SOFTWARE ENGINEERING LABORATORY SERIES

SEL-89-008

PROCEEDINGS OF THE SECOND NASA ADA USERS' SYMPOSIUM

NOVEMBER 1989

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SEL-89-008

PROCEEDINGS OF THE SECOND NASA ADA USERS' SYMPOSIUM

November 1989

GODDARD SPACE FLIGHT CENTER

Greenbelt, Maryland

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC, Systems Development Branch

The University of Maryland, Computer Sciences Department

Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that include this document.

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

AGENDA

SECOND NASA ADA USERS' SYMPOSIUM
NASA/GODDARD SPACE FLIGHT CENTER
BUILDING 8 AUDITORIUM
NOVEMBER 30, 1989

Summary of Presentations
R. Kester (CSC)

Session 1

Topic: NASA-Wide Activities

Session Leader: E. Seidewitz (NASA/GSFC)

Ada in NASA: Policy and Directions
F.E. McGarry (NASA/GSFC)

Ada and the Space Station
R. Nelson (Space Station Freedom Program Office)

Software Support Environment (SSE): Program Status
F. Barnes and D. Badal (Lockheed)

Session 2

Topic: Center and Project Activities

Session Leader: M. Stark (NASA/GSFC)

Ada in the SEL: Experiences with Operational Ada Projects
E. Seidewitz and M. Stark (NASA/GSFC)

The Application of CASE Technology and Structured Analysis to a Real-Time Ada Project
S. Cohen (GE/STGT)

Ada at JPL: Experiences and Directions
T. Fouser (JPL)

Ada and the OMV Project
W. Harless (TRW)

AGENDA (Cont'd)

Session 3

Topic: Space Station Activities

Session Leader: D. Littman (NASA/GSFC)

Lessons Learned: Prototyping with Ada for the Space Station Freedom Program
K. Rogers and L. Ambrose (MITRE)

Software Support Environment: Architecture and Design Overview
C. Carmody (PRC/GIS)

Flight Telerobotic Servicer
R. LaBaugh (Martin Marietta)

Lessons Learned in Prototyping the Space Station Remote Manipulator System Control Algorithms in Ada
P. Gacuk (SPAR Aerospace)

Appendix A - List of Attendees

Appendix B - Standard Bibliography of SEL Literature

SUMMARY OF PRESENTATIONS

R. Kester, CSC

SUMMARY OF THE SECOND NASA ADA USERS' SYMPOSIUM

On November 30, 1989, approximately 370 attendees gathered in Building 8 at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC) for the Second NASA Ada Users' Symposium. The symposium was created as a forum for NASA centers and contractors to exchange their ideas, plans, and experiences in using the Ada language or related methods and tools. It is sponsored by NASA/GSFC and hosted by the Goddard Ada Users' Group. Among the audience were representatives from 3 universities, 17 government agencies, 7 NASA centers, and 75 private corporations and institutions. Eleven papers were presented in three sessions:

- NASA-wide Activities
- Center and Project Activities
- Space Station Activities

SESSION 1 - NASA-WIDE ACTIVITIES

Ed Seidewitz of GSFC opened the symposium, welcomed attendees, and introduced the first speaker. The lead-off presentation, "Ada in NASA: Policy and Directions," was given by Frank McGarry, also of GSFC. McGarry described the four-step process of formulating NASA policies for Ada:

1. Assess current capabilities/needs/directions
2. Conduct an open review of findings/recommendations
3. Formulate the positions/recommendations of each NASA center/office
4. Develop an action plan for NASA

McGarry indicated that the first three steps have been completed. The response from NASA contractors has been very supportive of the recommendation that NASA adopt Ada, but issues, questions, and considerations were identified. The response from NASA centers has been mixed, with four centers supporting an Ada mandate, five supporting Ada but without mandate, four uncertain, and three opposed. The primary concerns expressed by NASA are the cost and maturity of

Ada technology. McGarry concluded by stating that completion of the NASA action plan has been understandably delayed by a change in NASA administration, but that some centers may move ahead and adopt Ada on a center-wide basis.

The second speaker, Robert Nelson of the Space Station Freedom Program Office, provided an update on the current status of the Space Station Freedom program and discussed where Ada fits in ("Ada and the Space Station"). Most systems are currently reviewing software requirements. A major impact on the program has been the reduction in the power budget for the in-orbit portion, which has caused reevaluation of the planned software and computer capabilities.

Nelson discussed the program-level risk management plan and noted that Ada was not identified as one of the top 10 risks, although it does appear on the list. An object-oriented architecture is evolving and a task team met recently to address some overall architectural issues. The current estimate of the total size of space station software is 10.5 million source lines of code (SLOC), mostly Ada. Nelson pointed to the number of interfaces and to the phased on-orbit assembly as representing significant software challenges for the Space Station Freedom program.

The final speaker in the first session, Frank Barnes of Lockheed, presented "Software Support Environment (SSE): Program Status." The intent of the SSE is to support the management of Space Station software development by NASA and its contractors. Barnes described the dissatisfaction of users with the current release of SSE. The overhead that results from managing any process, along with the immaturity of the current SSE release, account for much of this dissatisfaction. Although the final system is scheduled for release in mid-1993, much of its capabilities are needed now. Barnes described the current user interface as "user-surly," while noting that improvements in this area are planned for August 1990. The current release is also manually intensive, a problem that should be addressed by January 1991. Barnes summarized the SSE's major challenges as achieving a consensus among its many users, supporting multiple host systems, and providing capabilities that match the Space Station development schedule.

SESSION 2 - CENTER AND PROJECT ACTIVITIES

Ed Seidewitz of NASA/GSFC presented "Ada in the SEL: Experiences with Operational Ada Projects." The Software Engineering Laboratory (SEL) develops about 15 percent of its software in Ada for systems ranging in size from 68K to 170K SLOC. The SEL is currently in its fourth generation of Ada projects and some trends have been noted. The use of generic packages and user-defined types has increased, while the average size of packages and the use of tasks has decreased. Productivity, in statements per day, has been comparable to or lower than that typical of FORTRAN projects. However, a strong trend in the latest Ada projects toward increased reuse, attributable to Ada and Object-Oriented Design (OOD), has had the effect of greatly increasing effective productivity.

Seidewitz also described the results of a study porting a system from DEC VAX/VMS Ada to Alsys IBM/MVS Ada. The study indicates that even without designing for portability, the conversion of Ada code required only half as much time as would be expected for an equivalent FORTRAN system. Seidewitz concluded by identifying plans for future Ada work, which include a real-time embedded system; a large generalized flight dynamics support system; and studies of performance, reliability, and maintainability.

Discussing activities on the Second TDRSS Ground Terminal project (STGT), Sara Cohen of General Electric presented "The Application of CASE Technology and Structured Analysis to a Real-Time Ada Project." At the start of the project, all engineers and managers were trained in Ada. In addition, the effort was led by a core team experienced in Ada, large projects, and ground station development. These factors, along with the use of CASE technology, have contributed to the success of the project to date. Diagrams have proved an excellent means of communication among the team and with the customer. A data dictionary helped ensure consistent naming among team members. Not only did the CASE tool make design updates easier, it performed better consistency checking than would have been possible otherwise.

Cohen concluded by summarizing the benefits of using the CASE tool. The analysis and design products developed using the CASE tool made up about 80 percent of the Software Requirements Specification, which resulted in higher productivity than traditionally estimated. During preliminary design, the CASE model evolved

into the software design. The model also served as the basis for the system performance study and facilitated production of the software test plans.

Tom Fouser from the Jet Propulsion Laboratory (JPL) presented "Ada at JPL: Experiences and Directions." JPL develops systems for internal research and development and for the Department of Defense, as well as for NASA missions. Currently JPL is working on a number of Ada projects. When used by Ada experts for rapid prototyping, high productivity (18 statements per day) has been achieved. Several other projects have seen the early design phases take longer, but anticipate that later phases will be shortened. For training, JPL has used a combination of in-house courses presented over a period of weeks and intensive courses brought in from outside. In addition to development using DEC's VAX/VMS environment, there is an increased use of the Rational development environment. Also, a study performed for flight software found some deficiencies in performance and scheduling, but these have been overcome by using a non-Ada real-time kernel in conjunction with an Ada application.

Fouser observed several common features among the variety of projects using Ada. The general approach has been to staff a project with a mix of outside Ada consultants and in-house management and engineers. In this way the base of trained and experienced JPL engineers and managers is growing. Each project generally encounters some problems, but finds a satisfactory work-around. More and more projects are starting to use Ada; even those initially skeptical are beginning to consider it a viable option.

The last speaker of the morning sessions, Walton Harless of TRW, presented "Ada and the OMV Project." The Orbital Maneuvering Vehicle (OMV) is a remotely controlled vehicle that will be used to assist in the launch and retrieval of satellites beyond the shuttle's range. The flight segment contains about 14K to 20K SLOC, almost exclusively in Ada. The ground system uses a mix of languages, including Ada. Although initially proposed in FORTRAN and C, by the time of contract award the motivation to use Ada had increased; hence, the flight segment and ground system command and control software are being developed in Ada.

Harless characterized training for the project as including formal on-site training for managers, designers, and developers. After an extensive trade study, the

project selected the TLD VAX-to-1750A cross-development system. Prototyping on the TLD system indicated that some Ada features (e.g., tasking, variant records, and dynamic storage) were inappropriate for the flight segment. Harless described the porting of Ada software between VAX, Alliant, Sun, and PCs as relatively painless. However, when attempting to interface between dissimilar systems, he noted that tighter control must be used in specifying data representation. Harless concluded by stressing the importance of early prototyping with the target compiler in order to understand its strengths and weaknesses and thus guide the design.

SESSION 3 - SPACE STATION ACTIVITIES

Kathy Rogers of MITRE presented "Lessons Learned: Prototyping with Ada for the Space Station Freedom Program." The goal of the prototyping effort was to examine human interface factors and gain experience with Ada and OOD. Although OOD seemed to fit the problem and Ada design well, extra effort was required to translate from functional requirements to an OOD and again from the OOD to the data flow diagrams that the reviewers felt comfortable with. During coding, the project found that Ada's ability to separate specification from implementation facilitated independent development. The parallel evolution of the project's external interfaces, however, caused significant reworking of the simulation code that was used for testing, a consideration not included in the project plans. The project found that DEC's documentation and technical support were weak in interfacing with other languages and operating system services, an area where an expert consultant would have helped. Rogers concluded the presentation by stating that much was learned on the project, and overall Ada was found to be a good tool.

Next, Cora Carmody of Planning Research Corporation presented "Software Support Environment Architecture and Design Overview." Carmody began by reiterating the point made earlier by Barnes that the Space Station SSE must satisfy the often conflicting needs of creating a long-term, lower cost life-cycle and supporting immediate user needs in a variety of environments. The current design, based on stable standard interfaces, represents this balance. The architecture utilizes an object specification-driven definition of life-cycle products and processes.

The architecture is divided into four layers: Common User Interface Services, Environment Applications, Process/Object Management, and Platform. The Common User Interface Services provides both a graphical and a textual command interface. The Environment Applications layer provides the real tools from the user's viewpoint. The Process/Object Management layer provides access control, and the Platform layer implements a virtual machine that insulates the higher levels from the host system implementation. For performance reasons, the upper layers may bypass intermediate layers and use the Platform layer directly.

Carmody summarized the benefits of the object-oriented approach as reduction of life-cycle costs (by encouraging reuse and reducing maintenance costs) and enhanced system integrity and security.

Robert LaBaugh from Martin Marietta spoke next on their experiences developing Ada software for the "Flight Telerobotic Servicer." The Flight Telerobotic Servicer (FTS) is a sophisticated robot that will be able to perform remote servicing tasks controlled from the Space Station. The current estimate is that 224K SLOC will be developed for the FTS, entirely in Ada. The flight software represents the most significant category of code and will be implemented on a distributed system of 22 Intel 80386 microprocessors that control the motion of the robot in real time. The project is using the DDC-I Ada Compiler for 80386 protected mode, which allows full use of the 32-bit architecture. The flight software will run without any operating system, using the Ada run-time system to support interrupt handling, low-level I/O, tasking, and memory management.

LaBaugh stated that to date, their prototyping efforts have not uncovered any deficiencies in the Ada run-time system that preclude its use for real-time robotic control. LaBaugh closed by presenting the results of performance benchmarks on various control algorithms and matrix operations.

The final speaker of the symposium, Pete Gacuk of SPAR Aerospace, presented "Lessons Learned in Prototyping the Space Station Remote Manipulator System Control Algorithms in Ada." The Space Station Remote Manipulator System will be an advanced descendant of the space shuttle robot arm. The multiple goals of the prototyping effort examined issues related to life-cycle, methodologies, Ada development, technical communications, Ada performance, and configuration

management. Rather than use an informal or shortened life-cycle, the prototype effort used a full life-cycle with formal reviews and participation by product assurance and configuration management personnel in order to better represent the production environment and broaden the organization's experience.

Gacuk stated that the project adopted a hybrid methodology (a combination of NASA/GSFC GOOD and Booch OOD), as this provided a transition from structured analysis to OOD. They found that typical Ada and OOD diagrams did not provide a good basis for communications between software developers and hardware engineers or testers, but that multiple notations (data flows, Booch-grams, withing diagrams, tasking diagrams, timing diagrams, state diagrams, and exception diagrams) were needed to represent different aspects and to provide interdisciplinary communications. The initial performance of the prototype was disappointing; however, after profiling and some recoding, the desired performance was achieved.

Gacuk concluded by presenting some "lessons learned about lessons learned." First, good notes are needed throughout the process in order to document lessons learned. Second, the conclusions reached are likely to change during the process as you learn more. Third, lessons learned should be "stale dated," as they often lose their value over time. Finally, without champions who continue to present and push lessons learned, the lessons often go unlearned.

SESSION 1 — NASA-WIDE ACTIVITIES

Session Leader: E. Seidewitz, NASA/GSFC

Ada in NASA: Policy and Directions
F.E. McGarry

Ada and the Space Station
R. Nelson

Software Support Environment (SSE): Program Status
F. Barnes and D. Badal

"ADA IN NASA: POLICY AND DIRECTIONS"

F. McGarry, NASA/GSFC

Ada IN NASA -
POLICY AND DIRECTIONS

NOVEMBER 30, 1989

FRANK McGARRY
NASA/GSFC

K217.001

STEPS IN FORMULATING NASA POLICIES FOR Ada

(1) ASSESS CURRENT CAPABILITIES/NEEDS/DIRECTIONS . . . . . COMPLETED 4/89

- DETERMINE NASA S/W MANAGEMENT AND TECHNOLOGY POLICY/EXPERIENCES
- DEFINE APPROPRIATE S/W TECHNOLOGY AND NASA NEEDS
- DEVELOP PLAN FOR EVOLVING TO S-O-A S/W TECHNOLOGY

(2) CONDUCT OPEN REVIEW OF FINDINGS/RECOMMENDATIONS (FROM (1)). . . . . COMPLETED 6/89

- SOLICIT REVIEW OF AEROSPACE INDUSTRY
- OPEN DISCUSSION BY ALL NASA CENTERS/PROGRAM OFFICES

(3) FORMULATE POSITIONS/RECOMMENDATIONS BY EACH NASA CENTER/OFFICE . . . COMPLETED 8/89

- REVIEW REPORTS (1) AND COMMENTS/DISCUSSIONS (2)
- CARRY OUT INTERNAL (TO CENTER) ASSESSMENT
- RESPOND W/COMMENTS TO IRM EXECUTIVE SECRETARY

(4) DEVELOP ACTION PLAN FOR NASA . . . . . .TBD

- BASED ON (1), (2), (3)
- RESPONSIBILITY OF IRM COUNCIL
- SCHEDULED FOR FALL 1989

F. McGarry
NASA/GSFC

INTERNAL NASA STUDY

STEP ①

(JUNE, 1988) NASA IRM COUNCIL ESTABLISHES ASMAWG

- ASSESS NASA POSTURE ON S/W MANAGEMENT AND "Ada" TECHNOLOGY
- DEFINE MEANS TO BUILD KNOWLEDGE AND EXPERIENCE BASE
- DEVELOP PLAN CARRYING NASA TOWARD S-O-A S/W TECHNOLOGY

(MARCH, 1989) 2 REPORTS COMPLETED

- "Ada AND SOFTWARE MANAGEMENT IN NASA - ASSESSMENT AND RECOMMENDATIONS"
- "NASA EVOLVING TO Ada - 5 YEAR PLAN"

(APRIL, 1989) FINAL BRIEFING TO IRM COUNCIL

ACTIONS:
1. SCHEDULE SYMPOSIUM/FORUM - MAY
2. WRITTEN COMMENTS FROM IRM MEMBERS - JULY 1989
3. FORMULATE SPECIFIC ACTION PLAN - BY FALL 1989

FINDINGS* OF ASMAWG

1. NO S/W HORROR STORIES IN NASA
   - OFTEN ON CUTTING EDGE (E.G., SHUTTLE, MEASUREMENT, SSF)
   - GROWING AWARENESS BY ALL LEVELS OF MANAGEMENT
2. Ada IS APPROPRIATE FOR NASA
   - TO SUPPORT EVOLUTION IN S/W TECHNOLOGY
3. CURRENT TRAINING PROGRAMS INADEQUATE
4. Ada EXPERIENCE BASE CANNOT MEET CURRENT COMMITMENTS
5. NASA "INFRASTRUCTURE" REQUIRES CHANGE
   - NO HQ FOCUS FOR S/W ENGINEERING
   - UNPREPARED FOR S/W TECHNOLOGY TRANSITION
   - STANDARDS LACKING
6. SOME GOOD S/W RESEARCH, BUT ...
7. INSUFFICIENT MEASUREMENT PROGRAMS

*REFERENCE: "Ada AND SOFTWARE MANAGEMENT IN NASA - ASSESSMENT AND RECOMMENDATIONS" (4/89)

RECOMMENDATIONS* OF ASMAWG

1. NASA SHOULD MANDATE Ada (FOR ALL "MISSION" S/W)
   - 3 PHASE EVOLUTION
   - APPROPRIATE WAIVERS
   - LEARN FROM DoD
2. ALL CENTERS TO DEVLOP TRANSITION PLANS
   - DEFINE TRAINING/PHASING
   - CONTINGENCY/WAIVER PROCESS
3. DEVELOP NASA-WIDE SOFTWARE STANDARDS
4. ESTABLISH 2 SUPPORT "ORGANIZATIONS" (AT HQ)
   - "SOFTWARE ENGINEERING AND Ada IMPLEMENTATION TASK FORCE" (SEAITF)
   - "SOFTWARE PROCESS ENGINEERING TASK FORCE"
5. IMPLEMENT NASA-WIDE TRAINING PROGRAM
6. DEVELOP COMMON SUPPORT "ENVIRONMENT"
7. EXPAND/FOCUS S/W RESEARCH
8. BROADEN MEASUREMENT EFFORTS
9. IMPLEMENT CONTRACTOR INCENTIVES

*REFERENCE: "Ada AND SOFTWARE MANAGEMENT IN NASA - ASSESSMENT AND RECOMMENDATIONS" (4/89)

MISSION SOFTWARE*

MISSION SOFTWARE IS ALL SOFTWARE THAT IS CRITICAL TO THE DESIGN, PLANNING, OPERATION, CONTROL, OR TESTING OF ANY NASA FLIGHT PROJECT. IT COMPRISES ALL FLIGHT SOFTWARE AND ALL GROUND SOFTWARE THAT DIRECTLY INTERFACE WITH THE FLIGHT SYSTEMS OR COULD AFFECT MISSION PLANNING, CONTROL, OR OPERATIONS. MISSION SOFTWARE INCLUDES, FOR EXAMPLE, ALL SOFTWARE USED IN FLIGHT PLANNING, FLIGHT DYNAMICS, MISSION CONTROL, AND FLIGHT READINESS. IT ALSO INCLUDES ALL SOFTWARE USED TO SIMULATE, MODEL, OR TEST ANY OF THE FOREGOING SOFTWARE FUNCTIONS.

**AS DEFINED BY THE ASMAWG*

REVIEW FINDINGS AND RECOMMENDATIONS*

STEP ②

- REPORTS SENT TO ALL NASA CENTERS/OFFICES AND AEROSPACE CONTRACTORS
- 8 AEROSPACE CORPORATIONS ASKED TO REVIEW IN DETAIL AND PREPARE RESPONSE
- REPS FROM NASA ORGANIZATIONS ASKED TO ATTEND FORUM
- OPEN FORUM/SYMPOSIUM HELD MAY 31/JUNE 1 AT GSFC
    - 200 ATTENDEES
    - 8 FORMAL PRESENTATIONS (INDUSTRY)
    - OPEN DISCUSSION/PANELS (NASA)

*REFERENCE: "Ada AND SOFTWARE MANAGEMENT IN NASA: SYMPOSIUM/FORUM" (6/89)

Ada SOFTWARE MANAGEMENT IN NASA

OPEN FORUM/SYMPOSIUM
MAY 31 AND JUNE 1, 1989
DAY - 1 INDUSTRY PERSPECTIVE

PARTICIPANTS	REPRESENTATIVES
HUGHES	RAY WOLVERTON/BRUCE KRELL
IBM	JUDY FLEMING
CSC	DICK TAYLOR
MCDONNELL DOUGLAS	JOE McCABE
TRW	MIKE HOLLOWICH
GE	JEFF NEUFELD
LOCKHEED	KENT LENNINGTON
BOEING	WELDON JACKSON

NASA FORUM/SYMPOSIUM

INDUSTRY PERSPECTIVE (I)

- HUGHES
  - STRONG OVERALL SUPPORT FOR PLAN
  - MANDATE Ada - DEVELOP STANDARDS
  - NO "CONTRACTOR INCENTIVES" NECESSARY
- IBM
  - EVOLVE TO Ada (MANDATE?)
  - PRODUCE STANDARDS
  - FOCUS ON "REUSE INCENTIVES"
  - EMPHASIS ON LEVERAGE FROM COMPLETED WORK (SEI, DoD, ...)
- CSC
  - MANDATE Ada AND STANDARDS
  - STRONG GENERAL SUPPORT
  - HOW DO WE COORDINATE TRAINING (NASA - CONTRACTOR)?
- MCDONNELL DOUGLAS
  - CONCERNED ABOUT COST
  - LEVERAGE ON DoD EFFORTS
  - "...IN GENERAL SUPPORTS..."

NASA FORUM/SYMPOSIUM

INDUSTRY PERSPECTIVE (II)

- TRW
  - FOCUS ON "COMMON ENVIRONMENT"
    - QUESTIONS ON IMPLICATIONS AND VIABILITY
  - NO SPECIFIC POSITION ON Ada/STANDARDS/STRUCTURE/ (AT SYMPOSIUM)
- GE
  - MANDATE Ada/DEVELOP STANDARDS/COMMON ENVIRONMENT
  - STRONG OVERALL ENDORSEMENT
  - USE 2167X AS BASELINE
- LOCKHEED
  - BASED ON SSE PERSPECTIVE/SHOULD BE MODEL
  - STRONG SUPPORT FOR "TRANSITION MODEL" AND RECOMMENDATION
  - HOW DOES INDUSTRY PARTICIPATE IN "TASK FORCES"?
- BOEING
  - STRONG OVERALL SUPPORT
  - NO INCENTIVES SHOULD BE NECESSARY
  - IMPACT ON BOEING WOULD BE POSITIVE

NASA FORUM/SYMPOSIUM

NASA COMMENTS (I)

- JOHNSON (GARMAN)
  - OVERALL GENERAL SUPPORT
  - REMEMBER "NASA BUYS MOST SOFTWARE"
  - NEED STRONG FOCUS AT HEADQUARTERS
- OSSA (McMAHON)
  - COST OF IMPLEMENTATION WILL CAUSE OBJECTIONS
  - MANDATES WILL NOT WORK
- JPL (THORNTON)
  - ... FULLY SUPPORT ALL RECOMMENDATIONS ...
  - MUST DO COST ANALYSIS
  - INCENTIVES FOCUS SHOULD BE "REUSE"
  - NEED "DEMONSTRATION" OF Ada ADVANTAGE
- LANGLEY (KUDLINSKI)
  - STRONG SUPPORT FOR ALL RECOMMENDATIONS
  - CENTRALIZED (HQ) TASK FORCE ESSENTIAL
  - IMPLEMENTATION SHOULD BE ACCELERATED

NASA FORUM/SYMPOSIUM

NASA COMMENTS (II)

• GODDARD (DALTON)	• ISSUE IS SOFTWARE ENGINEERING NOT Ada ... (MANDATE NOT NECESSARY) • ORGANIZATION IS KEY TO SUCCESS OF TRANSITION. NEED MORE THAN "TASK FORCE"
• KENNEDY (HAHN)	• SSE CONCEPT IS TOO BROAD • DO NOT MANDATE Ada (KSC IS "C" SHOP)
• DoD (KOPP)	• BASED ON DoD EXPERIENCE NASA SHOULD "GO FOR IT" • NASA CAN LEVERAGE EXTENSIVE DoD WORK
• OSO (CARRO)	• PROPOSED TRANSITION TO Ada TOO LONG • NEED COST STUDY AND COST CONTROL

NASA FORUM/SYMPOSIUM
NASA COMMENTS (III)

- OAST (SMITH)
  - Ada TOO IMMATURE TO MANDATE
  - NEED "GOOD SOFTWARE ENGINEERING..."
  - CONCERN FOR COSTS
- MARSHALL (AICHELE)
  - NO NEW STANDARDS
  - NMI 2410.6 IS SUFFICIENT
  - NO Ada MANDATE
- LEWIS (SCHUBERT)
  - CONCERN FOR COSTS
  - "MANDATE" WILL RECEIVE RESISTANCE

PREPARE "OFFICIAL" CENTER/OFFICE RESPONSES ①

STEP ③

- AMES
  - SOME SUPPORT (SMAP IS GOOD)
  - Ada "PRINCIPLE LANGUAGE" NOT "STANDARD"
  - LIMIT METRICS TO "MEANINGFUL DATA"
- GODDARD
  - FULL AGREEMENT AND SUPPORT (MANDATE Ada)
    - REALIZE EXCEPTION
  - NEED MORE THAN "TASK FORCES"
- JPL
  - SOME SUPPORT AND AGREEMENT
  - "SUPPORT Ada" NOT "MANDATE"
  - SEND MONEY (INCENTIVES) TO CENTERS
  - LEARN FROM JPL EXPERIENCES
- JOHNSON
  - FULL AGREEMENT AND SUPPORT (MANDATE Ada +)
  - MUST SUPPORT STRONG CENTRAL OFFICE (HQ)

"OFFICIAL" CENTER/OFFICE RESPONSES (II)

(CENTER/OFFICE LETTERS)

- KENNEDY
  - MANY "FINDINGS" ARE UNSUBSTANTIATED
  - NO MANDATE
  - "WE SUPPORT NEED FOR INCREASED TRAINING"
- LANGLEY
  - GO FOR IT
  - NEED MORE THAN "TASK FORCE"
- LEWIS
  - GENERAL SUPPORT
  - 3 VIEWS SUBMITTED (2 OF 3 STRONG SUPPORT)
  - EMPHASIS ON HQ OFFICE (S.E.)
- MARSHALL
  - PRODUCE "...SUITE OF STANDARD LANGUAGES..."
  - SOME SUPPORT AND AGREEMENT
  - MUST LOOK AT COST
- STENNIS
  - FULL AGREEMENT AND SUPPORT
  - EXTEND SCOPE OF Ada USE BEYOND "MISSION SOFTWARE"

"OFFICIAL" CENTER/OFFICE RESPONSES (III)

(CENTER/OFFICE LETTERS)

- Q (OSR M&Q)
  - FULL AGREEMENT WITH FINDINGS
  - NO POSITION ON Ada MANDATE
  - EMPHASIZES NEED FOR HQ ORGANIZATION (1 FOCUS)
- R (OAST)
  - Ada IS TOO IMMATURE TO MANDATE
  - NEED MORE STUDY
- T (OSO)
  - "ENCOURAGE... Ada" NOT A MANDATE
  - "... OUR PROJECT MANAGERS ARE COMPETENT TO SELECT APPROPRIATE ..."
  - INCREASE SCOPE OF SMAP - BUILD HQ ORGANIZATION
- SMAP
  - EXPAND ROLE OF SMAP
  - NO INDICATION OF SUPPORT OR AGREEMENT

"OFFICIAL" CENTER/OFFICE RESPONSES (IV)

(CENTER/OFFICE LETTERS)

• M (OSF)	• PREMATURE TO MANDATE Ada • PRODUCE "... PORTFOLIO OF RECOMMENDED RECOMMENDED STANDARD LANGUAGES ..." • SUPPORT TAILORABLE STANDARDS • INCENTIVES NOT NECESSARY
• E (OSSA)	• NO Ada MANDATE • "EACH PROJECT MANAGER WILL MAKE A DECISION ON THE LANGUAGE..." • STRONG SUPPORT FOR HQ OFFICE TO SUPPORT S.E.
• S (OSS)	• GENERAL INTEREST
• B/D/H/L/X	• NOT APPLICABLE

RESPONSE SUMMARY
(MY INTERPRETATION)

OVERWHELMING SUPPORT/ AGREEMENT FROM INDUSTRY	• MANDATE Ada • NASA-WIDE STANDARDS • INCENTIVES NOT NECESSARY
NASA (AS WELL AS INDUSTRY) HAS TREATED THE STUDY VERY SERIOUSLY	
WIDE AGREEMENT (NASA)	• NASA SHOULD ADDRESS MAJOR SOFTWARE TECHNOLOGY ISSUES • NEED OVERHAUL OF INFRASTRUCTURE • NASA-WIDE STANDARDS DESIRABLE
NASA SPLIT IN Ada "MANDATE"	• COST • MATURITY • INERTIA

DEVELOP NASA ACTION PLAN
STEP ④

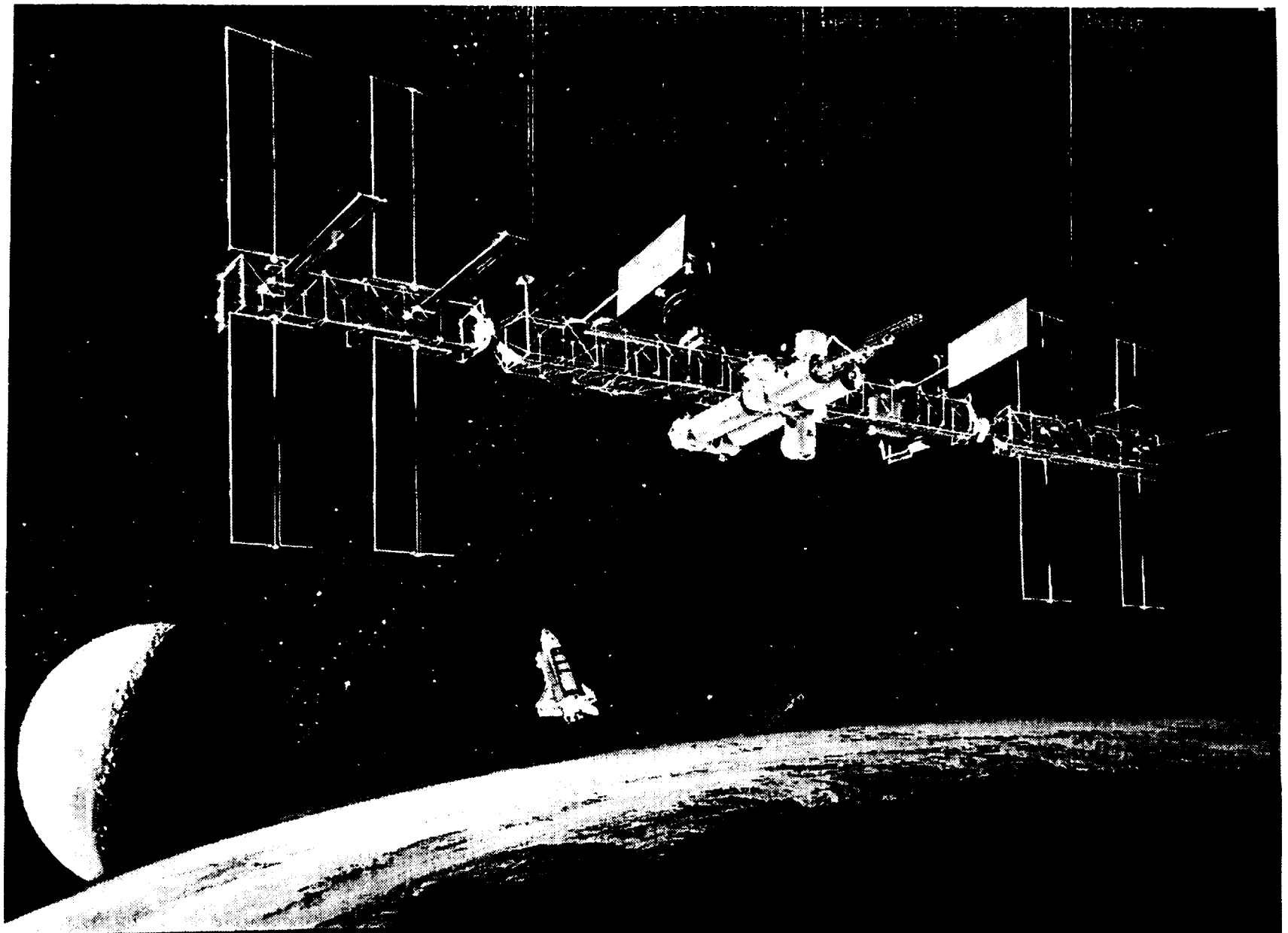
- RESPONSIBILITY OF IRM COUNCIL
  - AS STIPULATED BY FORMER CHAIR
  - ORIGINAL TARGET - FALL 1989
- BASED IN ASMAWG REPORTS AND REVIEW COMMENTS
- DEFINE STEPS/FUNDING/ORGANIZATION/TIMELINES
- ORIGINAL TARGET DATE - NOVEMBER 30, 1989

Ada IN NASA - POLICY AND DIRECTIONS
CURRENT (11/89) STATUS

- NO Ada POLICY YET IN EXISTENCE
  - 3 OF 4 STEPS COMPLETED
- ADMINISTRATION CHANGE HAS CAUSED SOME UNDERSTANDABLE DELAYS
  - STRUCTURE OF COUNCILS BEING REVIEWED
- 4 NASA CENTERS HAVE PROPOSED FULL COMMITMENT TO Ada
- HAVE BEEN SOME (FEW) ADDITIONAL PROJECTS COMMITTING TO Ada (E.G., STGT)
- SOME MOMENTUM BEING LOST
- POSSIBILITY OF FORMULATING "POLICIES" ON CENTER BASIS

"ADA AND THE SPACE STATION"

R. Nelson, Space Station Freedom Program Office



SPACE STATION FREEDOM UPDATE

R. Nelson
Space Station Freedom
Program Office

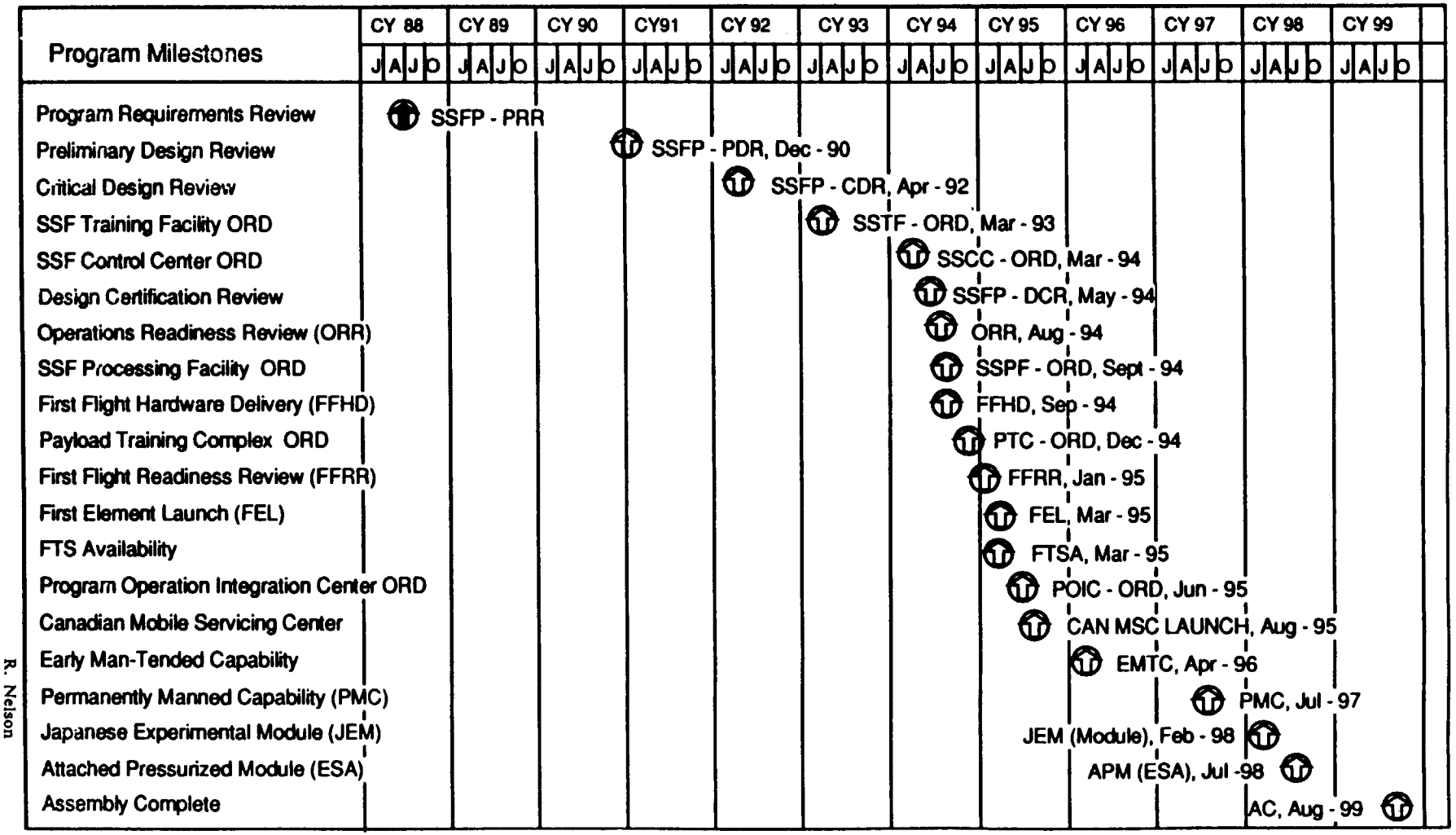
Presentation Outline

- Program Status/Overall Milestones
- Software Activities
  - Requirements Reviews
  - Architecture Task Team
  - Integration and Verification
  - Risk Management
  - Metrics Tracking
  - Ada Analytical Studies

National Aeronautics and Space Administration

Space Station Program Controlled Milestones

FREEDOM



SSFPO 342

11/07/89 M/AK SSO-8901997

BASELINE DOCUMENTS

LEVEL A SOFTWARE MANAGEMENT POLICIES

LEVEL I

PROGRAM DESIGN REQUIREMENTS DOCUMENT (PDRD)

ARCHITECTURE CONTROL DOCUMENTS (ACDs)
(One for each Distributed System)

LEVEL II

SYSTEM REQUIREMENTS DOCUMENT
(One for each Work Package)

LEVEL III

CONTRACT END ITEM SPECIFICATION (CEIs)
(One for each system and element)

SOFTWARE REQUIREMENTS SPECIFICATIONS (SRSs)

INTERFACE REQUIREMENT DOCUMENTS (IRDs)

LEVEL IV

R. Nelson
Space Station Freedom
Program Office

BASELINE REQUIREMENT CHARACTERIZATION

- **LEVEL A SOFTWARE MANAGEMENT POLICIES PROVIDES BASIC LIFE CYCLE REQUIREMENTS**
  - DOD 2167 BASED
  - IMPLEMENTED IN PRIME CONTRACTS (for the most part))
- **Ada BASELINE REQUIREMENT**
  - LEVEL I DIRECTIVE
  - PDRD, SRDs, CEIs
- **GENERAL SOFTWARE REQUIREMENTS IN PDRD**
  - USE OF SSE FOR DEVELOPMENT

SOFTWARE REQUIREMENTS REVIEW TOPICS

Software Requirements Review Topics

- CSCI hierarchy
- Functional overview of the CSCI
- Overall CSCI performance requirements, including those for execution time, storage requirements, reserve requirements, through-put, database access
- Results of analyses and prototyping which support the requirements of the CSCI
- Requirements traceability to governing documents
- Testing requirements that identify applicable levels and methods of testing for the software requirements that comprise the CSCI
- Quality factor requirements, i.e., reliability, efficiency, maintainability, testability, portability, interoperability
- Failure and fault tolerance requirements
- Milestones schedules for the development and test of the CSCIs
- Software safety risk assessment
- Software criticality assessment and associated verification planning

SOFTWARE REQUIREMENTS REVIEW TOPICS

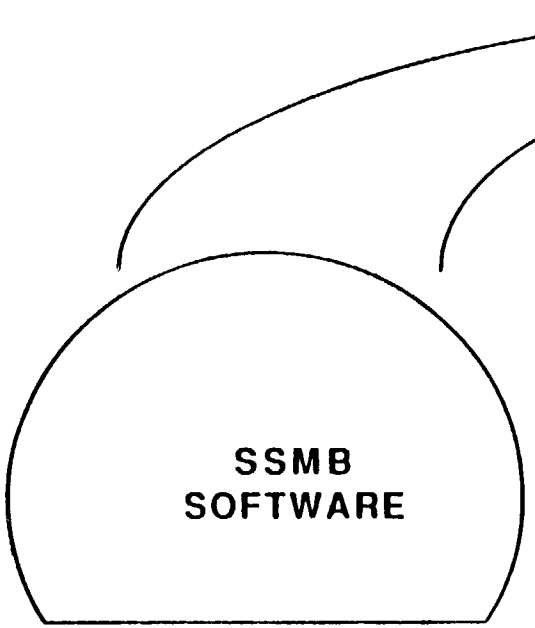
Software Requirements Review Topics (continued)

- Certification and integration planning/scheduling
- Software phasing resulting from the assembly sequence
- Requirements resulting from the assembly sequence
- Software Support Environment (SSE)/Software Production Facility (SPF) utilization
- Data flow between each of the software functions that comprise the CSCI
- Interface requirements between the CSCI and all other hardware and software configuration items both internal and external to the system
- Simulation/model requirements including needed levels of fidelity and development schedules
- User Support Environment *USE) utilization

SRR SCHEDULE

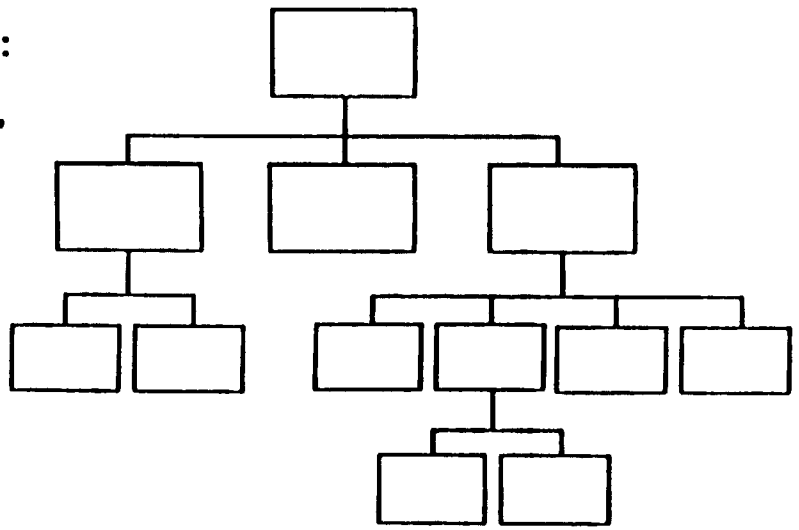
DATE	SYSTEM	LOCATION
11/07/89	OMGA	Houston, TX
11/09/89	C&T	Camden, NJ
11/14/89	Mobile Transporter	Huntington Beach, CA
11/16/89	MDD-Firmware	Phoenix, AR
11/28/89	Attitude Control	Clearwater, FL
12/05/89	GN&C	Huntington Beach, CA
12/12/89	Airlock & EVAs	Houston, TX
12/12/89	Docking & Berthing	Huntington Beach, CA
12/19/89	UL Equip Attach Sys	Huntingotn Beach, CA
01/02/90	Propulsion & Mast	Huntington Beach, CA
01/19/90	EMU	Houston, TX
01/28/90	FMS	Huntington Beach, CA

REQUIREMENTS FOR SSMB SOFTWARE ARCHITECTURE DEVELOPMENT

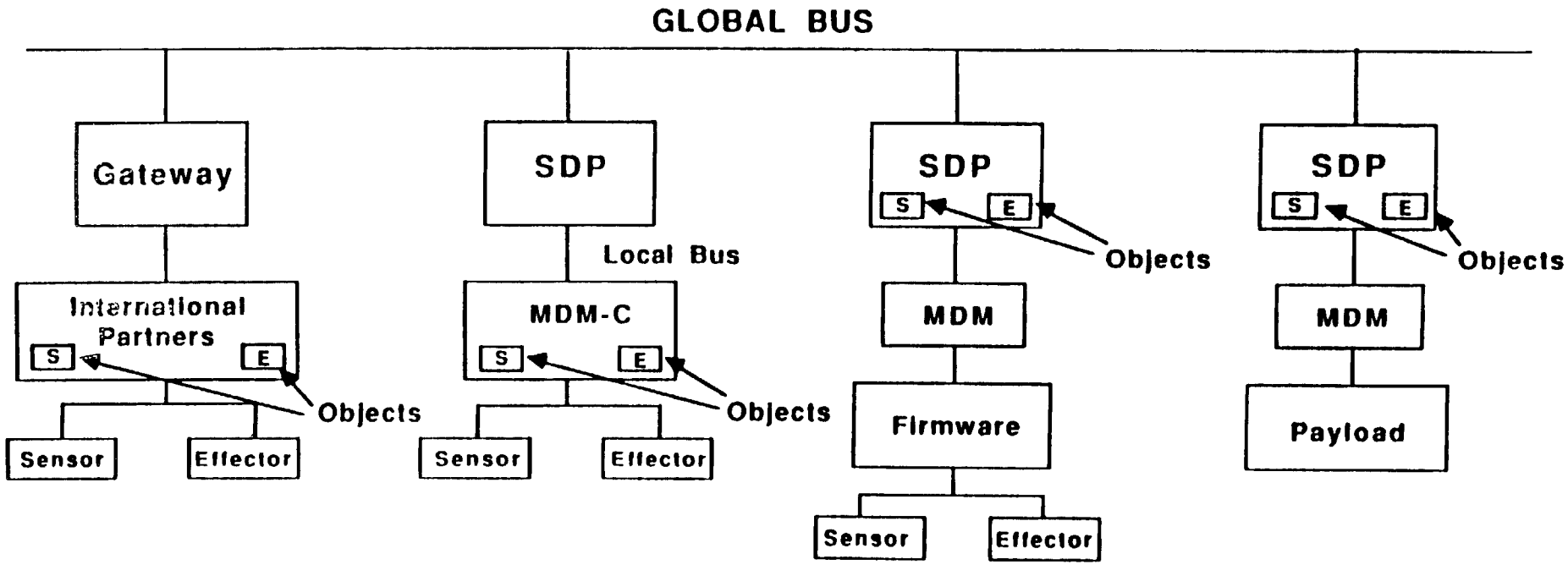


REQUIREMENTS NEEDED:

- Rules for "Building Block" definition - What are CSCIs?
- Definition of the relationship between the "Building Blocks" - What is the Command and Control Hierarchy
- Definition of the Data Structure - How is the data organized? How does a building Block access it?

