Temp #13592

NASA Conference Publication 2361

Space Station Software Issues

Report of a workshop held at
NASA Langley Research Center
Hampton, Virginia
August 20-21, 1984

NASA

NASA Conference Publication 2361

Space Station Software Issues

Edited by
Susan Voigt
NASA Langley Research Center

Sharon Beskenis
Kentron International, Inc.
Hampton, Virginia

Report of a workshop held at
NASA Langley Research Center
Hampton, Virginia
August 20–21, 1984

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1985

PREFACE

The Space Station Software Working Group, an ad hoc NASA committee, sponsored a workshop for NASA software persons at the Langley Research Center, August 20-21, 1984, to discuss a software development environment and other software issues needing attention during the planning and definition stages of the Space Station Program.

During the workshop, the twenty participants formed three working groups to deliberate on management and acquisition of software, software development environments, and software methodology and technology. Consensus was reached on seven major recommendations relative to software, and these were subsequently made to the Space Station Program. In addition, 21 issues were raised at the workshop and later refined by the participants.

These proceedings present these issues as well as recommendations on how to address them. The issues fall into the following five categories: software management, software development environment, software standards, information systems support for space station software developers, and a future software advisory board for the Space Station Program.

Also included, as background information, are presentations which were given on the Shuttle Software Production Facility and on Software Technology within NASA.

CONTENTS

ATTENDEES

Center	Workshop Participants and Mail Code
LaRC	Susan Voigt (MS 125) - workshop leader
	Sharon Beskenis (Kentron) - scribe
ARC	Bob Carlson (233-10)
GSFC	Joe Hennessy (510.1)
	Frank McGarry (582)
	Bob Nelson (522.1)
	Dolly Perkins (522.1)
JPL	Dave Callender (506-328)
	John McLeod (506-328)
	Art Zygielbaum (171-209)
JSC	Jim Raney (FD)
KSC	Tom Purer (CS-SED-22)
MSFC	John Wolfsberger (EB42)
NSTL	Joel Wakeland
HQ	Joe Bishop (TS)
	Ai Fang (EI)
	Dana Hall (DE)
	Rhoda Hornstein (TX)
	Ken Wallgren (RC)
	Bill Wilson (DR)

A position statement was also contributed by Ed Senn, LaRC.

1. INTRODUCTION

This report summarizes the results of a workshop held at the NASA Langley Research Center on August 20-21, 1984, on the topic "Space Station Software Development Environment." This workshop was sponsored by the Space Station Software Working Group (SWWG) for NASA software specialists to consider what is needed to support space station software development. The SWWG is an ad hoc committee under the Data Management Working Group of the NASA Headquarters Space Station Technology Steering Committee. The SWWG was formed to focus on software issues of importance to the space station, especially during the early planning and development phases.

The workshop had four objectives:

To consider the state of technology of software development environments appropriate to the space station

To deliberate on issues of standards, deliverables, integration, and control of contracted software

To learn from past NASA experience

To recommend software research directions and specific actions for space station decision makers.

The call for participation in the workshop went to all members of the SWWG and to the directors of all the NASA Centers. Attendees were requested to submit one page position statements on "lessons learned" from previous NASA software projects and on one of the following topics:

Software Technology

- Available or required for space station
- Space station requirements versus software technology available
- Technology options expected in 1987

Software Standards to impose or to provide

- Environment for development of software
- Common tools and program libraries
- Methodology
- Documentation

Contracting Issues

- What should the Government provide (GFE)?
- What deliverables should be required?

NASA Management of Software Acquisition

- Control of contracted development
- Integration and testing issues
- Database/data capture related to software development
- Management and decision support systems
- Software maintenance strategy

Twenty participants came from eight NASA centers and four Headquarters offices. Position statements were submitted by 19 people on lessons learned from past NASA projects and on software development issues. These position statements were used during the workshop to stimulate discussion and focus on issues. Some of these statements will be incorporated into the experimental NASA Software Information System being developed by the NASA Office of the Chief Engineer and accessible through RECON.

