. As discussed in the interface section above, do not treat this environment lightly.

In summary, Ada has what it takes for large systems.

SUMMARY

Models such as that presented above should help guide medium to large scale programs. Given the proper management support and risk management strategies, the processes and technology for tackling tomorrow's complex systems exists today.

REFERENCES

[1] Tom Peters, "Thriving on Chaos", Harper Collins.

[2] Dr. Steven Covey, "The Seven Habits of Highly Effective People", Simon and Schuster.

[3] F. Gregory Farnham and Kevin J. McSweeney, "Going to Sea with Ada", Defense Electronics, October 1994.

[4] F. Gregory Farnham, "Lessons Learned on BSY-2", Software Technology Conference, April 1993.

About the Author

Robert F. Sullivan Jr. is Vice President of Technology and Product Development at PROSOFT, Inc., located in Syracuse, NY. He is responsible for development and improvement of PROSOFT's state of the art Configuration Management product, XStream.

Blank page.

Version 2.0

DIRECT COMMENTS, QUESTIONS, OR REQUESTS FOR ADDITIONAL COPIES TO:

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, Utah 84056-5205
(801) 777-8045
E-mail: custserv@software.hill.af.mil

Blank page.