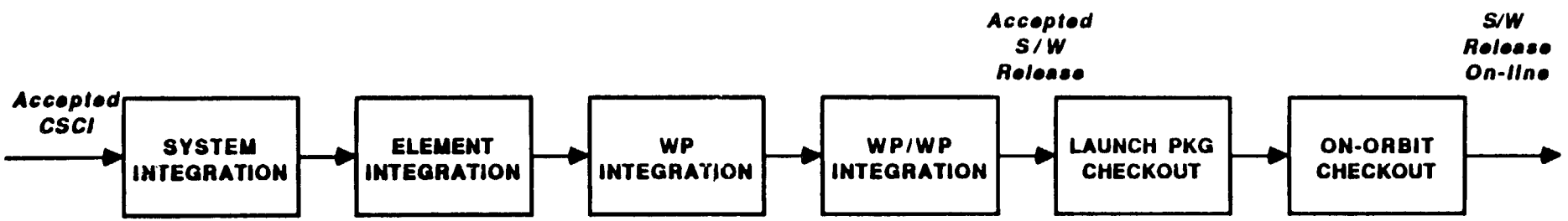


SSMB GLOBAL DATA STRUCTURE ORGANIZATION AND ACCESS CRITICAL TO CSCI DEVELOPMENT



- Object oriented data structure required
- Access time for objects will have impact on real-time performance
- Location of the objects impacts lower level update] traffic rates

SOFTWARE I&V PROCESS



CSCI/CSCI
- Integrate
- Verify
- Validate
- Accept

CSCI/HWCI (SYS)
- Integrate
- Verify
- Validate
- Accept

SYS/SYS
- Integrate
- Verify
- Validate
- Accept

SYS/ELEM (ELEM)
- No S/W Activity

ELEM/ELEM (WP)
- Integrate
- Verify
- Validate
- Accept

WP/WP (LP)
- Integrate
- Verify
- Validate
- Certify

LP CHECKOUT
- Checkout

ON-ORBIT CHECKOUT
- Checkout

System = Distributed System OR Element-Unique System

R. Nelson
Space Station Freedom Program Office

CCS
10/17/89

Figure 1

POTENTIAL
SYSTEM / SYSTEM SOFTWARE INTERFACES

SYSTEMS / SYSTEMS	WP	ECLSS	MAN SYSTEMS	INTERNAL TCS	ELEMENT MGRS	INTERNAL A/V	INTERNAL EPS	MECHANISMS S/W	FMS	AIRLOCK SERVICES	C&T	TCS	PROPULSION	GN&C	OMA	DMS	EVA	MT S/W	MECHANISMS S/W	CHeCS	FTS	APAE	EPS
	WP	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	4
ECLSS	1		X	X	X	X	X		X	X				X	X	X	X			X			
MAN SYSTEMS	1	X			X	X	X		X	X						X	X			X			X
INTERNAL TCS	1	X			X							X				X							
ELEMENT MGRS	1	X	X	X	X	X	X	X	X	X	X	X			X	X				X			X
INTERNAL A/V	1	X	X		X		X			X	X					X	X						
INTERNAL EPS	1	X	X	X	X	X		X	X							X							X
MECHANISMS S/W	1				X		X									X							
FMS	1	X	X		X		X						X	X	X	X							
AIRLOCK SERVICES	1	X	X			X										X	X						
C&T	2				X	X				X				X	X	X	X		X		X	X	X
TCS	2			X	X								X	X	X	X						X	
PROPULSION	2								X			X		X	X	X							
GN&C	2	X							X		X	X	X		X	X	X		X			X	X
OMA	2	X			X				X		X	X	X	X		X	X	X			X	X	X
DMS	2	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X
EVA/AIRLOCK	2	X	X			X			X	X	X			X	X	X				X			
MT S/W	2														X	X							X
MECHANISMS S/W	2													X		X							
CHeCS	2	X	X		X										X	X							
FTS	3										X	X			X	X							X
APAE	3										X	X		X	X	X							X
EPS	4		X		X		X				X			X	X	X		X			X	X	

= WP / WP SOFTWARE INTERFACE

X = SYSTEM / SYSTEM SOFTWARE INTERFACE

CCS
10/17/89

R. Nelson
Space Station Freedom
Program Office

SOFTWARE RELEASES PER ASSEMBLY SEQUENCE

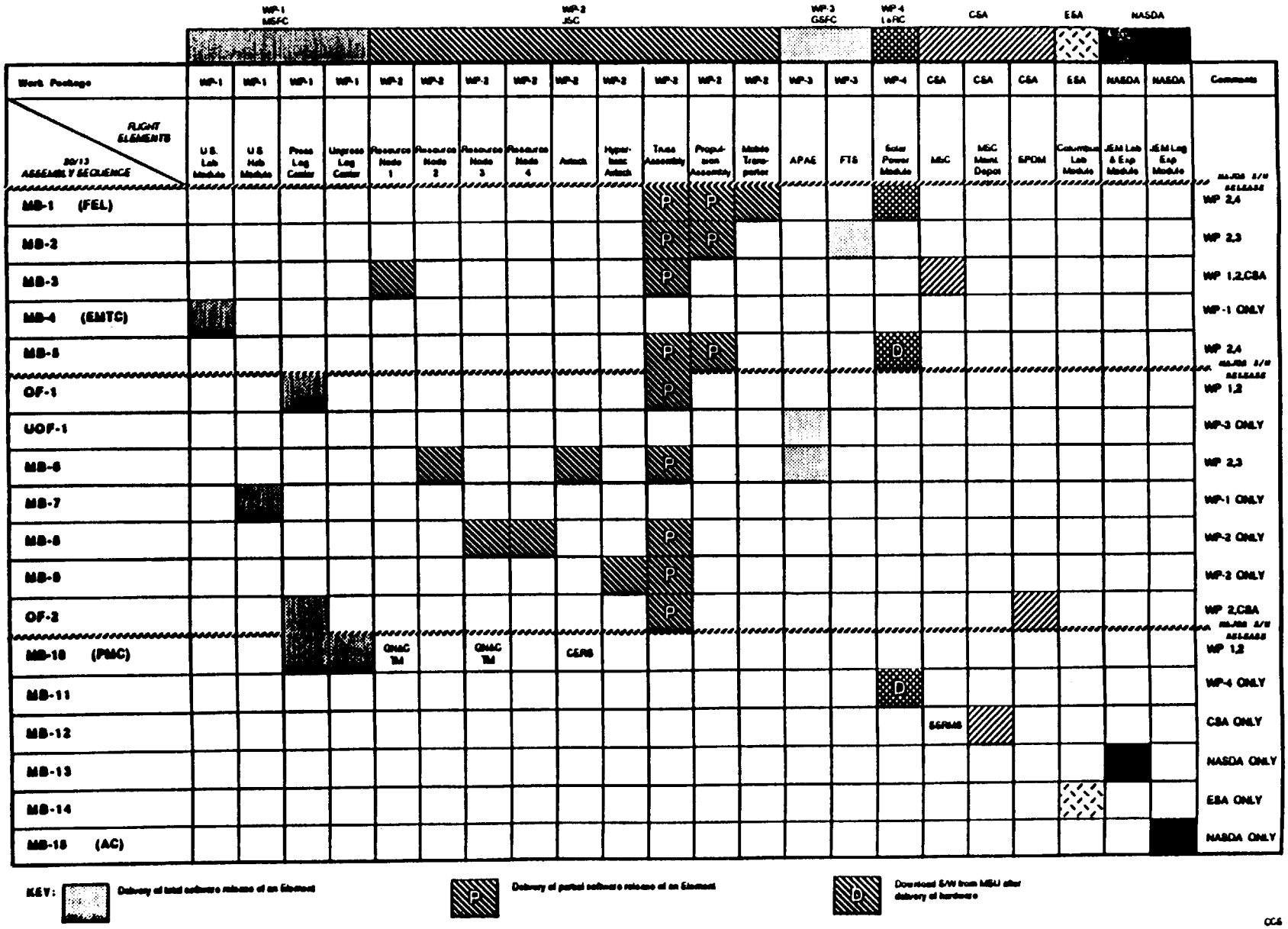


Figure 6

CCS
10/17/88

Software Risk Management

Accomplishments

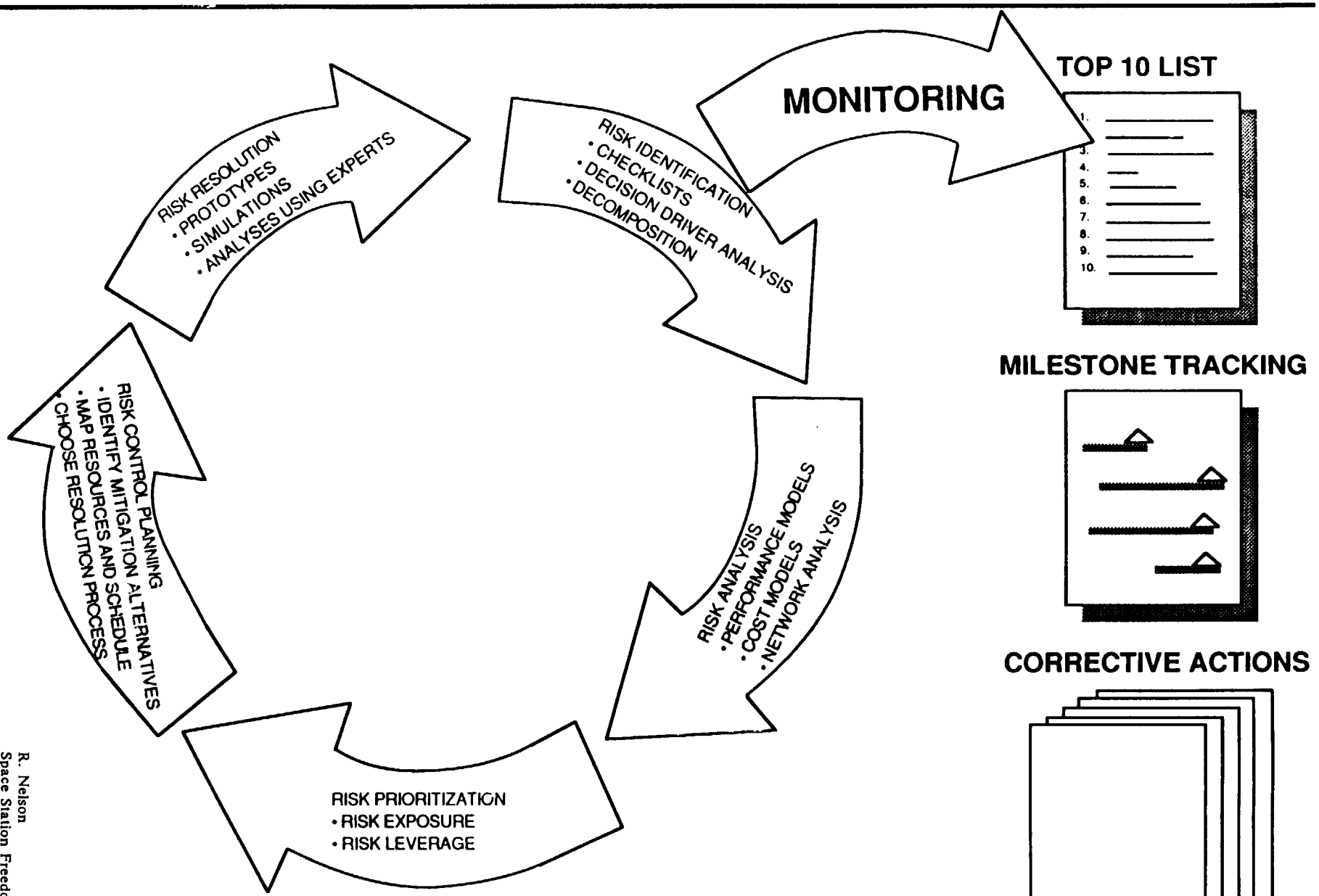
Risk Management Program implementation recommended by NRC to assess and control risks

Software Risks have been identified in Level III Software Management Plans

Further identification and prioritization has been performed by Level II and TRW

Software Risk Management Plan has been developed and presented to the SSFP Software Managers

RISK MANAGEMENT = RISK ASSESSMENT+ CONTROL + MONITORING



R. Nelson
Space Station Freedom Program Office

TOP 10 SSF SOFTWARE RISK ITEMS IDENTIFIED BY TRW

RANK	RI NUMBER	TITLE
1	R026	SSF Dependency on Data Management System (DMS)
2	R013	Inadequate Ops Concept to Support Architecture & Funct Decomposition
3	R010	Unrealistic schedules for Software Requirements Spec development
4	R005	Incorrect Schedule Data due to Underestimate of S/W Size and scope
5	R001	Lack of Level II Software Managment Personnel
6	R007	Inadequate Overall I&V for Staged Software Buildup
7	R004	Lack of Experienced SW Engineers with Flight Software Background
8	R030	Real-Time DMS Performance Shortfalls
9	R020	SSF S/W Functional Requirements Evolution
10	R033	Operating System Selection - real time shortfall

SSFP FROM A SOFTWARE PERSPECTIVE

FLIGHT SOFTWARE

SPACE STATION FREEDOM

9 DISTRIBUTED SYSTEMS
11 ELEMENTS
FTS
INTERNATIONAL PARTNER

} 2.4+ MSLOCs

GROUND - 8.0+ MSLOCs (including sims and support)

MSFC

- MPS — 1229 K SLOCs
- WP-1 GROUND — 191 K SLOCs

KSC

- TCMS — 1432 K SLOCs

GSFC

- WP-3 GROUND — 810 K SLOCs

INTERNATIONALS

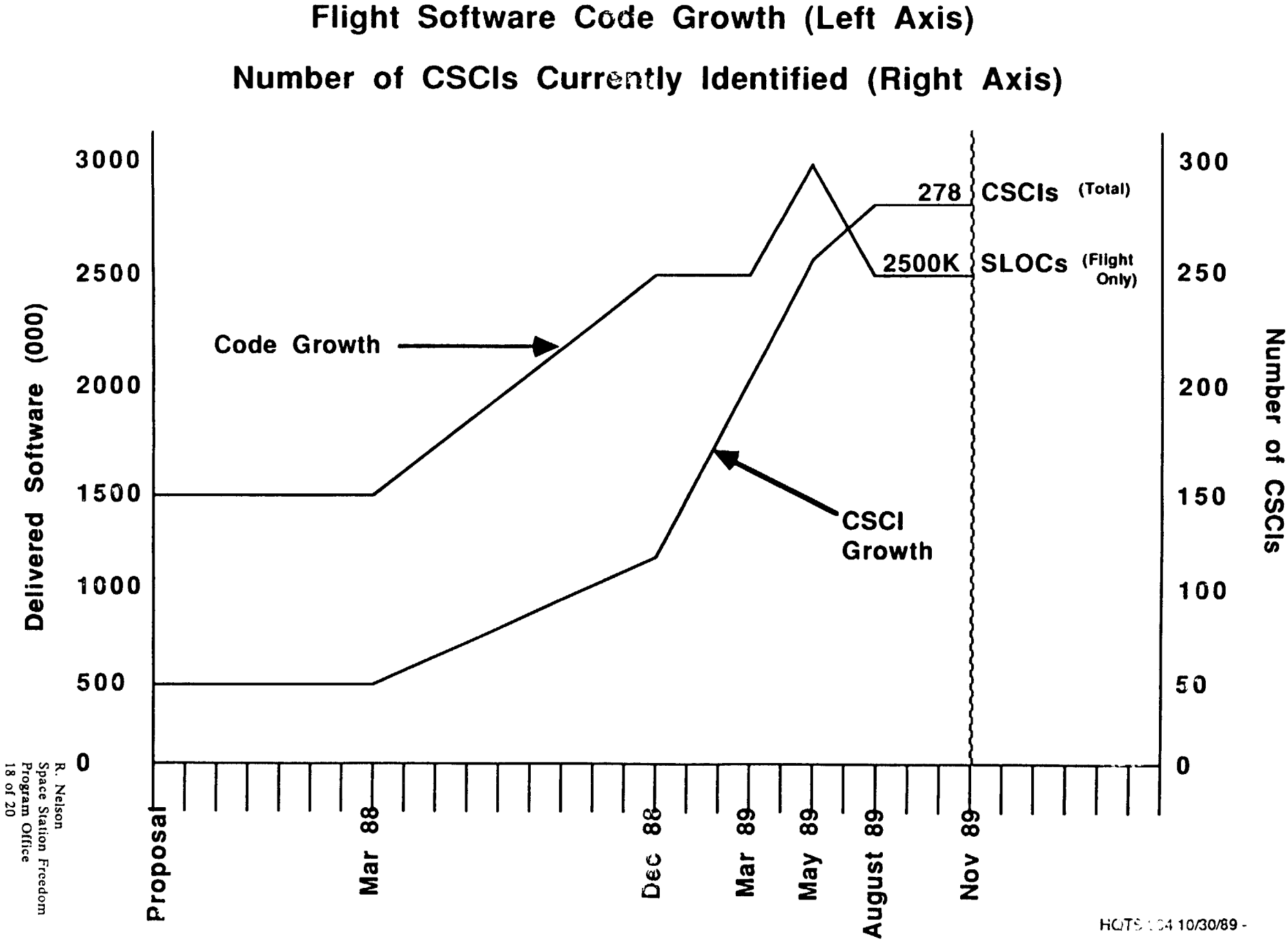
JSC

- SSCC — 1300 K SLOCs
- SSTF — 1350 K SLOCs
- WP-2 GROUND — 77 + K SLOCs
- OPAS — 1300 K SLOCs

- JAPAN (?)
- ESA (?)
- CANADA (?)

LeRC

- WP-4 GROUND — 387 L SLOCs


Flight Software Code Growth (Left Axis)
Number of CSCIs Currently Identified (Right Axis)
Delivered Software (000)
3000
2500
2000
1500
1000
500
0
Number of CSCIs
300
250
200
150
100
50
0
278 CSCIs (Total)
2500K SLOCs (Flight Only)
Code Growth
CSCI Growth
Proposal
Mar 88
Dec 88
Mar 89
May 89
August 89
Nov 89
R. Nelson
Space Station Freedom
Program Office
18 of 20
HQTS 104 10/30/89 -

SSFP SOFTWARE SIZE ESTIMATE BY SOFTWARE TYPE (KSLOCs)

	SOFTWARE TYPE				NUMBER OF CSCIs			
CENTER	FLIGHT	SIMULATION	GROUND	SUPPORT	FLIGHT	SIMULATION	GROUND	SUPPORT
MSFC	1039	156	1229	35	61	4	?	11
JSC	1139	76	3950	1	119	39	?	3
GSFC	230	67	500	242	3	4	9	4
LeRC*	49	100	37	250	3	1	2	4
KSC			803	629			10	1
TOTAL by TYPE	2457	399	6519	1157	186	48	21	23
10532 KSLOCs TOTAL					278 TOTAL CSCIs			

* POST SCRUB

? INSTITUTIONAL GROUND FACILITY CSCIs NOT CURRENTLY IDENTIFIED

I&V 006 11/8/89

Should there be a standard for the conversion of Ada source lines of code to computer memory usage?

- 24 used by most SSFP contractors, DEC
- Virginia Castor, Spec. Asst. for Software and Computer Technology, DoD warns that code expansion rates differ among compilers with greater than an order of magnitude range
- 14 Alsys Drhystone
- 53 to 86 GSFC Attitude Simulators (627 on VARS Simulator)

Recommendation: No SSFP-wide standard. Base sizing estimates on utilization of Ada constructs, specific application domain and compiler to be used.

"SOFTWARE SUPPORT ENVIRONMENT (SSE)

PROGRAM STATUS"

F. Barnes, Lockheed

2ND NASA ADA USERS SYMPOSIUM

SPACE STATION FREEDOM SOFTWARE SUPPORT ENVIRONMENT (SSE) STATUS AND CHALLENGES

F. N. BARNES

30 NOVEMBER 1989

CONTENTS

SSE PROJECT SCHEDULE

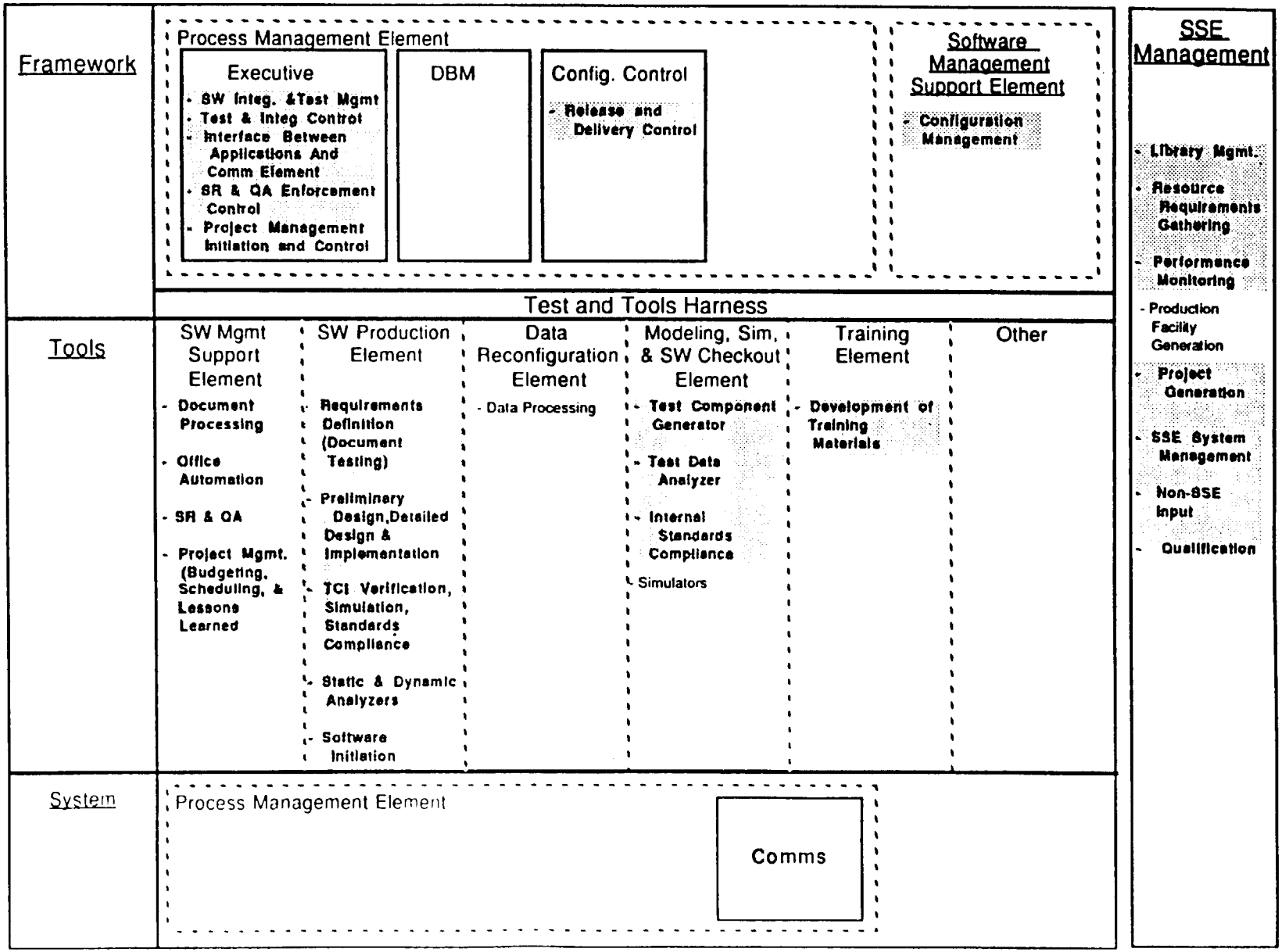
REVIEW OF SOME RECENT CHALLENGES

CURRENT AND UPCOMING CHALLENGES

F. Barnes
Lockheed

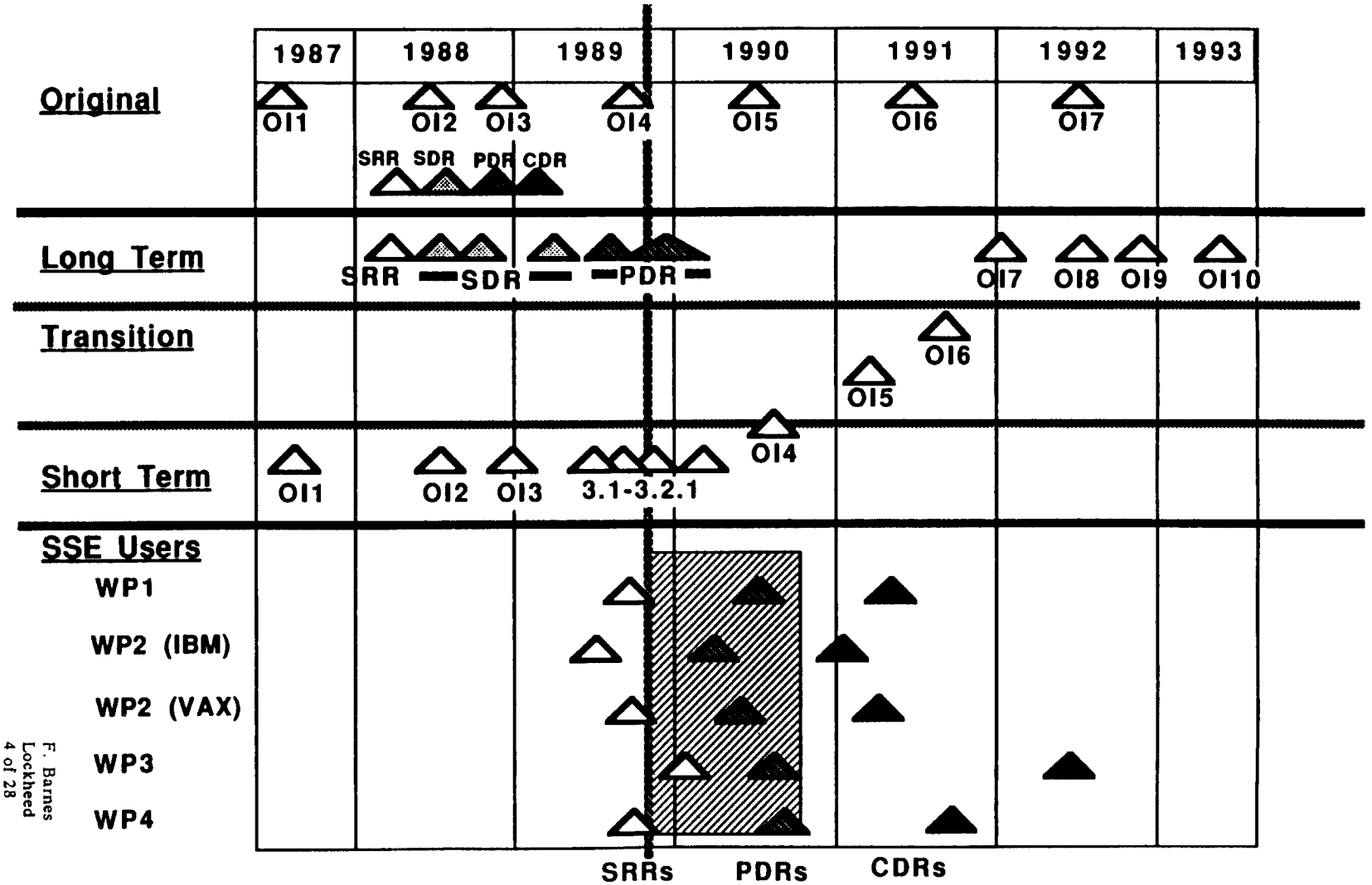
SSE PROJECT OVERVIEW

SSE Functional Capabilities



STATUS AND CHALLENGES

SSE PROJECT SCHEDULE



F. Barnes
Lockheed
4 of 28

STATUS AND CHALLENGES

SSE SYSTEM PRELIMINARY DESIGN REVIEW

BY PRE-ARRANGEMENT WITH NASA (LEVELS II AND IV) WE PLANNED A HIGH LEVEL PDR FOR AUGUST; COMPLETE DETAILS TO FOLLOW

DRLI 58, THE SSE ARCHITECTURAL DESIGN, PROVIDED A SOUND FOUNDATION FOR CONTINUING THE DESIGN OF THE FINAL SSE

IT INTENTIONALLY DID NOT PROVIDE THE LEVEL OF DETAIL TYPICALLY FOUND AT A PDR

PDR WAS CONSIDERED A SUCCESS, BUT LEFT THE COMMUNITY WITH MANY UNANSWERED QUESTIONS

F. Barnes
Lockheed
5 of 28

DETERMINATION OF OI CONTENT PHASING

HISTORY

- 2/89: SSE PROPOSED FUNCTIONALITY PHASING
- 5/89: WPS EXPRESS NEED FOR MORE SOONER
- 8/89: WPS SUBMIT NEW REQUESTS
- 8/89: SSE PROPOSES NEW PHASING
- 9/89: NASA PROPOSES PHASING; WPS DISAGREE
- 9/89: NASA "BASELINES" OI 4 CONTENT
- 10/89: NASA LEVEL 2 HOSTS OI 4/5/6 CONTENT MEETING
  - OI 4 REOPENED
  - SOFTWARE RELEASE 3.2.1 CREATED
  - OI 5 "ACCEPTED"
  - OI 6 DEFERRED

F. Barnes
Lockheed

STATUS AND CHALLENGES

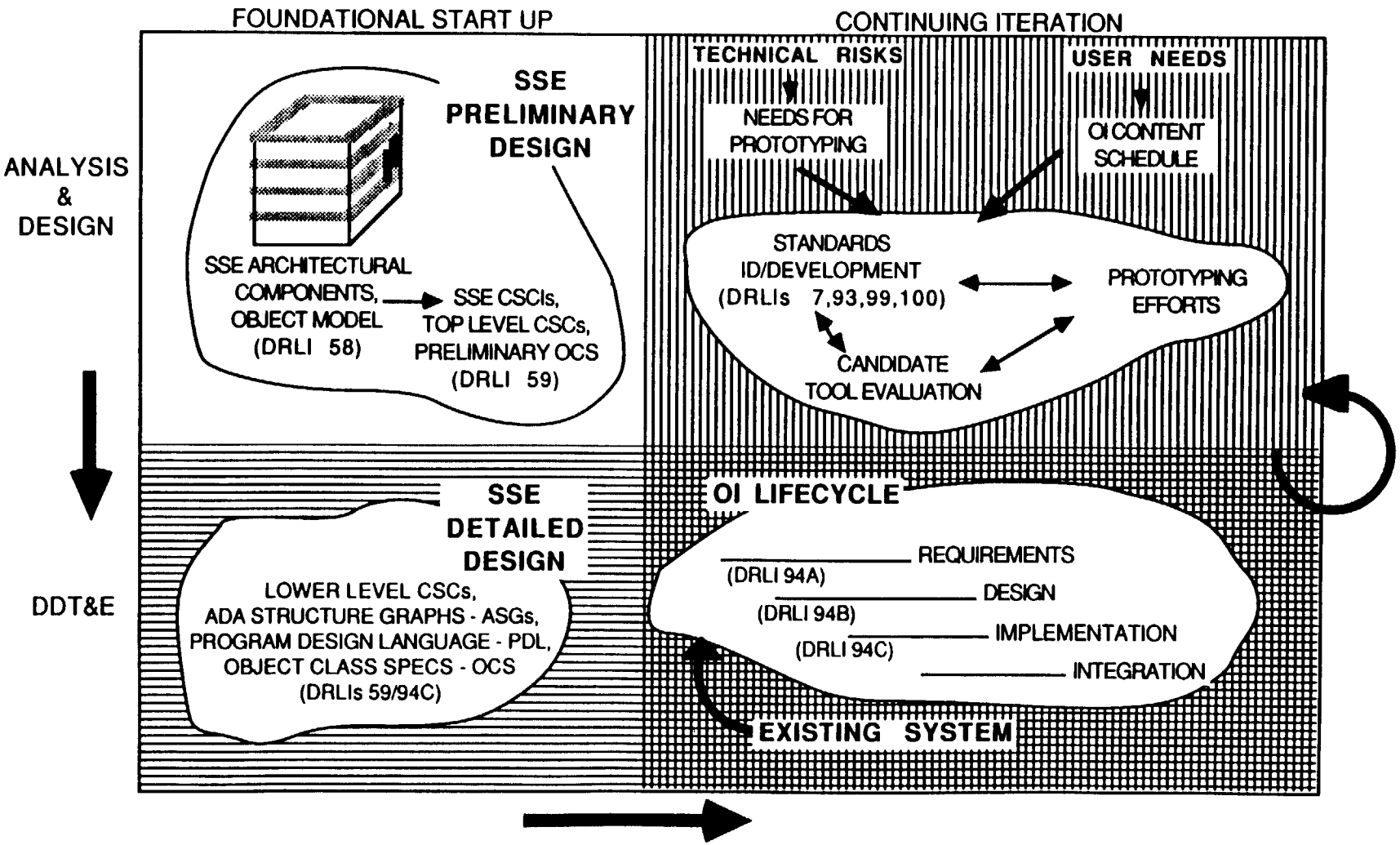
SYSTEM IMPLEMENTATION REVIEW

- FOCUS IS ON SSE OF 1992
- 4 REVIEWS OVER 3 MONTHS (12/89 THROUGH 2/90)
- LEVEL OF DETAIL -> LLCSC

F. Barnes
Lockheed
7 of 28

STATUS AND CHALLENGES

THE FOUNDATION OF START UP ANALYSIS, DESIGN AND DDTE ACTIVITIES FLOW INTO AN ONGOING ITERATION OF CONTINUING ANALYSIS DESIGN AND DDT&E ACTIVITIES



SPACE STATION FREEDOM
SOFTWARE SUPPORT ENVIRONMENT (SSE)

Ada COMPILER EVALUATION

STATUS REPORT

F. Barnes
Lockheed

D. L. B(Ada)L
10/4/89

AGENDA

PURPOSE:

CODE MANAGEMENT ENVIRONMENT:

GENERAL APPROACH:

COMPILER EVALUATION DETAILS:

- CROSS-COMPILERS
- HARDWARE
- PERFORMANCE SYSTEMS

SCHEDULE:

SUMMARY:

SSE SYSTEM PROJECT

PURPOSE:

TO PERFORM A TECHNICAL EVALUATION OF Ada COMPILERS FOR HOST, WORKSTATIONS, AND ON EMBEDDED REAL-TIME TARGET COMPUTERS FOR USE ON THE SSE SYSTEM. THIS REPORT PROVIDES STATUS AND TECHNICAL DETAILS ASSOCIATED WITH THIS CURRENT EFFORT.

THE PROGRAMMING-IN-THE-LARGE SCENARIO

The following is a description of the approach for handling the final tested Ada source code on the SSE with respect to how that code is cross-compiled and tested on the target computers.

A brief explanation of how (a proposed method) that code reaches the state where it is ready to be cross-compiled will be given first. The host-computer, either a VAX-8820 or IBM 3090, will have the software that coordinates development activity, including configuration management and building low-level integration test scripts. The host will be the focal point of the coding effort where work (i.e., permission to code a module) is checked-out and checked-in. After a developer receives permission from the host configuration management software to proceed with coding a module (could be more than one package), he will start the Ada coding on his workstation (either on Apollo, MacIntosh II, or IBM PS/2 using an Ada development tool (a smart editor such as Xinotech's Program Composer). A mechanism will be in place that allows current copies of the package specifications that the developer's module "withs' to be downloaded from the host to his workstation. This will allow the developer to build an Ada module that is semantically and syntatically correct by using the Ada compiler that is resident on the workstation. This should off-load some of the compilation load from the host to the workstation. After the developer has finished coding (has semantically and syntatically correct code) and has done some preliminary, non-run-time testing, he will check the module back into the host configuration management software. There a test build script will be developed and the code will be tested using drivers and/or a debugger either on the host or on a workstation. The code is iterated through this process until it is judged to be correct for this round of testing and is integrated with larger and larger code segments until final tested (for the host) code is produced.

During this testing process, some or all of the code will be downloaded to a non-bare target machine, that is a workstation(s) with an Intel 80386 processor, a POSIX compliant operating system and an Ada compiler. Some of these type of machines will be dedicated for this purpose and will be accessed by the developer/tester via an X-Windows interface and executable code will be built at this workstation using the resident Ada compiler. The code then will be run and tested there (still using the X-Windows interface that allows "logging in" from a remote workstation). This additional testing should identify some of the quirks in the executable code that are specific to an 80386 processor (which is the same as final bare target machine). It may be that some part of the deliverable code will actually run on this type of target (such as code meant for the crew console), in which case final testing could be completed using these types of workstations.

However, approximately only 2% of the code will ultimately be intended for bare-machine targets, that is computers without a conventional operating system. This is why you need an additional step that includes a way to build code for the target on a machine with an operating system and compiler by cross-compiling that code for an intended target (the compiled code will not be run on the machine compiling the code). Another program running on the remote machine (not the target) is needed to load the final executable binary code into the target machine's memory, to remotely start execution on the target and to remotely debug the code running on the target.

This approach produces executable code that is bit-for-bit compatible for the same source code whether that code was produced in a SPF with either an IBM or VAX host architecture.

Case Design Interface Format (CDIF) provides interoperability among the workstations and host (framework).

Tools like the Composer, X-Windows, AdaMat, and CDIF provide a robust Programming-In-The-Large environment.

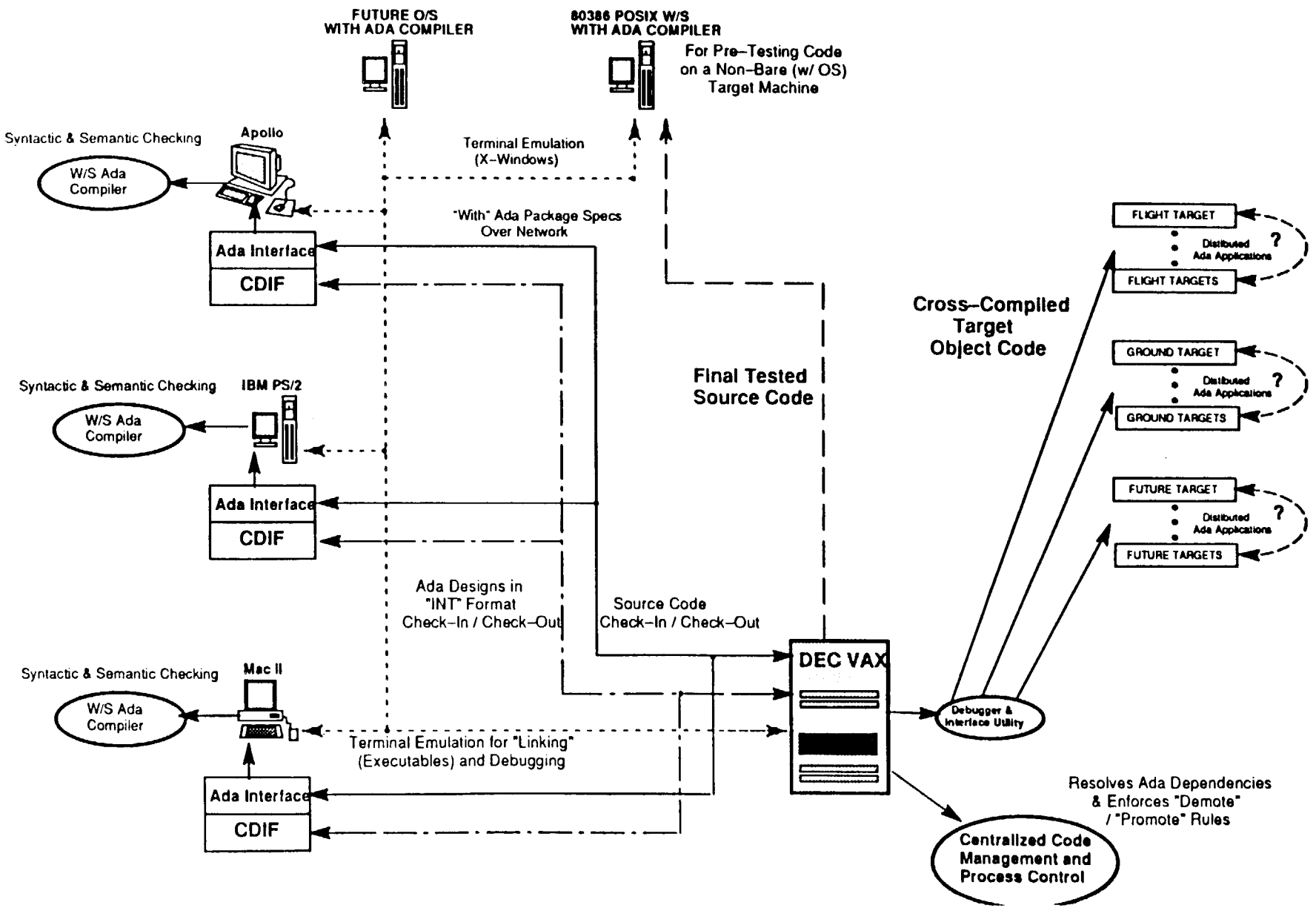
DEFINITION:

Programming-In-The-Large (PITL) is defined as the implementation of large scale software/hardware systems for both non-real-time and embedded real-time systems by many programmers in de-centralized areas using a distributed software implementation environment that conforms to the APSE Stoneman requirements while complying with well-established procedures, methods, and standards.

To accomplish this requires robust and very user-friendly tools such as Adamat, X-Windows, CDIF, Smart Editors, etc., that work in harmony and are completely interoperable within the system. A PITL environment will also include a reusable library and all the associated schemes used for the rapid retrieval, accessing, and browsing methods.

In order to provide the SSE with a robust and efficient PITL requires use of the most efficient and reliable Ada base and target compilers.

CODE MANAGEMENT ENVIRONMENT



SURVEY OF CANDIDATES:

As of June 1989, over 260 Ada compilers have been validated. The best form of information to determine the vendor and type of computers that these compilers were validated on comes from the "Ada Information Clearing House" newsletter. Another source is the "Language Control Facility Ada–Jovial" newsletter.

EVALUATION CRITERIA:

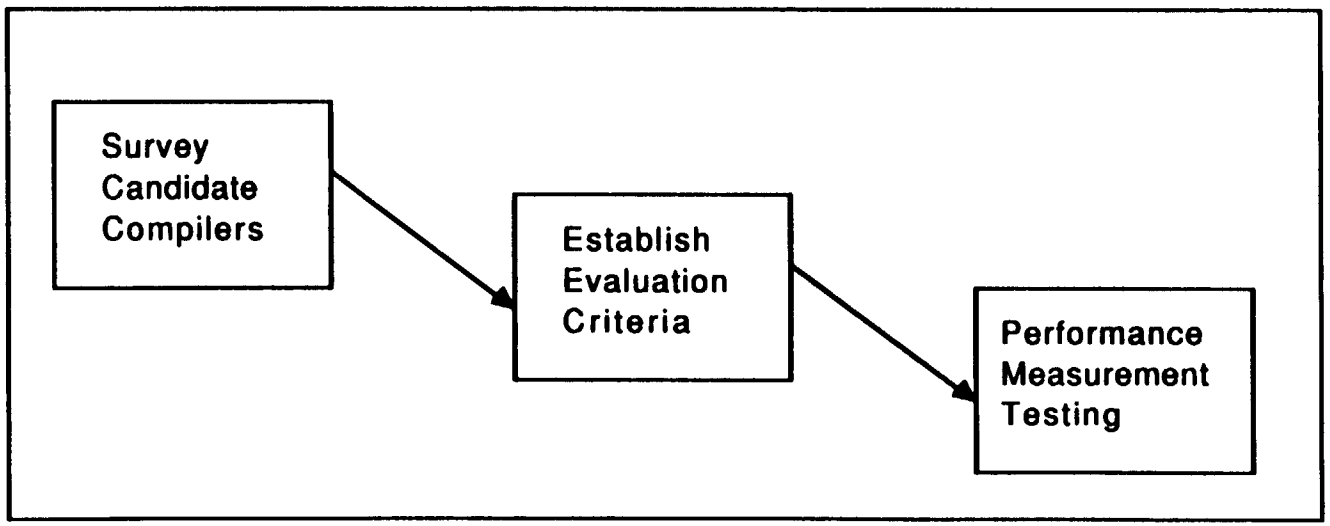
This task was accomplished by using the European Ada criteria guidelines, the ARTEWG CIFO, CRID and the SSE JSC 30500 SSE requirements documentation. This criteria was created by the SSE with the support of the System Environment Working Group (SEWG) that consists of JSC NASA, WP2, and IBM personnel. It includes the embedded real–time target, features, and issues.

PERFORMANCE SYSTEMS:

The following benchmark and performance measurement systems will be used for the cross–compiler evaluation:

1. Performance Issue Working Group (PIWG)
2. Ada Compiler Evaluation Capability (ACEC)
3. JSC Avionics Test Suite
4. Hard Ada Real Time Test Suite (HARTSTONE)
5. SSE Ada Test Suite (SATSTONE) – to be generated
6. In_Circuit_Emulator (ICE) and Software Analyst Workstation (SAW)

APPROACH:


Survey
Candidate
Compilers
Establish
Evaluation
Criteria
Performance
Measurement
Testing

Work that has been accomplished is represented by solid lines. Work to be accomplished is indicated with dotted lines.

MAINFRAME SUPPLIERS:

1. Tartan, Telesoft, Verdix, and DEC Ada were evaluated on the Vax 8820.

2. Telesoft, Alsys, and Intermetrics were evaluated on the IBM 3090-150E mainframe. Alsys was selected and a technical sole source justification was written.

3. Rational was evaluated. The Rational system is on SSEDF with an option to purchase one more.

WORKSTATION COMPILERS:

The following compilers were evaluated on the SSEDF workstation:

1.	MAC II	Meridian
2.	IBM PS2/60	Alsys, Janus and Meridian
3.	Apollo	Alsys, Verdix (Later)

Alsys is currently the baseline for Apollo and PS2/60. Alsys, Telesoft, and Meridian on the native MAC II to be evaluated later this year.

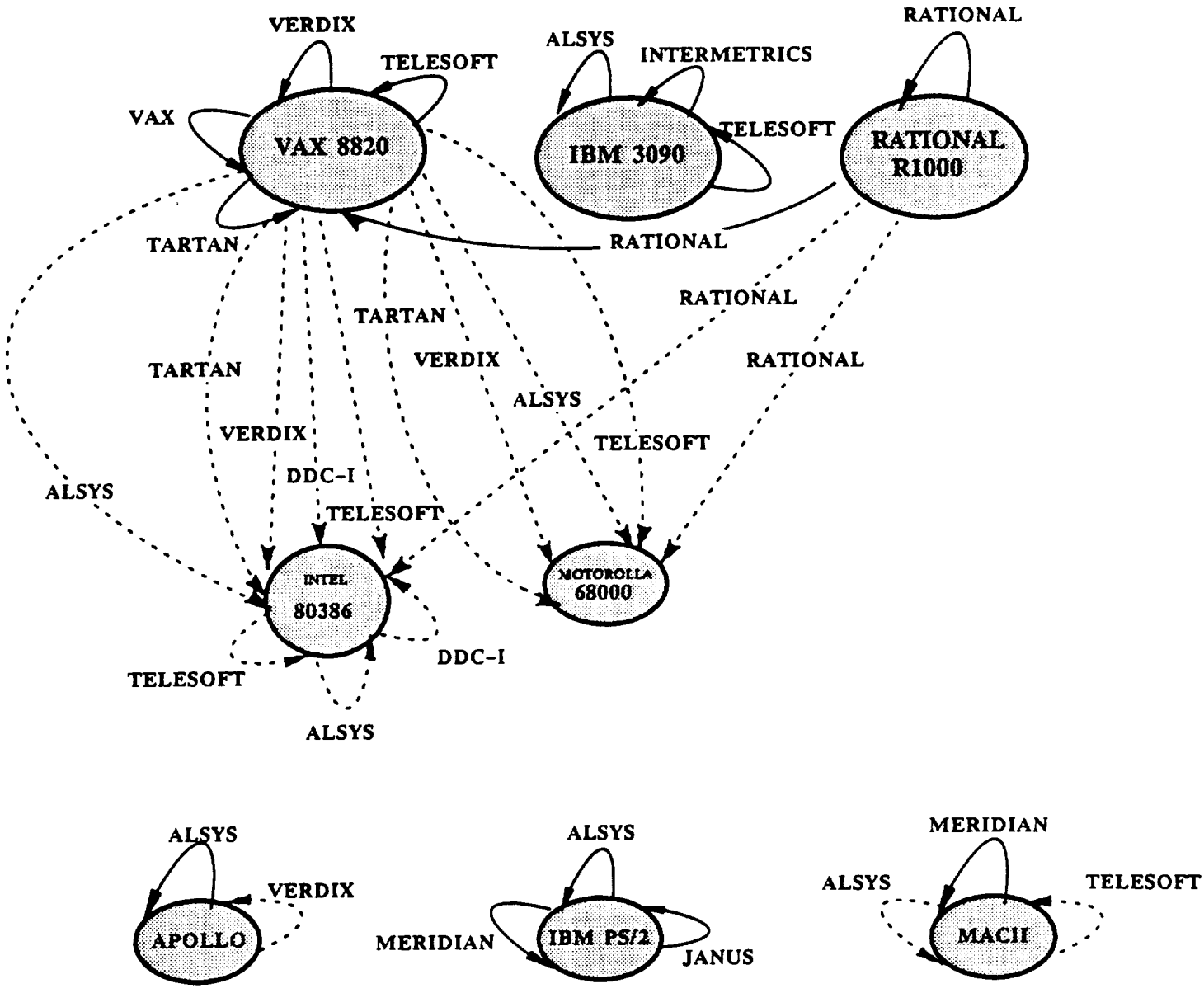
CROSS-COMPILERS:

The following cross-compilers will be evaluated from Vax 8820 to Target:

Vax 8820 to target 80386 - Alsys, Telesoft, Verdix, Tartan, DDCI

On the 680XX target, data is being gathered from other resources who already have performed an evaluation on the 68K.

ADA BASE AND CROSS-COMPILER EVALUATION REPORT



This figure shows the hardware configuration of the 80386 "bare" target test facility. It includes a PS2/80-111 to serve as the target. Also included is a PS2/60 and IBM AT to be used for the running of performance measurement systems.

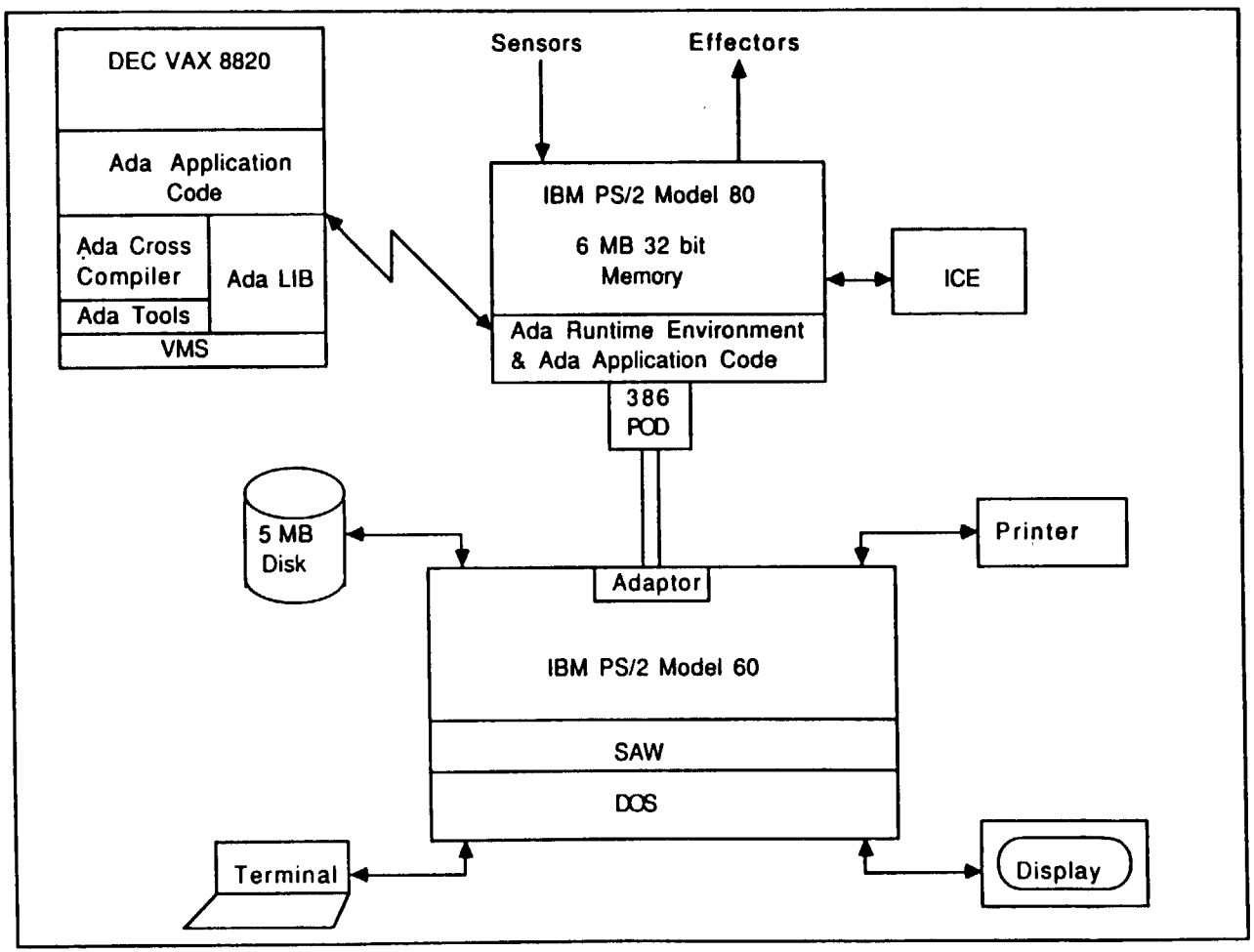
Ethernet and RS232 connectivity is provided on the Vax 8820.

The SAW will be used to obtain, in a non-intrusive manner, performance from the "bare" target as software downloaded is executed.

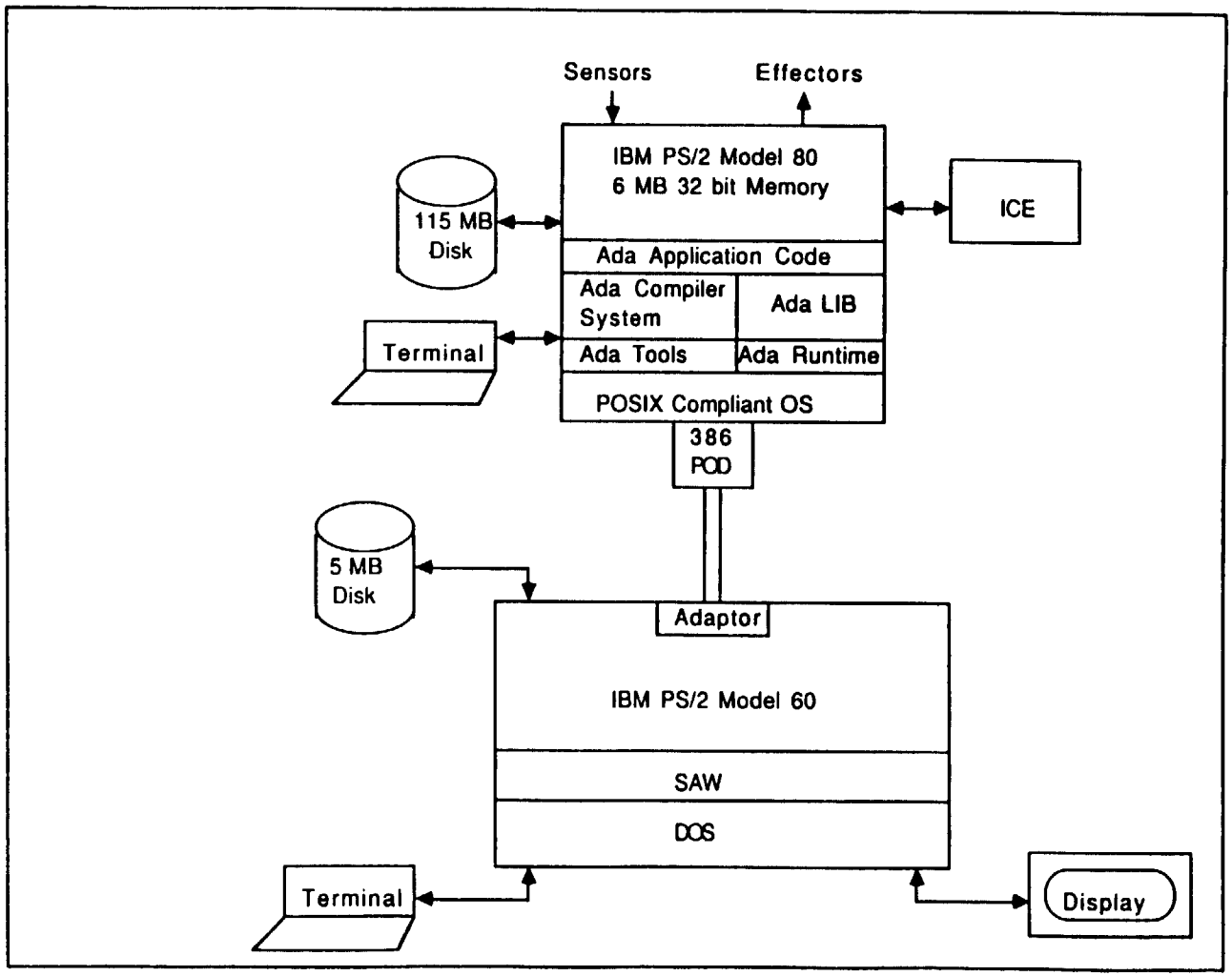
An In_Circuit_Emulator (ICE) will be used to jump-start the system. The general scenario used will be to jump-start the target system with an ICE and then use the SAW to obtain performance data.

The first set of evaluations will be be run on "bare" target and as the POSIX-compliant operating system becomes available, the compilers will be re-run with the OS, thus providing performance data that can be normalized to the "bare" configuration. The following two charts show each configuration.

SSE Ada CROSS-COMPILER 80386 TEST-SUITE
BARE TARGET CONFIGURATION



SSE Ada SELF TARGET 80386 TEST-SUITE WITH POSIX-COMPLIANT OS



The following performance measurement benchmark systems will be used during the cross-compiler evaluation period:

1. Ada Compiler Evaluation Capability (ACEC)

2. Performance Issue Working Group (PIWG)

3. JSC Avionics Test Suite

4. Hard Ada Real Time (HARTSTONE) Test Suite

During the evaluation process, the SSE has developed some benchmarks to verify SSE-specific requirements for embedded real-time systems. These benchmarks will be published as SSE Ada Test (SATSTONES) and current list is provided on the vu-foil.

SSE SYSTEM PROJECT

Ada PERFORMANCE TESTS FOR BARE EMBEDDED TARGETS

PIWG

CLOCK RESOLUTION (1)
CLASSICAL BENCHMARKING TESTS (14)
RUN TIME CHECKS (4)
TASK CREATION AND TERMINATION (3)
DYNAMIC ARRAY ALLOCATION (4)
EXCEPTION HANDLING (5)
PRAGMA PACK WITH BOOLEANS (4)
UNCHECKED CONVERSION (2)
REPRESENTATION CLAUSES (3)
PROCEDURES CALLS (11)
TASK RENDEZVOUS (8)
DELAY (1)

ACEC (EXECUTION TIME
AND CODE EXPANSION)

SIMPLE Ada STATEMENTS (812)
CLASSICAL BENCHMARKING TESTS (75)
APPLICATION PROGRAM EXAMPLES
AVIONICS (12)
DATA ENCRYPTION (11)
ELECTRONIC WARFARE (1)
RADAR (2)
KALMAN FILTER (1)
SIMULATION (8)
ERROR CORRECTING CODE (5)
OPTIMIZATIONS (7)
DELAY (14)
INTERRUPTS (10)
REUSE OF RECLAIMED SPACE (4)
TASK CREATION AND TERMINATION (2)
TASK RENDEZVOUS (78)
STACK CLEANUP AFTER EXCEPTION (1)

NASA/JSC AVIONICS

CLOCK (4)
TASK CREATION AND TERMINATION (5)
TASK RENDEZVOUS (10)
EXCEPTIONS (14)
IF/THEN/ELSE (3)
CASE (2)
DELAY (1)
PRAGMA PRIORITY (1)
CLASSICAL BENCHMARKING TESTS (4)

HARTSTONE

HARMONIC FREQUENCY PERIODIC TASKS
NON-HARMONIC FREQUENCY PERIODIC TASKS
HARMONIC FREQUENCY PERIODIC AND
APERIODIC TASKS
HARMONIC FREQUENCY PERIODIC TASKS
WITH SYNCHRONIZATION
HARMONIC FREQUENCY PERIODIC AND
APERIODIC TASKS WITH SYNCHRONIZATION

SATSTONE

CURRENTLY THE SSE HAS DEVELOPED 22 SATSTONES TO TEST SPECIAL ADA LANGUAGE FEATURES REQUIRED FOR THE SSE.

SAW – SOFTWARE ANALYSIS WORKSTATION

NON–INTRUSIVE MEASUREMENTS:

- SUB–PROGRAM CALLS
- OBJECT ALLOCATION
- EXCEPTIONS
- TASK ELABORATION, ACTIVATION & TERMINATION
- TASK SYNCHRONIZATION
- CLOCK EVALUATION
- TIME AND DURATION EVALUATIONS
- DELAY FUNCTION & SCHEDULING
- OBJECT DE–ALLOCATION AND GARBAGE COLLECTION
- INTERRUPT RESPONSE TIME

This chart reflects the tasks associated with the cross-compiler evaluation. For the first half of 1989, the following tasks were planned and have been completed.

1. Run all the benchmark test suites on SSEDF equipment.

2. With the SEWG involvement, determine the evaluation criteria to be used for the cross-compiler evaluation.

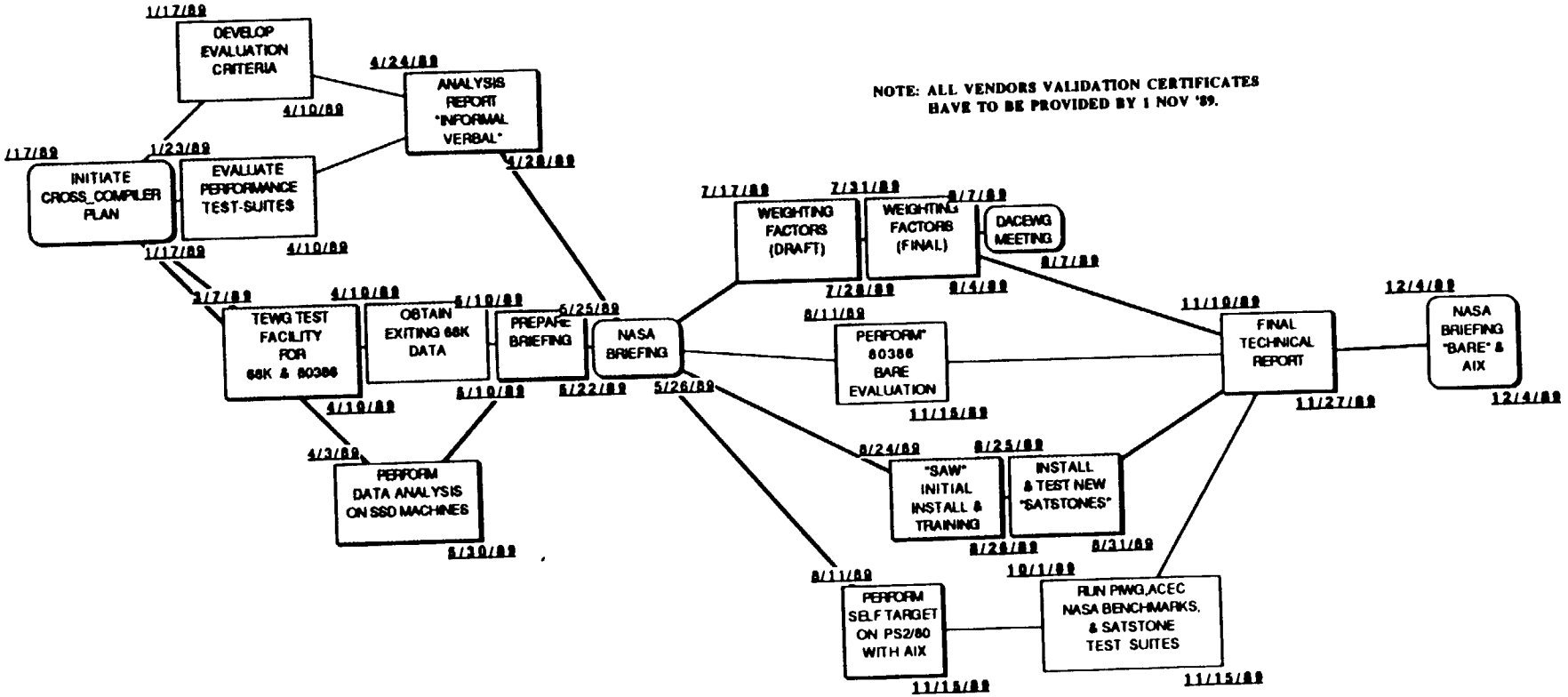
3. Set up a test facility for the evaluation process.

The above items were completed and a briefing to NASA provided late May 1989. The PS2/80 required for the evaluation period of performance is in final review cycle at NASA Level II SCMB, as of late May 1989. As of September 1989, the SSEDF Test Facility includes one PS2/80, one ELTECH 386, one IBM PC/XT, and two VT-100s. The facility includes RS232C and Ethernet connectivity

The evaluation on "bare" target and the evaluation of POSIX-compliant operating systems with self-targeted Ada compilers is in progress.

SSE SYSTEM PROJECT

Ada CROSS_COMPILER PLAN "DRAFT"



NOTE: ALL VENDORS VALIDATION CERTIFICATES HAVE TO BE PROVIDED BY 1 NOV '89.

* ALSYS RUNNING ON ELTECH 386 WITH DOS AND CROSSED TO THE PS2/80 "BARE".

NOTE: 1).RECEIVED ALSYS SELF TARGET ON PS2/80 WITH AIX ON 4 AUG '89.....RECEIVED AIX FROM NASA ON 4 AUG '89. RECEIVED DDCI SELF TARGET ON PS2/80 WITH AIX ON 14 AUG '89.....
2). LATEST PIWG NOT AVAILABLE AS OF 20 SEPT '89 WILL RUN WITH THE EARLIER VERSION.....

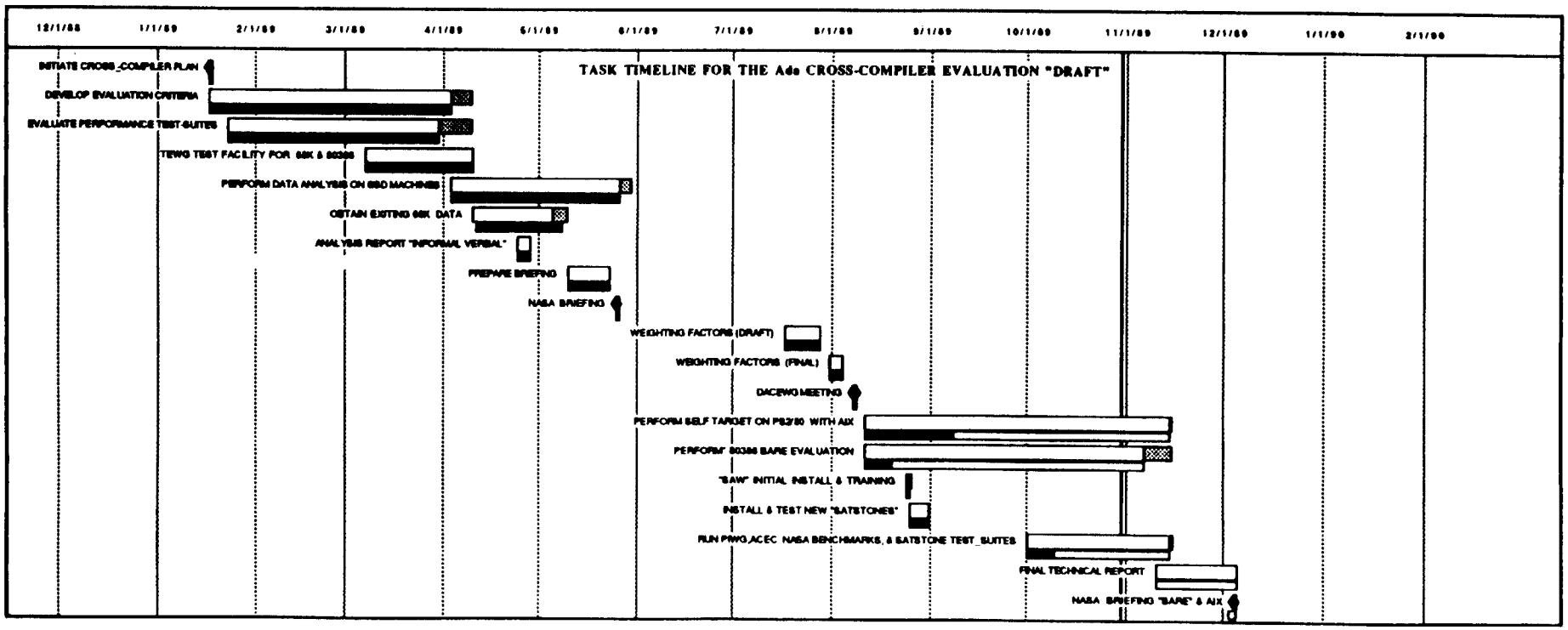
3). VERDIX COMPILERS VAX CROSSED TO "BARE" PS2/80 AND SELF TARGET ON PS2/80 WITH AIX SCHEDULED FOR 6 NOV '89 PENDING SSE RESOURCES AVAILABILITY...

NOTE: THE CROSS_COMPILERS AVAILABILITY FOR THE OS IS RISKY -SCHEDULE WISE.....
THE SSE WILL PROVIDE THE BEST 2 OUT OF 4 COMPILERS ON 4 DEC AND WILL PROVIDE TECHNICAL DATA ON CROSS_COMPILERS EVALUATED TO DATE..

THE POSIX OS SCHEDULE NEEDS TO BE RE_WORKED BY THE SSE & IBM.....

ORIGINAL PAGE IS OF POOR QUALITY

SSE SYSTEM PROJECT



SSE SYSTEM PROJECT

COMPILER	CONFIG.	VER #	BENCHMARK TEST_SUITE ACEC	PIWG	SATSTONE	NASA AVIONICS
ALSYS	ELTECH (DOS) CROSSED TO "BARE" PS2/80		3 NOV	3 NOV	3 NOV	3 NOV
ALSYS	SELF_TARGETTED PS2/80 (AIX)		*	*	*	N/A
DDCI	SELF_TARGETTED PS2/80 (AIX)		31 OCT	*	*	N/A
TELESOFT	SELF_TARGETTED PS2/80 (AIX)		3 NOV	*	*	N/A
VERDIX	SELF_TARGETTED PS2/80 (AIX)					N/A
VERDIX	VAX 8820 CROSSED TO "BARE" PS2/80					

* TASK COMPLETE

** VERDIX SCHEDULED TO BE AT SSE ON 6 NOV WITH COMPILERS

SUMMARY:

- DEVELOPMENT OF Ada COMPILER EVALUATION CRITERIA – COMPLETE
- PERFORMANCE MEASUREMENT SYSTEMS – COMPLETE
- EVALUATION OF STATE-OF-THE-ART Ada CROSS-COMPILERS TO BARE TARGETS – IN PROCESS
- WORK RELATING TO COTS OPERATING SYSTEM – IN PROCESS

SESSION 2 — CENTER AND PROJECT ACTIVITIES

Session Leader: M. Stark, NASA/GSFC

Ada in the SEL: Experiences with Operational Ada Projects
E. Seidewitz and M. Stark

The Application of CASE Technology and Structured Analysis to a Real-Time Ada Project
S. Cohen

Ada at JPL: Experiences and Directions
T. Fouser

Ada and the OMV Project
W. Harless

"ADA IN THE SEL: EXPERIENCES WITH OPERATIONAL ADA PROJECTS"

E. Seidewitz, NASA/GSFC

Ada IN THE SOFTWARE ENGINEERING LABORATORY: EXPERIENCES WITH OPERATIONAL Ada PROJECTS

ED SEIDEWITZ
NASA/GSFC

AND

MICHAEL STARK
NASA/GSFC

SOFTWARE ENGINEERING LABORATORY PRODUCTION ENVIRONMENT

TYPES OF SOFTWARE: SCIENTIFIC, GROUND-BASED, INTERACTIVE GRAPHIC, MODERATE RELIABILITY AND RESPONSE REQUIREMENTS

LANGUAGES: 75% FORTRAN, 15% Ada, 10% OTHER (PASCAL, C, ALC,...)

PROJECT CHARACTERISTICS:	**AVERAGE**	**(RANGE)***
DURATION (MONTHS)	26.0	(18-43)
EFFORT (STAFF-YEARS)	9.5	(02-30)
SIZE (1000 LOC)		
DEVELOPED	93.0	(25-250)
DELIVERED	102.0	(03-30)
STAFF (FULL-TIME EQUIV.)		
AVERAGE	5.4	
PEAK	10.0	
INDIVIDUALS	24.0	
APPLICATION EXPERIENCE (YEARS)		
MANAGERS	5.8	
TECHNICAL STAFF	4.0	
OVERALL EXPERIENCE (YEARS)		
MANAGERS	10.0	
TECHNICAL STAFF	8.5	

* SAMPLE OF 9 FORTRAN PROJECTS

E. Seidewitz
NASA/GSFC

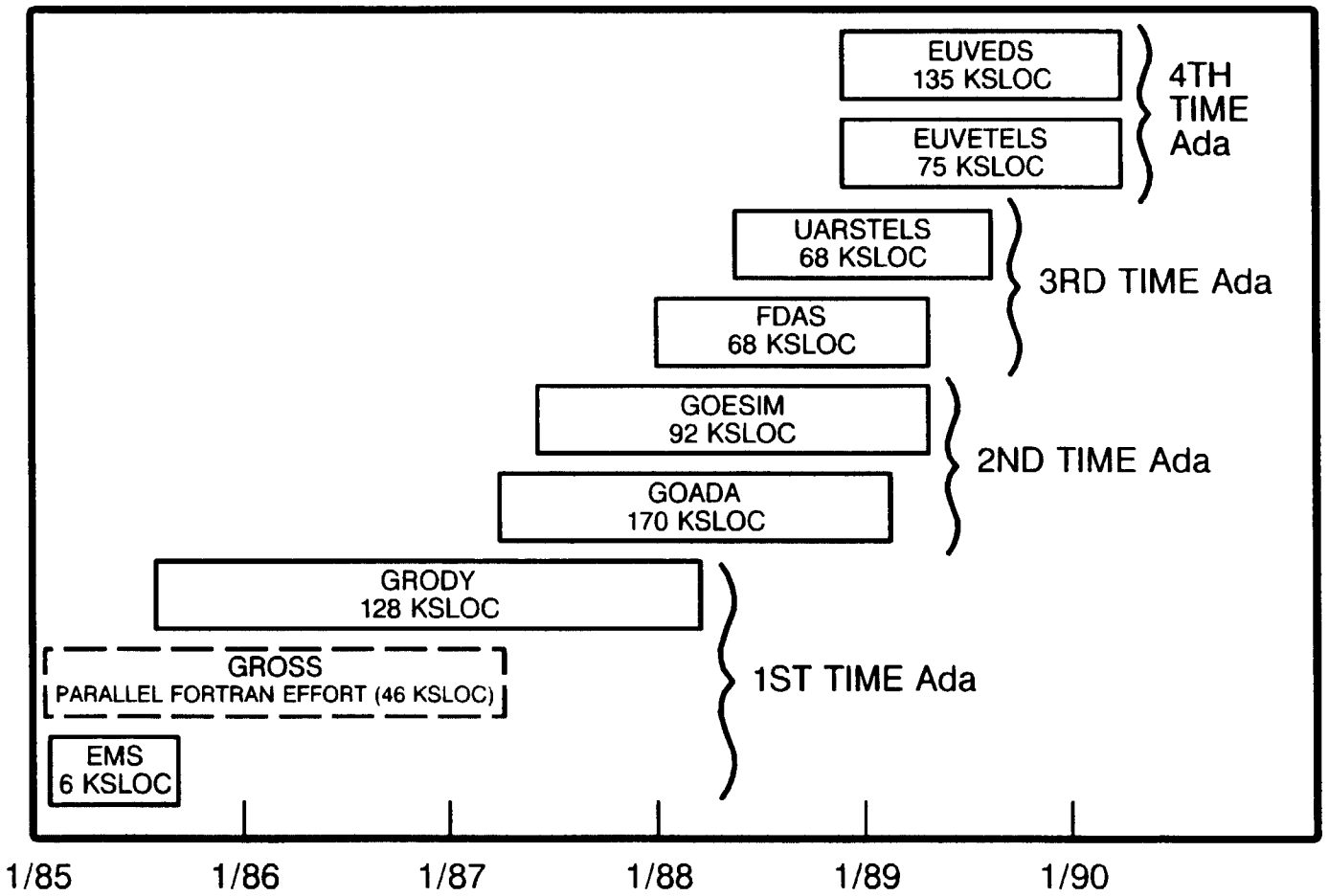
BACKGROUND

- Ada HAS BEEN UTILIZED ON 8 PROJECTS IN FLIGHT DYNAMICS DIVISION AT NASA/GSFC
- DETAILED DATA HAS BEEN COLLECTED/STUDIED BY THE SEL FOR ALL PROJECTS
- APPROACH TO TRAINING/DESIGN/IMPLEMENTATION/ TESTING HAS EVOLVED

POINTS TO BE ADDRESSED

- USE OF Ada FEATURES
- COST/PRODUCTIVITY
- REUSE
- ERROR CHARACTERISTICS
- PORTABILITY

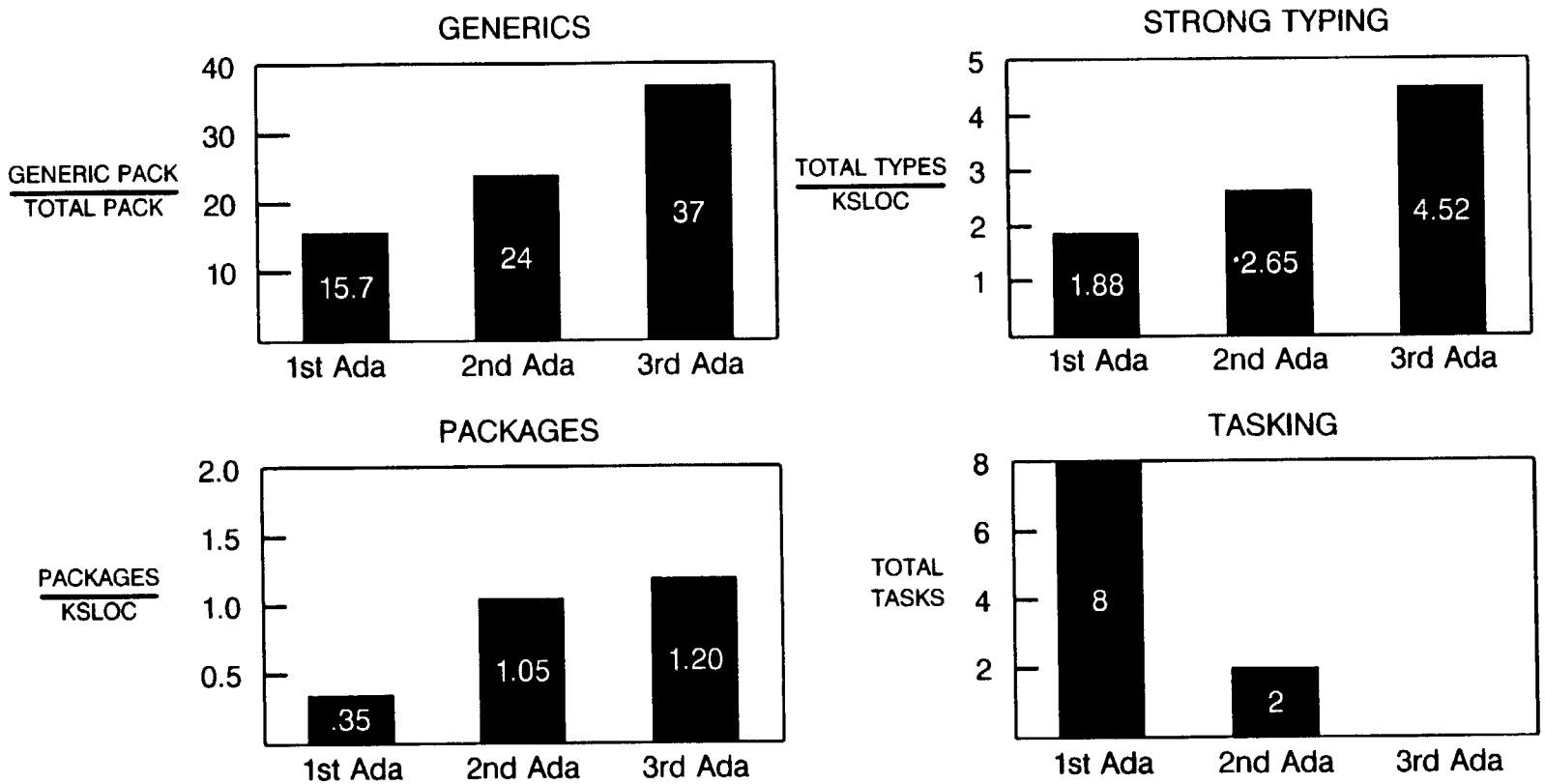
Ada PROJECTS IN FLIGHT DYNAMICS DIVISION



E. Seidewitz
NASA/GSFC

B270.004

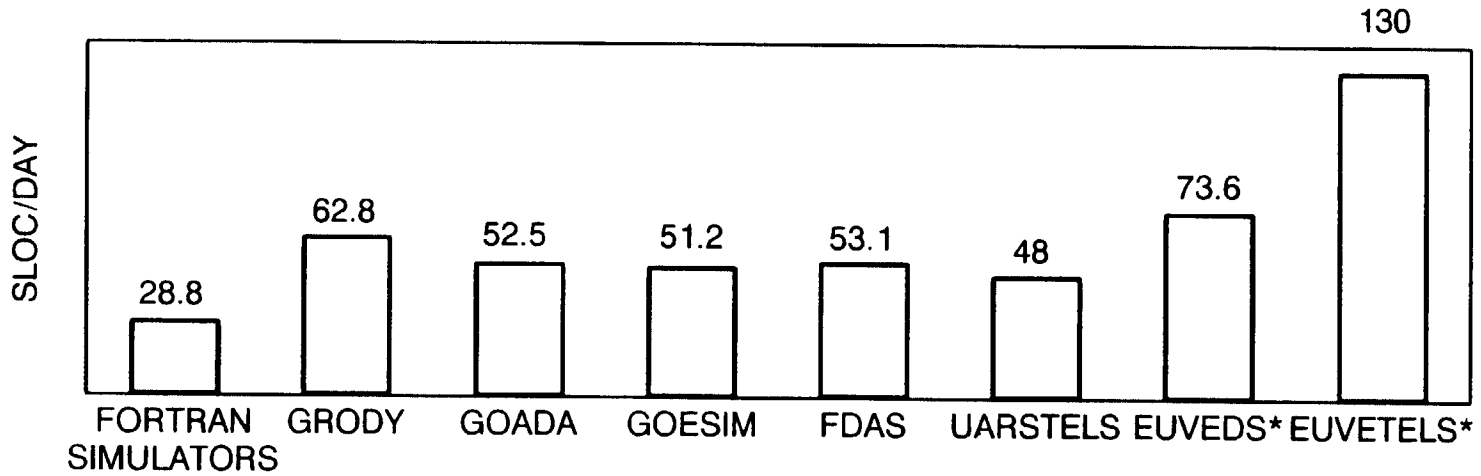
USE OF Ada FEATURES

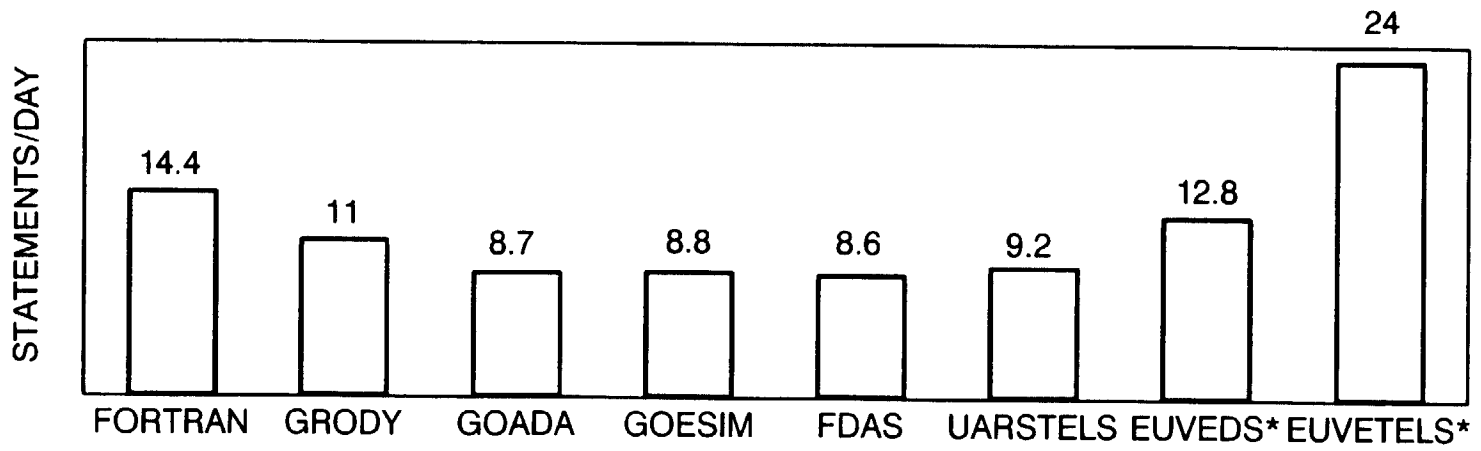


- USE OF Ada FEATURES CHANGES APPRECIATELY WITH EXPERIENCE
- NOT ALL FEATURE APPROPRIATE FOR APPLICATION

D217.018

COST/PRODUCTIVITY MEASURES



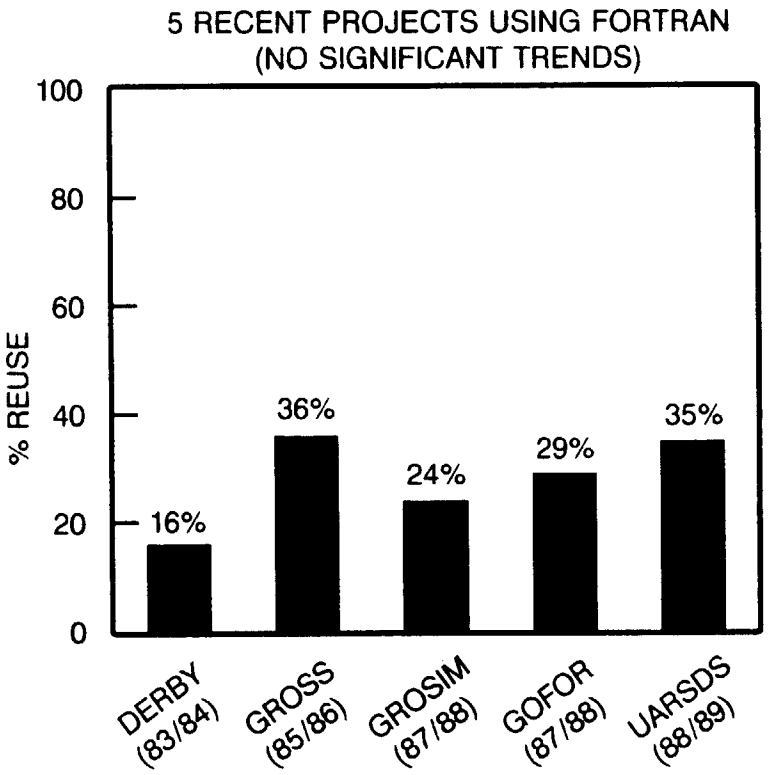


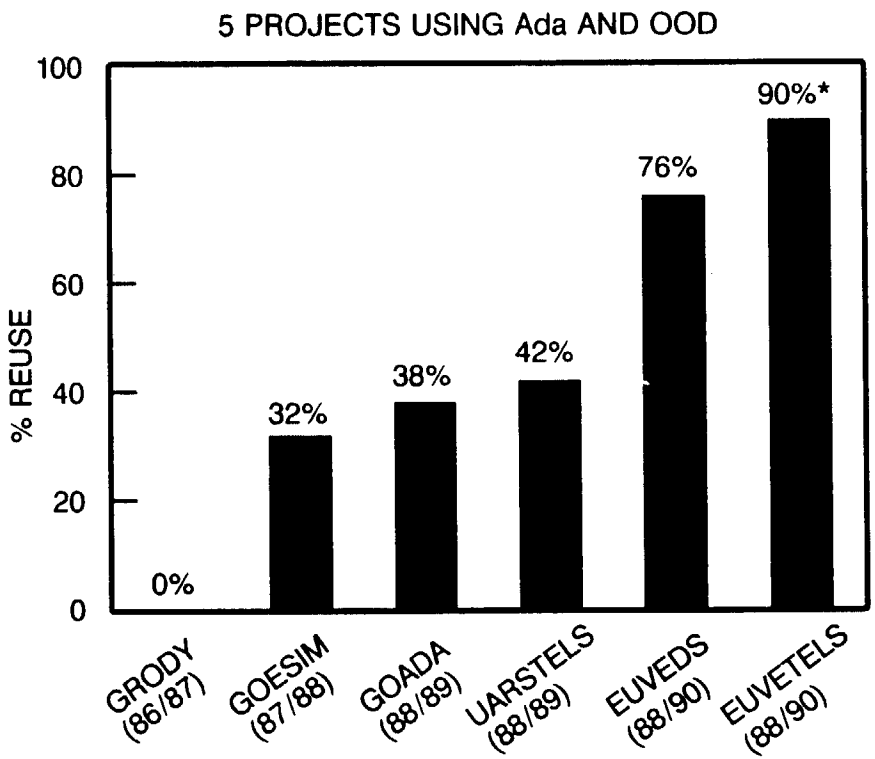
*PARTIALLY BASED ON ESTIMATES

E. Seidewitz
NASA/GSFC

B270.006

REUSE OF DYNAMIC SIMULATOR CODE



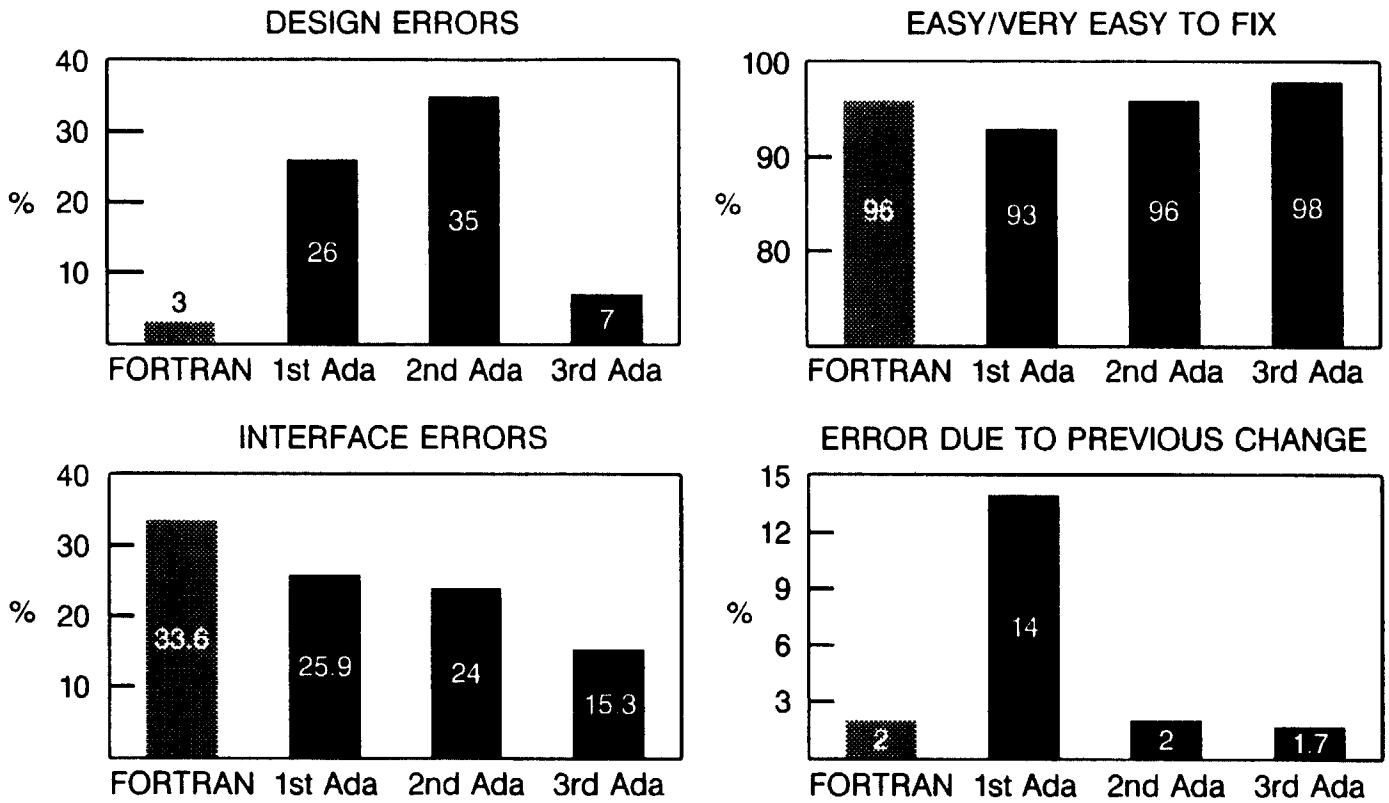


*EUVETELS SPECS WRITTEN AS A DERIVATION OF UARSTELS

E. Seidewitz
NASA/GSFC

B270.009

ERROR CHARACTERISTICS



- Ada ERROR PROFILE CHANGES WITH MATURITY OF USE
- Ada HELPS CUT INTERFACE ERRORS

B270.007

PORTABILITY STUDY

- 90 KSLOC (16K STMT)
- 561 COMPONENTS (14 FORTRAN)
- 6 STAFF YEAR DEVELOPMENT

- 83 COMPONENTS CHANGED
- 41 NEW COMPONENTS
- 2.5 STAFF-MONTHS TO COMPILE
- 4 STAFF-MONTHS TO EXECUTE

- BASELINE: ~1 STAFF-YEAR TO PORT EQUIVALENT FORTRAN SYSTEM
- KEY ISSUES
  - IMMATURITY OF Ada ENVIRONMENT ON IBM/MVS
  - USE OF VENDOR-SPECIFIC FEATURES DECREASES PORTABILITY
  - USE OF DATA TYPES INCREASES PORTABILITY

NEW FLIGHT DYNAMICS PROJECTS IN Ada

- TDRSS ON BOARD NAVIGATION SYSTEM (TONS) EXPERIMENT
  - SOFT REAL-TIME SOFTWARE FOR THE EXTREME ULTRA VIOLET EXPLORER (EUVE)
  - SIZE: ~45 KSLOC
  - EFFORT: ~5 STAFF YEARS
  - SCHEDULE: 1981-1991

- COMBINED OPERATIONAL MISSION PLANNING AND ATTITUDE SUPPORT SYSTEM (COMPASS)
  - HIGHLY GENERALIZED, CONFIGURABLE FLIGHT DYNAMICS SUPPORT SYSTEM
  - SIZE: ~1 MILLION LOC
  - EFFORT: ~250 STAFF YEARS
  - SCHEDULE: 1989-1996

ASSESSMENT

- INCREMENTAL ADOPTION OF Ada AND ASSOCIATED TECHNIQUES
  - DUE TO ORGANIZATIONAL LEGACY
  - 3 YEAR LEARNING PERIOD
  - 10 YEAR TRANSITION PERIOD
  - INITIALLY, ONLY SMALL CHANGES TO LIFE CYCLE PROCESS
- INCREASED REUSABILITY
  - DUE TO USE OF Ada AND OBJECT-ORIENTED DESIGN
  - REQUIRES SPECIAL PLANNING
  - DRIVES REDUCTION OF EFFECTIVE COST
- ONGOING STUDIES
  - PORTABILITY
  - PERFORMANCE
- FUTURE STUDIES
  - RELIABILITY
  - MAINTAINABILITY

"THE APPLICATION OF CASE TECHNOLOGY AND STRUCTURED ANALYSIS TO A REAL-TIME ADA PROJECT"

S. Cohen, GE/STGT

The Application of CASE Technology and Structured Analysis to a Real-Time Ada Project

Sara Cohen

General Electric/STGT
P.O. Box 8048 - Bldg. 25, Rm. 22S05
Philadelphia, PA 19101

Introduction

System requirements analysis frequently poses a challenge to a project development team. Traditional life-cycle methods used to analyze system and software requirements often result in lengthy text without graphical representation. This documentation is difficult to modify and check for consistency. The use of Structured Analysis (SA) has alleviated many of the disadvantages associated with traditional system and software requirements analysis methods. System and software requirements analysis using SA techniques is more complete, easily understood, precise and comprehensive. Benefits of SA are apparent in the System Requirements Analysis/System Design and Software Requirements Analysis phases. The experiences of the Second Tracking and Data Relay Satellite System (TDRSS) Ground Terminal (STGT) project to date have shown that the benefits of SA can be realized as far into the project life-cycle as the Detailed Design Phase. This paper will discuss the requirements analysis methodology exercised on the STGT project, as well as its benefits into the Detailed Design Phase.

Background

The Second Tracking and Data Relay Satellite System (TDRSS) Ground Terminal (STGT) is a real-time project which will contain over 450,000 lines of custom executable Ada code and approximately 2,000,000 lines of Commercial Off The Shelf (COTS) software. It provides the National Aeronautics and Space Administration (NASA) with tracking, telemetry and command of the TDRS, and high quality data communication service to the user community. The STGT mission requirements include implementing NASA's allocation of system resources, maintaining high quality forward and return links, providing system performance and status data to the Network Control Center (NCC), and providing efficient ground station maintenance to meet the high quality and availability requirements of the STGT users. STGT will provide high operational availability to the user community with less than fifty-five minutes down time per year (no more than 10 seconds at a time). A distributed architecture and multiple Computer Software Configuration Items (CSCIs) are employed to facilitate development, flexibility, maintenance, documentation, and control of the software necessary to operate and maintain the STGT.

Upon completing successful project reviews - a System/Subsystem Requirements Review in March 1989 and a Preliminary Design Review in August 1989, STGT is currently in the Detailed Design Phase. Software developers are identifying units according to DoD-STD-2167 and producing Ada package specifications and Ada Program Design Language (PDL). Subsystem Critical Design Reviews are scheduled to begin in January 1990 with the infrastructure CSCIs and continue through April 1990 for the remainding CSCIs. A system Critical Design Review is scheduled for June 1990. The Coding, Unit and CSC Testing Phase will begin in January 1990 and continue through October 1990.

STGT's software development team currently numbers fifty-five. Eleven STGT developers participated in an intensive Ada training program for one year prior to the start of full-scale STGT development. During this training, they completed at least one Ada project in technology directly applicable to STGT. In addition, they are experienced in the area of large-scale ground station projects. These software developers were assigned to lead the development of the CSCIs. All STGT software developers were required to complete an Ada individualized training program which required at least eighty hours. This self-paced program consisted of a multi-media curriculum including computer-based and text-based training, lectures, and hands-on application. Many STGT software managers either had previous Ada experience or completed the Ada training program. STGT software development management recognized the importance of management, as well as individual contributors, receiving software engineering and Ada design training specifically for real-time systems. Additionally, STGT software managers attended Ada management training classes.

Traditional Requirements Analysis vs. Structured Analysis

The early '70s introduced project management methods based upon models of the software development life-cycle. This called for documents to be produced at specific points within the life-cycle according to prescribed forms and standards or document content guidelines. These methods stressed the capture of documentation during the development process, but did not adequately provide for the continued usefulness of the documents. The specifications were often incomplete, inconsistent, incorrect and not always updated to reflect changes made to the system. Lack of formality, resulting in inconsistency, and lack of maintainablity have been identified as the problems which have limited the effectiveness of the life-cycle based methods.

It was obvious that more formal methods than those mentioned above were necessary if greater productivity was to be achieved. In the late '70s and early '80s, more formal methods were introduced, most notably structured analysis and structured design. These methods shifted the emphasis from later phases of the life-cycle to earlier ones. More time should be spent in the Requirements Analysis Phase, as well as the Preliminary and Detailed Design Phases. Less time and money would be spent in the Coding, Testing and Maintenance

Phases, since errors would be detected at the earliest possible time. Implementation would be easier and the resulting software would be of higher quality.

Tom DeMarco is credited with popularizing structured analysis. He explained that through the use of data flow diagrams and descriptions of data (i.e. data dictionary, process specifications and decision tables), one could build a systematic description of a system's logical (functional) and physical (implementation) aspects. Management could control the activities by conducting walkthroughs where data flow diagrams, the data dictionary, process specifications and decision tables could be manually validated. The data flow diagrams and the data dictionary became system document deliverables. According to DeMarco, if the person performing the structured analysis is doing so correctly, the structured specification would have the following qualities [1]:

1. It would be graphic. The data flow diagrams would present a meaningful, easily understood, picture of what is being specified.
2. It would be partitioned. The processes depicted on the data flow diagrams would represent the basic elements of the system.
3. It would be rigorous. The data dictionary would provide a rigorous document of the interfaces and the process specification would be rigorous as well.
4. It would be maintainable. Redundancy would be minimized and used in a controlled manner.
5. It would be iterative. The specification would be shared with the user and modified according to his/her needs until correct.
6. It would be logical, not physical. By eliminating elements that depend upon things such as hardware and vendor, one need not concern oneself with changes in physical thinking.
7. It would be precise, concise and highly readable.

Fulfilling these qualities, the structured specification would then exemplify the popular saying "a picture is worth a thousand words". A system specification properly decomposed into data flow diagrams would be more easily communicated than the traditional tonnage of requirements documentation.

With the methodology in place, there was a need for tools in order to provide automation. Without these tools, it would be less practical or economical to use formal system development methods. In the mid '80s, with the spread of desktop computers, a technology known as Computer Aided Software Engineering (CASE) was introduced. It was comprised of environments and tools which would allow the user to model a system from its initial user requirements through design and implementation. Tests could be applied in order to check for consistency, completeness and conformance to standards. In other words, these tools would assist the user in expressing his/her structured analysis and design models. They would *not* create the models for the user.

Benefits of Structured Analysis During the Requirements Analysis Phase

Given the complexity of the STGT, analyzing the system requirements was a challenge. The NASA Requirements Specification for STGT was analyzed using Structured Analysis (SA) techniques. Cadre Technologies Inc.'s **Teamwork/SA®** was selected as the CASE tool to support the structured requirements analysis process. This selection was due to GE Military & Data Systems Operations' (M&DSO) business association with Cadre Technologies Inc. Cadre Technologies Inc. is a large, stable company willing to accomodate M&DSO's needs. Their products met M&DSO's key requirements, among which was the capability to support multi-users and multiple platforms.

The system requirements analysis was rigorous, easily understood, precise and comprehensive. The system model's data dictionary served as a basis for hardware/software interface specifications. The system model itself provided a sound foundation upon which software requirements analysis could be performed.

Applying the same techniques as were applied in the systems requirements analysis phase, each CSCI developed a levelled model in which the requirements specific to the CSCI were captured. The models consisted of multiple levels of data flow diagrams reflecting corresponding levels of functional detail. Individual requirements were enumerated in the process specifications. Intra-CSCI, as well as, inter-CSCI data flows were defined in the data dictionaries. The use of **Teamwork/SA®**'s checking capability ensured that the data flow diagrams were syntactically correct and that they balanced with their child data flow diagrams and process specifications. The software requirements specifications produced were precise, concise and highly readable.

Teamwork/SA® did, however, have one limitation. Since the number of users accessing a particular data base simultaneously was limited to eight, it was decided that each CSCI would develop its own model. All of the CSCIs could not be accomodated in one data base. This meant that each CSCI would have its own data base. All consistency checking between CSCIs was performed manually. A manual procedure was used where data dictionaries were merged and definitions were checked for consistency. This procedure, unfortunately, was not 100% foolproof. An automated procedure would have been more efficient.

The software requirements models and data dictionaries provided for 80% of the SRSs' content. Interleaf Publishing Software was used to produce the SRS documents. The production of the SRSs was enhanced due to the software utilities, commercial and in-house, available to incorporate the **Teamwork®** models into Interleaf documents.

Teamwork/SA® is a registered trademark of Cadre Technologies, Inc.

Software developers produced the SRSs using these tools, leaving the Technical Publications group the task of merely applying cosmetic changes.These SRSs were delivered to the customer at the System/Subsystem Requirements Review.

Based upon GE's initial Ada project experiences and modern software engineering principles, it was projected that STGT would spend 10–15% of its software hours in the Requirements Analysis Phase. In fact, over 9% of STGT's budgeted software hours were spent performing requirements analysis.

The development of the requirements models produced some unexpected benefits. The graphics depicted in these specifications proved to be an excellent means of communication with NASA during the requirements analysis walkthroughs. They were equally helpful to convey ideas or work out issues with colleagues. In addition, the data flow diagram models served as good training medium as new STGT personnel familiarized themselves with the system.

Benefits of Structured Analysis During the Preliminary Design Phase

The basic goals of the Preliminary Design Phase were to develop a top-level design for each CSCI reflecting the requirements specified in the SRSs and to develop a lower-level design for Computer Software Components (CSCs) which were identified as critical elements of the design. Critical elements were defined as those required at an early date for the development of other CSCs, those having a long development period and/or those having performance requirements that would be especially critical.

Employing an object-oriented design approach tailored to the needs of STGT, the first step of preliminary design was to identify objects, physical and abstract, in the system. The objects would contain state data and provide operations on that data. Object-oriented CSCs encapsulated objects within a CSCI. Object-oriented design, however, had its limitations. Concurrency would not be addressed until object implementation. Overall system concurrency would not be addressed and control was not obvious. In order to address these limitations, process-oriented CSCs were identified. The process-oriented CSCs encompassed major portions of the processing to be employed by the CSCI.

The identification of concurrent processes within STGT was based upon an "Edges-In" approach, where the processes necessary to control the external devices were identified first. The rules used to identify concurrent processes were:

1. External devices: These processes were designed as simple device drivers to run at the speed of the device.
2. Functional cohesion: Closely related functions were combined into a single process.
3. Time-critical functions: High-priority processes were identified due to their time criticality.

4. Periodic functions: Separate processes were identified for periodic functions to be activated at the proper time intervals.
5. Computational requirements: Low-priority processes were identified for functions which were not time critical and often computationally intensive.
6. Temporal cohesion: Functions performed during same time period or immediately following certain events were combined into a single process.
7. Data base functions: Functions which needed access to a shared data base were aggregated into a single process with mutual exclusion as the access mechanism.

With the design approach in place, the software requirements model was an excellent starting point for the "carving process". Transforms and their decompositions were examined with the rules described above in mind. Process-oriented CSCs were identified as a result of this technique. Many of the data stores in the model were initial object candidates. Figure 1 illustrates the results of the "carving" process. The data flow diagram depicted is a high level data flow diagram. This data flow diagram, in addition to its child data flow diagrams were used to carve out the Ground Equipment object-oriented CSC, Perform Operator Initiated Testing process-oriented CSC, Maintain Service Status Data process-oriented CSC, Perform Failover and Automatic Fault Isolation process-oriented CSC, Monitor Ground Equipment process-oriented CSC and Control Ground Equipment process-oriented CSC.

During the Preliminary Design Phase, two activities, other than software design, benefited by the results of the Requirements Analysis Phase. A system performance study was conducted and used the software requirements models to define transactions. In addition, the software test group found that production of the software test plans was facilitated by the clarity of the requirements.

Benefits of Structured Analysis During the Detailed Design Phase

Entering the Detailed Design Phase, the initial goals were to refine the CSCs, identify Ada tasks and VMS processes and finally to select units, generate Ada package specifications and Ada PDL. The role of the software requirements model was smaller than in the previous phases. The models represented a solid understanding of the physical requirements. In a number of areas, however, software developers felt the need to address implementation issues, prior to unit selection. This was satisfied by further decomposing the software requirements models. Again, the "carving" technique was employed. Data stores, as well as transforms or groups of transforms were prime candidates for units.

Lessons Learned

To date, the STGT team, GE and NASA, feels that the requirements analysis performed on this program was successful. There have been, however, a couple of lessons learned from this experience.

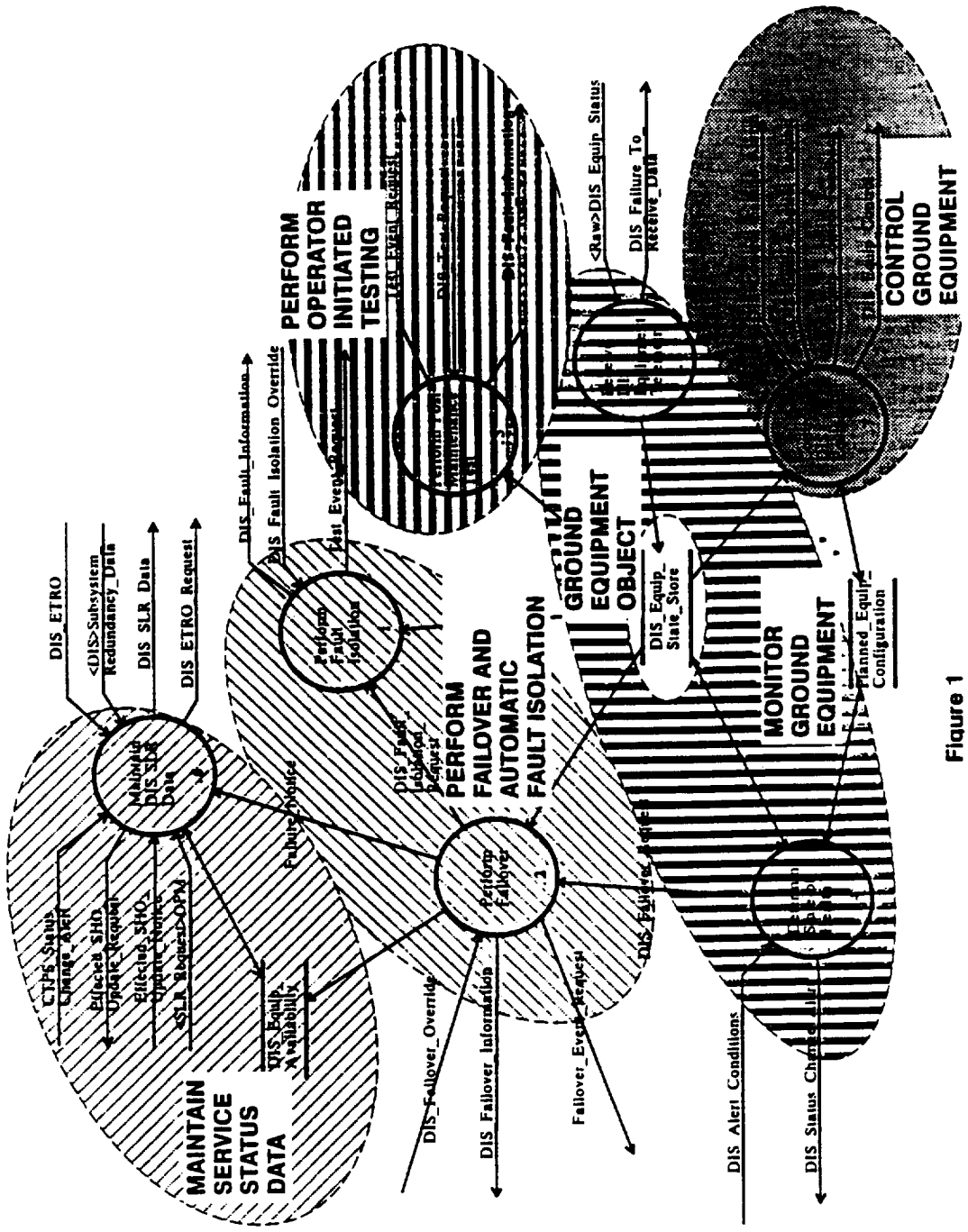


Figure 1

- The use of modern software engineering principles on a program whose schedule has been set with the traditional emphasis of effort in the later project phases (e.g. Code and Unit Test Phase) can cause a conflict. Typically, in this situation, not enough time is allocated to the Requirements Analysis Phase causing requirements analysis to be continued into the Preliminary Design Phase.
- It is not easy to separate implementation from requirements issues when developing a requirements analysis model. However, when design concepts are incorporated into the requirements model, the SRSs have to be repeatedly updated during the course of the project to reflect changes in the design! Additionally, as derived requirements are incorporated into the models, software developers spend much of their time balancing the models. This interrupts precious time which could be spent on design activities.

Conclusion

"Is SA suitable for an Ada project to be designed using an object-oriented approach?" is a question often asked. The experiences of STGT have shown that using SA for a large-scale Ada project resulted in a rigourous, precise and comprehensive requirements analysis. The software requirements models were useful in many areas – directly and indirectly related to the software development process. It is our feeling that the use of SA played a key role in the success of STGT's requirements analysis. The effect of using SA was so great that its benefits were realized into the Detailed Design Phase.

References

1. DeMarco, T., *Structured Analysis and System Specification*, Yourdon Press, New York, 1978.
2. Chikofsky, E.J. and Rubenstein, B.L., *"CASE: Reliability Engineering for Information Systems"*, IEEE Software, March 1988, pp. 11–16.
3. Booch, G., *Software Engineering With Ada – Second Edition*, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, 1987.
4. Hanner, M.A., *" CASE Tools – Productivity for the Masses"*, DEC Professional, December 1988, pp. 38–47.
5. Washenko, R.A., *CASE Tools Evaluation*, TIS 87CIT016, November 1987.
6. Cohen, S. *"CASE – Methodology and Tool"*, Proceedings GE Software Engineering Conference, May 1989.
7. Nielsen, K. and Shumate, K., *Designing Large Real–Time Systems with Ada*, McGraw–Hill Book Company, New York, New York 1988.

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Agenda

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- An Introduction To Second TDRSS Ground Terminal (STGT)
- Traditional Requirements Analysis *vs.* Structured Analysis (SA)
- Benefits of SA and CASE Tools Usage During Project Phases:
  - Requirements Analysis
  - Preliminary Design
  - Detailed Design
- Conclusion

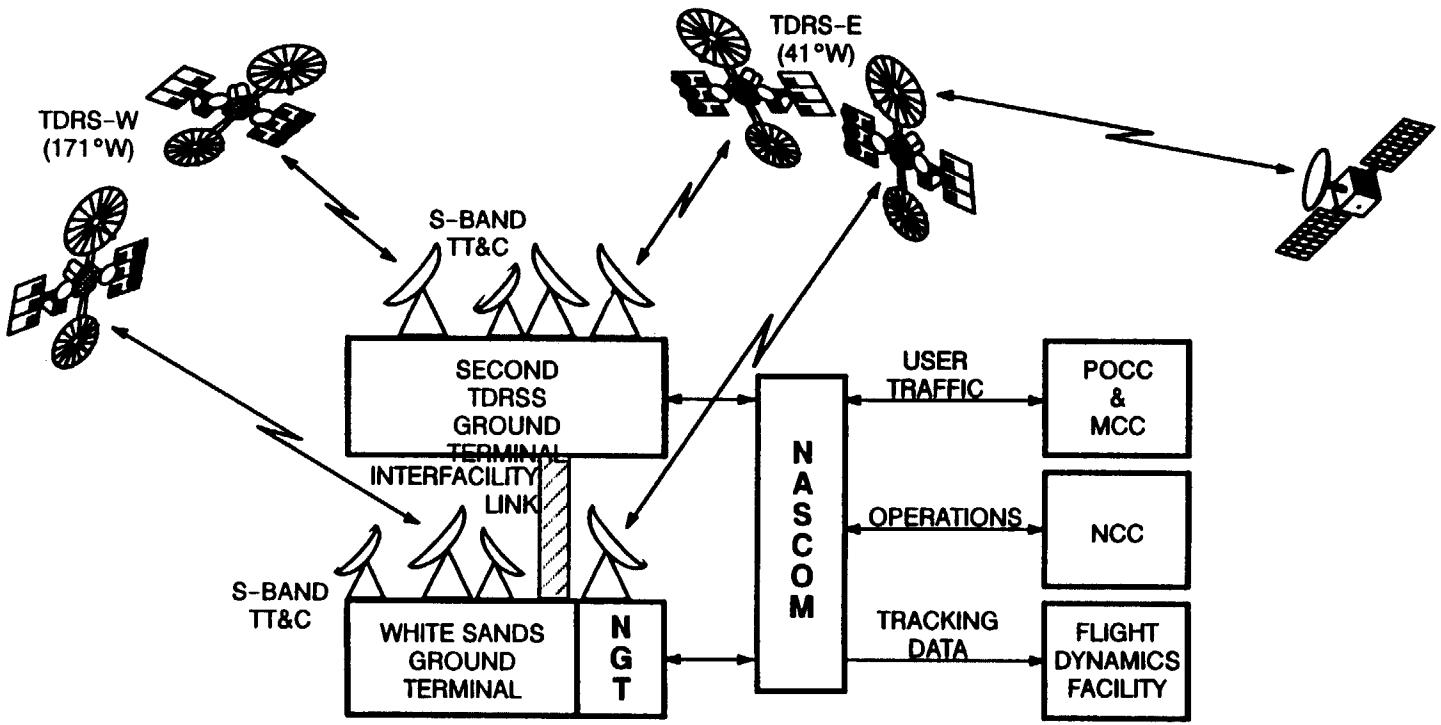
S. Cohen
GE/STGT
10 of 24

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Second TDRSS Ground Terminal (STGT)

STGT
Second TDRSS
Ground Terminal

November 30, 1989



- PROVIDES NASA TRACKING, TELEMETRY AND COMMAND (TT&C) FOR TDRS
- PROVIDES DATA COMMUNICATION SERVICES TO USER COMMUNITY
- DISTRIBUTED ARCHITECTURE - 12 VAX CLUSTERS
- PROVIDES HIGH OPERATIONAL AVAILABILITY TO USER COMMUNITY
  - LESS THAN 55 MINUTES DOWN TIME PER YEAR
  - NO MORE THAN 10 SECONDS AT A TIME
- BACKUP TO WHITE SANDS GROUND TERMINAL

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

STGT – Software

STGT
Second TDRSS Ground Terminal

November 30, 1989

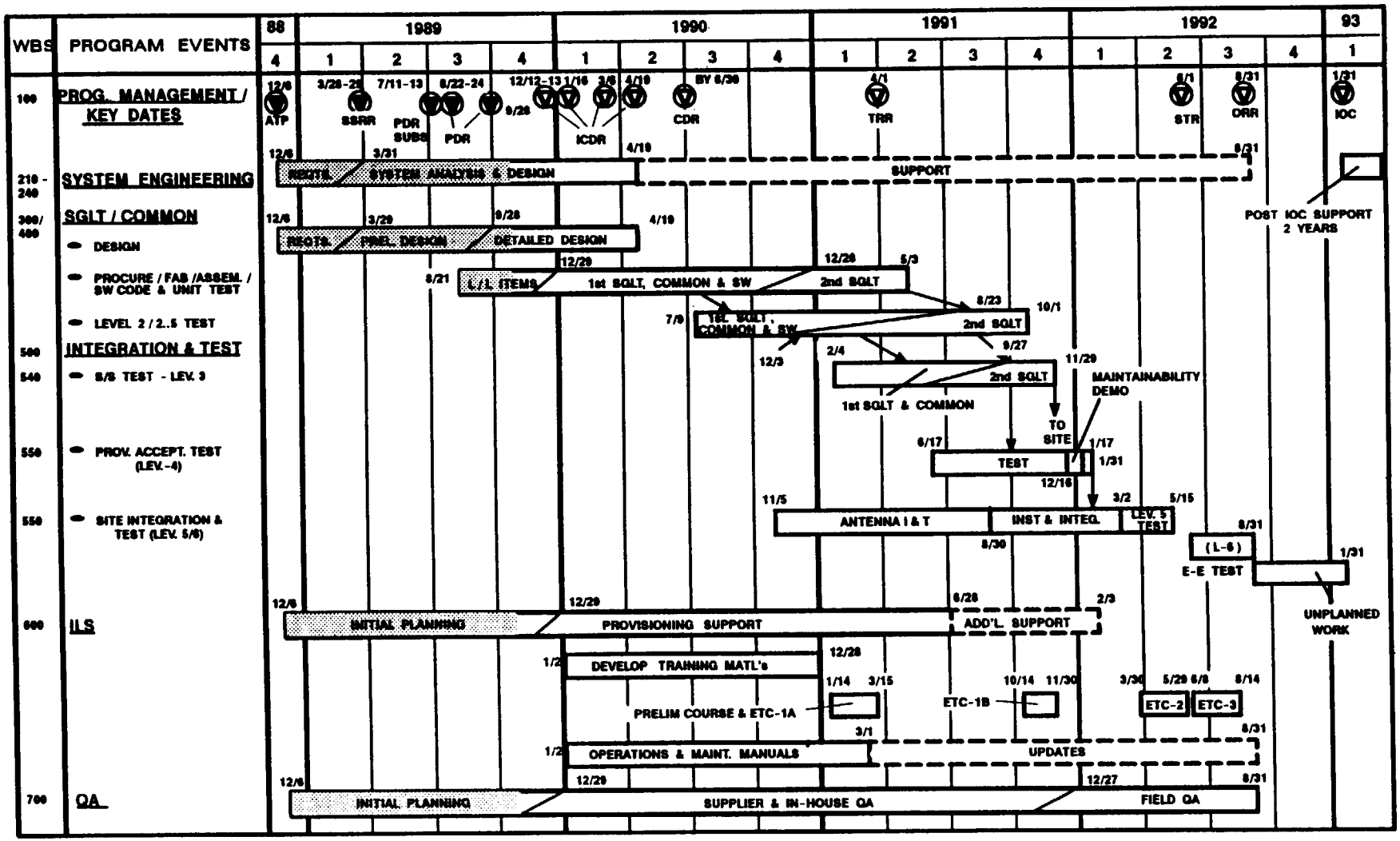
- **Large Scale Real-Time Ada Project**
  - **Over 450,000 lines of executable Ada Code**
  - **Over 2,000,000 lines of Commercial Off The Shelf (COTS) software**
  - **8 CSCIs**
- **Staffing**
  - **PDR/CDR Staff – 55**
  - **Code/Unit Test Staff – 90**
- **Experience**
  - **CSCI development lead by Ada Core Team members with large scale project/ground station experience**
  - **Developers Ada certified**
  - **Significant software engineering, Ada design, and Ada management training**
  - **Managers trained in software engineering and Ada – major contributor to success**

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

STGT – Project Status

STGT
Second TDRSS Ground Terminal

November 30, 1989



The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Traditional Requirements Analysis vs. SA

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- **Traditional Requirements Analysis Methods:**
  - **Lack of formality**
  - **Documents difficult to modify and check for consistency**
- **Structured Analysis:**
  - **More formal methods than the traditional approach**
  - **Specifications are precise, concise, highly readable and consistent**

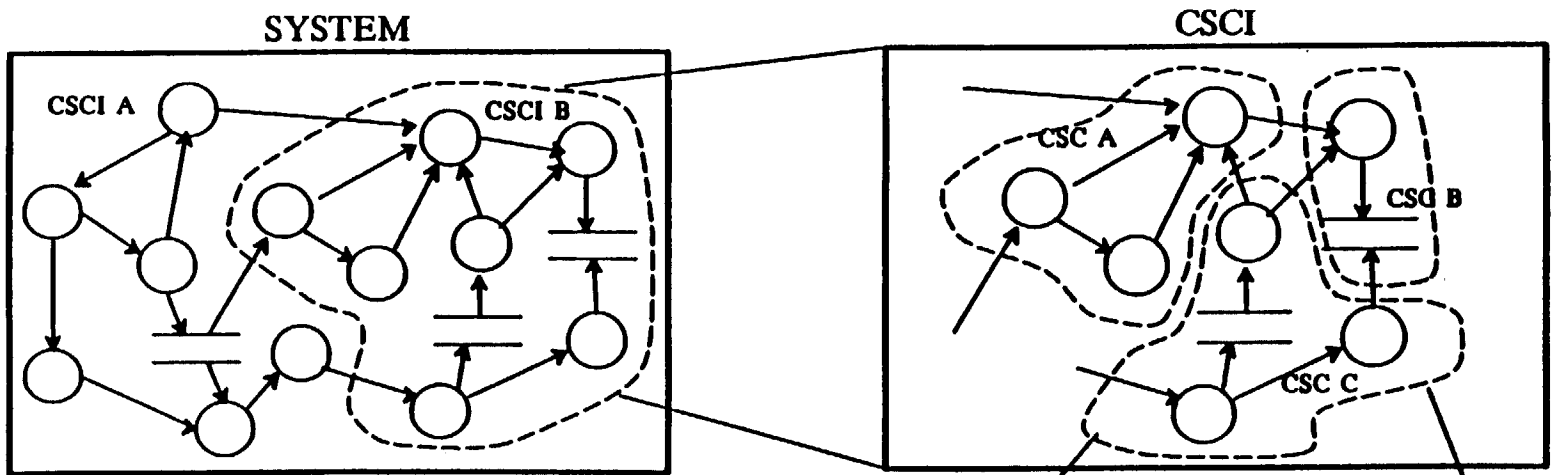
" A PICTURE IS WORTH A THOUSAND WORDS "

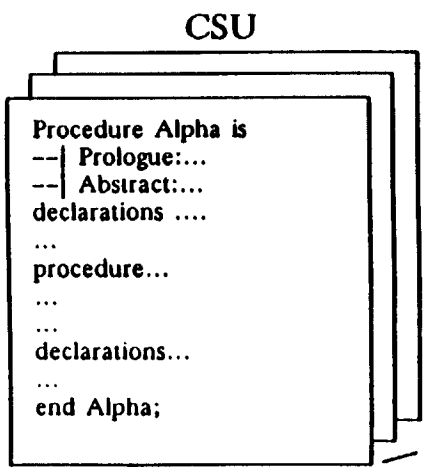
The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

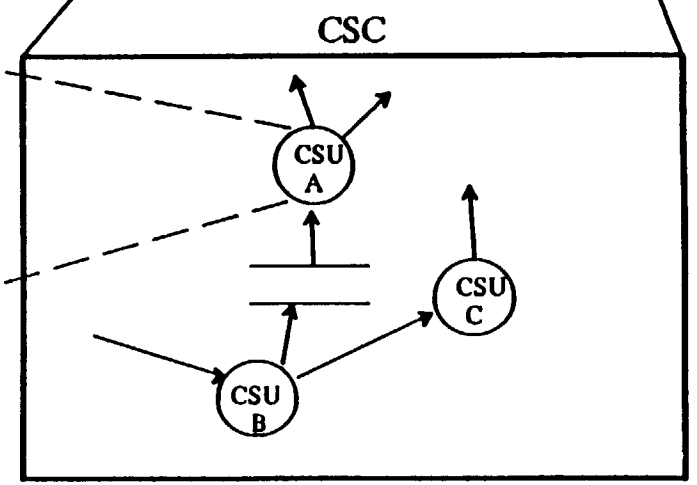
Evolution of System Components

STGT
Second TDRSS
Ground Terminal

November 30, 1989







The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Benefits of SA During Requirements Analysis

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- **Software Requirements Analysis Techniques Similar To Those Used For System Requirements Analysis**
- **Levelled Models Allowed For Depicting Basic Elements Of The System:**
  - **High level transforms "exploded" into lower level transforms**
  - **Enumeration of individual requirements in process-specifications**
- **Data Dictionary Allowed for:**
  - **Clear definitions of data flows and CSCI-to-CSCI interfaces**
  - **Unique name for each data item**

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Benefits of Tools During Requirements Analysis

STGT
Second TDRSS Ground Terminal

November 30, 1989

- **CASE Tool, Team*work*/SA®, Used to Support Analysis**
  - **Automated SA documentation process**
  - **Graphical editor eased modification**
  - **Ensured consistency**
- **Software Requirements Specification (SRS) Production:**
  - **Models provided substantial content (80%)**
  - **Use of Interleaf Desktop Publishing Software to expedite process**

Team*work*/SA® is a registered trademark of Cadre Technologies, Inc.

The Application of CASE and SA To A Real-Time Ada Project...

Related Benefits of Models – Requirements Analysis

STGT
Second TDRSS Ground Terminal

November 30, 1989

- **Data Flow Diagram Models Served As:**
  - ***Excellent*** **communications medium with customer during walkthroughs**
  - ***Excellent*** **communications medium with colleagues during meetings**
  - ***Good*** **training medium for new STGT personnel**

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Productivity During Requirements Analysis

STGT
Second TDRSS Ground Terminal

November 30, 1989

- **Project Management Models Predict That 10-15% Software Hours To Be Spent On Software Requirements Analysis**
- **STGT Software Hours Spent On Software Requirements Analysis Were 9.5% of Our Budgeted Software Hours**
- **Even Given Productivity Enhancement, Traditional Program Milestones Didn't Accomodate The Time Necessary To Complete Requirements Analysis**

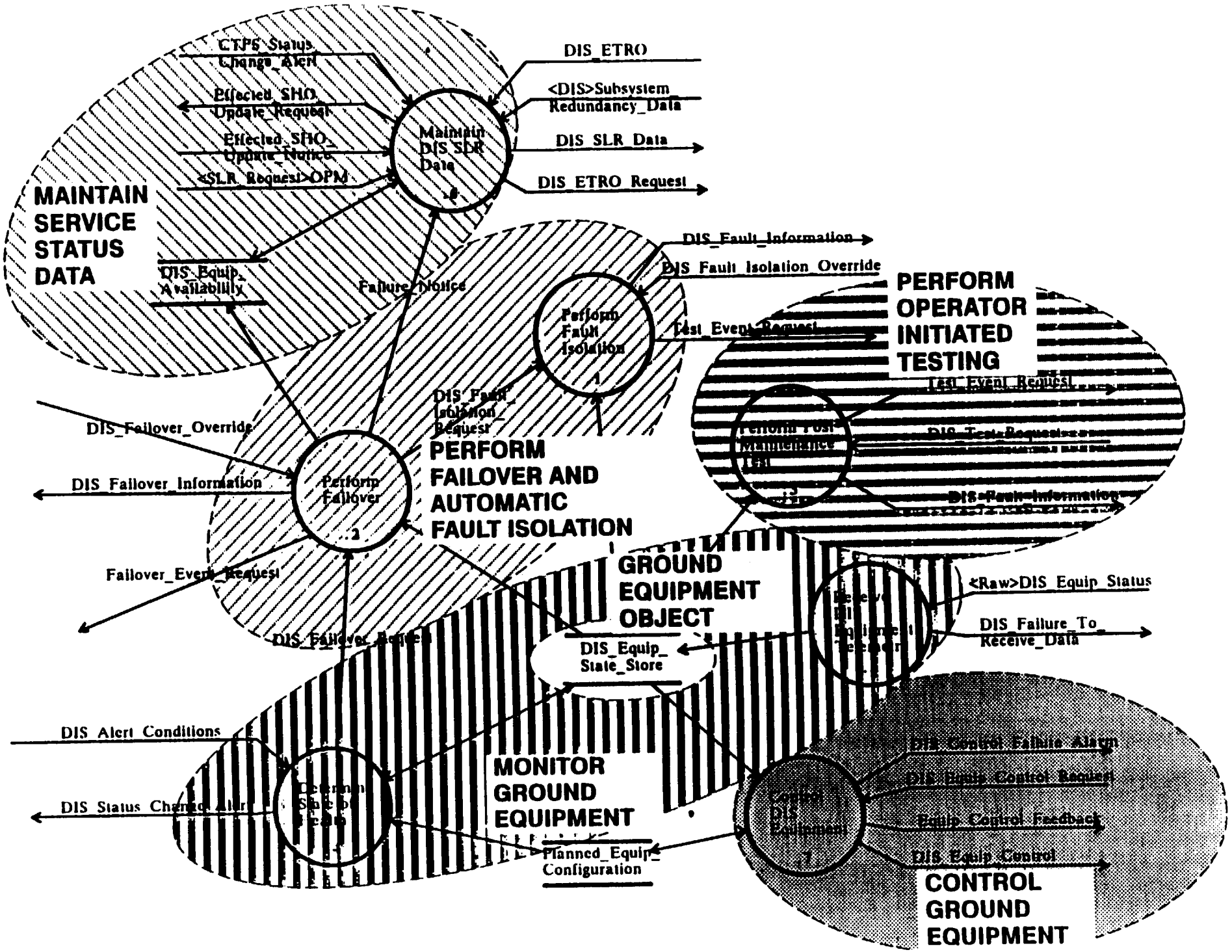
The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Benefits of SA During Preliminary Design

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- **Requirements Model Starting Point for Design**
- **Selection of Object and Process-oriented Computer Software Components (CSCs)**
  - **Application of "Edges-in" approach**
  - **Groups of transforms and their decompositions "carved" into CSCs**
  - **Data stores were initial object candidates**
- **Data Dictionary Further Refined to Reflect Interfaces Between CSCs**
- **When Design Concepts Incorporated Into Requirements Model, SRSs Repeatedly Updated**



The Application of CASE and SA To A Real-Time Ada Project...

Related Benefits of Models – Preliminary Design

STGT
Second TDRSS Ground Terminal

November 30, 1989

- **Software Requirements Models Used to Define Transactions for System Performance Study**
- **Clarity of Requirements Facilitated Software Test Plan Production**

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Benefits of SA During Detailed Design

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- **Ada Tasks and VMS Processes Identified Using:**
  - **"Edges-In" Approach**
  - **Examination of Concurrency Issues**
- **Unit Selection Performed Following:**
  - **Ada tasks *vs.* VMS processes identification**
  - **Inter-CSCI interface refinement**
- **Further Decomposition of SA Models to Address CSCI Functionality in More Detail Where Appropriate**

The Application of CASE Technology and Structured Analysis To A Real-Time Ada Project

Conclusion

STGT
Second TDRSS
Ground Terminal

November 30, 1989

- **Benefits of SA In Requirements Analysis Phase Are Obvious – Benefits in Other Phases Are Less Obvious**
- **Requirements Models Are Useful *AFTER* Requirements Analysis Phase**
- **Clarity of Requirements Early In Program Facilitates Test Plan Production**
- **Requirements Models Provided Substantial SRS Content**
- **Requirements Models Are Excellent Means of Communication With Customer**
- **Requirements Models Serve As Good Training Medium**
- **A Requirements Model With Design Flavor Can Be A Documentation Burden!!**

"ADA AT JPL: EXPERIENCES AND DIRECTIONS"

T. Fouser, JPL

JPL

Ada AT JPL: Experiences and Directions

Thomas J. Fouser

Software Resource Center (SORCE)

November 30, 1989

JPL

Ada AT JPL: Experiences and Directions

Agenda:

Global Decision Support System (GDSS)

Real-Time Weather Processor (RWP)

DSN Ground Communication Facility (GCF)

Internal Ada Training

Ada Flight Software Study

Future Ada Activities

Summary

Global Decision Support System:

Background

- 300K lines of Ada in 1000K SLOC system
- command and control system
- distributed data base DEC System
- DEC development environment
- "rapid prototype" development

Ada Issues

- early decision to go to Ada
- no in-house Ada experience; had to use contractors
- DEC Rdb access problems in Ada; FORTRAN workarounds
- Ada compile time slow (up to 5x FORTRAN); run time okay

Global Decision Support System (Cont'd):

Productivity observations

- Ada productivity high - 18 SLOC per day
- on balance, Ada not detrimental
- can not be considered an Ada "first use"

JPL Ada capability gains

- developed Ada management experience
- technology and personnel transfer to RWP
- three trained JPL programmers

  (large loss of contractor experience)

Real-Time Weather Processor:

Background

- near real-time, highly distributed DEC system
- DEC development environment; rigorous methodology
- >75K SLOC Ada; 3 builds
- tailored commercial communications software

Ada Issues

- late decision to go to Ada
- hired 2 key Ada contractors; trained 26 (half JPLers)
- Ada tasking, asynchronous processing caused problems
  - efficient workarounds found
- used DEC Windows (Ada binding not mature)

Real-Time Weather Processor (Cont'd):

Productivity observations

- early phases have taken longer
- later phases should be shorter
- quality expected to be high

JPL Ada capability gains (when RWP completed)

- will have 10-12 JPL engineers experienced in Ada
- additional 6-8 supervisors/managers trained in Ada

Ada AT JPL: Experiences and Directions

Ground Communications Facility:

Background

- world-wide real-time data communications system
- Encore/Gould development environment
- 110K SLOC Ada with high degree of common software
- "rapid prototype" development

Ada Issues

- late decision to go to Ada (post H/W procurement)
- minimal in-house experience
- Encore/Gould Ada not integrated well with OS
- memory usage higher than expected

Ground Communications Facility (Cont'd):

Productivity observations

- longer learning curve; novices wrote AdaTRAN
- overall productivity high
- longer design phase; integration/checkout shorter

JPL Ada capability gains (when completed)

- 11-13 programmers with both Ada and DSN experience
- software cognizant design engineers and managers experienced in Ada

Ada AT JPL: Experiences and Directions

Internal Ada Training:

Short-term, intensive courses brought in

- **Richard Bolz 1 week "S/W Engineering with Ada" (5x)**
- **Richard Bolz 1 week "Advanced Ada" course (1x)**

In-house courses

- **"Fundamentals of Ada" offered 4 times**
- **"Design with Ada" offered 1 time**
- **"Concurrency with Ada" in development**
- **"Management of Ada Projects" in development**
- **"Rational Fundamentals" offered 6 times**
- **"Advanced Rational" offered 4 times**

Project-specific training also

Ada Flight Software Study:

Study task:

- objective to evaluate Ada on JPL Flight Project platforms
- trained experienced flight software programmers
- benchmarked "typical" functions on NSC32xxx

Preliminary findings:

- solutions and performance very vendor dependent
- poor showing in key need areas
  - fully pre-emptive tasking
  - deterministic scheduling
- non-Ada real-time kernel plus Ada application software may have possibilities (functional and performance)

T. Fouser
JPL

Ada AT JPL: Experiences and Directions

Future Ada Activities:

TASKS	Ada KSLOC	TOTAL KSLOC	Ada PROG	DEVELOPMENT ENVIRONMENT
DoD/ASAS Prototype	5	5	4	Rational
DSN/NOCC Upgrade*	50	200	8	Rational
DoD/EUCOM	>250	?	?	Rational

* Real-time tasks with the most demanding real-time activities written in C or assembly language

Ada AT JPL: Experiences and Directions

Summary:

- "precursor" projects building JPL expertise
- "wait and see" attitude slowly changing due to successes
- some see Ada as one of several viable languages for JPL
- some organizations "bullish" on Ada

"ADA AND THE OMV PROJECT"

W. Harless, TRW

OMV Overview and Ada Lessons Being Learned

Walton Harless

Abstract. The Orbital Maneuvering Vehicle (OMV) project is involved in the development of an unmanned, remote control, reusable utilitarian space vehicle and the associated support subsystems. The vehicle is to be deployed and recaptured by the Space Shuttle. All functional requirements are derived from a set of Design Reference Missions which describe a composite set of overall capabilities. The development effort is managed by the Marshall Space Flight Center with a launch date scheduled for late 1993. The operational flight software and the ground control software are being developed in Ada. These software systems are currently in PDR phase. This paper discusses some of the Ada related observations that have been made to date.

OMV Background. The OMV project is the result of a need to extend the capability of the Space Shuttle to meet an anticipated set of diverse requirements as they evolve for the Space Station and other orbiting space platforms. There is an existing Shuttle capability to place and retrieve satellites in low earth orbit. Servicing of platforms and vehicles at higher orbits becomes considerably more impractical or impossible. The OMV capability is responsive to this need and the various OMV configurations provide a flexibility over a wide range of mission requirements.

The OMV is an unmanned vehicle that is deployed from the Shuttle and piloted from a ground station or commanded from the Space Station. The vehicle may be configured to accommodate differences in payload mass, mission length and mission duration. The OMV may be space based for an extended period between missions and refueled or serviced on orbit. Vehicle navigation is highly automated by means of the on board guidance and control software and the mission sequencing capability. The actual docking phase of rendezvous operations is accomplished by a man-in-the-loop pilot that controls the vehicle through a ground based pilot interface.

The ground station provides the pilot with mission critical data and vehicle control in real time via the NASCOM link. The on board radar information, position data and video image are displayed on the pilot station chromatics graphics terminal. All other information in the vehicle downlink is available to the pilot for analysis. This data is also made available by the ground control system in real time and historically for the various mission support and monitoring functions. The actual real time control of the vehicle is accomplished by the pilot with the custom hand controllers and switches the comprise the pilot station controls.

Software. The two basic software categories for the OMV program are the Flight Software and the Ground Software. The major portion of flight software is that which resides in the OMV on board computer (OBC) and it will be primarily Ada (95%). The remainder of that which is called flight software is non Ada software and it

consists of the embedded OMV subsystem firmware and the control software intended to reside in the Space Station computer network.

There are seven different software entities that comprise the ground portion of the OMV effort. The software for five of these areas will be largely comprised of that which has previously existed on other related programs or was developed in prototyping efforts early in the OMV program. These software systems provide the flight planning, pilot training and much of the test and integration capability. However, the actual Mission Operations Software that will reside in the Ground Control Console is to be developed entirely in Ada. Also, any additional test sets that are to be developed for use with the Electronics Ground Support Equipment (EGSE) will be written in Ada.

Of significance is the fact that the most visible and critical portions of the OMV software are to be developed in Ada. These are the Operational Flight Software and the Mission Operations Software. These two subsystems complement each other as the air and ground portions of the real time OMV capability. The flight software provides the typical on board GNC and communications functions as well as mission sequencing, status monitoring and redundancy management. The flight machine is the CDC 444RR (1750A) with a dual CPU.

The ground control software is to be hosted in the ground control console (GCC) which is comprised of two VAX 3600 machines that communicate with each other through a DMA link. In addition to pilot display and control, the ground software will provide all telemetry and command processing, data management, operations control and data analyst services.

OMV Ada Evolution. The conclusion of a programming language evaluation study during the proposal phase of the OMV project stated that Ada was the most suitable high order language (independent of hardware and other considerations) when compared to FORTRAN or JOVIAL. However, at the time of the original study (1984), additional selection factors such as existing hardware availability, software development environment maturity and the existence of "reusable" FORTRAN code for implementing OMV algorithms were enough to tilt the scale in favor of a FORTRAN implementation for both the flight and ground systems. The existing FORTRAN implementations of closely related algorithms and test systems represented a considerable amount of cost savings in the overall developed system.

Nevertheless, by joint agreement at the beginning of the contract, the suitability of Ada to the OMV project was to be reviewed. The continuing evaluation strengthened the original conclusion that Ada best met the language requirements for both ground and flight software on OMV. Then in an OMV language trade study released in March 1987, a revised conclusion stated that Ada was not only the best choice, but that no other language offered a defensible alternative for a development effort whose anticipated useful life extends beyond the end of this century. The maturity of the available support had progressed to a very credible stage. The availability of Ada experience and Ada training also looked attractive. With emphasis on the fact that Ada was the language of choice for development efforts of significant expected life span,

the decision was made to implement the major development efforts of the OMV project in Ada. This meant that the Operational Flight Software and the Mission Operations Software were targeted as Ada development efforts.

Development Teams. The flight and ground software development efforts are two distinct segments of the overall effort. It is proper to describe them in different terms since they are separated by geography as well as experience base. The flight team is largely composed of personnel with extensive experience in real time flight systems. The actual language experience is mostly assembly language with a significant amount of non Ada high order language experience. Before the OMV project, there was virtually no prior Ada experience.

The ground team background contains noticeably more application experience as opposed to real time development. The experience in high order language is much more prevalent than assembly as well. Most of the ground team had been involved with at least one previous Ada development effort, although the software systems produced were applications, software tools and environments.

All of development team members have been involved in extensive Ada training. The types and amount varied considerably. There has been a considerable amount of accredited course work as well as various types of seminar involvement, in house training and hands on experience.

Significant Implementation Factors. There are several factors that are a part of the OMV program by design that are noted at this point since these factors have a significant influence on Ada experiences thus far. Observations concerning the Ada experiences are made with these factors in mind.

1750A Flight Machine Architecture. The originally intended Litton 4516 target machine selection virtually eliminated Ada from consideration due to lack of availability of an Ada tool set. At the time of the post award language study, there existed at least four validated compilers for the 1750A and three others were to be validated shortly.

VAX/VMS Ada Development Environment. The VMS Ada environment is the system of choice as the development host for both the flight and ground software efforts. It is generally agreed that VMS is one of the most mature environments available for Ada development.

TLD Toolset. Of those systems available, the VAX/VMS hosted version of the TLD cross compiler and toolset was chosen for the flight software development effort. The factors in the decision involved the existence of the TLD Interpretive Computer Simulation for the 1750A architecture, the proximity of the TLD (Terry L. Dunbar) Corporation to the development effort and the reasonable cost.

Prototyping/Benchmarking Exercise. The decision to acquire the TLD toolset provided both a motivation and opportunity to perform a comprehensive set of prototype/benchmark tests. These tests were to evaluate the efficiency of the TLD toolset from an operational

point of view and to evaluate the efficiency of the code generated by the compiler for each language construct. Also, the benchmarks were used to evaluate the TLD 1750A simulator and run time kernels. The results from this effort were captured in the Software Standards and Procedures Document for the OMV project.

Prototype Conclusions. The overall impression of the TLD compiler is that it is very efficient and reliable. The compiler is intelligent in optimizing source code into efficient object code. There are problems and needed enhancements, but no show stoppers and the support from the vendor has been extremely responsive. The development environment is reasonable for developing software for a generic 1750A architecture. There will be enhancements required in order to emulate unique characteristics of the actual target machine. Basically, these enhancements are in the areas of simulating the timing and memory conflict associated with a dual CPU, more flexibility with regard to interrupts, a more representative I/O system and provisions for input to the 1750A simulation from an outside program.

The implementation of several Ada constructs were judged as inappropriate for use in the flight system. Generally the associated expense of such structures was cited as the offending characteristic. A detailed list of these recommendations are available, however the more significant constructs to be avoided appear to be the tasking, variant record, private formal parameters in generics and access types.

In general, any constructs that utilize dynamic memory allocation are not regarded as desirable in the flight system software. Reasons for this go beyond the concern in terms of memory and CPU expense. Of significance is that it is desirable for program execution to be deterministic for verification purposes. It is also desirable for memory to not be dynamically allocated so that everything in memory can be in a known location for telemetry fetching purposes and to accommodate on orbit patching.

Observations to Date. The VAX/VMS development environment for both the flight and ground software systems is performing very satisfactorily. The ground team uses as a basis for comparison their prior development experience in a number of environments including the Alliant, SUN and various PC hosted environments. The compiler and linker are very efficient and reliable in terms of user interface as well as the generated code. The symbolic debugger is very mature providing an extremely useful run time environment. The CMS provides a flexible library system that tracks the various modules and their change history, enforces user control of modules and provides group and class operations.

The eventual porting of software to the respective target machine is not expected to be an issue. In the case of the flight effort, the executable image will simply move from the TLD simulated environment to the target machine. The initial impressions of the simulator have been that it provides an accurate rendering of a generic 1750A target computer. Tailoring of the simulator to more closely represent the actual flight machine is expected to have been completed well in advance of the test and integration phase. The relatively small percentage of flight

software that is expected to be implemented in assembly language may be integrated within the simulator environment.

The ground software migration will be from an initial development environment on the VAX 780 to an environment on one of the VAX 3600 series computers. This move will be conducted when the hardware is available in a timeframe well before the test and integration phase. The experience of the ground team with porting Ada source code between various vendors and models of hardware indicates that this should be a relatively painless procedure regardless of the development phase in which it takes place. The ground control software is to be implemented entirely in Ada. The only exception is the Job Control Language file that actually boots the system.

The generally accepted view that the integration and test phase of an Ada development effort may be significantly reduced in comparison with the integration phase of other languages is supported by the prior Ada experience of the ground software team. The observations to date indicate that even with the very large complex systems, source code that compiles cleanly will run comparatively well the first time that it is integrated. The problems associated with inconsistencies in definition and with unexpected side effects are greatly minimized. The mature ACS will eliminate problems associated with incompatible object versions and compilations in general are simplified by the Ada packaging concepts.

Observations on Constructs. The largest single factor in determining the desirability of an Ada construct for a particular application appears to be the maturity of the compiler and the associated Ada environment. Currently, the second leading factor is the set of constraints with the target system although the importance of this may diminish as the Ada language tools continue to mature and the hosting hardware improves. These trends would tend to reduce the number and magnitude of target host constraints. Finally, an increasingly significant factor is the experience and training of the developers. As the tools and hardware improve, the familiarity of the developers with the language becomes the significant factor in determining the degree to which the Ada capability is fully exploited.

For purposes of observation, consider that there are three basic levels or categories that describe the usage of Ada constructs on a particular development effort. These are: 1) Constructs that are avoided - a diminishing yet stubborn group 2) Constructs that are applauded - those that are used universally throughout the effort 3) Constructs that are Contested - those for which no universal opinion exists. The members of these categories for the OMV project are determined officially and otherwise by the factors described in the previous paragraph. A discussion of representative members of these categories follows. The list is not exhaustive because, among other reasons, membership sets continue to be dynamic.

In the category of Constructs that are Avoided, Tasking is the most notable member. For the OMV, project the reasons are numerous and typical. There is a considerable amount of expense in terms of memory and CPU with all implementations of tasking. For the flight software, there is a desire to avoid all dynamic memory constructs

and those that would prohibit a deterministic execution. At the noise level, there is distaste for the fact that the implementations of tasking do not resemble virtual tasks in many respects.

Many constructs are applauded and for the reasons that were intended by the language design. Most notable are the packaging and information hiding constructs. Also the Ada exception handling approach offers a very clean and efficient approach to anomalous conditions. The concept of generic code is often applauded and cited as having great potential in the reusability arena; however, in the OMV effort and from experience, it would appear that there will not be an overwhelming amount of generic code in the final product.

Many constructs are contested for reasons ranging from prohibitive circumstances to preference. For example, variant records are prohibited in the flight code because the TLD compiler generates a tremendous amount of control code. However, in the ground control software, there are a number of message passing applications that are greatly simplified with the use of this construct. The Separate construct instruction to the compiler has experienced difficulty in the early VAX/VMS implementations as well as the TLD compiler. This may have influenced decisions to eliminate large employment of this construct although temporary usage of it during various development stages is quite common for convenience reasons. Mandates concerning uniform usage of context clauses receive mixed reviews from the developers. The trade off is in the area of readability versus self documentation. The relative merits of many other constructs are weighed in terms of utility and readability versus perceived or actual expense. These types of constructs include Arrays with initial values, IF statements with compound conditions and Private types as formal parameters in generics.

Overall Observations. The overall impressions with the Ada language are extremely good. This is not unexpected considering that the language of choice for the OMV project apart from non language constraints has been Ada since the proposal phase of the program. Experience since the program start continues to reinforce this position. Some of the observations follow although this is not to be considered an exhaustive list.

Prototyping. The prototyping/benchmarking exercise that was conducted shortly after the Ada implementation decisions were made produced a number of significant benefits. In addition to identifying the constraints that existed for an Ada implementation, the exercise provided an extremely good hands on learning experience. A similar exercise should be considered in the early stages of any Ada development task and especially where performance characteristics of Ada for the particular target are unfamiliar.

Maintainability. The enhanced maintainability characteristic of Ada is generally recognized. In addition, observations of the team members from previous efforts as well as the OMV effort thus far substantiate the opinion that the code is inherently very readable. The strong typing and information hiding characteristics of the

language minimize opportunities for breaking existing software in any modification or update process.

Automation. The opportunities for automation of project related activities are enormous. The structured, coherent and consistent nature of the language are of course intended to support a language oriented tool set. Of particular note in the OMV effort has been the favorable results achieved with the use of portions of the ADADL development tool set. The developers have utilized automated assistance such as cross reference aids, pretty printers and requirements allocation trackers. The PDR documentation for the flight and ground software PDRs was generated almost entirely by the DOCGEN tool. The actual development environment of the average Ada Compilation System (ACS) possess the capability to automate compilations, provide class and version operations and generate change histories. The availability of state of the art development automation as well as commercially available software packages appears to be guaranteed with the Ada language. For instance, the OMV project is utilizing a commercial data base system (SYBASE) server for all ground software data base related activities as the result of an extensive trade analysis between a large number of potential candidates. The selected system is a fast, mature and reliable system that interfaces with Ada extremely well.

A final observation concerning automation opportunities in the Ada environment concerns the capability for automatically generated graphical representations of structure and control flow from source code. Typically the Ada developers are very satisfied with Ada PDL as a very reasonable and utilitarian representation of overall structure and control flow. However, there exists at present and for the foreseeable future a very significant demand for graphical representations of the software for use in interface with management, systems engineering and the customer.

OMV OVERVIEW

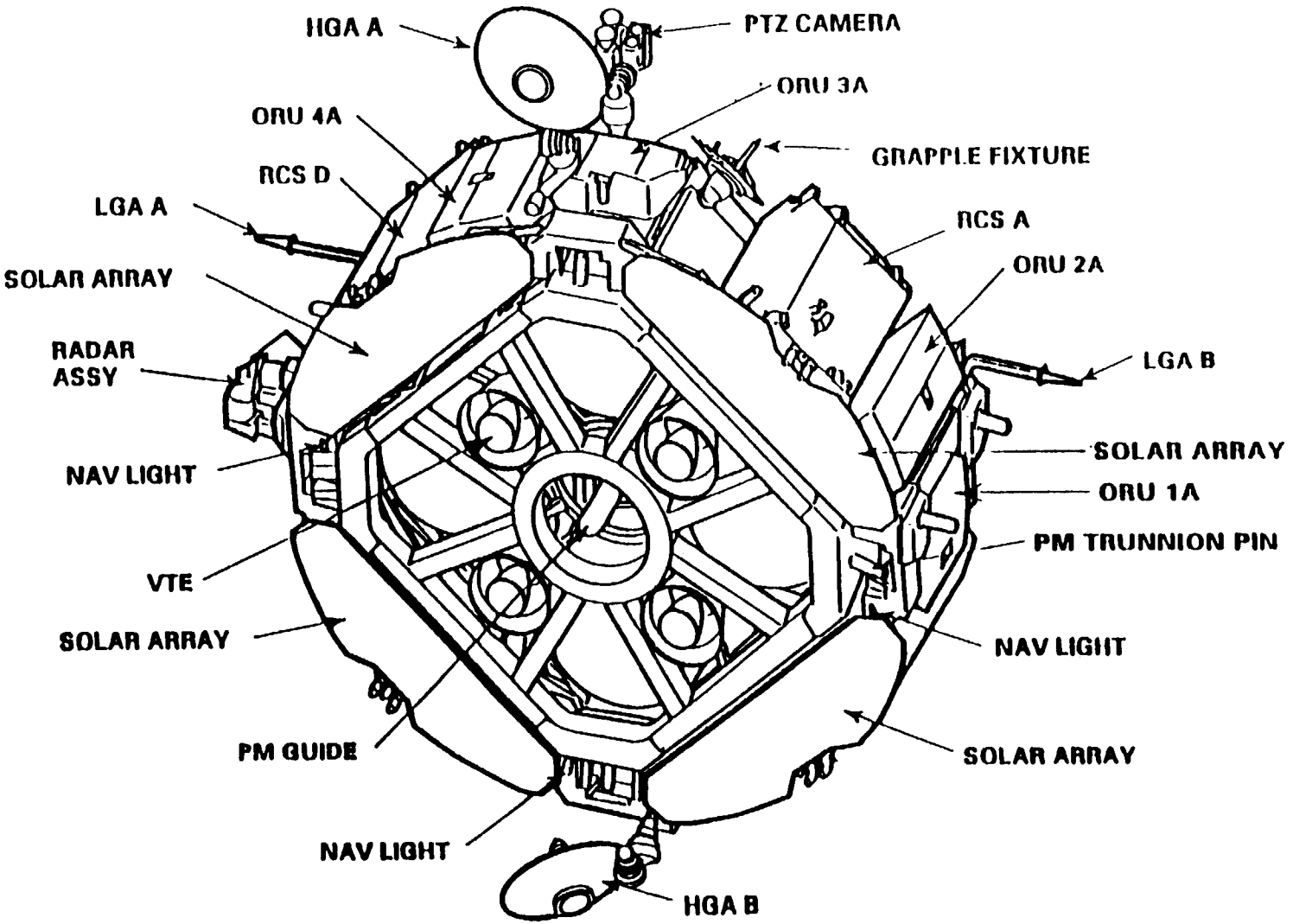
AND

Ada LESSONS BEING LEARNED

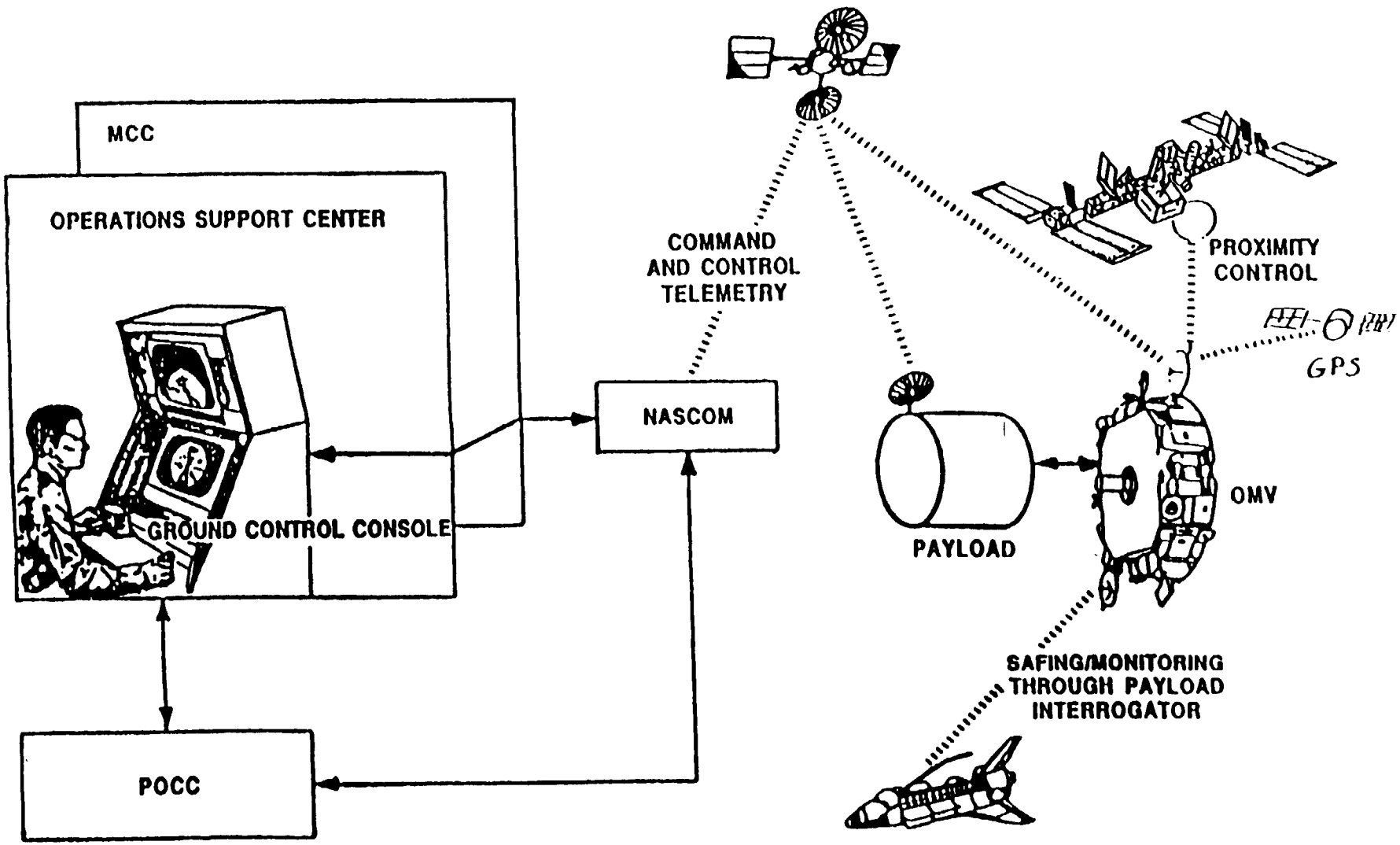
W. Harless
TRW

November 30, 1989

ORBITAL MANEUVERING VEHICLE (OMV)

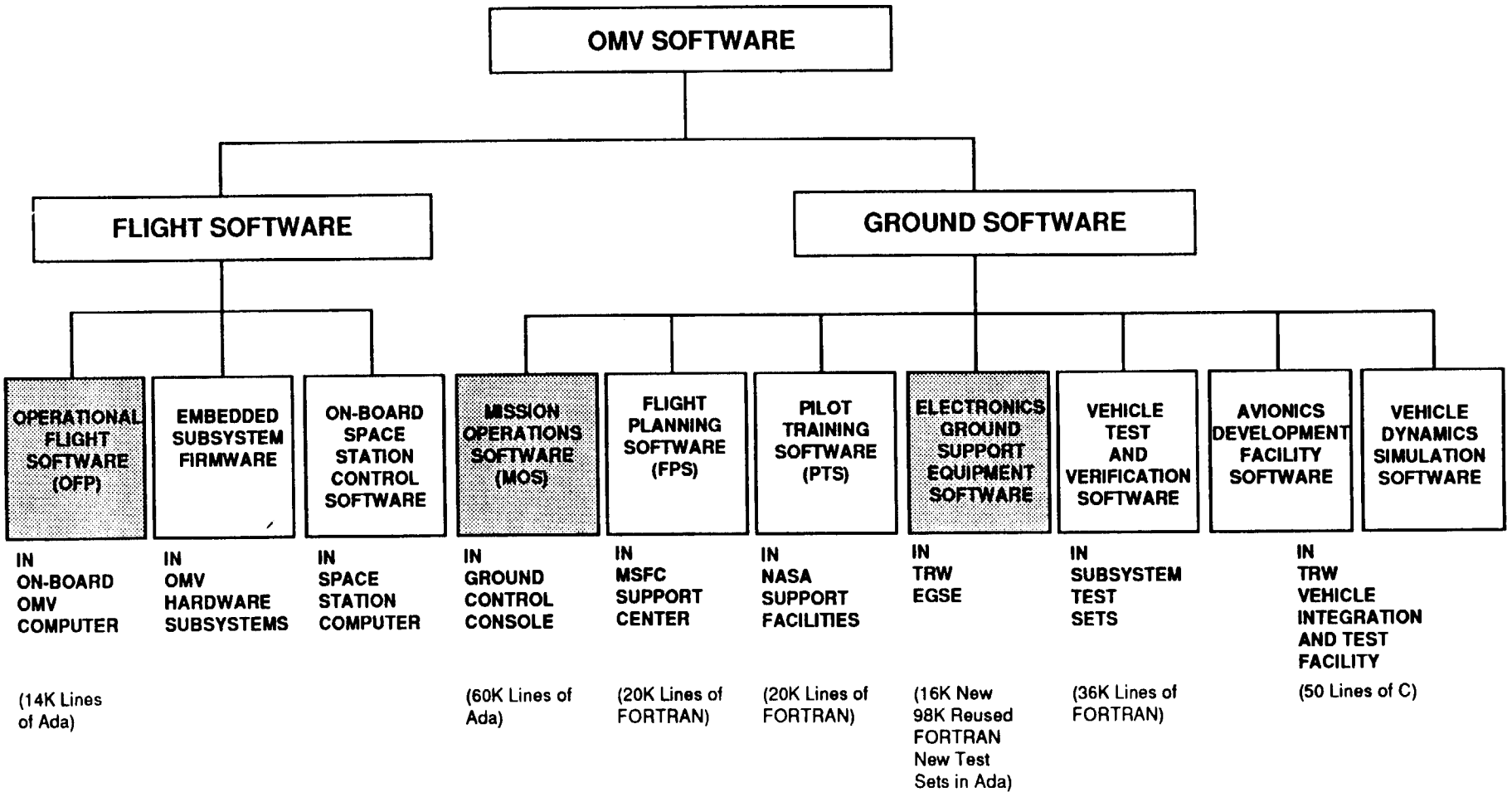


OMV OPERATIONAL COMMUNICATIONS

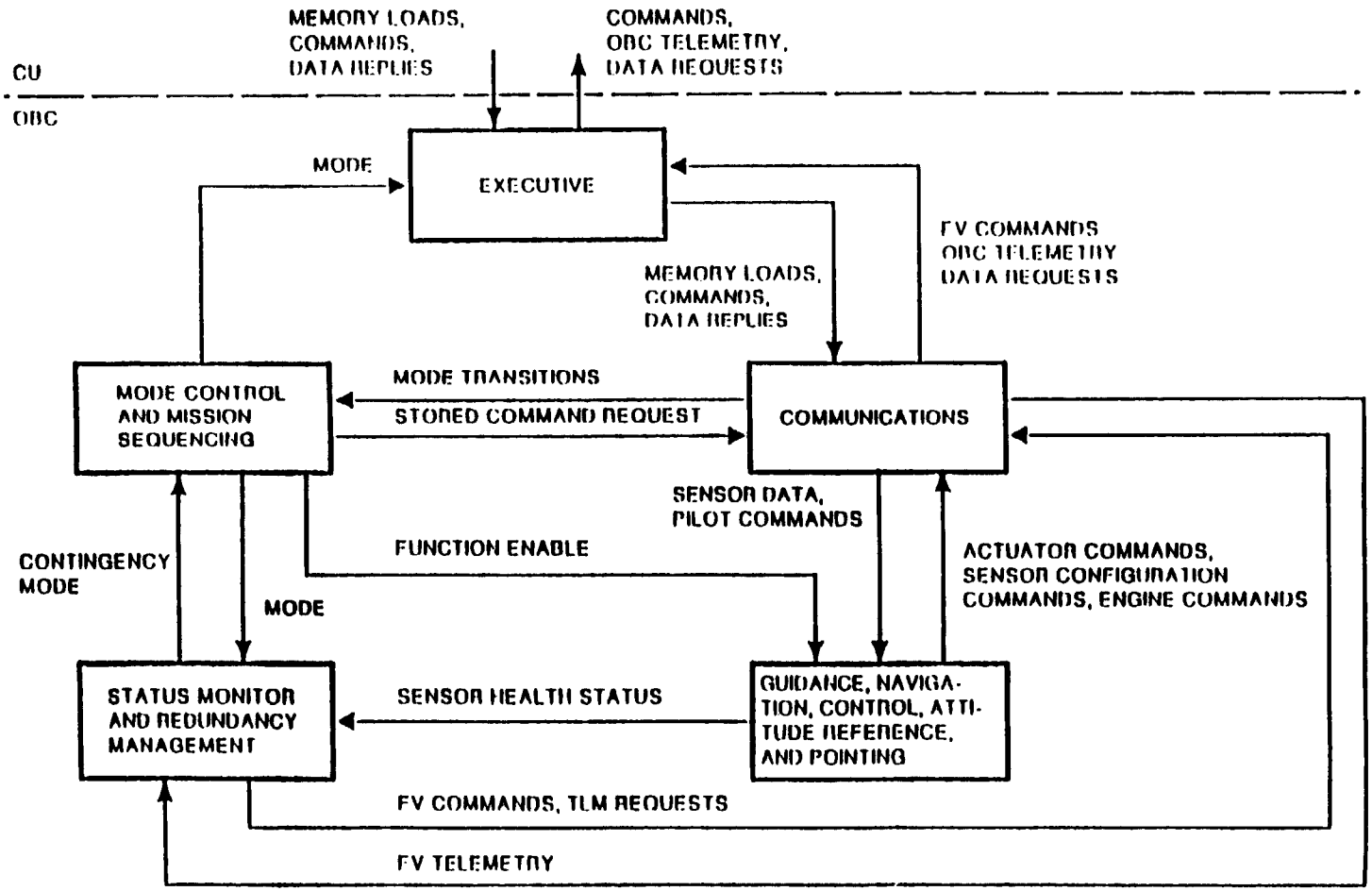


W. Harless
TRW
10 of 22

OVERALL OMV SOFTWARE STRUCTURE

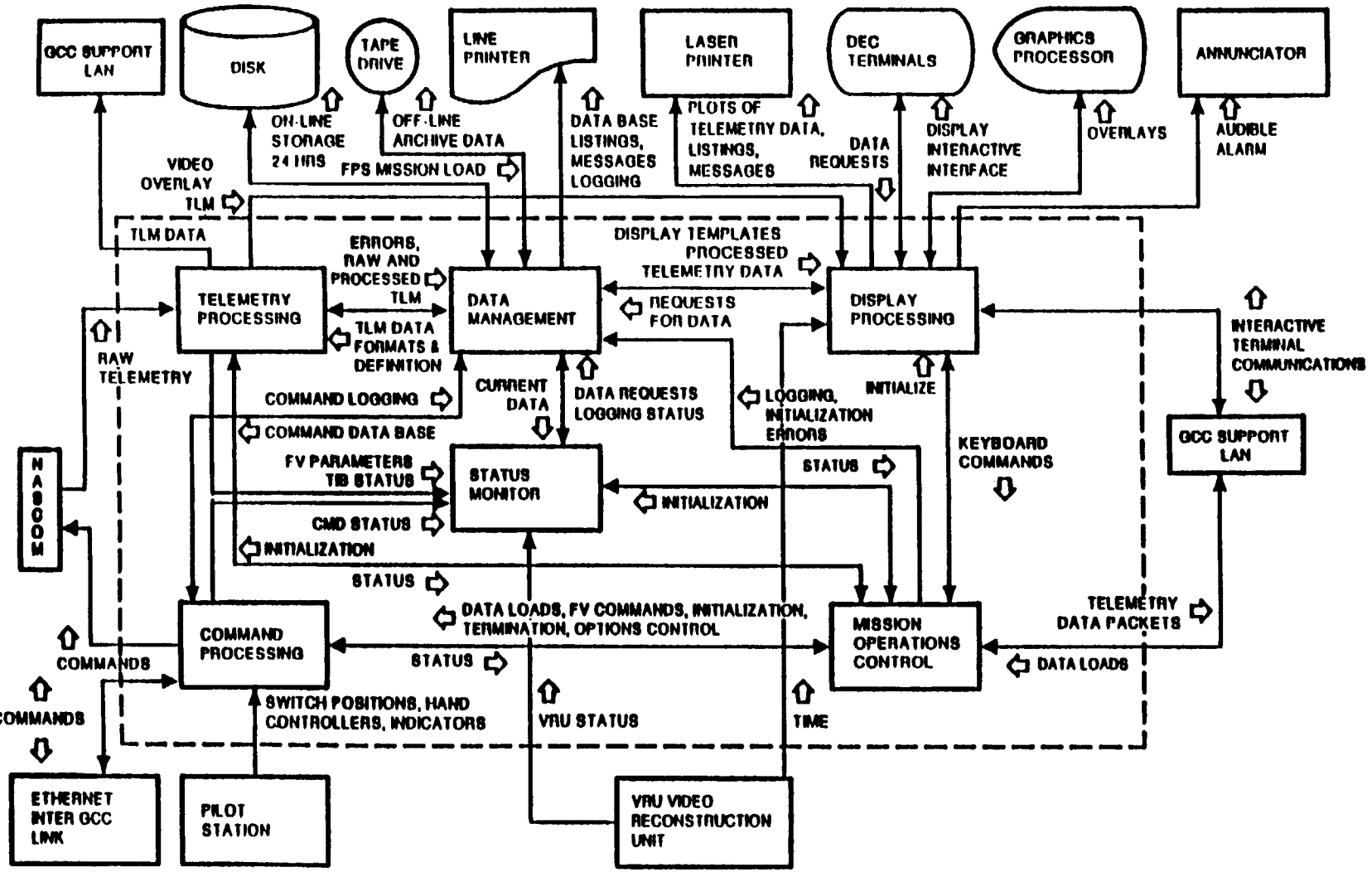


TOP-LEVEL OFP SOFTWARE DATA FLOW



W. Harless
TRW
12 of 22

MOS FUNCTIONAL BLOCK DIAGRAM



OMV Ada EVOLUTION

ORIGINAL PROPOSAL BID FORTRAN

- Known Tools & Experience
- Reuse of Existing Design and/or Software
- Cost Savings Associated with Reuse

NASA REQUESTED Ada STUDY

- Maturity of Available Support Credible
- Significant Ada Experience Available
- Paybacks in Long Term Maintenance

COMPROMISE CONCLUSION

- Flight Software in Ada
- Mission Operations Software in Ada
- New Test Sets for EGSE in Ada
- Remaining Software in FORTRAN or 'C'

W. Harless
TRW
14 of 22

DEVELOPMENT TEAM BACKGROUND

FLIGHT TEAM

- Extensive Flight Software Development Experience
- Mostly Assembly Language Implementations
- Significant Amount of HOL Experience
- Minimal Ada Experience
- All Development Team Candidates Have Participated in Ada Training

GROUND TEAM

- Diversified Development Backgrounds
- Significant Real Time Experience
- Majority of Team from HOL Arena
- Majority of Team from Previous Ada Application Experience
- Virtually No Real Time Ada Experience
- All Development Team Candidates Have Participated in Ada Training

EXTENSIVE Ada TRAINING

FORMAL/ACCREDITED COURSE WORK

- Up to 4 Semesters

INTENSIFIED SEMINAR/WORKSHOP INVOLVEMENT

- For Core of Project Development Teams

ON-SITE FORMAL TRAINING FOCUSED TOWARD PROJECT ROLES

- Ada for Managers
- Ada for Designers
- Ada for Developers

HANDS-ON TRAINING

- In-House after hours Training/Exercises
- PC-Hosted Tutorial
- Previous Projects

W. Harless
TRW
16 of 22

Ada IMPLEMENTATION FACTORS

1750A ARCHITECTURE (CDC 444R) TARGET MACHINE

- Ada enabled by revised Flight Machine Choice

VAX 8650 AND VAX 780 DEVELOPMENT HOSTS

- Mature Ada Environment
  - Compares well with SUN, Alliant, Various PC Hosts
- TLD Toolset
  - Compiler (Cross Compiler that Produces 1750A Object Code)
  - Debug Tool
  - Simulator (CDC 444R Interpretive Computer Simulator)

PROTOTYPING/BENCHMARK EXERCISE

- Evaluation of TLD Efficiency
- Ada Constructs Trade-off Analysis
- Evaluation of Tool Set

CONCLUSIONS FROM PROTOTYPING

PLEASED WITH TLD MATURITY

- Minimal Errors
- Toolset Reasonable
- Good Support

SOME CONSTRUCTS TOO "EXPENSIVE" FOR REAL TIME APPLICATIONS

- Tasking
  - Too Much CPU, Too Much Code
  - Although Other Tasking Characteristics Unatractive
    - Deterministic Requirement
    - I/O Wait in Task Disables Entire Process
- Variant Records
  - Nasty Amount of Code Generated
- Objects of Private Type as Formal Parameters to Generics
  - Reentrant Requirement Has Prolific Code Generation Results

SOME CONSTRUCTS IMPRACTICAL FOR FLIGHT REQUIREMENTS

- Dynamic Memory Constructs
  - TLM Fetching
  - Patching
  - Debug

OBSERVATIONS TO DATE (Including prior Ada Efforts)

- VAX/VMS IS MOST MATURE/RELIABLE Ada ENVIRONMENT
  - Compiler
  - Debugger
  - ACS

TRANSPORTING OF DEVELOPED S/W RELATIVELY PAINLESS

- VAX 780, 8800
- Alliant
- SUN
- PC

PORTING FROM DEVELOPMENT TO TARGET IS NON ISSUE

- 95% Ada Implementation for Flight (VAX 8650 -> CDC 444R)
- 100% Ada Implementation for Ground (VAX 780 -> VAX 3600)

INTEGRATION TIME SIGNIFICANTLY REDUCED

- Compilable Code $\cong$ correct code

W. Harless
TRW
19 of 22

OBSERVATIONS TO DATE (Including prior Ada Efforts) (Continued)

SOME CONSTRUCTS STILL AVOIDED

- Tasking

SOME CONSTRUCTS APPLAUDED

- Exception Handlers
- Packages
- Information Hiding

SOME CONSTRUCTS CONTESTED

- Use Clauses vs Dot Notation
- Separates
- Variant Records
- Arrays with Initial Values

OVERALL OBSERVATIONS

PROTOTYPING/BENCHMARKING EXERCISE WITH CANDIDATE COMPILER VERY PROFITABLE

- Identify Constraints
- Good Learning Experience

MAINTAINABILITY BENEFITS OBVIOUS

- Readable
- Side-Effects/Nuances Regulated

AUTOMATION OPPORTUNITIES ATTRACTIVE

- ADADL
  - One of many Ada Design tools
  - Cross reference
  - Pretty Print
  - Document Generator (DocGen for Preliminary Design)

OVERALL OBSERVATIONS (Continued)

- DEC ACS/CMS

  - Automated Recompilation
  - Elimination of incompatible versions
  - Library services
  - Change History
  - Class/Version Operations

GUARANTEED VENDOR SUPPORT

- Database
- CASE
- Methodology

GRAPHICS PRODUCED FROM SOURCE CODE VERY DESIRABLE

- Developers happy with PDL
- Demand from reviewers

  - Management
  - System Engineering
  - Government

SESSION 3 — SPACE STATION ACTIVITIES

Session Leader: D. Littman, NASA/GSFC

Lessons Learned: Prototyping with Ada for the Space Station Freedom Program
K. Rogers and L. Ambrose

Software Support Environment: Architecture and Design Overview
C. Carmody

Flight Telerobotic Servicer
R. LaBaugh

Lessons Learned in Prototyping the Space Station Remote Manipulator System Control Algorithms in Ada
P. Gacuk

"LESSONS LEARNED: PROTOTYPING WITH ADA FOR THE SPACE STATION FREEDOM PROGRAM"

K. Rogers, MITRE

LESSONS LEARNED:
PROTOTYPING WITH ADA FOR THE SPACE STATION FREEDOM

Kathy Rogers
Leslie Ambrose
The MITRE Corporation
1120 NASA Road 1
Houston, Texas 77058

ABSTRACT

Developing prototype software in Ada leads to some conclusions about the language as well as the available methods and services. Results from this project address the use of the Ada language in a network environment intended to emulate that which will exist onboard the Space Station Freedom. Conclusions are drawn concerning the strengths and weaknesses of Ada for prototyping projects.

PURPOSE OF THIS PAPER

This paper documents the lessons learned as a result of building the Human-Computer Interaction Lab (HCIL) Ada Executive (HAE). The HAE is a specialized program which obtains data from a testbed network built to evaluate candidate services and resources that will be required of the Data Management System (DMS) for the Space Station Freedom Program (SSFP). The HAE supplies data to the HCIL Multi-Purpose Application Console (MPAC) prototype which evaluates the presentation of data and information.

This paper is an attempt to glean from the HAE that information which may have applicability to an audience larger than that which was directly involved in the effort. The experience of prototyping a relatively small system in Ada, in a network environment, provided insight into challenges that might be expected when developing larger software systems, especially those systems that might exist on the Space Station Freedom. To that end, many details of the project's history have been omitted and other aspects have been elaborated in a manner which is not in proportion to the actual effort.

This paper will focus on the process of developing the Ada production and test software that was used as part of the larger testbed effort to evaluate various data services and resources[1]. After a brief history of the prototype, this paper addresses the Ada and software engineering issues confronted by the prototype.

BACKGROUND ON THE HCIL MPAC PROTOTYPE

The electronic component of a workstation onboard Space Station Freedom is an MPAC. The effective use of this instrument will be important to crew productivity. As such, the National Aeronautics and Space Administration (NASA) wished to investigate the human factors issues associated with the presentation of information on the MPAC. The DMS testbed at the Johnson Space Center (JSC) provided the necessary resources and data for MPAC analysis as part of the System (OMS) Integration effort.

---

[1] This project was done as part of contract NAS9-18057, project 3100K. This paper is a partial summary of a larger report, "Lessons Learned: Object Oriented Methodologies, Ada, The Data Management System Testbed, and Prototyping", JSC 23903, September 1989.

The OMS Integration effort is an ongoing series of prototype demonstrations, focussing on establishing interoperability between Space Station Freedom system simulations. It runs in the environment of the DMS Testbed, which is based upon an Apollo token ring network with Ethernet. The configuration of the DMS Testbed for OMS Integration Demonstration 3A is shown in Figure 1. System simulations are hosted on

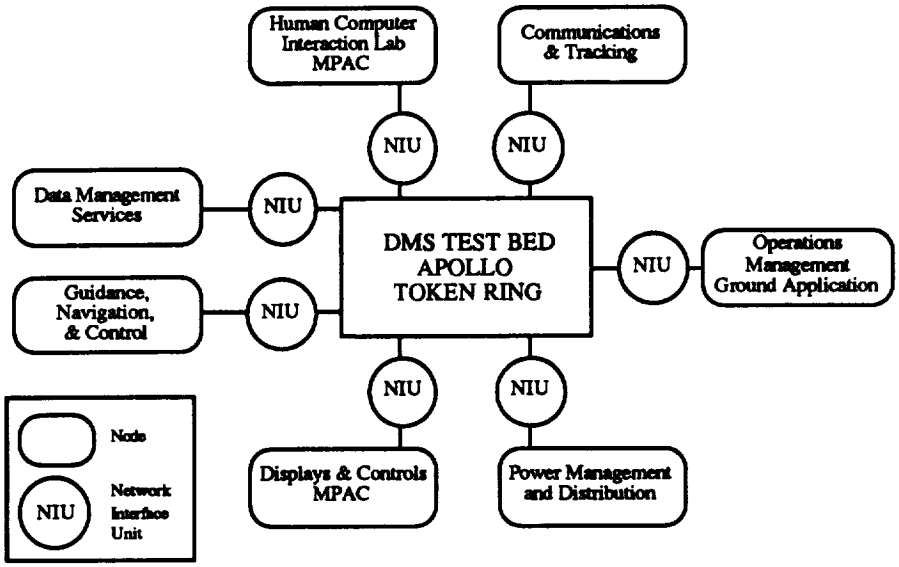


Figure 1. OMS Integration Demonstration 3A Participants

the various nodes of this network. It was in this environment that the HCIL MPAC Prototype was to become a player, focussing on the data from the Guidance, Navigation, and Control (GN&C) node. The OMS Integration effort represented a good source of realistic data which could serve as the basis for analysis of candidate MPAC displays.

A User Interface Management System (UIMS) is a piece of software which facilitates the development of consistent, effective user interfaces by providing development tools and a runtime environment to present the products of those tools. The BLOX® UIMS is controlled by user-defined state tables, which respond to interface commands by invoking user-defined programs. The combination of the commercial BLOX package, its external tables, and the display process programming, will be referred to collectively as BLOX.

The need for the HAE arose due to the fact that the services and resources provided by the DMS Testbed did not match the needs of the BLOX UIMS. The HAE acts as a middleman, requesting data from the DMS Testbed network (in the proper format) and making it available in the proper format for BLOX. Figure 2 shows the relationship of the HAE software to the other HCIL MPAC Prototype components.

® BLOX is a product of Template.

Had the Testbed services provided a more tailorable, extensible interface, there would have been no need to develop additional software.

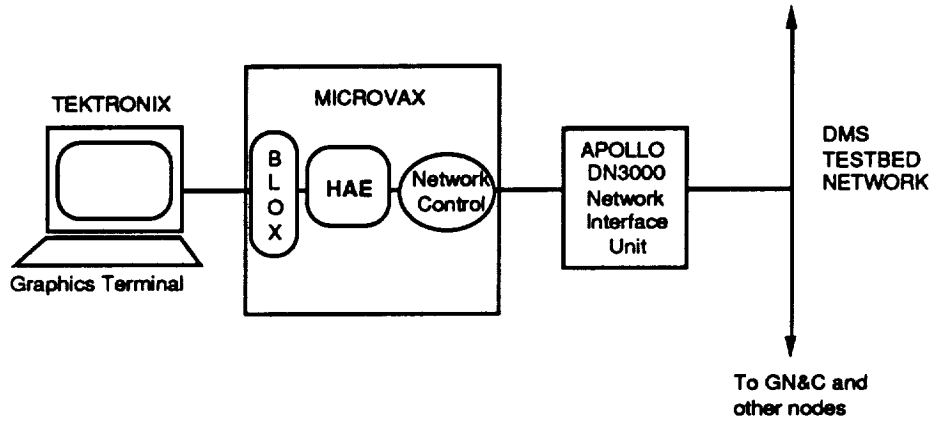


Figure 2. HCIL MPAC Prototype Components

HAE OVERVIEW

The HAE is a relatively small piece of software, consisting of approximately 2000 Ada statements of unique source code[1]. It was developed under VMS® 4.7 on a VAX® cluster, which included a VAX 785 and 8810. It was run on a MicroVAX®. The prototype configuration is depicted in Figure 3. Its executable code occupies 95K bytes and the additional object file which is linked into the BLOX program occupies 53K. No code metrics were kept during this effort, but an informal evaluation of libraries upon completion revealed that approximately 6500 lines of test code had been written.

The HAE provides services to BLOX through a series of procedure calls. The interface was designed to accommodate communication between two very different languages, C and Ada, and is simple and straightforward as a result. The HAE services include 1) periodically gathering data values from the network, 2) returning those values to the user, 3) returning the type and units of available data, and 4) stopping the HAE.

The HAE performed satisfactorily in its intended demonstration. It handled data updates from the network at the rate of 0.3 messages per second, where each message contained up to 48 data elements. The BLOX system takes several seconds to update the screen. The HAE provides data values in approximately 1/4 second per query.[1] This is fast enough for the present application, since it is not a "hard" real time system. The end user has expressed satisfaction with the system.

---

[1] Reused software is counted only once, and software from outside sources is not counted at all. The statistics on Ada statements were generated by counting semicolons which occur outside of comments and parentheses. The overall statement statistics can be found in Appendix A.

® MicroVAX, VAX and VMS are registered trademarks of Digital Equipment Corporation.

[1] This average is based upon tests using a test harness which accesses and prints system time immediately before and after issuing a call to the HAE.

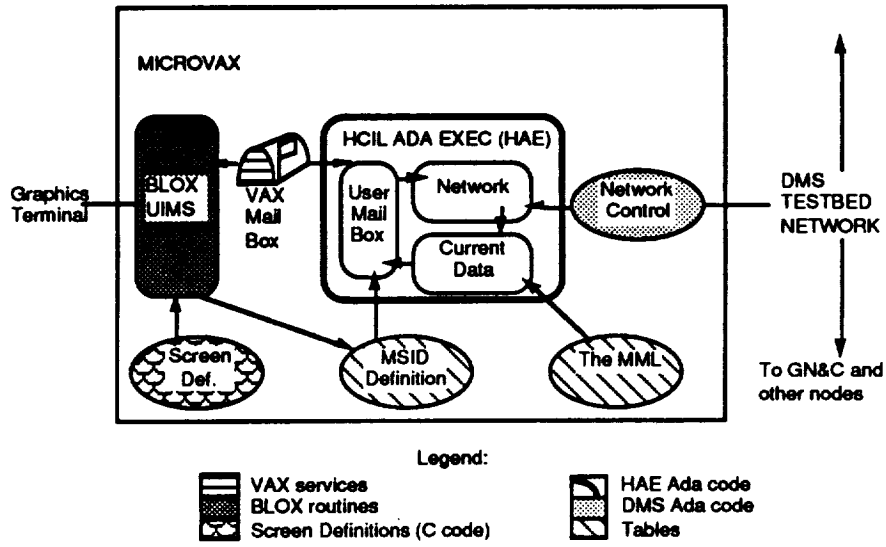


Figure 3. HCIL MPAC Prototype Software Architecture

The HAE has not been fully exercised, but so far has proven to be robust (i.e., it has not failed to perform its intended functions). It was designed to support queries to multiple systems, but this capability has not been exercised. The overall design of the HAE was found to be sound and resulted in a functional product.

ADA IMPLEMENTATION LESSONS

One of the goals of this task was to gain some firsthand impressions of the Ada language. Since the use of Ada has been mandated for the Space Station Freedom Program, the Spacecraft Software Division (SSD) at JSC was interested in exploring software engineering and Ada. The observations of the language as an implementation tool are presented in this paper[2].

The use of Ada for this project was quite successful. In many ways, the properties of the language, particularly information hiding, contributed to the success of the prototype. The problems encountered do not appear to impinge upon the usefulness of Ada as a software engineering tool or as a prototyping language. They do have implications, however, for required tools, approaches, and support systems.

The remaining discussion on Ada implementation issues is organized into two categories: successful aspects of our use of Ada, and areas which require attention in order to achieve positive results.

Successes

Ada proved to be a workable language whose use offered some specific benefits to this project. We had positive experience with Ada in the areas of information hiding, readability, language-to-language interface,

[2] Other issues such as object oriented methodologies, the Data Management System testbed, and prototype project management are described in the full report, JSC-23903.

and tasking. We also found that performance was not a problem in this system. Information hiding and readability were anticipated benefits of the language. The remainder, however, were areas where we were initially apprehensive about success. Instead, we found them to be straightforward.

Ada's *information hiding* capabilities facilitated independent development of components. The first step in the coding process was to formalize the design into Ada package specifications. The MITRE team did modify these specifications twice due to changes in types, but the consequences were minor and limited mainly to recompilation and linking. Eventually, a set of stub programs was created to allow unit testing. We possibly could have enjoyed even more of the benefits of this feature of Ada by building these stubs earlier in the project.

The use of modular specifications facilitated independent work, with each team member coding separate units. Eventually these were combined for system testing, which worked surprisingly well. No interface problems were encountered which could be attributed to the use of Ada; rather, Ada greatly facilitated this step.

The only interface problem was due to the use of integer codes to return error conditions back to the BLOX program (e.g., 0=Successful, 1=MSID_Not_In_MML). This resulted in tight coupling between several modules because the information about each value's meaning had to be contained in each module involved. This was a memory burden for the programmer as well. In an all-Ada system, this would have been neatly solved by use of an enumerated type coupled with the use of exceptions and handlers.

The Ada code for this project proved to be *readable*, at least from the perspective of the team members. We had one code review and several occasions where one member had to pick up the work of the other due to absence. Our observation was that, even with minimal use of comments, the structure and purpose of the code was accessible to the moderately informed reader. A strong reminder of this characteristic of Ada was provided when we had to make alterations to the screen definition code for BLOX, which was written in C. Minor changes proved to be a difficult undertaking. This comparison is somewhat suspect since neither team member is an expert C programmer, but both had experience with the language and the advantage of having reviewed the code with the author.

When formal prologues were included in the code, programmers making modifications were inspired to record their changes. When there was no prologue, change comments were rarely added. Providing a prologue at the beginning of the coding phase and contributing information incrementally was much easier than recreating it upon completion of coding. Consequently, we recommend the use of prologues on all files. As only a minimal amount of information was required, the sample prologue in Figure 4 could be used as a basis.

Other prologue information, which could be part of the abstract, should include the performance characteristics of the compilation unit (e.g., whether it was optimized to minimize the use of storage or minimize execution time), and the effects of the use of the object (e.g., side effects, exceptions raised, exceptions handled).

Part of this project involved an *interface between the C and Ada* languages. The independent HAE process wrote to and read from VAX VMS mailboxes. Ada procedures to access these mailboxes were made available to the BLOX C program via the DEC-supplied pragma *EXPORT*. This process worked well, due to the hospitable environment provided by the VAX VMS operating system.

```
-- Package Name: <Ada package name>, <operating system file name>
-- System Specification:
--      <hardware> running <operating system> <version/revision>
-- Abstract: <reference the design document section or page>
--      <description should include discussion of object >
--      <description of modifications should be added as needed>
-- Author(s):
--      <name> <affiliation>
-- Modification History:
--      Created: <date>
--      Modified: <date(s)>
```

Figure 4. Sample Ada Prologue

One minor restriction was found: only base types could be used at the interface of procedures to be exported. Initially, we used restricted integer types which made sense from an Ada perspective, but which caused errors when used from C. As a consequence, we had to modify some package specifications and deal directly with DEC-supplied types for floating point numbers (i.e., *g_float* or *h_float* instead of *float*). This obviously impinges on the customary benefits of Ada: we suspect that such loss at a language-to-language interface is inevitable. Where possible, such interfaces should be avoided in order to maximize the benefits of using Ada.

Evaluation of *tasking* was not originally called out as an area of investigation within this project; however, the design appeared to be most easily addressed using tasks. Each of the top-level, concurrent objects could be a task, with its methods supplied by task entries. Overall, tasking worked well in spite of our concerns that it would add a level of technical difficulty. We were able to consider each unit independently, working from specifications.

We shielded our task entries with procedure calls (Figure 5). Unshielded tasks can be aborted from anywhere within their scope, an undesirable situation for our purposes. The shielding offers the added benefit of reduced runtime overhead. If we were required to reduce the code size, the abort semantics could be removed from the task semantics since they could not be used. The shielding technique also allowed us to replace the task packages with non-tasking dummy versions. In these dummies the procedure calls, instead of calling task entries, merely supplied hard-coded values. This was helpful for testing and indicates that, had tasking proved to be a problem, we could have substituted a non-tasking version without affecting the remainder of the system.

Much has been said on the performance weaknesses of Ada. Our requirements were not strenuous, but we had to provide *adequate performance* to update a display screen fast enough for a waiting human user. We did not experience any performance problems attributable to Ada.

File I/O on the VAX was quite slow, but that is a characteristic of the system, not the language implementation. For example, one procedure call causes the HAE to open a file, read its contents and create some data structures. This is the slowest procedure call, taking approximately 2 seconds to execute.

```
...
package Master_Measurement_List is
        ...
        procedure Supply_MSID_Spec(...);
        ...
end Master_Measurement_List;

...
package body Master_Measurement_List is
        ...
        task The_MML_Process is
                entry Supply_MSID_Spec(...);
        ...
        end The_MML_Process;

        procedure Supply_MSID_Spec(...) is
        begin
                The_MML_Process.Supply_MSID_Spec(...);
        end Supply_MSID_Spec;

        task body  The_MML_Process is
        ...
                select
                        accept Supply_MSID_Spec(...)
                        do  ...
                        end Supply_MSID_Spec;
                or
                        ...
                end select;
        ...
        end The_MML_Process;

begin
        null;
end Master_Measurement_List;
```

Figure 5. Shielding Task Entries

Areas to Watch

Support is critical to effective use of any new tool. Although both members of the development team had experience in Ada development, neither was expert in the VMS environment. There were a number of times when having an expert in the combination of Ada and VMS would have greatly facilitated the development process. The usefulness to a project of one system expert should not be under rated.

Testing progressed slowly and a large amount of test code was generated, more than three times as many statements as the product. The features of Ada which make it good for large, complex projects—in particular, strong typing— slowed down the testing process. Test scaffolding took longer to create than was anticipated based on experience with other languages. "Quick and dirty" is only relatively possible with Ada. Test programs must be carefully constructed: packages must be instantiated to allow debugging print statements, types must be matched, values of records must be fully redefined with each change. This rigor is arguably a virtue, since better testing programs might lead to a stronger product. However, the additional time was not built into the schedule of this task. In a prototype, whose life span is expected to be short, it is not clear whether this rigor is a net benefit. It is therefore important to manage the whole process, planning in extra time and taking a more formal approach to testing than might otherwise be done. In a larger project, success or failure could hinge on the creation and management of test structures.

A *repository of existing Ada code* was available for reuse on our host machine. The NASA Ada Software Repository provided for the acquisition and analysis of Ada software (developed by, for, and outside of NASA) for possible distribution and reuse within NASA. Ultimately the software within the NASA Ada Software Repository will be used to seed the reuse libraries of various NASA software development efforts. Tools to qualify software items into the library, classify them according to their projected uses, and retrieve them from the library based on the users' specifications are in the analysis and evaluation stages. At the current time, the software carries no warranties because of the disparate sources of software and the diversity of conditions under which the software may be used.

Although we encountered some difficulty, we did reuse two programs from the repository, after making the minor corrections required to compile them. These programs accounted for approximately 7% of the final product. Even with the searching, analyzing, correcting, and testing, this process was much faster than developing the code. Providing libraries of reusable code is complex, but judging even by our limited use of such services, it is a potentially powerful aid to programming.

Configuration management is another difficult task in software engineering. We used the VAX's file numbering scheme coupled with named directories and paper records. Better configuration management tools and practices would have been useful during this project, although we experienced no real disasters. No code was irretrievably lost, but time was occasionally taken up in finding it. In short, we were vulnerable to machine failure or human error at many points in the project. Only the stability and small size of the design team, the relatively short development time, and a healthy dose of luck kept us from a major loss of work.

We advocate almost complete avoidance of the *USE* clause in Ada. This clause, which allows the programmer to reference parts of library packages without supplying their full names, leads to confusion for anyone attempting to read the code. Our recommendation is based upon the experience of tracing references to find the source of a certain type or procedure—a frustrating exercise. The DMS Services supplied on the testbed include the Data Acquisition and Distribution Service (DADS) and the Network Operating System (NOS). These services, used by testbed participants to effect data transfer, are supplied via a series of procedures which may be incorporated into a node's code. The sample programs demonstrating the use of these procedures employ the USE clause to the whole compilation unit and supply only local names for the type and procedure names. Since ten or more libraries are imported by these programs, the programmer is left with the options of examining the specifications of each to find the source of each type and procedure, or of including all the referenced libraries in his/her own code.

There are very few, if any, good uses for the *USE* clause. Currently available pretty printers do not provide the capability to fully qualify references, to our knowledge. Even if that capability became available, it may still be inappropriate to use the *USE* clause because it can increase compile time for resolving overloaded subprograms. Not only must a compiler find an appropriate reference (in terms of subprogram name, number of arguments, type of arguments, etc.), it must assure that there is no other subprogram that could possibly fit the profile. Therefore, it has to look at everything that has been referenced by a *USE* clause.

Fully qualified names are generally straightforward and readable for procedures and data types. However, they are less convenient for infix operators. In an application which requires an overloaded infix operator (or whenever package names become cumbersome), the *RENAMES* construct is preferable to the *USE*. For example, consider overloading the "+" operator for addition of two matrices. Direct referencing of the operator is clear but awkward:

```
Result := Math_Routines."+"( Left, Right)
```

The renames construct, however, allows a more natural format:

```
-- declarative part
function "+"(L, R:  Some_Type) return Some_Type renames Math_Routines."+";
...
-- sequence of statements
Result := Left + Right;
```

Additionally, only the "+" operation would be visible, eliminating the readability and compilation problems within the scope where the "+" is needed. To make other subprograms visible, other renames clauses would have to be used.

In a tasking program, *robustness of tasks* becomes very desirable. User-defined exceptions whose handlers caused tasks to fail were difficult to diagnose and were removed from the final system. There was no situation encountered in this project where failing was preferable to continuing to run, although one can imagine a program such as a robot driver where the opposite might be true.

The *experience base* of users familiar with both VMS and Ada is limited, and DEC's Ada is not as mature as some of their other languages. We encountered a number of areas where we were unable to find other users who had exercised the capabilities we needed, particularly in the use of VAX system services. At least part of these troubles can be attributed to our lack of experience with VMS. The DEC documentation was a source of problems for two reasons: the necessary information was spread out over a number of manuals (five in one case[1]); and the Ada documentation contained many inaccuracies. Needless to say, this slowed down our efforts.

The VAX *debugger* changed the observed behavior of the system, particularly in matters related to timing. Code compiled with the debug option produced different results from that compiled without debug, even if run in the */nodebug* mode. Fortran programmers report that this is a known characteristic of the debugger, so this does not appear to be an Ada-only problem. It should be noted that, even with this

[1] The problem was the use of mailboxes. The manuals were the following: VAX Ada Programmers Run-Time Reference Manual; VAX Ada Language Reference Manual; VAX VMS System Services; VAX VMS I/O User's Reference Manual; Developing Ada Programs on VAX/VMS.

problem, the debugger was a useful tool. However, the confusing results combined with our unwise faith in the tool slowed down the resolution of some problems. In this, and other implementation problems, we found it helpful to use the technique of keeping a "lab book" describing the problem encountered, steps taken to isolate the problem, and the success or failure of each attempt. This book fostered deliberate debugging and acted as a communication device during the absence of a team member.

Some apparently *inconsistent results* were obtained. One particularly annoying problem was the failure of some code to "scale up". It seemed nonsensical to claim that the mailbox reads and writes worked in a small program but not when included in the whole system, yet that was the observed behavior. A systematic set of tests finally revealed the culprit, as shown in Figures 6 and 7. Calls to the package VAX_Mailbox_Services, which invoked VAX system services, failed when done from a subprogram defined within the scope of the main procedure. The simple test program had its calls in line and therefore avoided this problem. We are unaware of any place where this is documented.

In a process which uses tasks, direct calls to VMS general input/output system service routines *$QIO and $QIOW* from a single task result in blocking the whole process. DEC Ada tasking is implemented using a run-until-blocked method. According to the VMS documentation, the system does not recognize that the individual task is waiting and therefore does not allow any other task to run (Digital, February 1985). In order to avoid this affect, the user is directed to employ the DEC-supplied package Tasking_Services which avoids this problem. An additional work-around is offered by the DEC-supplied pragma *TIME_SLICE*. This allows the programmer to control the maximum amount of time given to any one task.

SUMMARY

This project resulted in a working prototype which met its objectives of being a participant in the OMS Integrated Demonstration and of providing a tool for examining data presentation issues. The HAE software itself is a functional piece of software which could be used to attach another node to the DMS Testbed, at least until the Testbed Services are upgraded in 1990. Any UIMS or other software which can call Ada procedures in a VAX VMS environment could make use of the HAE.

In the process of creating one portion of the prototype, the HAE, lessons were learned on the techniques and tools used to create the software as well as about the process of prototyping itself. Ada is a useful language whose benefits were clearly recognized during this project. Its information hiding and tasking were particularly helpful: the use of Ada specifications allowed independent development of packages, and the tasking model fit well into a design of independent functional units. Each portion could be considered in isolation.

Some planning, however, is required to avoid potential trouble spots. The youth of the language means that the support systems, both human and machine, are not as mature as is desirable. The availability of good programming tools and of a repository of indexed and qualified reusable code would greatly speed project development. Further, the aspects of Ada which are its strengths, such as compiler-enforced typing, mean that quick and dirty development is only relatively possible. This does not mean that it is not a good prototyping language, but the schedule impacts must be considered. A second prototype, implemented by the same team, would undoubtedly be much faster to develop than the first.

```
--This program does not work.
--
with Text_IO;
with Starlet;
with VAX_Mailbox_Services_Without_Tasks;

procedure User_Mailbox_Manager is
        package VMS renames VAX_Mailbox_Services_Without_Tasks;

        Receiving_Channel   Starlet.Channel_Type;
        Sending_Channel    : Starlet.Channel_Type;

        procedure Start_Mailbox(    R_Channel       : out Starlet.Channel_Type;
                                    S_Channel       : out Starlet.Channel_Type ) is
                From_BLOX           : VMS.HAE_or_BLOX := VMS.HAE;
                For_BLOX            : VMS.HAE_or_BLOX := VMS.BLOX;
                Start_Status        : VMS.HAE_Status := 0; -- o.k.
                S_Mailbox_Status    : VMS.Status_Value := VMS.Success;
                R_Mailbox_Status    : VMS.Status_Value := VMS.Success;
                Sending_Channel     : Starlet.Channel_Type; --for Sending mailbox

        begin
                VMS.Create_Mailbox( For_BLOX,
                                    S_Mailbox_Status,
                                    S_Channel ); -- for sending
                VMS.Create_Mailbox( From_BLOX,
                                    R_Mailbox_Status,
                                    R_Channel ); -- for receiving
        end Start_Mailbox;

begin
                Start_Mailbox(Receiving_Channel, Sending_Channel);
end User_Mailbox_Manager;
```

Figure 6. Indirect Call Which Fails

```
-- This program works.
--
with Text_IO;
with Starlet;
with VAX_Mailbox_Services_Without_Tasks;

procedure User_Mailbox_Manager is
        package VMS renames VAX_Mailbox_Services_Without_Tasks;

        R_Channel : Starlet.Channel_Type;
        S_Channel : Starlet.Channel_Type;

        From_BLOX          : VMS.HAE_or_BLOX := VMS.HAE;
        For_BLOX           : VMS.HAE_or_BLOX := VMS.BLOX;
        Start_Status       : VMS.HAE_Status := 0; -- o.k.
        S_Mailbox_Status   : VMS.Status_Value := VMS.Success;
        R_Mailbox_Status   : VMS.Status_Value := VMS.Success;
        Sending_Channel    : Starlet.Channel_Type; --for Sending mailbox

begin   -- procedure Start Mailboxes and Network

        VMS.Create_Mailbox(  For_BLOX,
                             S_Mailbox_Status,
                             S_Channel );  -- for sending

        VMS.Create_Mailbox(  From_BLOX,
                             R_Mailbox_Status,
                             R_Channel );  -- for receiving

end User_Mailbox_Manager;
```

Figure 7. Direct Call Which Works

APPENDIX A

STATISTICS ON MODULES AND ADA STATEMENTS

During the process of writing the code, information was not captured about versions or number of lines written or tested. Instead, after-the-fact analysis has been done on the rather *ad hoc* library structure which was in place upon the successful completion of the product. We attempted to garner some information about reuse of code from external sources, the creation of code that was reusable within the context of the product, and the amount of test code required to produce a working result.

The following assumptions were made in examining the libraries:

- It is not feasible to determine reuse below the module level.
- If a module has a unique name, it is counted as a unique piece of code, recognizing that much code was duplicated.
- If two modules of the same name have different numbers of statements, they are considered to be two different modules.

The modules were classified by purpose (*product*, *discarded*, or *test*) and by source (*new*, *reused*, or *new used multiple times*). For each category, the number of modules and Ada statements was determined. The following table contains the overall results:

	Statements	Modules	Statements/Module
Total	11802	188	62
Product	2212	33	67
Test	6581	118	55
Discarded	3009	37	81
Reused	228	4	57
New	11419	179	63
New Multiple	155	5	31

To examine just the product code for reuse, the following table was developed:

	Statements	Modules	Statements/Module
Product	2212	33	67
Reused	174	2	87
New	1929	27	71
New Multiple	109	4	27

Some ratios of possible interest follow:

	Statements	Modules
Test / Product	2.9	3.5
Reused Product / Total Product	0.08	0.06
New Multiple / Total Product	0.05	0.12
New Multiple+Reused / Total Product	0.13	0.18

These ratios reveal that the amount of test code is roughly three times the size of the product. Reused code from external sources accounted for about 7% of the code, while code employed multiple times was 5% of product statements and 12% of modules. The combined multiple purpose code accounted for 13-18% of total product code.

BIBLIOGRAPHY

Ambrose, L.A. and K. L. Rogers (September 1989), "Lessons Learned: Object Oriented Methodologies, Ada, The Data Management System Testbed, and Prototyping," JSC-23903, Houston, TX: The MITRE Corporation.

Boehm, B.W. (1981), *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall.

Booch, G. (1987[a]), *Software Components with Ada*, Menlo Park, CA: Benjamin/Cummings Publishing Company, Inc.

Booch, G. (1987[b]), *Software Engineering with Ada, Second Edition*, Menlo Park, CA: Benjamin/Cummings Publishing Company, Inc.

Cohen, N. (1986), *Ada as a Second Language*, New York, NY: McGraw-Hill.

Digital Equipment Corporation (February 1985), *VAX Ada Programmer's Run-Time Reference Manual*, Maynard, MA: Digital Equipment Corporation.

Lockheed Engineering and Sciences Company (August 1988), "The End-to-End Test Capability (ETC) Test Bed Nodal Interface Control Document (ICD) (Initial Phase of Demonstration No. 3)," LESC-25117, Houston, TX, Job Order 33-102, Contract NAS9-17900.

NASA Goddard Space Flight Center (May 1987), *Ada Style Guide, Version 1.1*, Software Engineering Laboratory Series, SEL-87-002, Greenbelt, MD: Goddard Space Flight Center.

Nissen, J. and P. Wallis (1984), *Portability and Style in Ada*, Cambridge, MA: Cambridge University Press.

Robinson, T. (March 1988), "Requirements for Project Management in a High Productivity Software Engineering Environment," *unpublished*, Houston, TX: The MITRE Corporation.

Swartout, W. and R. Balzer (July 1982), "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM*, XXV:7, pp. 438-440.

Ulmer, J. (June 1989), "OMS Integration Test Bed With OMGA Control," Test Report (Demonstration 3A), Job Order 052-32-369, Houston, TX: TRW.

ACRONYM LIST

ACS	Ada Compilation System
ADS	Ancillary Data Service
ANSI	American National Standards Institute
DADS	Data Acquisition and Distribution Service
DEC	Digital Equipment Corporation
DMS	Data Management System
GN&C	Guidance, Navigation and Control
HAE	HCIL Ada Executive
HCIL	Human Computer Interaction Laboratory
JSC	Johnson Space Center
MML	Master Measurement List
MPAC	Multi Purpose Application Console
MSID	Measurement/Stimulus Identifier
NASA	National Aeronautics and Space Administration
OMS	Operations Management System
SSE	Software Support Environment
SSD	Spacecraft Software Division
SSFP	Space Station Freedom Program
UIMS	User Interface Management System
UMM	User Mailbox Manager

Lessons Learned: Prototyping with Ada for the Space Station Freedom

Kathy Rogers
Leslie Ambrose

30 November 1989

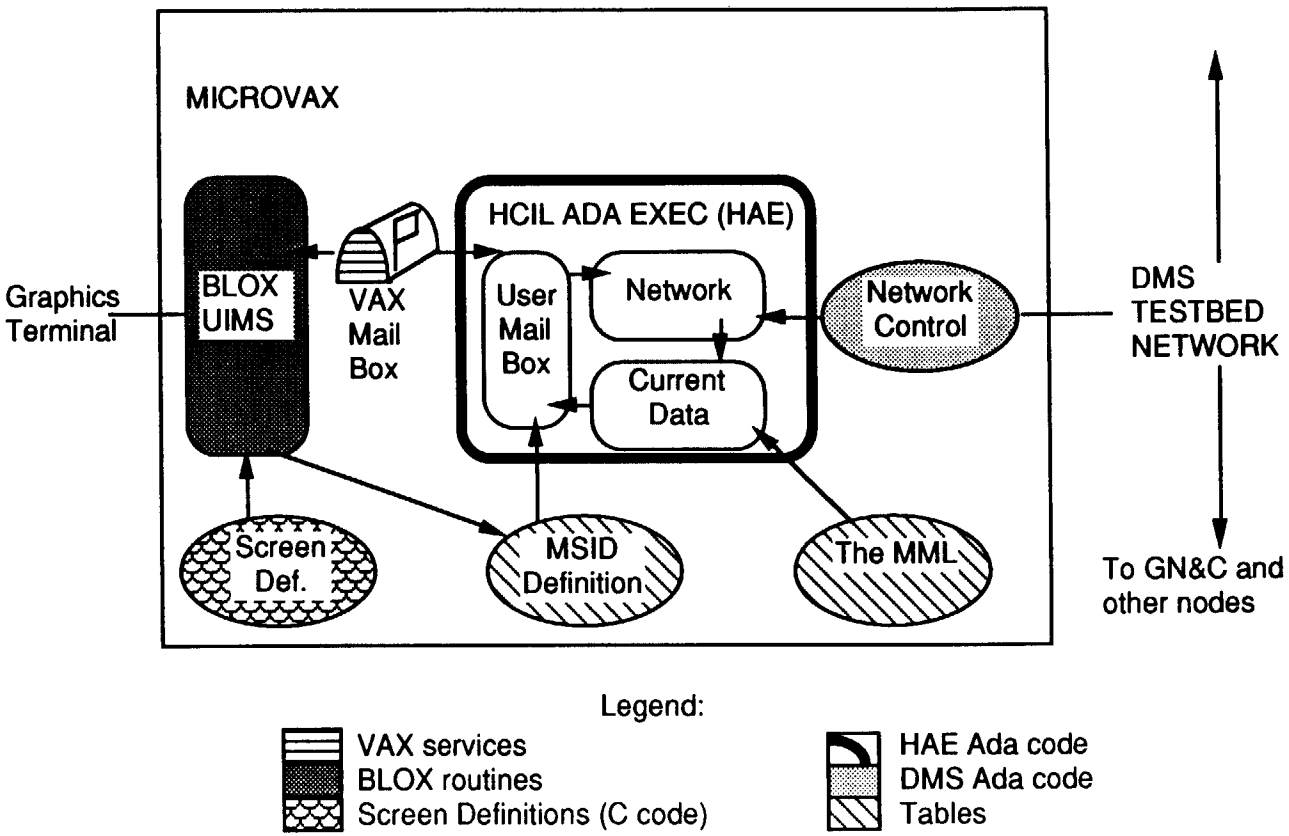
MITRE

History

- Motivation for This Effort
  - Need to examine human factors issues for the Multi-Purpose Application Console (MPAC)
  - JSC SSD desire to explore software engineering issues associated with Ada and Object Oriented Methodologies
  - Existence of the Operations Management System (OMS) Integration Group working on the DMS Testbed
  - Availability of realistic data from DMS simulations
  - Exploration of user interface issues in the DMS

K. Rogers
MITRE
18 of 29

How the Human Computer Interaction Lab (HCIL) MPAC Works



K. Rogers
MITRE
19 of 29

Statistics on the HCIL Ada Executive (HAE)

- Consists of 2000 Ada statements of unique code (approximately 6500 additional statements of test code)
- Handles network communication at 1 message/3 seconds, each message containing a maximum of 48 Measurement Stimulus Identifiers (MSIDs)
- Provides MSID values to BLOX at rate of 0.25 seconds per query

Ada Lessons - Successes

- *Information hiding*: stubs and specifications facilitated smooth integration of separately coded units
- *Readability*: team members found code accessible; advocate use of formal prologues
- *C to Ada Interface*: DEC's pragma *EXPORT* had only one restriction, only base types could be used
- *Tasking:* facilitated OOD by providing a natural model of the system
- *Performance*: no problems attributable to Ada

K. Rogers
MITRE
21 of 29

Ada Lessons - Potential Difficulties

- Effective use of tools requires support (page 7)
- Test creation takes longer than with other languages (page 8)
- Standards to support Ada coding development are needed (page 9)
  - *Use* clause example
- DEC-specific Problems Slowed Progress (page 12)

K. Rogers
MITRE
22 of 29

Effective Use of Tools Requires Support

- *Environment support* is critical: could have used an Ada and VAX expert
  - Documentation was not always clear
- Better *development tools* needed than full screen editor and compiler/linker
  - Tool training was lacking
- *Configuration management* could have been a problem
  - Project size resulted in small problems

K. Rogers
MITRE

Testing Creation Takes Longer

- Ada requires more time for coding test
  - I/O instantiations
  - Other test scaffolding
- Prototyping sometimes conflicts with good software engineering
  - Quality requires time

K. Rogers
MITRE
24 of 29

Standards to Support Ada Development

- Ada *code repository* is useful
  - Recommend descriptive indices and certified quality
- Integer return codes increase coupling
- Prologues encourage program documentation during development rather than after the fact
- *Use* clause increases complexity and reduces readability
  - Recommend coding standard restricting its use

K. Rogers
MITRE
25 of 29

Rationale for Restricting the *Use* Clause

- Issue:
  - Imported program's types and procedures may be used without supplying fully qualified names. This was done extensively within the DMS Services
- Problems:
  - Tracing references is difficult
  - Compile time increases since compiler must search all referenced library units to ascertain that there is no ambiguity
- Solution: use *Renames* construct
  - To allow infix notation
  - To shorten extremely long names

Renames Clause as an Alternative

- To allow infix notation
  - Without *use* or *renames*

    ```
    Result := Math_Routines."+"(Right, Left) ; -- awkward
    ```

  - With *renames*

    ```
    function "+" (R, L: some_type)  renames Math_Routines."+";
    ...
    Result := Left + Right;  -- better
    ```

- To shorten extremely long names

  ```
  package The_MML renames The_Master_Measurement_List;
  ```

K. Rogers
MITRE
27 of 29

DEC-Specific Problems

- DEC user experience with system service calls from Ada is limited
- Debugger changes behavior of the code
- VAX system services to create mailboxes fail when done from a subprogram
- VMS general I/O services block whole process

K. Rogers
MITRE

Conclusions on HCIL MPAC Prototype

- Ada is a useful, powerful tool
  - Information hiding and tasking were known benefits
  - Prototyping capabilities were good for HAE application
- Planning needed to leverage Ada
  - Coding standards would improve language use
  - Environment-specific problems should be anticipated
  - Development support requires tools and tool training
  - Expert consultant on system service issues is desirable
  - Testing requires adequate schedule allowances

K. Rogers
MITRE
29 of 29

"SOFTWARE SUPPORT ENVIRONMENT: ARCHITECTURE AND DESIGN OVERVIEW"

C. Carmody, PRC/GIS

2ND NASA ADA USERS SYMPOSIUM

SSE ARCHITECTURE AND DESIGN OVERVIEW

C. L. CARMODY

30 NOVEMBER 1989

SSE ARCHITECTURE AND DESIGN

Contents

- SSE Architecture
- SSE Design Status/Overview

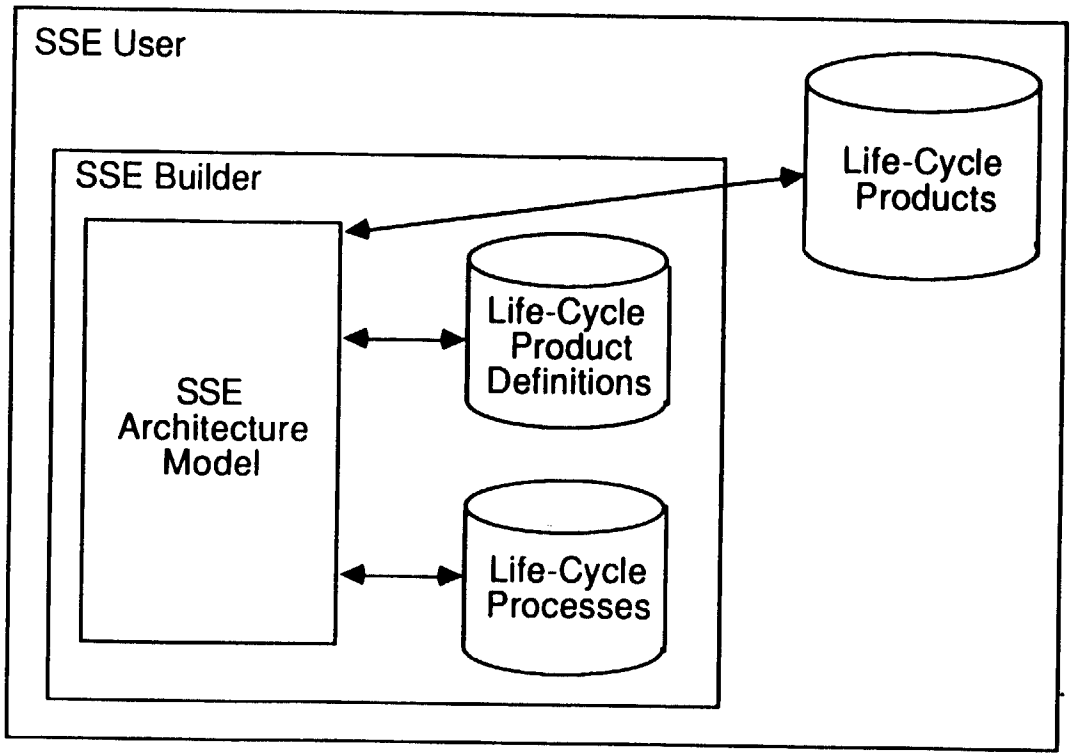
SSE ARCHITECTURE AND DESIGN

SSE Architecture as documented in DRLI 58; the SSE Architectural Design

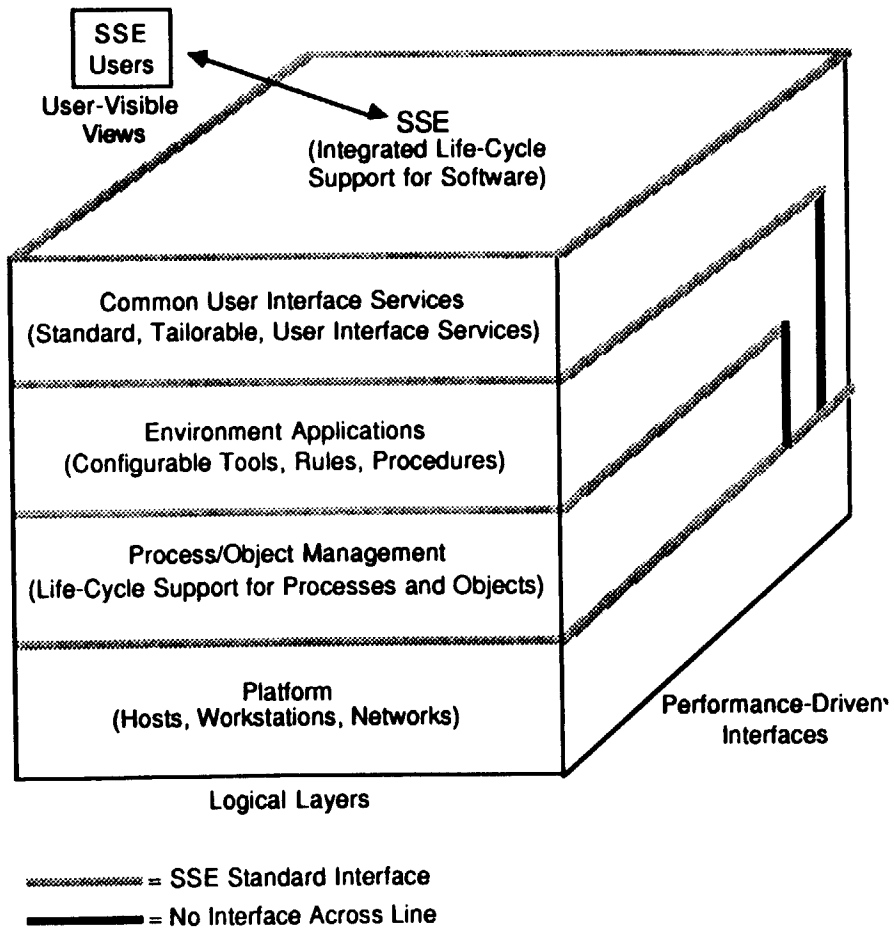
- SSE Architectural Model
- SSE High Level Object Model
- High Level CSCI Structure
- Key Attributes of the Architectural Design

SSE PROJECT OVERVIEW

Top-Level View of SSE Architecture

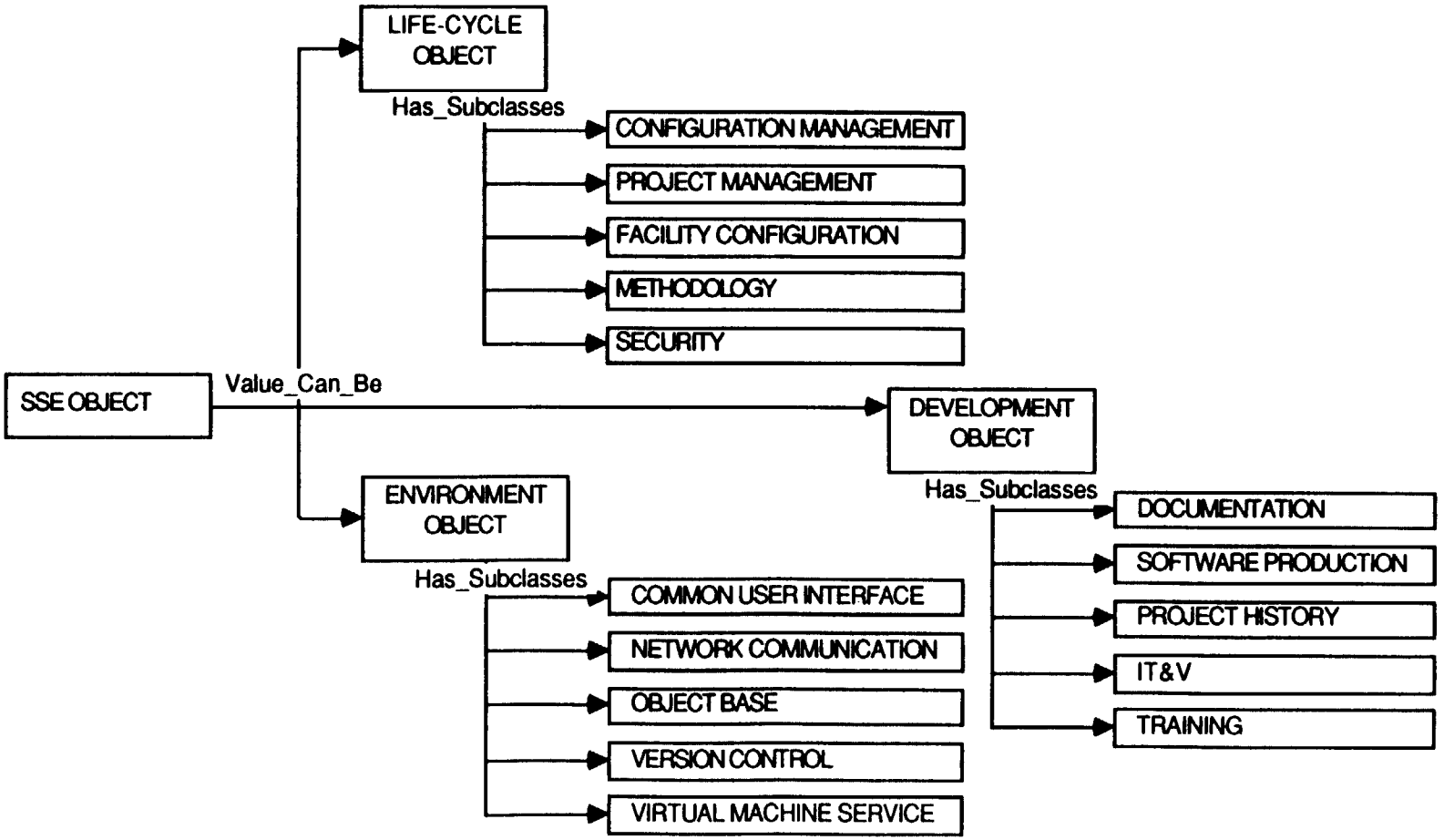


SSE Architectural Model



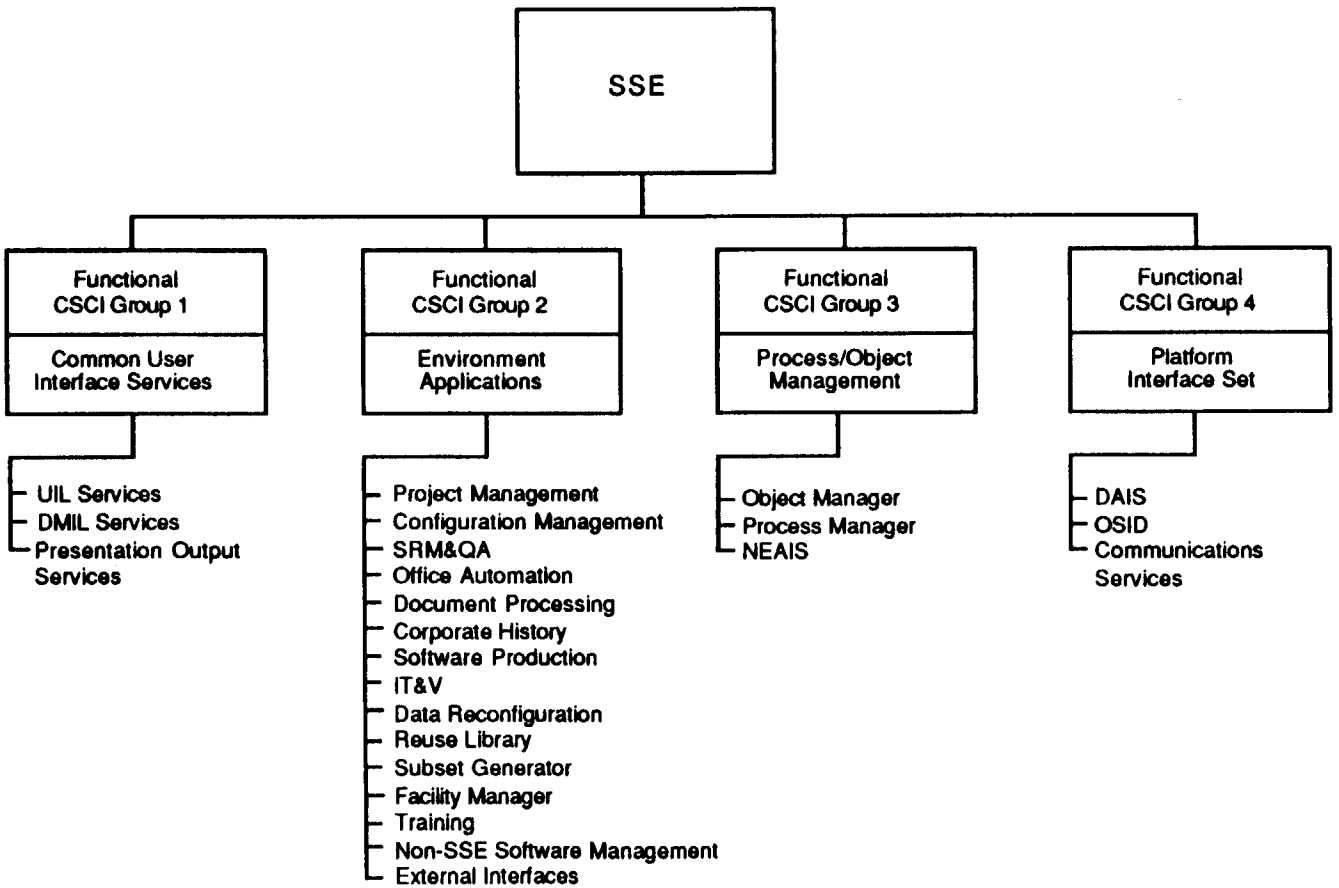
SSE ARCHITECTURE AND DESIGN

The High Level Object Model



SSE ARCHITECTURE AND DESIGN

The High Level CSCI Structure



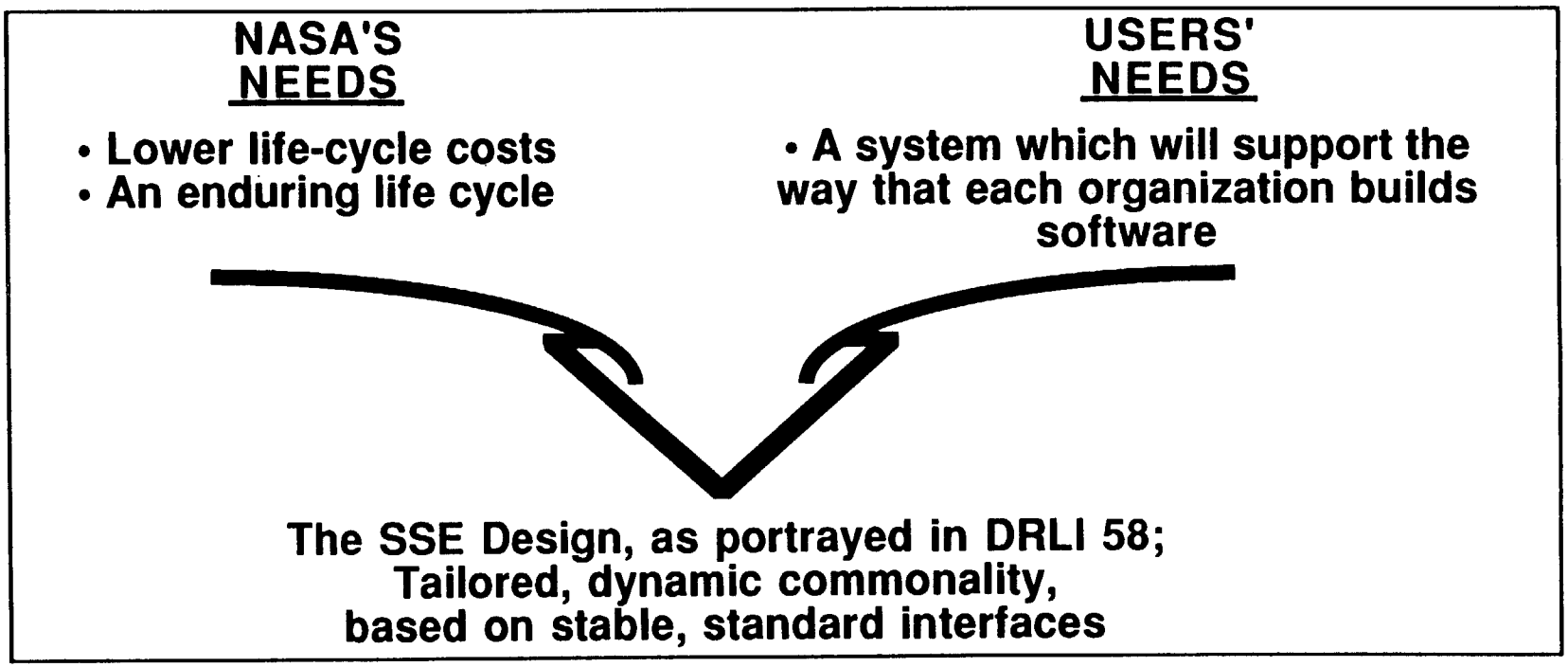
SSE ARCHITECTURE AND DESIGN

Key Attributes of the Architectural Design

Major goal of the SSE Design

Significance of Object Oriented Approach

SSE ARCHITECTURE AND DESIGN



The SSE Architectural Design must satisfy both long-term requirements for technical viability, and immediate, user-oriented needs. These two demands are often in conflict.

SSE ARCHITECTURE AND DESIGN

Significance of Object Oriented Approach

Reduction of total life-cycle costs

- improving programmer productivity
- encouraging reuse
- reducing maintenance costs

Enhanced security

- systems resist both accidental and malicious corruption

SSE ARCHITECTURE AND DESIGN

SSE Design Overview

Design Products Prepared for each CSCI/CSC

Some Example Design Products

SSE ARCHITECTURE AND DESIGN

Design Products Prepared for each CSCI/CSC

- Data Flow Graphs (DFGs)
  - "Cloud Graphic"
  - Notes
- User Functionality Trees (UFTs)
  - User Functionality Screens (UFSs)
  - Response/Behavior Information
- Object Model
  - Object and Class Specification (OCS)
- Risk Analysis
- Make/Buy/Adapt Recommendation
- Standards

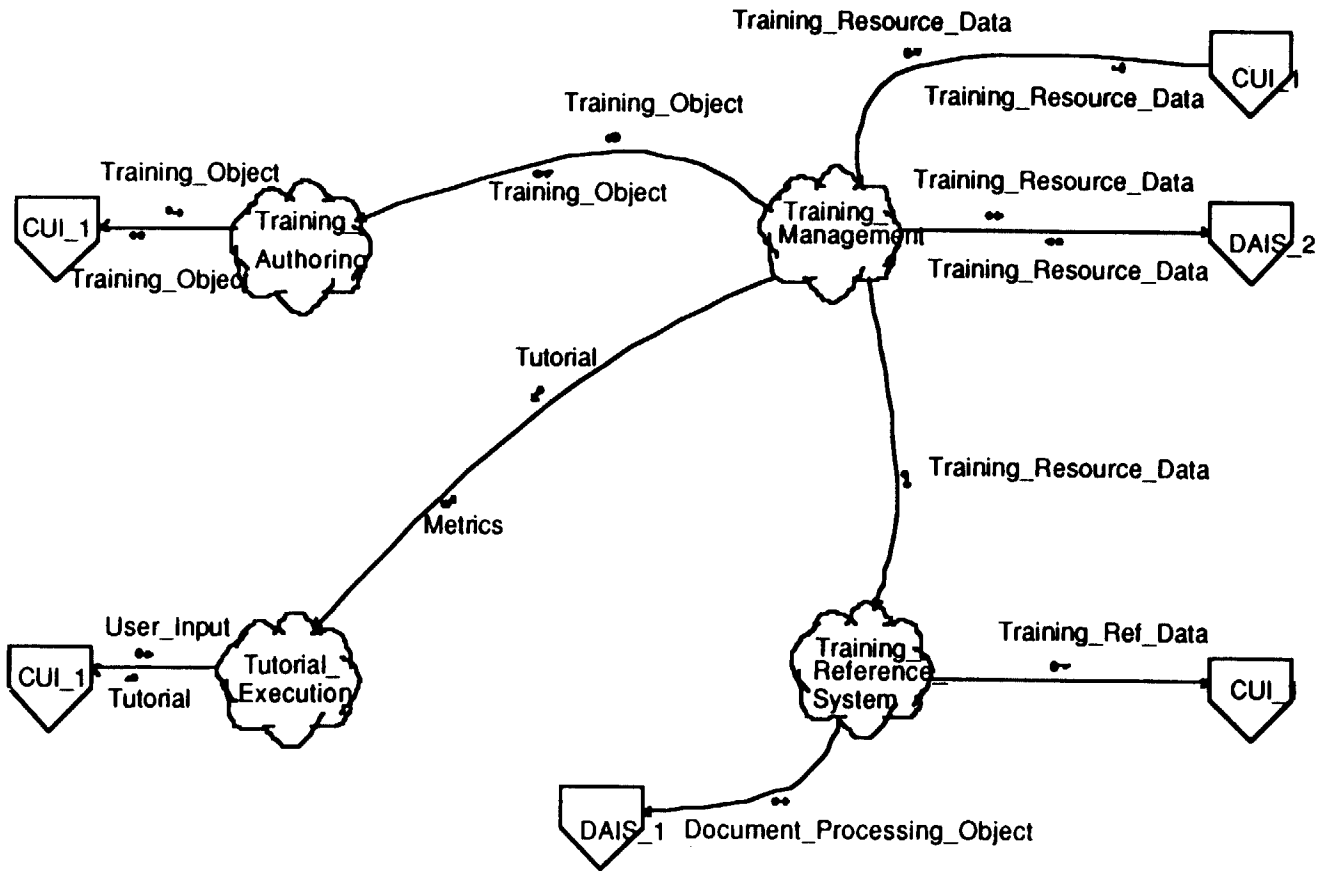
SSE ARCHITECTURE AND DESIGN

Some Example Design Products

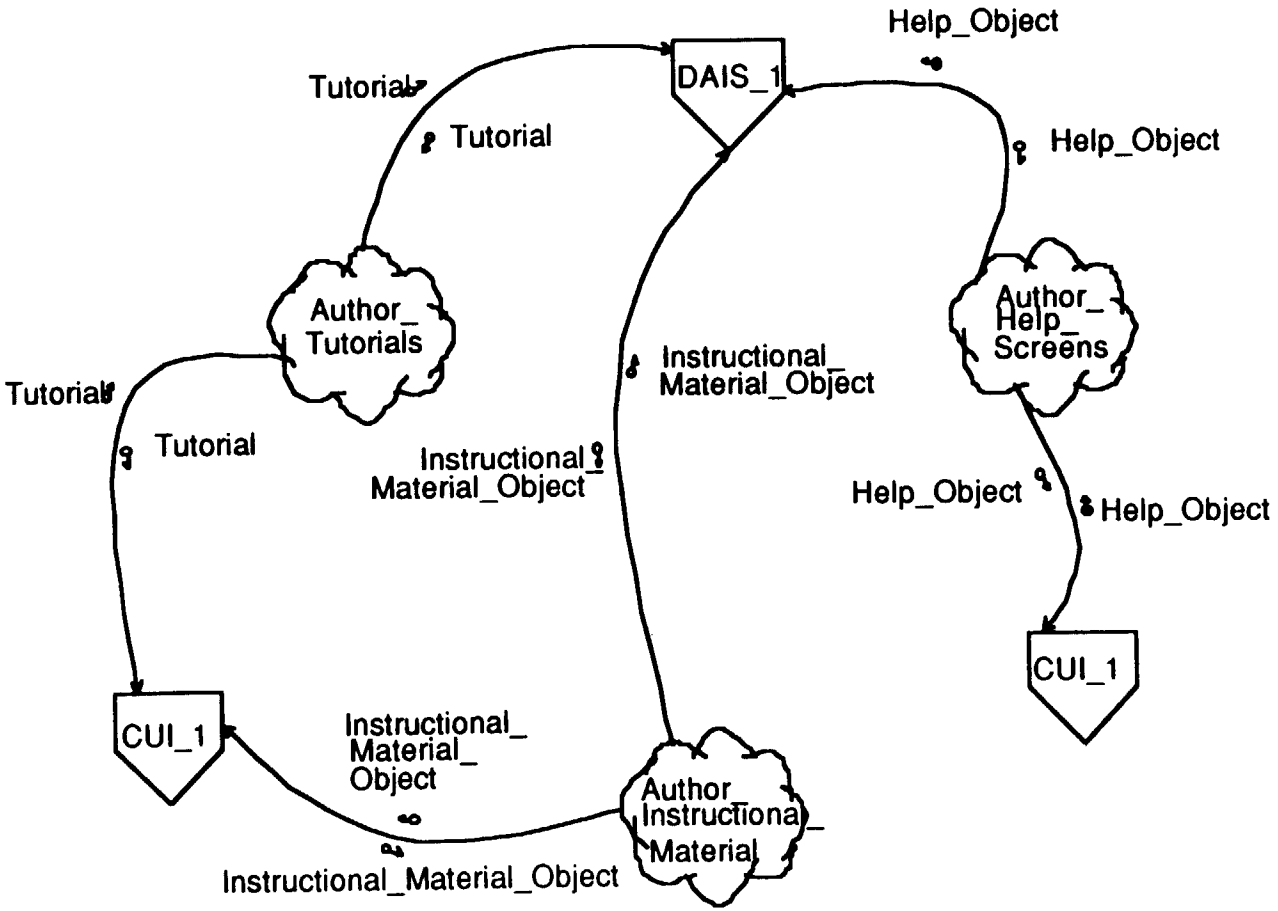
- Data Flow Graph for the Training CSCI
- Data Flow Graph for the Training CSCI Authoring CSC
- User Functionality Tree for the Training CSCI (partial)
- SSE Object Class Specification - Outline

SSE ARCHITECTURE AND DESIGN

Data Flow Graph for the Training CSCI



Data Flow Graph for the Training Authoring CSC



SSE ARCHITECTURE AND DESIGN

User Functionality Tree for the Training CSCI (partial)

Author	Tutorial	Attributes	Create
			Modify
			Delete
			View
		Lesson	Create
			Modify
			Delete
			View
			Import Data
	Help Screens	For Tool Name/ID	Create
			Modify
			Delete
			Import Data
		For Procedure/Policy	Create
			Modify
			Delete
			Import Data
	Instructional Material	Create	Course ID
			Type
			Inst Material Data
		Modify	
		Delete	
		Import Data	

SSE ARCHITECTURE AND DESIGN

SSE Object Class Specification - Outline

1.0 Precise and Concise Description

1. Abstraction/Definition - the object's role in the "real" world of software
2. Purpose/Rationale
3. Other Characteristics: Indicate whether the object is static or dynamic, and whether it is a metaclass.

Do not include parameters, constants, exceptions, or operations.

2.0 Static Representation

Semantic Network for the object class, including relations, constraints, other class, and "is_a" class if appropriate. Attributes are allowed using the "has_attribute" relation.

3.0 Operations

Suffered Operations table:
Operation, Response Behavior;
split by Constructors, Selectors, Iterators
Required Operations table:
Operation, Description

SSE Object Class Specification - Outline, continued

4.0 Dynamic Representation

State transition diagrams
Include state information as needed, in the form of notes.
(most entries to be provided at Detailed Design)

5.0 Constants

To be provided at Detailed Design

6.0 Abstract Interface Specification

Operation, Parameters, Range, Direction, Parameter Class or Description, Exceptions

Flight Telerobotic Servicer

Robert J. LaBaugh
Martin Marietta Astronautics Group
Space Systems
Denver, Colorado

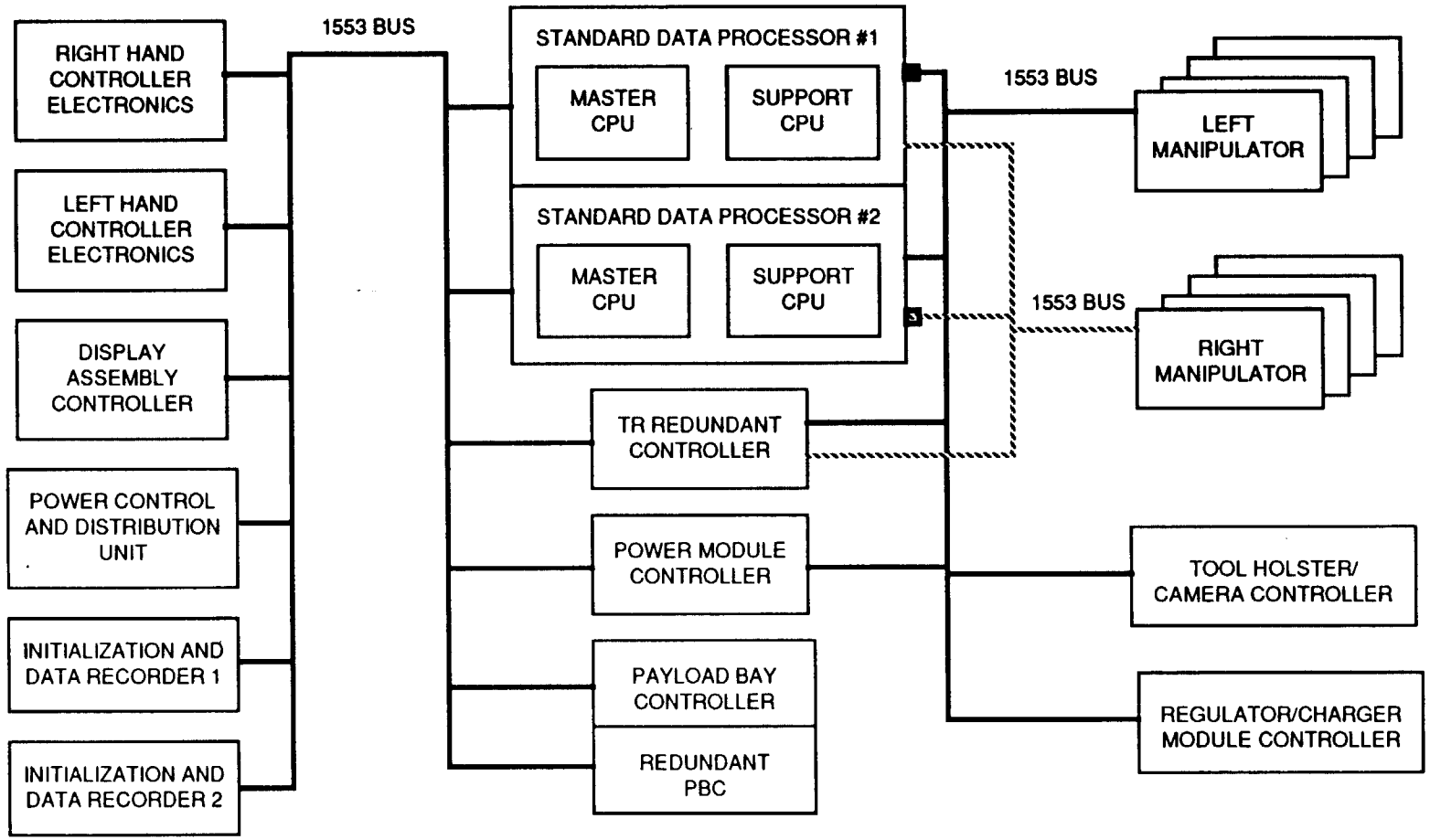
"FLIGHT TELEROBOTIC SERVICER"

R. LaBaugh, Martin Marietta

Software Categories

- Flight Software
  - 60 K Lines of Code
  - Telerobot Control, Operator Interface, Safety
- Simulator
  - 9 K Lines of Code
  - Graphical Display of Robot and Workspace
- Trainer
  - 30 K Lines of Code
  - Hydraulic Manipulator for 1 G Operation
- Electrical Ground Support Equipment
  - 75 K Lines of Code
  - Test and Test Support Code
- Engineering Test System
  - 50 K Lines of Code
  - Development Test Bed
- All Software Written In Ada
- Extensive Reuse of Flight Code In Simulator, Trainer, EGSE, ETS

Flight Computer Architecture



Allocation of Flight Software Elements to Processors

SDP Master CPU
- Manipulator Primitive Control
- Part of Manipulator Servo Control
- End Effector Primitive Control
- Hand Controller Primitive Control
- Part of Hand Controller Servo Control

SDP Support CPU
- Telerobot Control
- Manipulation
- Perception
- Telerobot Functional Control
- Part of Manipulator Servo Control
- Teleoperation
- Part of Workstation Communications

Display Assembly Controller
- Workstation Functional Control
- Workstation Power
- Part of Workstation Thermal

Hand Controller Electronics
- Part of Hand Controller Servo Control

Wrist Roll/Tool Controller
- End Effector Changeout Mechanism Control
- Tool Servo Control
- Wrist Camera Control
- Part of Manipulator Servo Control

Other Joint Controllers
- Part of Manipulator Servo Control

Allocation of Flight Software Elements to Processors (cont)

Redundant Controller
- All of Safety

Tool Holster/Camera Controller
- Holster Control
- Head Camera Control
- Contamination Sensors

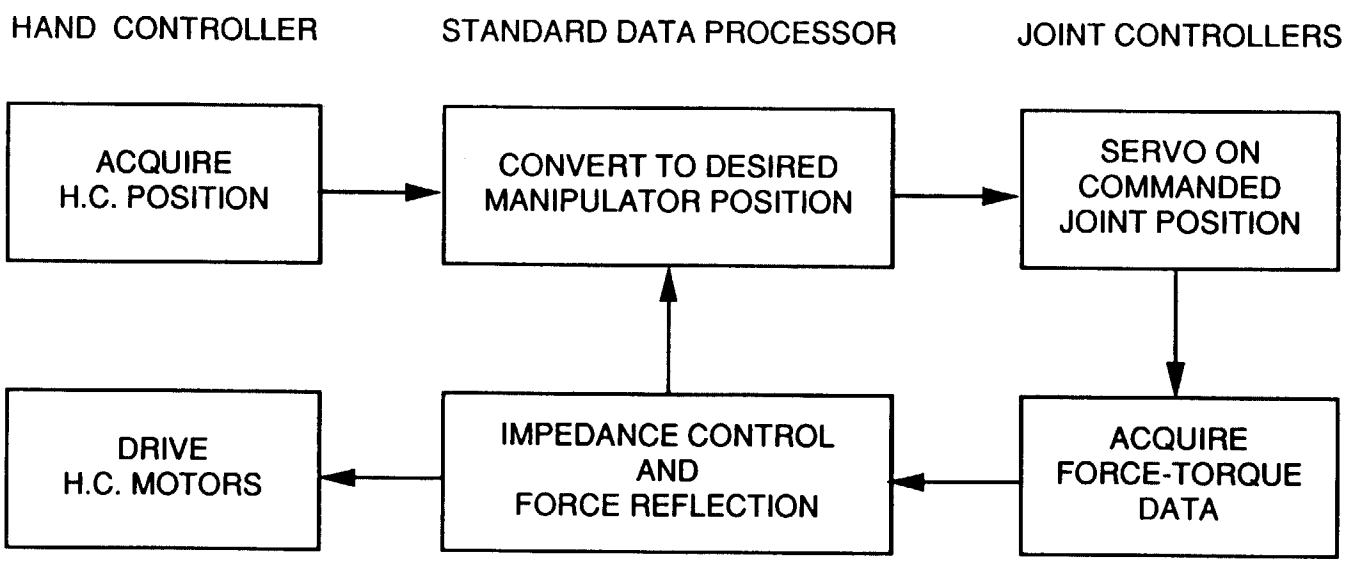
Power Module Controller
- Part of Telerobot Power Subsystem

Regulator/Charger Module Controller
- Part of Telerobot Power Subsystem

Payload Bay Controller
- Video Switch Control
- Global Camera Control
- Part of Workstation Thermal
- Part of Workstation Communications
- Video Control
- Caging Mechanism Control
- Part of Telerobot Power Subsystem

Redundant Payload Bay Controller
- Video Switch Control
- Caging Mechanism Control
- Part of Telerobot Power Subsystem

Control Algorithms Processing/Data Flow



- SDP Cartesian Control Loop Frequency of 50 - 100 Hz
- Joint Controller Servo Loop Frequency of 150 - 200 Hz

Development Environment

- VAX 6000 Model 420
  - VMS Operating System
  - DEC Ada Compiler for Initial Checkout
- Macintosh II Workstations
- DDC-I Ada Compiler for 80386 Protected Mode
  - 32 Bit Architecture
  - Strict Priority Based Task Scheduling
  - Package to Interface to 80387 Math Functions
  - Package to Interface to Multibus II Message Passing Protocol
  - Symbolic Debugger for Breadboard Flight Computers
- Intel 80386 In-Circuit Emulators
  - Initial Hardware Checkout
  - Low Level Software/Hardware Checkout
- PC Based 1553 Test System
  - Send, Receive, or Monitor 1553 Messages

Bare Machine Ada Run-Time System

- Not Using An Operating System
  - SDP Baselined To Run AIX (Unix Based)
  - Unix Not Designed For Real Time Applications
    - Complexity and Overhead Associated With Interrupt Handling
    - Timing Is Nondeterministic
  - Same Compiler And Run-Time System Used For All Processors

- Ada Run_Time System Capabilities
  - Initialization
  - Storage (Memory) Management
  - Input And Output (Low Level IO)
  - Interrupt Handling
  - Tasking (Synchronization And Communication)
  - Time Management

- Implications
  - Application Program Must Implement Some "Operating System" Functions
    - Interface To Peripherals
    - Desirable For Embedded Systems

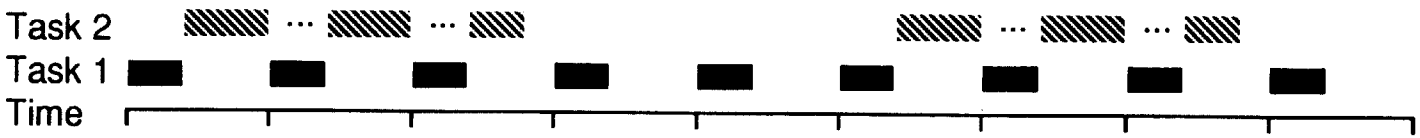
Prototyping Efforts

- Hardware
  - Wirewrap Prototypes of Controller Design
    - CPU Memory Board
    - MIL-STD-1553B Interface
    - RS-232 Port Added for CRT/Debugger
  - Intel Multibus II 80386 Single Board Computers
  - Space Station Standard Data Processor Due December

- Software
  - NASREM Interfaces and Levels of Control
  - Low Level Hardware Interface Checkout
  - MIL-STD-1553B Interface Control
  - Operator Interface
  - Control Algorithms (Time Critical)
  - Cyclic Execution Control
  - Data Management Scheme

Cyclic Operations

- Interrupt Handler
  - Timer or 1553 Message
  - Ada Task Entry
  - Counts Interrupts To Create Different Cycle Times (E.G. 200 Hz, 50 Hz, ...)
  - Synchronizes Communications And Access To "Global" Parameters
- Ada Task For Each Frequency
  - Highest Priority Assigned To Highest Frequency
  - Infinite Loop With "Start" Entry Called By Timer Interrupt Handler
  - Must Execute Within Cycle



Preliminary Timing Data

Function	Breadboard CPU Time (8 MHz)	Scaled CPU Time (16 MHz)	Intel SBC CPU Time (16 MHz)
Digital Filters: (6 element vectors)			
4th order	1.196 msec	.598 msec	.650 msec
3rd order	.892	.446	.500
1st order	.260	.130	.125
Inertia Matrix Calculation	38.763	19.382	17.640
Forward Kinematics	2.604	1.302	1.370
Inverse Kinematics	2.713	1.357	1.140
Matrix-Vector:			
Quaternion to A-matrix	.379	.190	.197
A-matrix to Quaternion	.233	.117	.117
Quaternion Multiply	.298	.149	.197
Vector Add	.106	.053	.047
Matrix Vector Multiply	.231	.116	.101
Matrix-transpose Vector Multiply	.355	.178	.168
Matrix Matrix Multiply	.578	.289	.261
Euler Angle Transformations:			
Euler Angle to A-matrix	.489	.245	
Euler Angle to Quaternion	.275	.138	

"LESSONS LEARNED IN PROTOTYPING THE SPACE STATION REMOTE MANIPULATOR SYSTEM CONTROL ALGORITHMS IN ADA"

P. Gacuk, SPAR Aerospace

SPAR

Lessons Learned In Ada Prototyping

Canadian Space Applications

Presenter: P. Gacuk

Prototype Team:

P. Gacuk P. Harding P. Lombardo S. Prasad C. Trainor S. Vetter

Spar Aerospace Limited

Remote Manipulator Systems Division

1700 Ormont Drive, Weston, Ontario, Canada

(416) 745-9680

ACM SIG Ada Conference Ottawa Meeting

August 11th, 1989

REPRESENTED

2nd NASA Ada Users Symposium

Goddard Space Flight Center

Greenbelt, Maryland

November 30th, 1989

Presentation Structure – Part I:

Prototyping In General:

- Why Prototype?
- Definitions
- Declarative Aspect
- Techniques
- Life Cycle Issues

P. Gacuk
SPAR Aerospace

Presentation Structure – Part II:

SPAR

Prototyping The Space Station Remote Manipulator System (SSRMS) Control Algorithms In Ada:

- Simple Description Of SSRMS Control Algorithms
- Software Risks Addressed
- Prototype Metrics
- Lessons Learned (Organized According To Life Cycle)

P. Gacuk
SPAR Aerospace

Part I

Prototyping In General

P. Gacuk
SPAR Aerospace
4 of 62

Why Prototype?

- To Reduce Risk
- To Learn By Doing
- To Disseminate Lessons Learned Which May Be Useful For The Successful Completion Of Future Projects

P. Gacuk
SPAR Aerospace

Definitions:

SPAR

Webster's New World Dictionary

- **proto**

  "1. first in time, original, primitive"

- **prototype**

  "1. the first thing or being of its kind; original; model; pattern; archetype

  2. a person or thing that serves as a model for one of a later period"

- **model**

  "1.b) a preliminary representation of something, serving as the plan from which the final, usually larger, object is to be constructed...

  d) a hypothetical or stylized representation

  e) a generalized, hypothetical description... used in analyzing or explaining something"

P. Gacuk
SPAR Aerospace
6 of 62

Definitions Of Prototyping:

SPAR

Brooks – 1987

- **Prototyping**

"part of the iterative specification of requirements"

- **Prototyping Software System**

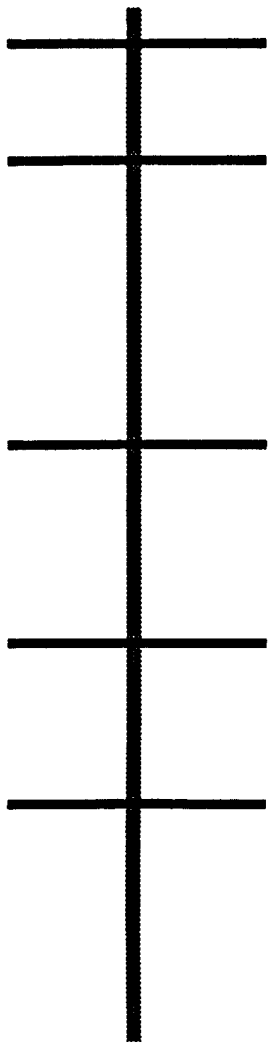
"one that simulates the important interfaces and performs the main functions of the intended system, while not necessarily being bound by the same hardware, speed, size, or cost constraints"

Gacuk – 1989

- **Prototype Declaration**

"prior to embarking on the prototype, declare what the prototype will be used for after the prototyping activity is complete"

Prototyping Techniques:



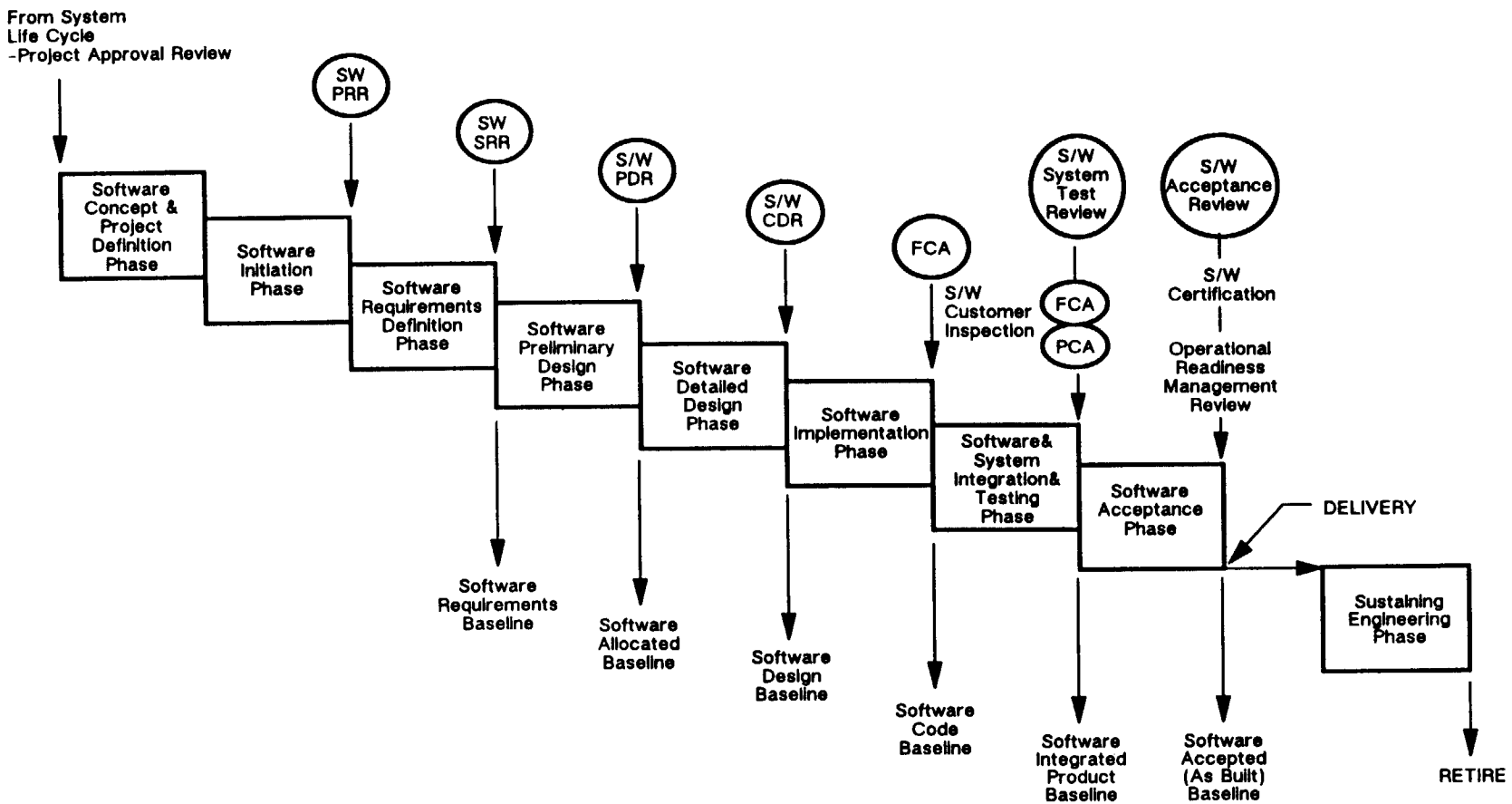
LOW LEVEL OF DECLARATION

- **Paper Prototype**
  - ☐ Visualize Form & Fit
- **Simulation**
  - ☐ Specialized Prototyping Tools
  - ☐ More Application Specific
  - ☐ Simulation Languages
  - ☐ Determine Performance Effects Of Design Constraints & Decisions
- **Proof Of Concept**
  - ☐ Build With The Intention To Throw It Away
  - ☐ Not Production Quality
  - ☐ Evaluate Hardware & Software Design Or New Technologies
- **Existing Building Blocks**
  - ☐ Assemble From Existing Components
  - ☐ Postpone Speed & Memory Efficiency Issues
- **Incremental Build**
  - ☐ Fully Develop A Portion
  - ☐ Partially Develop System Level Functionality
  - ☐ Incrementally Replace Scaffold Hardware / Software With Production Quality

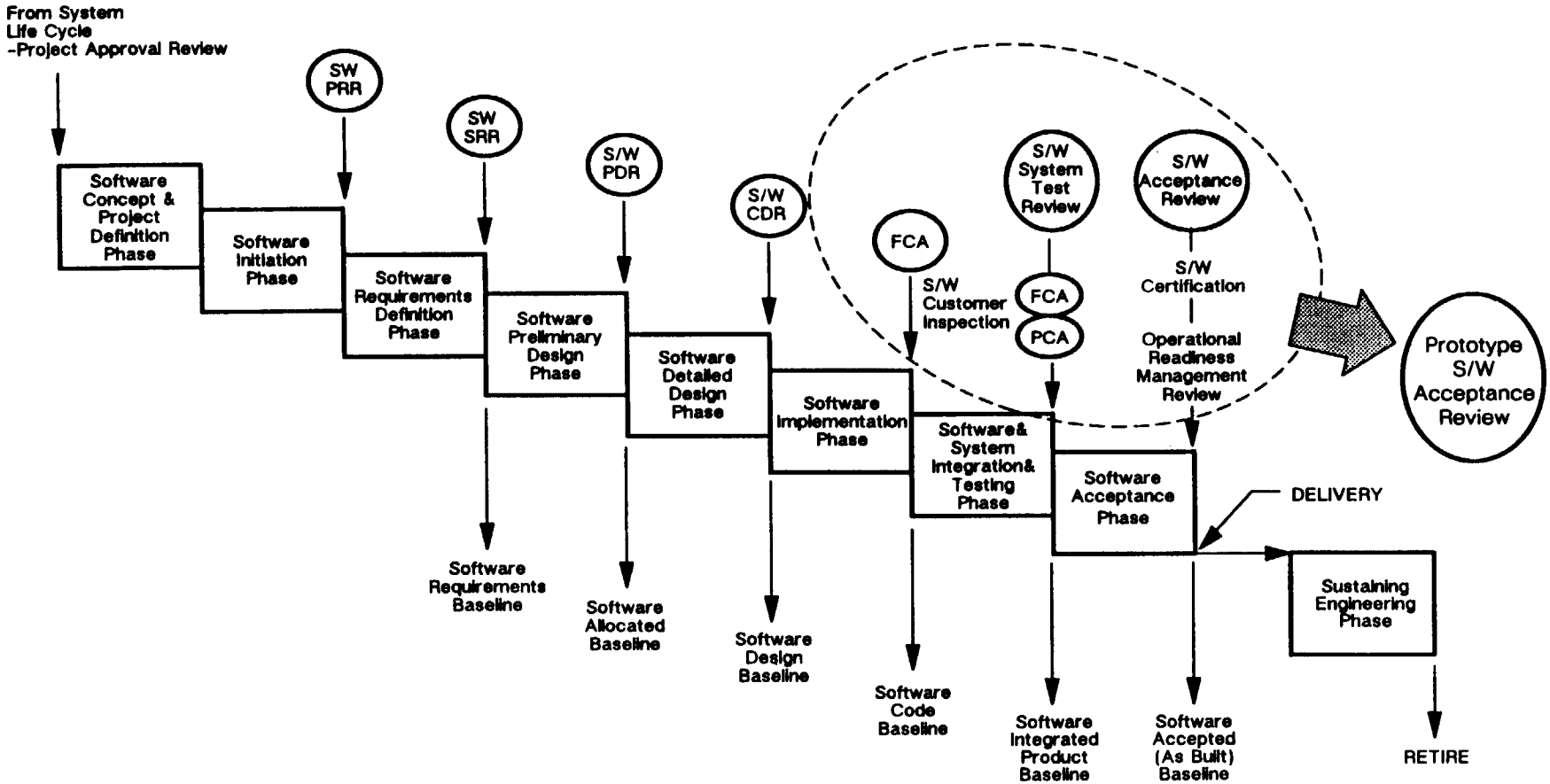
HIGH DEGREE OF DECLARATION

P. Gacuk
SPAR Aerospace

NASA Space Station Freedom Program Software Life-Cycle

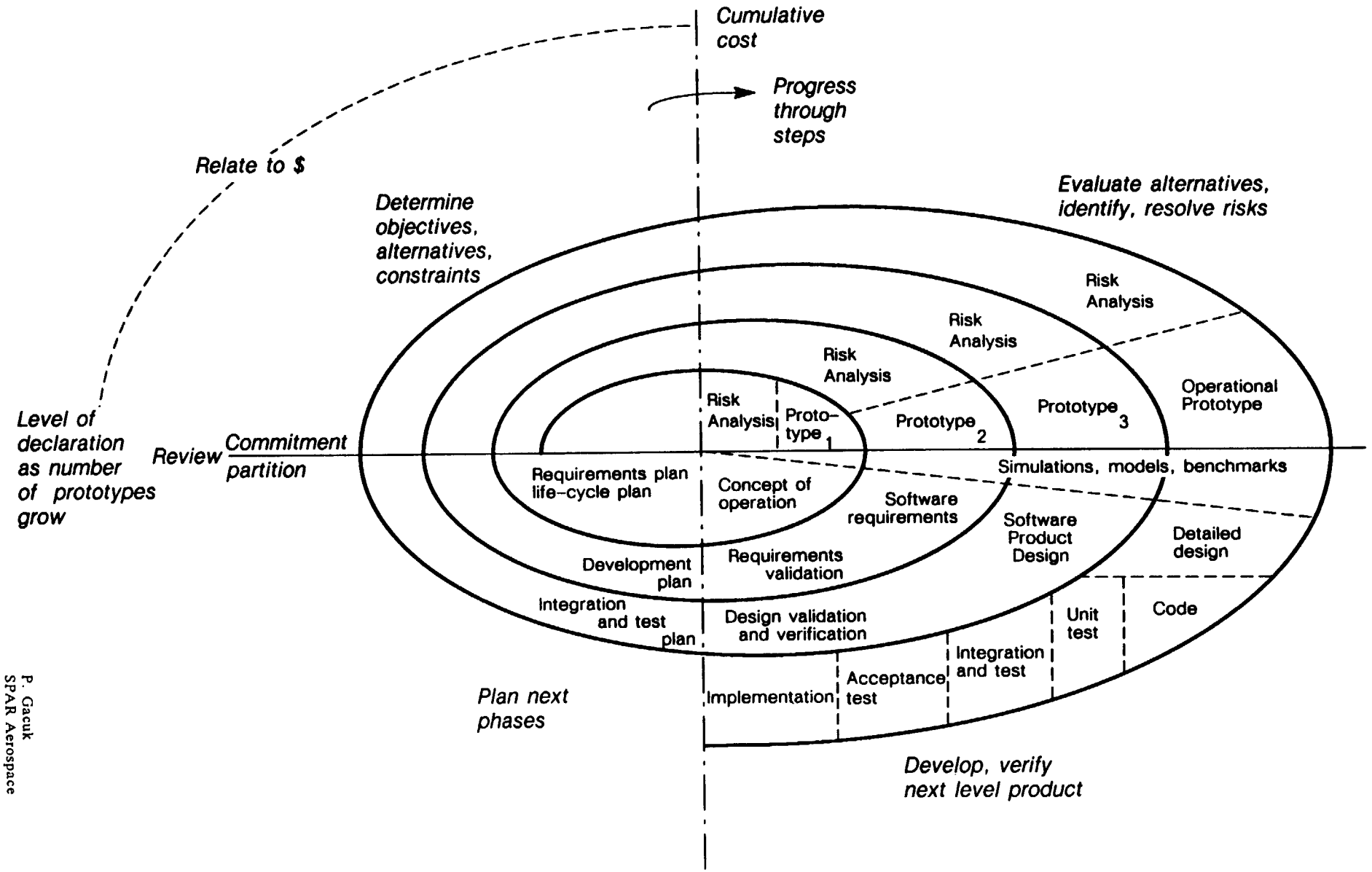


Modified NASA Space Station Freedom Program Software Life-Cycle for Canadian Space Station Software Prototyping



Preferred Software Prototype Life-Cycle

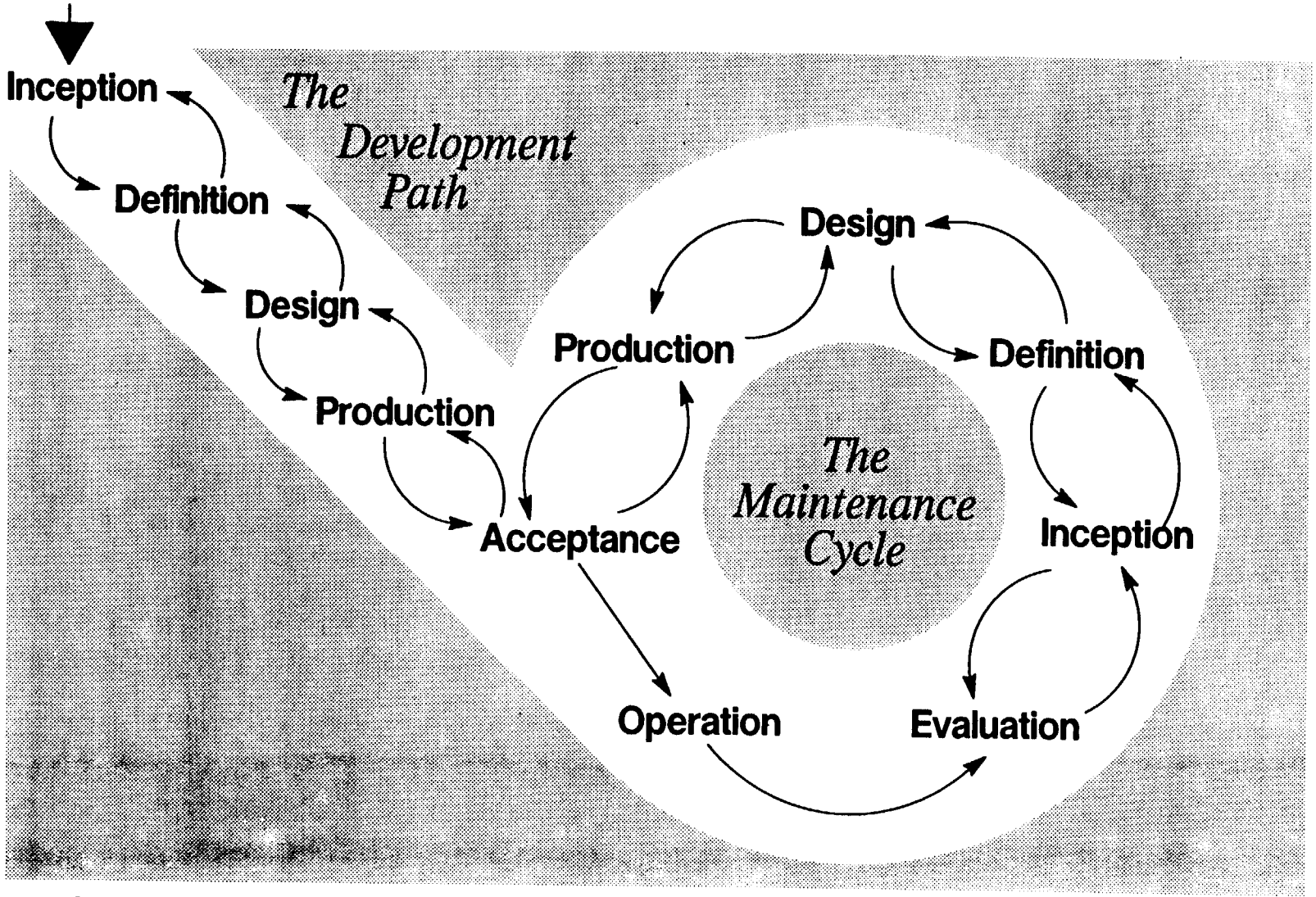
SPIRAL MODEL (Boehm, 1986)



Preferred Software Production Life-Cycle

SPAR

b-Model (Birrell and Ould, 1985)



Emphasis on cost and length of maintenance

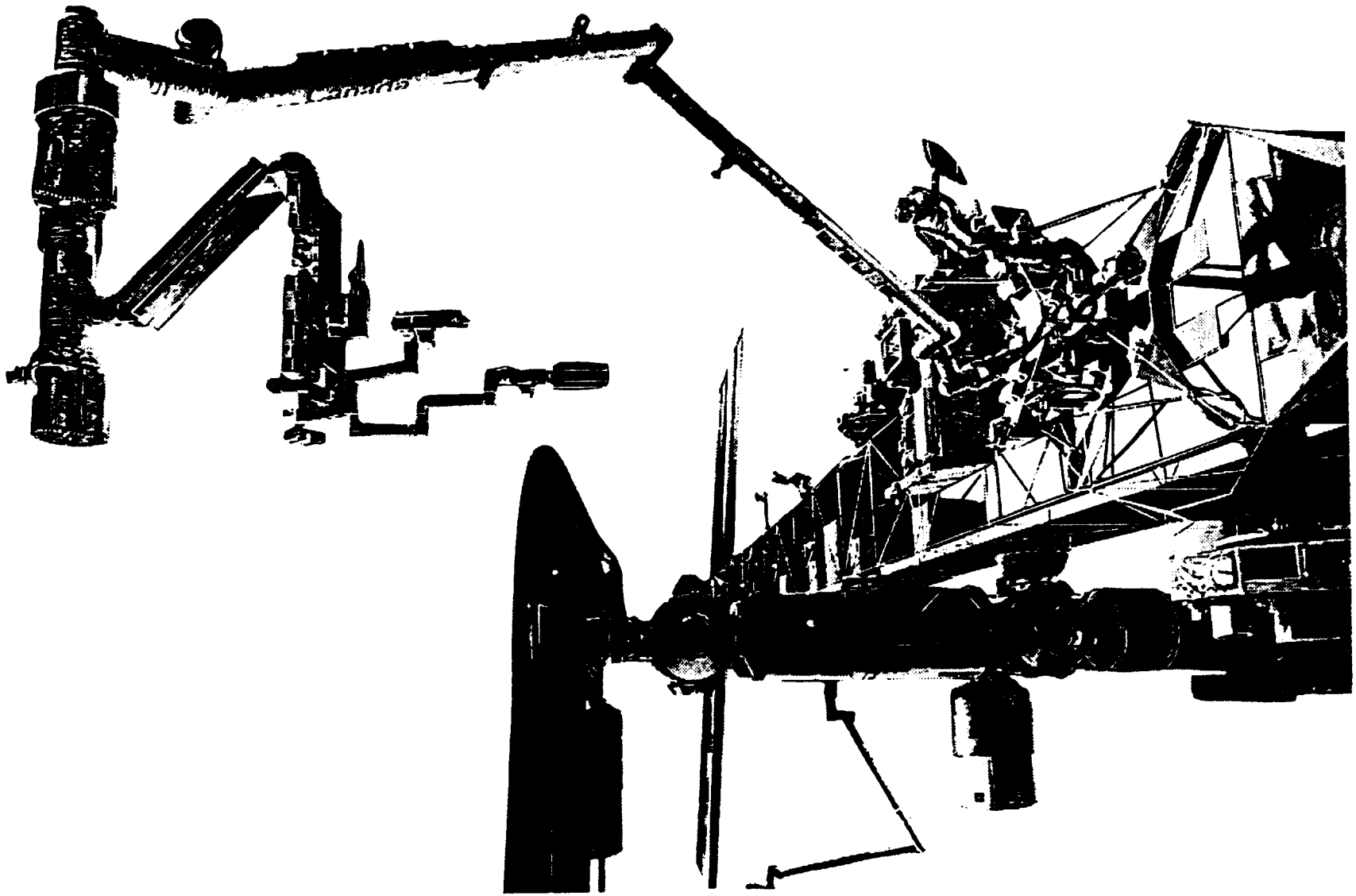
P. Gacuk
SPAR Aerospace

Part II

Prototyping
The Space Station Remote Manipulator System (SSRMS)
Control Algorithms In Ada

P. Gacuk
SPAR Aerospace

Mobile Servicing Centre (MSC)



V-SS-078

Software Goals Or Risks Addressed By The Prototype: SPAR *

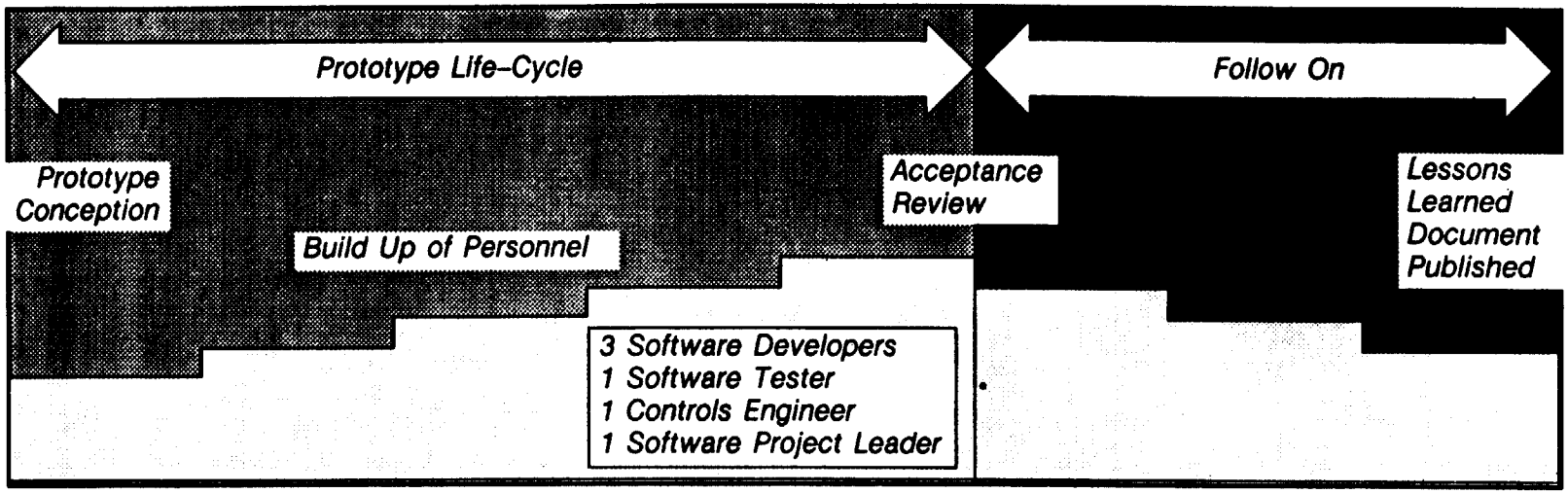
- Software Life Cycle Issues
- Ada Development Issues
- Methodology Issues
- Engineering Interdisciplinary Communication Issues
- Ada Execution Issues

Plus

- Provide Ada Code For A Configuration Management Prototyping Exercise

P. Gacuk
SPAR Aerospace

Prototype Metrics



Effort		Activities
Software Engineering:	16 Person Months	Lessons Learned Meetings Methodology Revist
Technical Management:	3 Person Months	Run Time Performance Improvement
Total:	19 Person Months	Compile Notes and Meeting Discussions into Lessons Learned Document

Lines of Code

	Semicolon Count	Carriage Return Count
Application (delivered) Code	3693	13461
Test Code	2663	6872
Total	6356	20333

Record notes and lesons learned during entire life-cycle

Hybrid Mix Of Methodologies & Methods Used During Different Phases:

SPAR

- Tailor Methods Associated With Methodologies Where Necessary

  If a methodology does not provide a totally adequate means of representing required information, then the methods should bend, through additional notation for example, rather than bending the problem or design to fit the methods

P. Gacuk
SPAR Aerospace

Requirements Analysis & Definition Phase:

Ward & Mellor's Structured Analysis For Real–Time Systems – Evaluation:

SPAR

- Methodology's Artifacts (Transformation Schemas, State Transition Diagrams, Transformation Specifications, & Data Dictionary) Adequately Describe The Logical Model
- Methods Are Well Documented
- Supported By Automated Tools
- Difficult To Generate Transformation Schemas Which Were "Executable" (Volume 1 Chapter 9 of Ward & Mellor, Structured Development For Real–Time Systems)

P. Gacuk
SPAR Aerospace

SPAR

Difficulty In Generating "Executable" Ward & Mellor, Transformation Schemas

- Ensuring Data Flow Triggers Make Sense
- Deciding Between Time–Continuous & Time–Discrete Flows
- Deciding Whether To Include Hardware
- Deciding When To Use Data Stores Rather Than Simple Data Flows, & How To Use Them To Represent Saving Of Values From Previous Cycles
- Indicating Actions Taken Each Time A Bubble Is Enabled
- Indicating Sequencing Of Instantaneous Events
- An Automated Tool To Animate The Executable Behavior Is Needed

Requirements Analysis & Definition Phase:

Engineering Interdisciplinary Communication:

- Evaluate The Appropriateness Of A Methodology's Artifacts As A Means Of Communicating Technical Information

  Can Systems & Controls Groups Understand Software Artifacts? (Transformation Schemas, Object Diagrams, Buhr Diagrams, Etc.)

Conversely

Can The Software Group Understand Systems & Controls Artifacts? (Technical Notes, Control Loop Diagrams, Etc.)

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

OOD = Oh Oh Dilemma! (To Some)

= Object Oriented Development (To Others)

Preliminary / Detailed Design Phase:

SPAR

POD = Package Oriented Design

More Germane When The Expected Implementation Language Is Ada

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

Hybrid Design Methodology:

- Started With Seidewitz & Stark, Goddard, (General Object Oriented Development (GOOD 1986))
- Investigated Buhr (System Design With Ada 1984)
- Investigated Booch (Software Components With Ada 1987)
- Settled On A Hybrid Mix Of Considerations From GOOD & Booch
- Buhr's Concepts Are Being Incorporated Into Our Adopted Methodology During Our Current Multi-Tasking Prototype

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

General Object Oriented Development (GOOD)–Paraphrased:

- Methodology For Transitioning From Requirements Analysis (Data Flow Diagrams) To An Object Oriented Design
- Simple Definition Of An Object (Has State & Operations)
- Objects Are "Carved Out" Of Data Flow Diagrams ("Abstraction Analysis")
- Starts With Centre Of Data Flow Diagrams And Works Outward
- Identification Of Objects Is Done By Evaluation Of Abstraction Levels, With Compromises Allowed For Simplicity Of Design.

  Abstraction Levels, In Order Of Importance, Are:

  1) Entity Abstraction (Intuitive Concept Of An Object In Problem Domain)
  2) Action Abstraction (Set Of Operations Which Perform Similar Actions)
  3) Virtual Machine Abstraction (Seniority Levels)
  4) Coincidental Abstraction (None)

- Centralized Vs Decentralized Control Decision Left To The Designer

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

General Object Oriented Development (GOOD) – Evaluation:

- Abstraction Analysis Of GOOD Was Not Helpful
- Discussions In GOOD Manual Are Good

Problems Encountered With Abstraction Analysis

- Confusion Over What An "Entity" Is
- Difficulty Identifying The Central Entity
- Objects Could Be Chosen Better From Intuition Or Relation To The Real World Than By Going Through The Abstraction Analysis

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

General Object Oriented Development (GOOD) – Evaluation:

SPAR

Discussions In GOOD Manual Are Good

Particularly...

- Properties Of Good Objects
- Use Of Virtual Machine Hierarchies
- Pros & Cons Of Centralized Vs Distributed Control

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

Booch – Paraphrased:

- Identify Objects & Their Externally Visible Attributes
- Identify Operations Suffered By & Required By Each Object
- Establish Visibility Of Each Object
- Establish Interfaces Between Objects

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

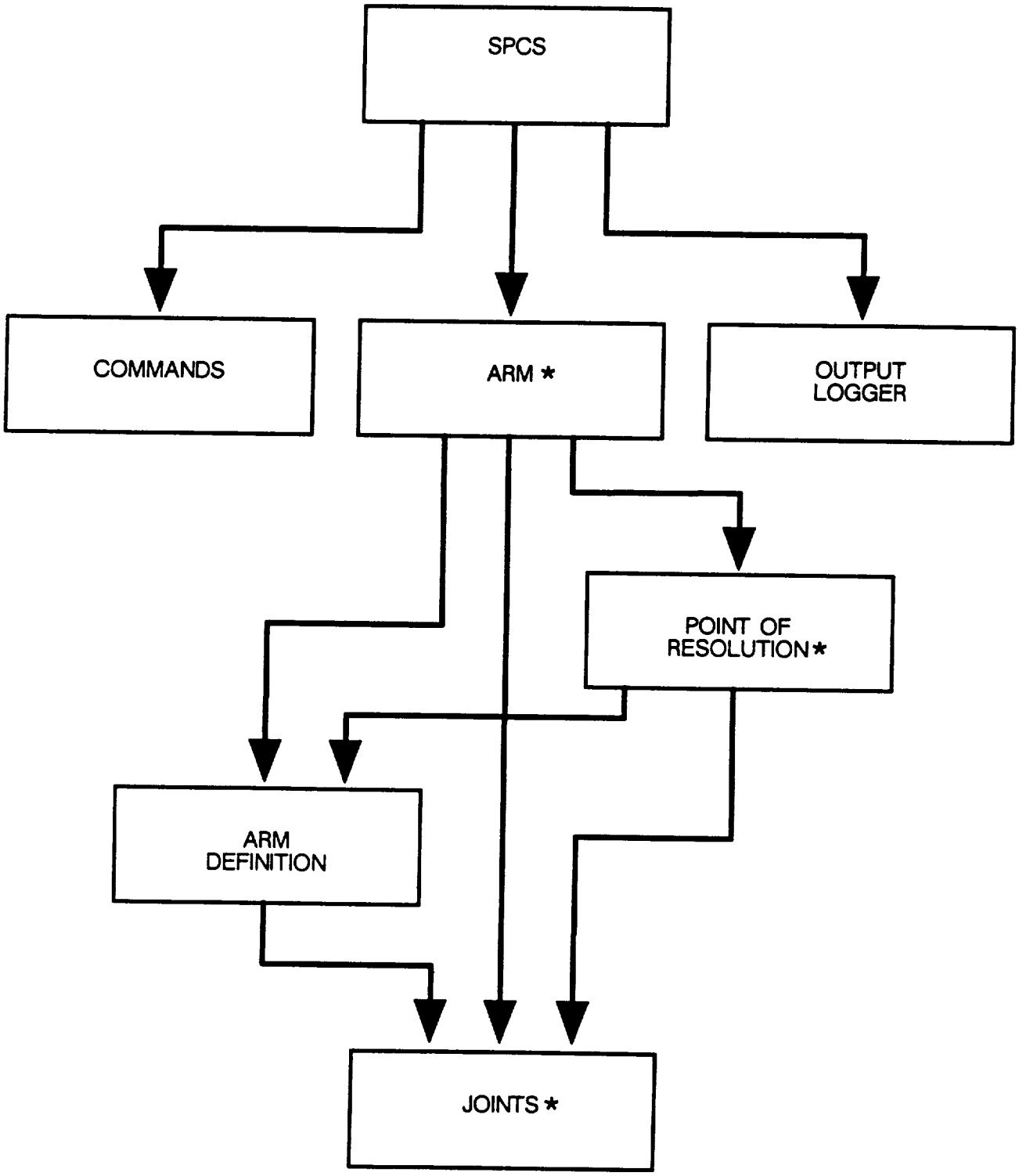
Hybrid Methodology Adopted:

SPAR

- Hierarchical Layered Object Diagrams
- Seniority Hierarchy Of Objects Upon A Single Diagram
- Composition Hierarchy Of Objects On Different Layers Of Diagrams (Each Higher Level Object May Be Composed Of Lower Level Objects)

Object Diagram

SPAR



*Composite Objects
(Note: The arrows indicate direction of control.)

Preliminary Design Phase:

Hybrid Methodology Object Diagrams:

SPAR

- Arrows On Object Diagrams Show Control
- Point From The Object Requiring The Operations To The Object Suffering The Operations
- Arrow Does Not Represent "View Of The World" Or Ada "Withing"

P. Gacuk
SPAR Aerospace

Preliminary Design Phase:

Hybrid Methodology Object Diagrams:

SPAR

- Finding Good Sub–Objects is Difficult
- Top–Down & Bottom–Up Composition Should Be Employed
- Use a Waiver List To Simplify Object & Withing Diagrams
- Low Level Objects Map Strongly To Ada Packages

Preliminary / Detailed Design Phase:

OOD – Evaluation:

SPAR

- Object Oriented Design Has Clear Advantages Over Other Techniques, Especially For Software To Be Coded In Ada

However...

- Methods Need To Be Distilled From The Methodology

P. Gacuk
SPAR Aerospace

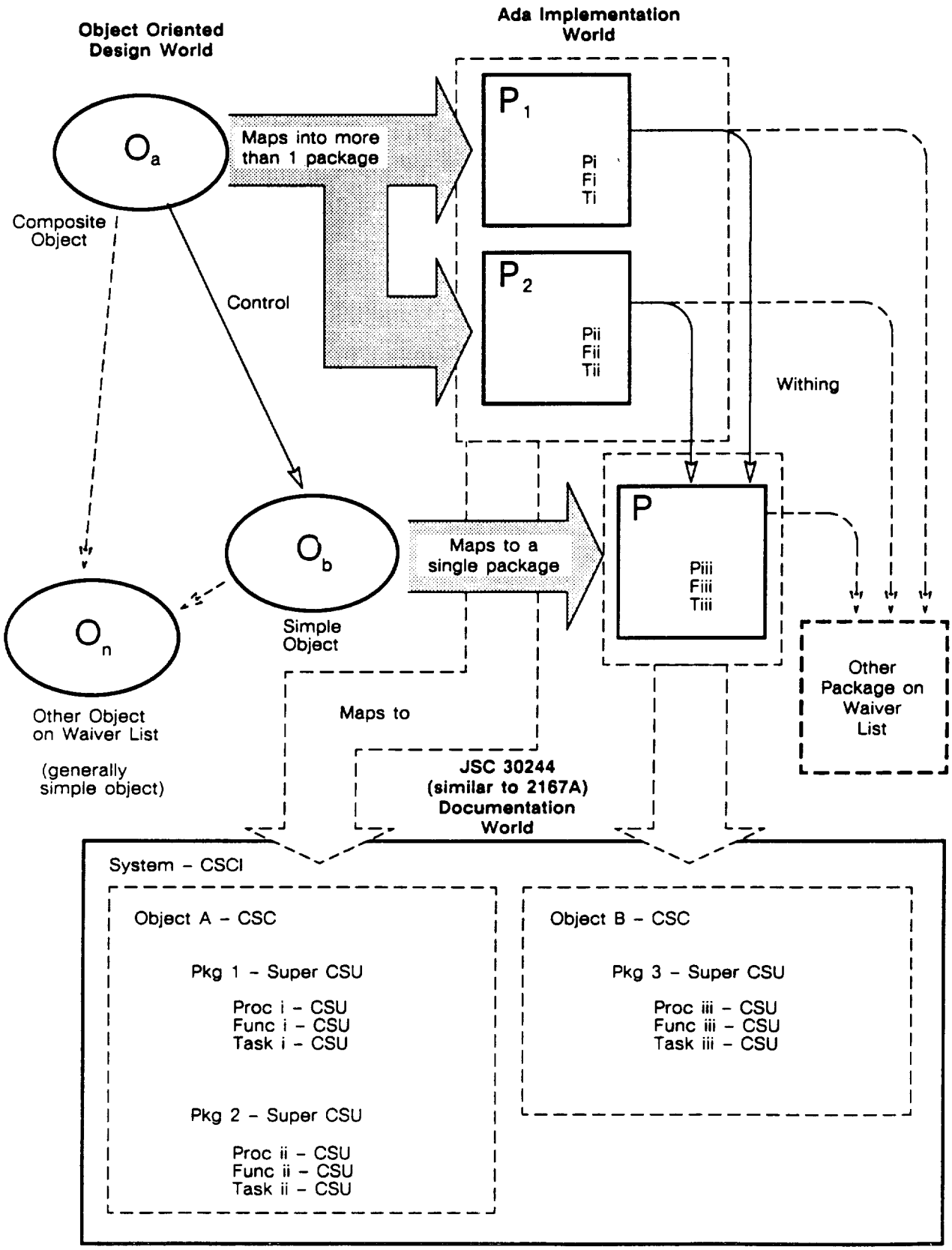
Transition From Structured Analysis / Structured Design To Object Oriented Design:

SPAR

- Can Certainly Be Made
- Reoccurring Concept Of Carving Objects From Data Flow Diagrams In Many Methodologies
- Match Is Less Than Ideal
- High Level Of Logistics Required To Maintain Traceability
- Presently Investigating Object Oriented Requirements Analysis

P. Gacuk
SPAR Aerospace

Old Approach of Mapping Ada Units to CSCs & CSUs



Detailed Design Phase:

Computer Software Organization:

SPAR

"Computer software organization. The contractor shall decompose and partition each CSCI into Computer Software Components (CSCs) and Computer Software Units (CSUs) in accordance with the development method(s) documented in the Software Development Plan (SDP). The contractor shall ensure that the requirements for the CSCI are completely allocated and further refined to facilitate the design and test of each CSC and CSU..." [DOD–STD–2167A: 4.2.5]

Computer Software Component (CSC)

"A functional or logically distinct part of a Computer Software Configuration Item (CSCI)." [DOD–STD–2167A: 3.7]

Computer Software Unit (CSU)

"The smallest logical entity specified in the detailed design which completely describes a single function in sufficient detail to allow implementing code to be produced and tested independently of other units. Units are the actual physical entities implemented in code." [DOD–STD–2167A: 3.23]

"An element specified in the design of a Computer Software Component (CSC) that is separately testable." [DOD–STD–2167A]

Detailed Design Phase:

Mapping Ada Units To CSCs & CSUs Is Difficult:

- Old Approach Introduced The Concept Of Super CSUs

CSCI <---> Entire Computer Program

CSC <---> Collection Of Packages (Forming A Composite Object)

Super CSU <---> Ada Package

CSU <---> Ada Procedure, Function, Or Task Within A Package

Problem With Super CSUs Concept

- CSU Is Now Not Necessarily "Separately Testable" As Required By 2167A (Eg. "Pop" Function In A Stack Package Is Not Testable Without The "Push" Procedure)

Detailed Design Phase:

Current Thinking On Mapping Ada Units To CSCs & CSUs:

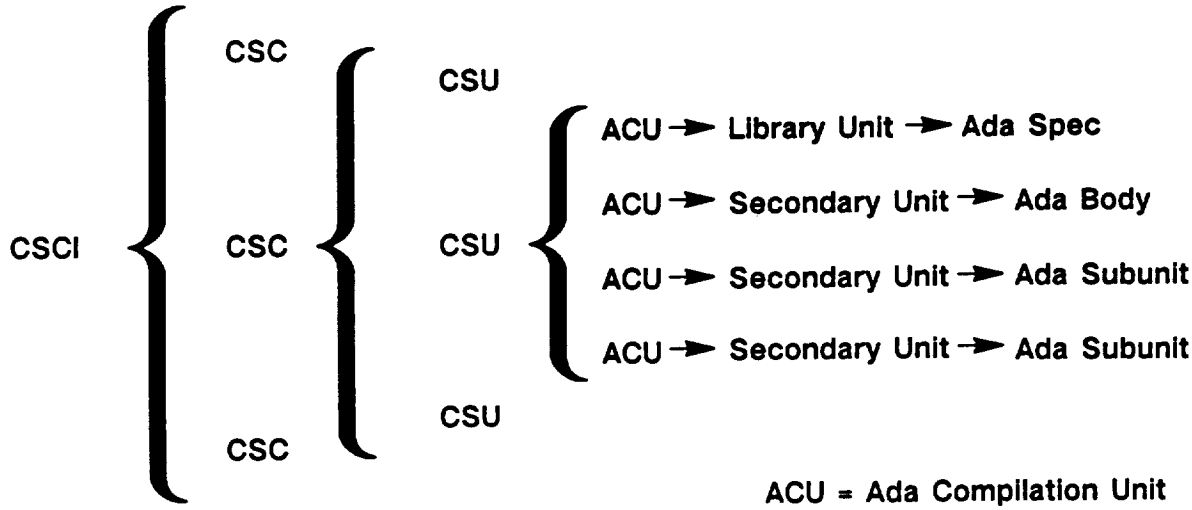
- An Ada Unit Which Composes A Collection Of Packages Is Needed
- Rational's Concept Of A Subsystem Seems To Satisfy This Need

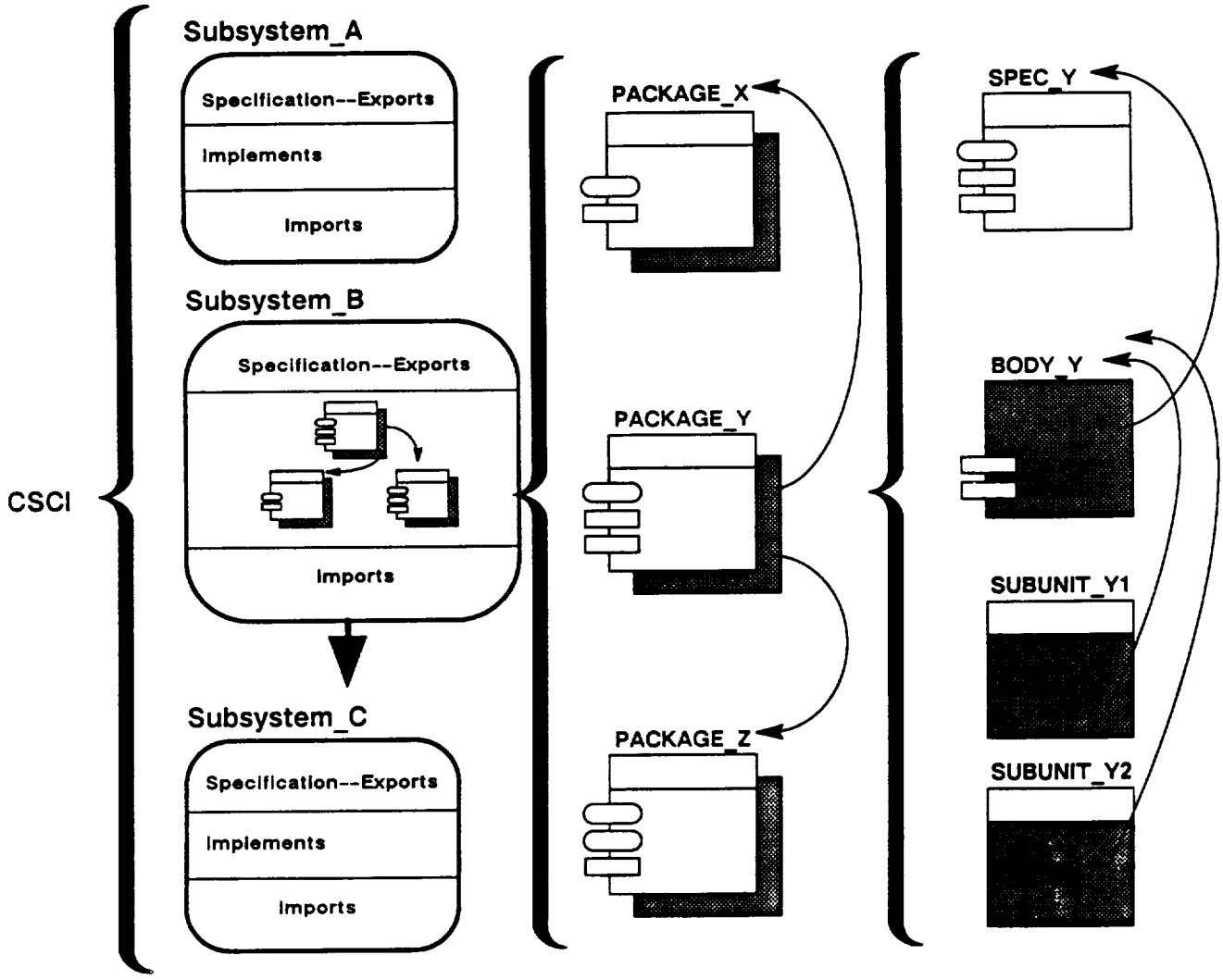
Plus...

- The Export Aspect Of A Subsystem Controls Package Dependencies

P. Gacuk
SPAR Aerospace
38 of 62

Proposed Approach of Mapping Ada Units to CSCs & CSUs





ORIGINAL PAGE IS
OF POOR QUALITY

Detailed Design Phase:

Multiple Views Of The Design Are Required:

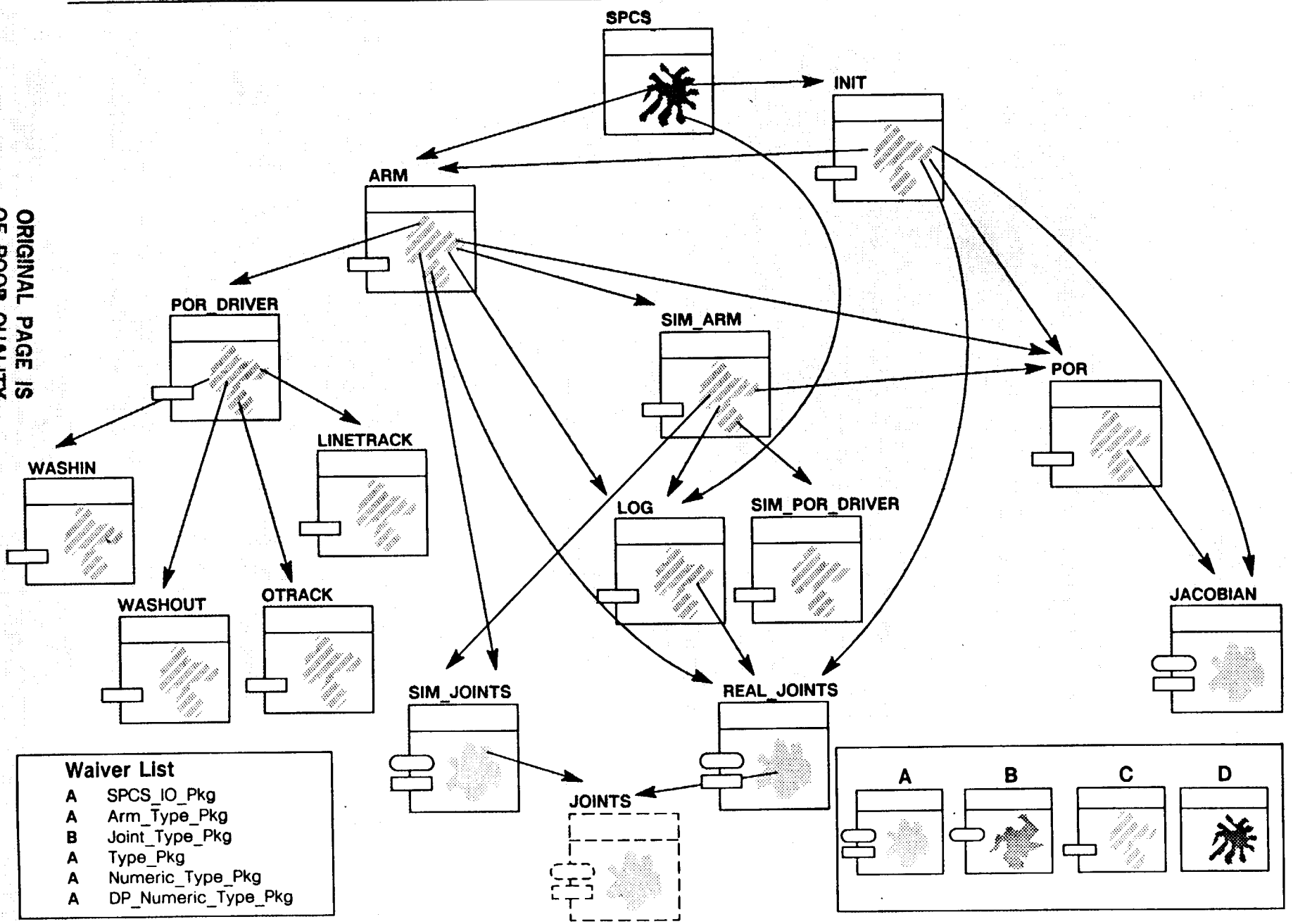
- Object Diagrams
- Data Flow Diagram Fragments
- Control Flow Diagram Fragments
- State Transition Diagrams
- Withing Diagrams
- Exception Diagrams
- Tasking Diagrams
-
-
-

Etc.

P. Gacuk
SPAR Aerospace

Withing Diagram

SPAR *

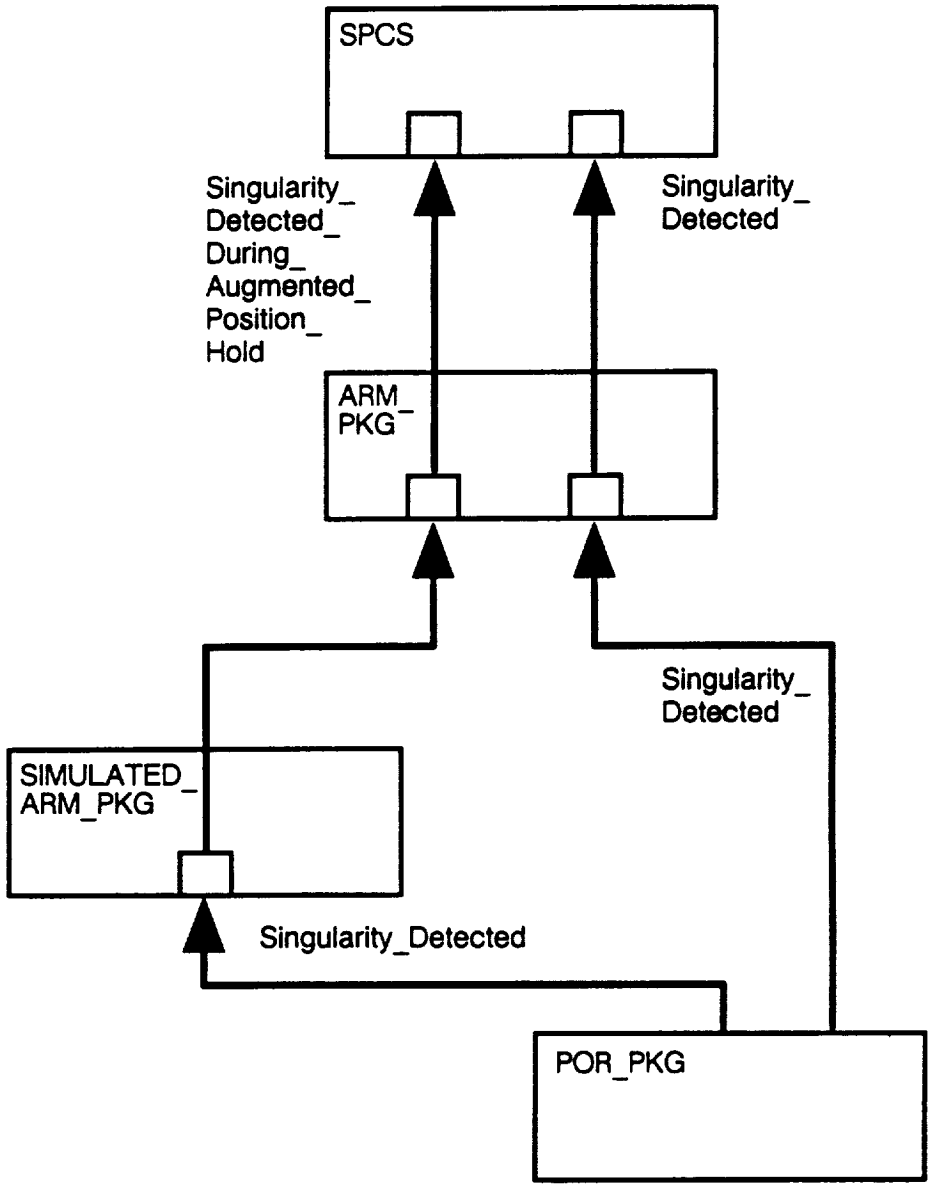


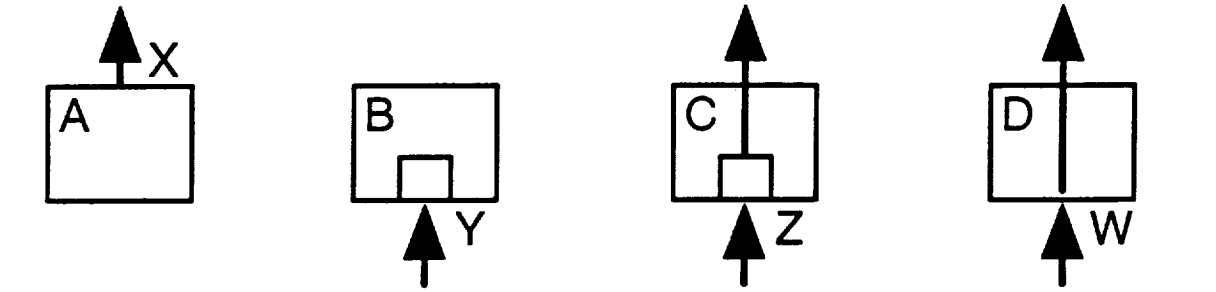
Waiver List

A	SPCS_IO_Pkg
A	Arm_Type_Pkg
B	Joint_Type_Pkg
A	Type_Pkg
A	Numeric_Type_Pkg
A	DP_Numeric_Type_Pkg

ORIGINAL PAGE IS
OF POOR QUALITY

Exception Diagram





Package A generates exception X

Package B handles exception Y

Package C handles and re-raises exception Z

Package D does not handle and therefore propogates exception W

Detailed Design Phase

Documentation:

SPAR

Several Of The Traditional Document Templates (JSC 30244 or 2167A) Require Extensive Rework

- Impact Of Ada
- Addition Of Useful Guidance For The Writers & Readers
- Newer Standards Currently Being Proposed By NASA (Software Support Environment (SSE), Common US, Canada, Japan, European Standards)
- Design Document Templates Require Considerable Modifications For OOD & Ada

Detailed Design Phase:

Design Document Modifications For OOD & Ada:

SPAR

Section Required To Describe The Object Oriented Design Via

- Object Diagrams
- Object Descriptions
- Object Attribute Tables
- Object Operation Tables

P. Gacuk
SPAR Aerospace

Detailed Design Phase:

Design Document Modifications For OOD & Ada:

SPAR

The Indented List Of CSCs & CSUs

- Should Indicate Which Items Are Generic Instantiations
- Should Indicate Which Items Will Be Obtained From A Reuse Library

Detailed Design Phase:

Design Document Modifications For OOD & Ada:

SPAR

"Textual" Detailed Design Portion (Section 4) Problems

- Very Time Consuming To Complete
- Inappropriate For OOD & Ada
- Requires Documenting Aspects Of The Design Which Could Be Better Described Using A Combination Of
  - ☐ Object Diagrams
  - ☐ Withing Diagrams
  - ☐ Exception Diagrams
  - ☐ Tasking Diagrams
  - ☐ Ada Specifications With Embedded Comments
  - ☐ Program Design Language (PDL)
- Problem Identifying Overloaded Subprograms

P. Gacuk
SPAR Aerospace

Detailed Design Phase:

Program Design Language (PDL):

- English Comments Using Ada Structures (Loops, Blocks, Etc.)
- Includes Compilable Ada Specifications Defining Types And Interfaces
- High Degree Of Correlation Between PDL & Code

P. Gacuk
SPAR Aerospace
47 of 62

Implementation Phase:

Impact Of Ada:

SPAR

- Use Of Ada Did Not Increase Development Time

However...

- Allowances Must Be Made For A Learning Curve
- Phased Training In Software Engineering Techniques And Their Application To Ada Is Very Important

P. Gecuk
SPAR Aerospace

Implementation Phase:

Ada Style Guidance:

SPAR

- The NASA Ada Style Guide Provides Very Good Guidance

P. Gacuk
SPAR Aerospace

Ada Style Guidance:

SPAR

- Limit Use of "Use" Clause
  Use Only For Packages Which Define The Infix Operators (*, +, Etc.).

- "Renames" Clauses Can Be Avoided By Intelligent Choice Of Package, Type & Subprogram Names

- Procedure & Function Argument Names Should Be Chosen So That Calls To Them Read In An English–Like Manner

P. Gacuk
SPAR Aerospace

Implementation Phase:

Ada Style Guidance:

SPAR

Subprogram Naming Convention

- Functions Should Be Named Using Nouns
- Functions Returning True Or False Should Be Named With Names Which Contain Forms Of The Verb "To Be", Such As Cursor_Is_Inside_Window
- Procedures Should Be Named Using Verbs

P. Gacuk
SPAR Aerospace

Ada Feature Usage Guidance:

SPAR

Strong Typing Advantages / Disadvantages

- An Important Decision Is How Many Basic Types To Use
- It Is Easy To Create Too Many Types, Which Results In Having To Define Specific Operations For All The Legal Combinations Of These Types
- Reliability Due To Strong Typing Must Be Analyzed Vs Package Complexity Due To The Number Of Operations Required To Be Defined

P. Gacuk
SPAR Aerospace

Implementation Phase:

Ada Feature Usage Guidance:

SPAR

Problems With Derived Types

- Derived Types In Ada Are Not Very Flexible
- There Is No Good Way Of Selecting Which Operations One Wants To Derive From The Parent Type, Since They Are All Derived Automatically
- Nor Is There A Way Of Deriving Operations Involving Two Derived Types

Implementation Phase:

Ada Feature Usage Guidance:

SPAR

- Get The Basic Data Types Right

  Changes To The Low Level Types Packages Are Costly As They May Cause Many Recompilations And Retesting

- Avoid Nesting

  Nested Units Make Independent & Non-intrusive Testing More Difficult

  Nested Units Cannot Be Easily Reused

P. Gacuk
SPAR Aerospace

Implementation Phase:

Avoiding Recompilations Guidance:

SPAR

- Be Aware Of Forced Recompilations
- Maintain A Withing Diagram Of Dependencies Between Compilation Units And Access Impact Of Unit Change
- Develop Generics First As Non–Generics
- Use Separate Compilation Of Subunits Freely

P. Gacuk
SPAR Aerospace

Implementation Phase:

Configuration Management:

- The Ada Language Itself Provides Important Safeguards On The Compilation Dependencies Between Various Units

However...

- Setting Up Configuration Management, Tools & Procedures Which Are Integrated With Ada Compilers Has Proven To Be A Very Difficult Issue

P. Gacuk
SPAR Aerosp

Implementation Phase:

Configuration Management Problems:

"Make" Tools Are Required

- Some Compilers Provide Automatic Recompilation Of Units Which Have Been Entered Into An Ada Library

But...

- Provide No Help Ordering Source Files When They Are First Being Compiled
- Automatic Recompilation Of Units Already Compiled Into The Ada Library May Not Work Properly If The Dependencies ("Withs") Have Been Modified

P. Gacuk
SPAR Aerospace

Implementation Phase:

Configuration Management Problems:

- Different Vendors Use Different Ada Library Management Schemes

  Problem When Using Different Host & Target Compiler Vendors And Attempting To Have A Unified Configuration Management Scheme

P. Gacuk
SPAR Aerospace
58 of 62

Integration & Test Guidance:

SPAR

Generally, Integration Was Easier Than For Comparable Projects In Fortran Or C

- Largely Due To Ada's Compile Time Checking Of Interfaces
- Testing Ada Units & Programs Is Not Significantly Different From Other Languages

However...

- Attention Should Be Paid To Ada–Specific Features During Testing (Eg. Exceptions, Tasks, Etc.)

Follow On Phases:

Run Time Performance:

- Initial Run Time Performance Performance Was Disappointing
- Profiler Run To Carry Out Subprogram Execution Analysis (Identify Candidates For Optimization)

P. Gacuk
SPAR Aerospace
60 of 62

Follow On Phases:

Results Of Initial Code Optimizations:

SPAR

(Timing Test – Determine The Cycle Time Required To Calculate Successive Joint Rate / Position Commands)

Original Code, Normal Case (With No Joints Frozen):	230 Ms
Expansion Of Loops In Vector And Matrix Subroutines:	183 Ms
Expansion Of Loops In Pseudoinverse Subroutine:	127 Ms
Expansion Of Loops And Elimination Of Unnecessary Geometry Calculations:	58 Ms
Expansion Of More Loops And Optimization Of Joint Rate Calculation:	43 Ms
Streamlining The Joints Package	29 Ms

- All Modifications Made To Ada Code, No Assembler Written
- IBM PS/2 Model 70, 2.2 MIPS (Intel 80386/7 Chip Set) Best Current Model Of Proposed Space Station Embedded Data Processor
- Ada Compiler Used Generates Intel 80286/7 Code

P. Gacuk
SPAR Aerospace
61 of 62

Future Prototyping Efforts:

SPAR

- Continue To Distill Methods From Methodologies
- Object Oriented Requirements Analysis
- More Booch Based Object Oriented Design
- More Buhr Based System Task Design
- More Ada Run Time Analysis
- Involvement Of Verification & Validation Personnel
- Involvement Of Software Product Assurance Personnel

APPENDIX A — LIST OF ATTENDEES

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

ADAMS, NEIL .....BENDIX FIELD ENGINEERING CORP.
AIKENS, STEPHEN D.....DEPT. OF DEFENSE
ALANEN, JACK .....SOHAR, INC.
AMBROSE, LESLIE .....THE MITRE CORP.
ANDERSEN, BILL .....DEPT. OF DEFENSE
ANDERSON, FRANCES .....STANFORD TELECOMMUNICATIONS, INC.
ANDERSON, MARSHALL .....DEPT. OF DEFENSE
ANDREOTTA, DONALD J.....NASA/HEADQUARTERS
ANGIER, BRUCE .....INSTITUTE FOR DEFENSE ANALYSIS
APPLEGET, PATRICIA .....WESTINGHOUSE ELECTRIC CORP.
ARBOGAST, GORDON W.....DEFENSE COMMUNICATIONS AGENCY
ARMSTRONG, MARY .....IIT RESEARCH INSTITUTE
ARMSTRONG, ROSE .....MOUNTAINET, INC.
ASHTON, ANNETTE .....NAVAL SURFACE WEAPONS CENTER
ATKINS, EARL .....ELECTRONIC WARFARE ASSOCIATION
AZUMA, KENNETH I.....FORD AEROSPACE CO.

BACHMAN, SCOTT .....DEPT. OF DEFENSE
BAILEY, KIRK .....COMPUTER SCIENCES CORP.
BARBER, TOM .....COMPUTER SCIENCES CORP.
BARDIN, BRYCE M.....HUGHES AIRCRAFT CO.
BARKSDALE, JOE .....NASA/GSFC
BARNES, DAVID .....UNISYS CORP.
BARNES, FRANK .....LOCKHEED MISSILE & SPACE CO.
BARRY, GLEN .....EBA, INC.
BASSMAN, MITCHELL J.....COMPUTER SCIENCES CORP.
BEARD, ROBERT M.....COMPUTER SCIENCES CORP.
BECK, HANK .....JET PROPULSION LAB
BENEDICT, ROBERT J.....BOOZ, ALLEN & HAMILTON, INC.
BENITEZ, MEG .....DEPT. OF DEFENSE
BERRENS, MIKE .....TELEDYNE BROWN ENGINEERING
BEWTRA, MANJU .....CTA, INC.
BLAND, SKIP A.....UNISYS CORP.
BOND, PAUL .....SAIC
BOOTH, ERIC .....COMPUTER SCIENCES CORP.
BREDESON, MIMI .....SPACE TELESCOPE SCIENCE INSTITUTE
BREDESON, RICHARD W.....OMITRON, INC.
BRENNEMAN, DALE .....COMPUTER SCIENCES CORP.
BRESLIN, MARK .....GENERAL ELECTRIC CORP.
BRINKER, ELISABETH .....NASA/GSFC
BROWN, HARROLD E.....NASA/MSFC
BROWN, MARTY .....COMPUTER SCIENCES CORP.
BROWN, NEIL F.....DEPT. OF DEFENSE
BROWN, OTIS .....GRUMMAN
BUCKLEY, JOE .....COMPUTER SCIENCES CORP.
BUELL, JOHN .....COMPUTER SCIENCES CORP.
BUNCH, ALEDA .....SOCIAL SECURITY ADMINISTRATION
BURLEY, RICK .....NASA/GSFC
BUSBY, MARY B.....IBM
BUTLER, MADELINE J.....NASA/GSFC

CAKE, SPENCER C.....HQ USAF/SCTT
CARLISLE, CANDACE .....NASA/GSFC

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

CARMODY, CORA .....PLANNING RESEARCH CORP.
CARPENTER, MARIBETH B.....CARNEGIE MELLON UNIVERSITY
CARRIO, MIGUEL .....TELEDYNE BROWN ENGINEERING
CASASANTA, RALPH .....COMPUTER SCIENCES CORP.
CASE, ROBERT .....DEPT. OF DEFENSE
CATO, WILLIAM .....HQ USAF/SCTT
CERNOSEK, GARY J.....MCDONNELL DOUGLAS SPACE SYSTEMS CO.
CHANG, JOAN .....COMPUTER SCIENCES CORP.
CHEDGEY, CHRIS .....SPAR AEROSPACE CO.
CHU, MARTHA .....COMPUTER SCIENCES CORP.
CHU, RICHARD .....FORD AEROSPACE CO.
CHUNG, ANDREW .....FAA TECHNICAL CENTER
CHURCH, VIC .....COMPUTER SCIENCES CORP.
CISNEY, LEE .....NASA/GSFC
COHEN, HERBERT E.....AMSAA
COHEN, SARA .....GENERAL ELECTRIC CORP.
COLEMAN, MONTE .....DEPT. OF THE ARMY
COLSTON, RAYNETT .....COMPUTER SCIENCES CORP.
COOLEY, JAMES .....NASA/GSFC
COUCHOUD, CARL B.....SOCIAL SECURITY ADMINISTRATION
COVER, DONNA .....COMPUTER SCIENCES CORP.
CRAFTS, RALPH .....SS&T, INC.
CRAINE, BOB .....LOGICON, INC.
CRAMBLITT, FRANK .....IIT RESEARCH INSTITUTE
CRAWFORD, STEW .....
CREASY, PHIL .....MCDONNELL DOUGLAS ASTRONAUTICS CO.
CREEGAN, JIM .....FORD AEROSPACE CO.
CREPS, DICK .....UNISYS CORP.
CROKER, JOHN .....LISAN CORP.
CUCE, ROBERT J.....DEFENSE COMMUNICATIONS AGENCY
CUESTA, ERNESTO .....COMPUTER SCIENCES CORP.
CUTTS, ROY D.....DEPT. OF DEFENSE

D'AGOSTINO, JEFF .....OAO CORP.
DAKU, WALTER .....VITRO CORP.
DANGERFIELD, JOSEPH W.....TELESOFT
DANIELL, WALTER E.....IBM
DAVIS, TIM .....NASA/GSFC
DECKER, WILLIAM .....COMPUTER SCIENCES CORP.
DEGRAFF, GEORGE .....GRUMMAN
DEMAIO, LOUIS .....NASA/GSFC
DEMEO, JOSEPH R.....FEDERAL AVIATION AGENCY
DEMILLO, RICH .....NATIONAL SCIENCE FOUNDATION
DERENZO, BILL .....TARTAN LABS
DEVARAJ, SAVITHRI .....COMPUTER SCIENCES CORP.
DEVLIN, MIKE .....CONCURRENT COMPUTER CO.
DEWBRE, DOYLE .....DEPT. OF DEFENSE
DIGNAN, DAVID M.....DEPT. OF DEFENSE
DIKEL, DAVID .....FOCUSED ADA RESEARCH
DOUGLAS, FRANK J.....SOFTRAN,INC.
DUBIN, HENRY C.....U.S. ARMY OFFICE OF TEST & EVAL. AGEN
DUNIHO, MICKEY .....DEPT. OF DEFENSE
DUREK, TOM .....TRW

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

DUTTINE, VALERIE .....NASA/GSFC

EGLITIS, JOHN .....LOGICON, INC.
ELLIOTT, DEAN F.....SWALES & ASSOCIATES INC.
ELLIS, WALTER .....IBM
EMEIGH, MICHAEL .....LOGICON, INC.
EMERSON, CURTIS .....NASA/GSFC
EMERY, RICHARD .....VITRO CORP.
ERB, DONA M.....THE MITRE CORP.
ESHLEMAN, LAURA .....DEPT. OF DEFENSE
ESKER, LINDA .....COMPUTER SCIENCES CORP.
EUSTICE, ANN .....IIT RESEARCH INSTITUTE

FAFF, TIM .....IIT RESEARCH INSTITUTE
FEERRAR, WALLACE .....THE MITRE CORP.
FERNANDEZ, AL .....COMPUTER SCIENCES CORP.
FINK, MARY LOUISE A.....EPA
FISHER, TOM .....BOOZ, ALLEN & HAMILTON, INC.
FISHKIND, STAN .....NASA/HEADQUARTERS
FORSYTHE, RON .....NASA/WALLOPS FLIGHT FACILITY
FOURROUX, KATHY .....TELEDYNE BROWN ENGINEERING
FOUSER, THOMAS J.....JET PROPULSION LAB
FOX, EILEEN M.....IDE
FRIEND, GREGG .....COMPUTER SCIENCES CORP.

GACUK, PETER .....SPAR AEROSPACE CO.
GAFFKE, WILLIAM E.....PROJECT ENGINEERING, INC.
GALLAGHER, BARBARA .....DEPT. OF DEFENSE
GARCIA, ENRIQUE A.....JET PROPULSION LAB
GARY, ALAN V.....TELEDYNE BROWN ENGINEERING
GIESER, JIM .....VITRO CORP.
GILL, CHARLES W.....COMPUTER SCIENCES CORP.
GILLILAND, DENISE .....STANFORD TELECOMMUNICATIONS, INC.
GILYEAT, COLIN .....ADVANCED TECHNOLOGY, INC.
GIRAGOSIAN, PAUL .....THE MITRE CORP.
GLASS, JEFF .....PROJECT ENGINEERING, INC.
GLIES, MARK .....TELESOFT
GODFREY, SALLY .....NASA/GSFC
GOUW, ROBERT .....TRW
GRAHAM, MARCELLUS .....NASA/MSFC
GRAYBEAL, KYLE .....FEDERAL AVIATION AGENCY
GREEN, DAVID .....COMPUTER SCIENCES CORP.
GRIMALDI, STEVE .....BOOZ, ALLEN & HAMILTON, INC.
GRONDALSKI, JEAN .....COMPUTER SCIENCES CORP.
GRONECK, MIKE .....IBM
GUENTERBERG, SHARON .....PLANNING RESEARCH CORP.

HAIN, GERTRUD .....SABAS
HAIN, KLAUS .....SABAS
HALL, DANA .....SYSTEMS ENGINEERING AND SECURITY, INC
HALTERMAN, KAREN .....NASA/GSFC
HAMILTON, JOHN R.....FEDERAL AVIATION AGENCY
HAND, ROBERT .....GRUMMAN

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

HANG, BAILEY T.....BALLISTIC RESEARCH LAB
HARLESS, WALTON N.....TRW
HARRIS, BERNARD .....NASA/GSFC
HAYES, CAROL .....UNISYS CORP.
HEASTY, RICHARD .....COMPUTER SCIENCES CORP.
HEFFERNAN, HENRY G.....EDP NEWS SERVICES
HELLER, GERRY .....COMPUTER SCIENCES CORP.
HENDRICK, ROBERT B.....COMPUTER SCIENCES CORP.
HENRY-NICKENS, STEPHANIE .....NASA/GSFC
HERBOLSHEIMER, CHARLES .....FEDERAL AVIATION AGENCY
HEYL, NORMAN F.....U.S. GENERAL ACCOUNTING OFFICE
HIGGINS, HERMAN .....DEPT. OF DEFENSE
HILL, MIKE .....MARTIN MARIETTA
HILL, VICKI .....THE MITRE CORP.
HIOTT, JIM .....UNISYS CORP.
HOLLADAY, WENDY T.....NASA
HOLLOWAY, MICHAEL .....NASA/LARC
HOOTEN, MONICA .....FORD AEROSPACE CO.
HOUSTON, SUSAN .....LISAN CORP.
HSU, JAMES .....INFORMATION DYNAMICS, INC.
HUDSON, WENDY .....CONCURRENT COMPUTER CO.
HUTCHISON, GREG .....IBM

IRELAND, THOMAS .....TEKTRONIX DEFENSE SYSTEMS

JAHANGIRI, MAJID .....COMPUTER SCIENCES CORP.
JENKINS-BNAFA, JOVITA .....TRW
JENKINS-HUNTER, CARA R.....FEDERAL AVATION AGENCY
JESSEN, WILLIAM .....GENERAL ELECTRIC CORP.
JOHANNSON, HANK .....FORD AEROSPACE CO.
JONES, CARL .....SCIENCE APPLICATIONS, INC.
JONES, DAVID .....UNISYS CORP.

KARLIN, JAY .....PROJECT ENGINEERING, INC.
KASPUTYS, JACKIE .....NATIONAL SCIENCE FOUNDATION
KELLY, JOHN C.....JET PROPULSION LAB
KENNEDY, ELIZABETH A.....ROCKWELL INTERNATIONAL
KESTER, RUSH .....COMPUTER SCIENCES CORP.
KETCHUM, HARRY .....STATISTICA, INC.
KILE, THOMAS .....DEPT. OF THE ARMY
KIM, ROBERT D.....COMPUTER SCIENCES CORP.
KIMMINAU, PAMELA S.....DEPT. OF DEFENSE
KIRKPATRICK, MARK .....CARLOW ASSOC.
KOPP, ALLAN .....TELESOFT
KRAHN, MARGIE .....DEPT. OF DEFENSE
KREIDER, ROBERT .....NASA/HEADQUARTERS
KREMER, AUDREY .....IBM
KRIEGMAN, DAVID .....SRA CORP.
KUDLINSKI, ROBERT A.....NASA/LARC
KUNKEL, HENRY .....BOEING AEROSPACE CO.

LABAUGH, MODENNA .....MARTIN MARIETTA
LABAUGH, ROBERT .....MARTIN MARIETTA

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

LANDIS, LINDA .....COMPUTER SCIENCES CORP.
LAVALLEE, DAVID .....FORD AEROSPACE CO.
LEE, JOHN A.....GENERAL DYNAMICS
LEFEVRE, JEANNE .....UNISYS CORP.
LEVITT, DAVID S.....COMPUTER SCIENCES CORP.
LIGHT, WARREN .....CTA, INC.
LIN, CHI Y.....JET PROPULSION LAB
LITTLEWOOD, CHRISTOPHER .....MARTIN MARIETTA
LIU, JEAN C.....COMPUTER SCIENCES CORP.
LIU, KUEN-SAN .....COMPUTER SCIENCES CORP.
LOCKMAN, ABE .....GTE
LOESH, BOB E.....JET PROPULSION LAB
LONGENECKER, SALLY .....COMPUTER SCIENCES CORP.
LUCZAK, RAY .....COMPUTER SCIENCES CORP.
LaMARSH, MARGO .....NASA/LARC

MADDOCK, KAREN R.....TECHNOLOGY PLANNING, INC.
MADISON, DAVE .....IIT RESEARCH INSTITUTE
MADSEN, KENT .....UNIVERSITY OF CALIFORNIA
MALAY, SUSAN .....PLANNING ANALYSIS CORP.
MALTHOUSE, NANCY .....LOGICON, INC.
MARCINIAK, JOHN .....CTA, INC.
MARGONO, JOHAN .....COMPUTER SCIENCES CORP.
MARKS, TOM .....DEPT. OF DEFENSE
MARSHLICK, MICHAEL .....COMPUTER SCIENCES CORP.
MARTIN, GEORGE W.....PROJECT ENGINEERING, INC.
MARTINEZ, BILL .....FORD AEROSPACE CO.
MARVRAY, ESMOND .....NASA/GSFC
MATHIASEN, CANDY .....UNISYS CORP.
MAURY, JESSE .....NASA/GSFC
MCCLURE, MARTY .....BENDIX FIELD ENGINEERING CORP.
MCDERMOTT, TIM .....COMPUTER SCIENCES CORP.
MCDONALD, BETH .....DEPT. OF DEFENSE
MCGARRY, FRANK .....NASA/GSFC
MCKENNA, JOHN J.....DEPT. OF DEFENSE
MCWEE, HARRY .....DEPT. OF DEFENSE
MEDEIROS, EDWARD .....COMPUTER SCIENCES CORP.
MERIFIELD, JAMES .....ADVANCED TECHNOLOGY, INC.
MICKEL, SUSAN .....GENERAL ELECTRIC CORP.
MILLER, JOHN .....COMPUTER SCIENCES CORP.
MILLER, KENNETH .....THE MITRE CORP.
MOONEY, PAT .....IBM
MOYLEN, ALDEN .....COMPUTER SCIENCES CORP.
MULLER, ERICH .....SPARTA, INC.
MYERS, MONTGOMERY .....UNISYS CORP.
MYERS, PHILIP I.....COMPUTER SCIENCES CORP.

NARROWS, BERNIE .....BENDIX FIELD ENGINEERING CORP.
NELSON, ROBERT W.....NASA/HEADQUARTERS
NICKENS, DON O.....HARRIS SPACE SYSTEMS CORP.

O'BRIEN, DAVID .....CONCURRENT COMPUTER CO.
O'MALLEY, JAMES .....HGO TECHNOLOGY

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

O'MALLEY, RUTH E.....HGO TECHNOLOGY
O'NEIL, BOB T.....NASA/HEADQUARTERS

PAGE, GERALD .....COMPUTER SCIENCES CORP.
PAJERSKI, ROSE .....NASA/GSFC
PALMER, JAMES G.....APPLIED PHYSICS LAB
PARESO, SAM .....HGO TECHNOLOGY
PASCIUTO, MIKE .....NASA/HEADQUARTERS
PELNIK, TAMMY M.....THE MITRE CORP.
PEREZ, FRANK .....UNISYS CORP.
PFLARTER, DAVE .....MCDONNELL DOUGLAS CORP.
PLETT, MICHAEL E.....COMPUTER SCIENCES CORP.
PLUNKETT, THERESA .....DEPT. OF DEFENSE
POLE, THOMAS .....SOFTWARE PRODUCTIVITY CONSORTIUM
PORTER, ADAM A.....UNIVERSITY OF CALIFORNIA
POTTER, WILLIAM .....NASA/GSFC
PRESSMAN, TOM .....STRICTLY BUSINESS COMPUTER SYSTEMS
PRESTON, DAVID .....IIT RESEARCH INSTITUTE
PRISEKIN, JULIA .....IIT RESEARCH INSTITUTE
PURCELL, ELIZABETH .....THE MITRE CORP.

QUANN, EILEEN S.....FASTRAK TRAINING, INC.

RADOSEVICH, JIM .....NASA/HEADQUARTERS
RANADE, PRAKASH V.....COMPUTER SCIENCES CORP.
RANEY, DALE L.....UNISYS CORP.
RAPP, DAVE .....DEPT. OF DEFENSE
REDDY, JAY .....STRICTLY BUSINESS COMPUTER SYSTEMS
RIGTERINK, PAUL .....COMPUTER SCIENCES CORP.
RITTER, SHEILA J.....NASA/GSFC
ROBESON, THERESA .....IIT RESEARCH INSTITUTE
ROGERS, KATHY .....THE MITRE CORP.
ROSENZWEIG, DAVE .....HARRIS SPACE SYSTEMS CORP.
ROTTERMAN, GENE .....GENERAL DYNAMICS
ROY, DAN .....FORD AEROSPACE CO.
RUDOLPH, RUTH .....COMPUTER SCIENCES CORP.
RUMPL, WILLIAM M.....COMPUTER SCIENCES CORP.
RUSKIN, LESLIE .....COMPUTER SCIENCES CORP.
RUTEMILLER, OREN G.....STANFORD TELECOMMUNICATIONS, INC.

SABATINO, RICK .....OMITRON, INC.
SAUBLE, GEORGE .....OMITRON, INC.
SCHELLHASE, RONALD J.....COMPUTER SCIENCES CORP.
SCHOENBORN, BOB .....STATISTICA, INC.
SCHUETZLE, JIM .....FORD AEROSPACE CO.
SCHULER, MARY P.....NASA/LARC
SCHWARTZ, KAREN D.....GOVERNMENT COMPUTER NEWS
SCOTT, STEVE .....UNISYS CORP.
SEAVER, DAVID P.....PROJECT ENGINEERING, INC.
SEIDEWITZ, ED .....NASA/GSFC
SEVERINO, TONY .....GENERAL ELECTRIC/RCA
SHAW, CHARLES E.....CENTURY COMPUTING, INC.
SHAWE, M. .....BENDIX FIELD ENGINEERING CORP.

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

SHEKARCHI, JOHN .....COMPUTER SCIENCES CORP.
SHEPPARD, SYLVIA B.....NASA/GSFC
SHI, JEFF .....RMS TECHNOLOGIES, INC.
SHOAN, WENDY .....NASA/GSFC
SHYMAN, STEVEN .....INSTITUTE FOR DEFENSE ANALYSIS
SIEGERT, GREG .....IIT RESEARCH INSTITUTE
SILBERBERG, DAVID .....NATIONAL COMPUTER SECURITY CENTER
SIMMONS, BARBARA .....DEPT. OF DEFENSE
SIMONS, MARK .....NASA/GSFC
SLACK, IKE .....MCDONNELL DOUGLAS ASTRONAUTICS CO.
SLEDGE, FRANK .....GTE
SMITH, GENE .....NASA/GSFC
SMITH, OLIVER .....EG&G WASC, INC.
SMITH, PAUL H.....NASA/HEADQUARTERS
SOLOMON, CARL .....ST SYSTEMS CORP.
SPENCE, BAILEY .....COMPUTER SCIENCES CORP.
SQUIRE, JON .....WESTINGHOUSE ELECTRIC CORP.
SQUIRES, BURTON E.....CONSULTANT
STARK, MICHAEL .....NASA/GSFC
STEGER, WARREN .....COMPUTER SCIENCES CORP.
STEINBACHER, JODY .....NASA/JPL
STOKES, ED .....COMPUTER SCIENCES CORP.
STUART, ANTOINETTE D.....DEPT. OF THE NAVY
SUBOTIN, ROSA .....COMPUTER SCIENCES CORP.
SULLIVAN, JOHN D.....FEDERAL AVIATION AGENCY
SUN, ALICE .....THE MITRE CORP.
SWALTZ, LEON .....IBM
SWEIGERT, DAVID .....DAEDALEAN
SZULEWSKI, PAUL .....C. S. DRAPER LAB, INC.

TASAKI, KEIJI .....NASA/GSFC
TAUSWORTHE, BOB .....NASA/JPL
TAVASSOLI, NAZ .....COMPUTER SCIENCES CORP.
TAYLOR, GUY .....FLEET COMBAT DIRECTION SYSTEMS
THACKREY, KENT .....PLANNING ANALYSIS CORP.
THOMPSON, JOHN T.....FORD AEROSPACE CO.
THORNTON, THOMAS .....NASA/JPL
TRAYSYELUE, WEISNER .....COMPUTER SCIENCES CORP.
TSAGOS, DINOS .....GRUMMAN
TZENG, NIGL .....NASA/STX

UPPERT, DICK .....GRUMMAN
URBINA, DANIEL .....FORD AEROSPACE CO.

VALETT, JON .....NASA/GSFC
VAN METER, DAVID .....LOGICON, INC.
VEHMEIER, DAWN R.....OASD(P&L)WSIG
VIENNEAU, ROBERT .....KAMAN SCIENCES CORP.
VOIGT, DAVID .....BENDIX FIELD ENGINEERING CORP.
VOIGT, SUSAN .....NASA/LARC

WALIGORA, SHARON R.....COMPUTER SCIENCES CORP.
WALKER, CARRIE K.....NASA/LARC

SECOND NASA Ada USERS SYMPOSIUM ATTENDEES

WALKER, GARY N.....JET PROPULSION LAB
WALKER, JOHN .....IIT RESEARCH INSTITUTE
WATSON, BARRY .....IIT RESEARCH INSTITUTE
WAUGH, DOUG .....IBM
WEEKLEY, JIM .....FORD AEROSPACE CO.
WEISMAN, DAVID .....UNISYS CORP.
WELBORN, RICHARD P.....STANFORD TELECOMMUNICATIONS, INC.
WENDE, ROY .....FAIRCHILD SPACE CO.
WESTON, WILLIAM .....NASA/GSFC
WILDER, DAVID C.....DEPT. OF DEFENSE
WILLIAMS, CHERYL .....CTA, INC.
WILSON, BILL M.....QUONG ASSOC.
WILSON, RUSSELL .....BOEING AEROSPACE CO.
WITTIG, MIKE .....IIT RESEARCH INSTITUTE
WONG, ALICE A.....FEDERAL AVIATION AGENCY
WOOD, DICK .....COMPUTER SCIENCES CORP.
WOODWARD, HERBERT P.....TRW FEDERAL SYSTEMS GROUP

YANG, CHAO .....NASA/GSFC
YOUNG, LEON .....WESTINGHOUSE ELECTRIC CORP.

ZAVELER, SAUL .....U.S. AIR FORCE
ZELKOWITZ, MARV .....UNIVERSITY OF MARYLAND
ZOCH, DAVID .....FORD AEROSPACE CO.

APPENDIX B — STANDARD BIBLIOGRAPHY OF SEL LITERATURE

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-004, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-78-302, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3), W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1), W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-806, Annotated Bibliography of Software Engineering Laboratory Literature, M. Buhler and J. Valett, November 1989

SEL-83-001, An Approach to Software Cost Estimation, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-83-106, Monitoring Software Development Through Dynamic Variables (Revision 1), C. W. Doerflinger, November 1989

SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, Proceedings From the Ninth Annual Software Engineering Workshop, November 1984

SEL-85-001, A Comparison of Software Verification Techniques, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, R. Murphy and M. Stark, October 1985

SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985

SEL-85-004, Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, R. W. Selby, Jr., May 1985

SEL-85-005, Software Verification and Testing, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, Proceedings From the Tenth Annual Software Engineering Workshop, December 1985

SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

SEL-86-002, General Object-Oriented Software Development, E. Seidewitz and M. Stark, August 1986

SEL-86-003, Flight Dynamics System Software Development Environment Tutorial, J. Buell and P. Myers, July 1986

SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986

SEL-86-005, Measuring Software Design, D. N. Card, October 1986

SEL-86-006, Proceedings From the Eleventh Annual Software Engineering Workshop, December 1986

SEL-87-001, Product Assurance Policies and Procedures for Flight Dynamics Software Development, S. Perry et al., March 1987

SEL-87-002, Ada Style Guide (Version 1.1), E. Seidewitz et al., May 1987

SEL-87-003, Guidelines for Applying the Composite Specification Model (CSM), W. W. Agresti, June 1987

SEL-87-004, Assessing the Ada Design Process and Its Implications: A Case Study, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-008, Data Collection Procedures for the Rehosted SEL Database, G. Heller, October 1987

SEL-87-009, Collected Software Engineering Papers: Volume V, S. DeLong, November 1987

SEL-87-010, Proceedings From the Twelfth Annual Software Engineering Workshop, December 1987

SEL-88-001, System Testing of a Production Ada Project: The GRODY Study, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988

SEL-88-003, Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis, K. Quimby and L. Esker, December 1988

SEL-88-004, Proceeding of the Thirteenth Annual Software Engineering Workshop, November 1988

SEL-88-005, Proceedings of the First NASA Ada User's Symposium, December 1988

SEL-89-002, Implementation of a Production Ada Project: The GRODY Study, S. Godfrey and C. Brophy, September 1989

SEL-89-003, Software Management Environment (SME) Concepts and Architecture, W. Decker and J. Valett, August 1989

SEL-89-004, Evolution of Ada Technology in the Flight Dynamics Area: Implementation/Testing Phase Analysis, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard, C. Brophy, November 1989

SEL-89-006, Collected Software Engineering Papers: Volume VII, November 1989

SEL-89-007, Proceedings of the Fourteenth Annual Software Engineering Workshop, November 1989

SEL-89-008, Proceedings of the Second NASA Ada Users' Symposium, November 1989

SEL-89-101, Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1), M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

SEL-90-001, Database Access Manager for the Software Engineering Laboratory (DAMSEL), M. Buhler and K. Pumphrey, March 1990

SEL-RELATED LITERATURE

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[7]Basili, V. R., Maintenance = Reuse-Oriented Software Development, University of Maryland, Technical Report TR-2244, May 1989

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

[7]Basili, V. R., Software Development: A Paradigm for the Future, University of Maryland, Technical Report TR-2263, June 1989

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," Proceedings of the First Pan-Pacific Computer Conference, September 1985

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," Journal of Systems and Software, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," Proceedings of the International Computer Software and Applications Conference, October 1985

[4]Basili, V. R., and D. Patnaik, A Study on Fault Prediction and Reliability Assessment in the SEL Environment, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," Communications of the ACM, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and J. Ramsey, Structural Coverage of Functional Testing, University of Maryland, Technical Report TR-1442, September 1984

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P--A Prototype Expert System for Software Engineering Management," Proceedings of the IEEE/MITRE Expert Systems in Government Symposium, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press, 1979

[5]Basili, V., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," Proceedings of the 9th International Conference on Software Engineering, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," Proceedings of the Joint Ada Conference, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, June 1988

[7]Basili, V. R., and H. D. Rombach, Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment, University of Maryland, Technical Report TR-2158, December 1988

[2]Basili, V. R., R. W. Selby, Jr., and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., Comparing the Effectiveness of Software Testing Strategies, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," Proceedings of the NATO Advanced Study Institute, August 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, July 1986

[5]Basili, V. and R. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, December 1987

[2]Basili, V. R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: IEEE Computer Society Press, 1978

[5]Brophy, C., W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the Joint Ada Conference, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," Proceedings of the Washington Ada Technical Conference, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, October 1985

[5]Card, D., and W. Agresti, "Resolving the Software Science Anomaly," The Journal of Systems and Software, 1987

[6]Card, D. N., and W. Agresti, "Measuring Software Design Complexity," The Journal of Systems and Software, June 1988

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," IEEE Transactions on Software Engineering, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," ACM Software Engineering Notes, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: IEEE Computer Society Press, 1983

[5]Doubleday, D., "ASAP: An Ada Static Source Code Analyzer Program," University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," Proceedings of the 1988 Washington Ada Symposium, June 1988

Hamilton, M., and S. Zeldin, A Demonstration of AXES for NAVPAK, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

Jeffery, D. R., and V. Basili, Characterizing Resource Data: A Model for Logical Association of Software Data, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," Proceedings of the Tenth International Conference on Software Engineering, April 1988

[5]Mark, L., and H. D. Rombach, A Meta Information Base for Software Engineering, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[5]McGarry, F., and W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," Proceedings of the Sixth Washington Ada Symposium (WADAS), June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," Proceedings of the Hawaiian International Conference on System Sciences, January 1985

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (Proceedings), March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," Proceedings of the Eighth International Computer Software and Applications Conference, November 1984

[5]Ramsey, C., and V. R. Basili, An Evaluation of Expert Systems for Software Engineering Management, University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," IEEE Transactions on Software Engineering, March 1987

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," Proceedings From the Conference on Software Maintenance, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[7]Rombach, H. D., and B. T. Ulery, Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL, University of Maryland, Technical Report TR-2252, May 1989

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," Proceedings of the 21st Hawaii International Conference on System Sciences, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," Proceedings of the CASE Technology Conference, April 1988

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications, October 1987

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," Proceedings of TRI-Ada 1989, October 1989

Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," Proceedings of the Joint Ada Conference, March 1987

[7]Sunazuka. T., and V. R. Basili, Integrating Automated Support for a Software Management Cycle Into the TAME System, University of Maryland, Technical Report TR-2289, July 1989

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

[5]Valett, J., and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," IEEE Transactions on Software Engineering, February 1985

[5]Wu, L., V. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," Proceedings of the Joint Ada Conference, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," Journal of Systems and Software, 1988

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

NOTES:

[1]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985.

[4]This article also appears in SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986.

[5]This article also appears in SEL-87-009, Collected Software Engineering Papers: Volume V, November 1987.

[6]This article also appears in SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988.

[7]This article also appears in SEL-89-006, Collected Software Engineering Papers: Volume VII, November 1989.