During the opening session, James Raney reported on the JSC Space Station Information System (SSIS) concept, Susan Voigt presented a list of software development environment (SDE) elements, and Frank McGarry discussed software technology within NASA and the state of practice versus the state of the art.

Three working groups were formed to identify issues and to develop recommendations:

Management and Acquisition of Software (John McLeod, chair)

Software Development Environment (Frank McGarry, chair)

Methodology and Technology (Dana Hall, chair)

Twenty-one issues and associated recommendations were identified by these 3 groups. These are included in the five sections of these proceedings under the topics: Software Management, Software Development Environment, Software Standards, Information Systems, and the Future Role of the SWWG.

The final session of the workshop was a plenary session where the working groups reported on their results. Several issues raised were considered very important by the participants and there was consensus that these should be brought to the attention of the Space Station Program Office (Code S in NASA Headquarters) as soon as possible.

MAJOR PROGRAMMATIC RECOMMENDATIONS:

P1 There should be a person in level A (the headquarters Space Station Office), cognizant of software requirements and developments for the space station, who can provide programmatic guidance, budget formulation, and policy on all space station software matters, as well as to assure that a software management plan is developed and implemented.

P2 The Software Working Group (SWWG) should be given official status as an intercenter advisory group to the space station software manager.

P3 A space station software management plan should be developed as soon as possible. A special interest group has been established within the SWWG that will create a recommended draft of the needed software management plan.

P4 The relationship of the Technical and Management Information System (TMIS) (also known as the Management Communication and Data System) to the space station software development environment should be determined, including the correlation between the two and how they are to interact.

MAJOR TECHNICAL RECOMMENDATIONS:

T1 A space station Software Development Environment (SDE) is needed very soon to support both Phase B and Phase C/D contracting activities. (The SDE would consist of computer-aided tools for developing compatible software for the space station project.) It should be defined and modeled as soon as possible.

T2 A strategy for selecting the space station software language(s) should be determined, including evaluation and selection criteria, study of the relevance of Ada, and identification of other candidate languages.

T3 Requirements definition and analysis should be under way for the SDE, the TMIS, and the Space Station Information System. These systems are interrelated, and it is imperative that there be regular communication among the various system planners and developers to assure that good software engineering practices are used and duplication efforts are minimized or eliminated.

2. OPENING SESSION

Following the introduction of all the participants and a brief discussion of the objectives and format of the workshop, several presentations were made to provide background for the workshop deliberations.

2.1. Plans for Space Station Information System

James L. Raney, Johnson Space Center, discussed the current concept of the Space Station Information System (SSIS), see Figure 1. The SSIS consists of a network with a variety of processor nodes attached. Each node has one or more standard processing units (SPU), one or more universal workstations (UWS), and application subsystems. The core SSIS software includes the network operating system, the user interface, and common support software. He displayed a schedule for the software development environment (SDE) which shows the SDE being developed in parallel with the definition study contracts and available prior to application software development.

2.2. Lessons Learned From the Shuttle Software Development

Over the last decade, much experience has been gained with the Shuttle software at JSC, and Jim Raney offered to describe some lessons learned. He described the current Shuttle Software Production Facility (SPF). His slides are included in Appendix A.

2.3. Elements of a Software Development Environment

Susan Voigt, LaRC, led a discussion on the contents of a software management plan, the generic elements of a software development environment, and tool characteristics and considerations. The National Bureau of Standards Special Publication 500-78: NBS Programming Environment Workshop Report, June 1981, was cited as a good reference. The NBS workshop report defines several levels of tool environments from "Fig Leaf" to "Spacesuit", and these were used to stimulate discussions. Several modifications to the list of elements of a software development environment were made by participants and these are reflected in the section of this document on Software Development Environment Issues.

2.4. Software Technology Within NASA

Frank McGarry, Goddard Space Flight Center, discussed software technology within NASA, contrasting the state of the practice with the state of the art. His slides are included in Appendix B. His point was that it takes a long time to move technology from the idea stage to real practice and one must be selective in choosing the right practices for the right problems.

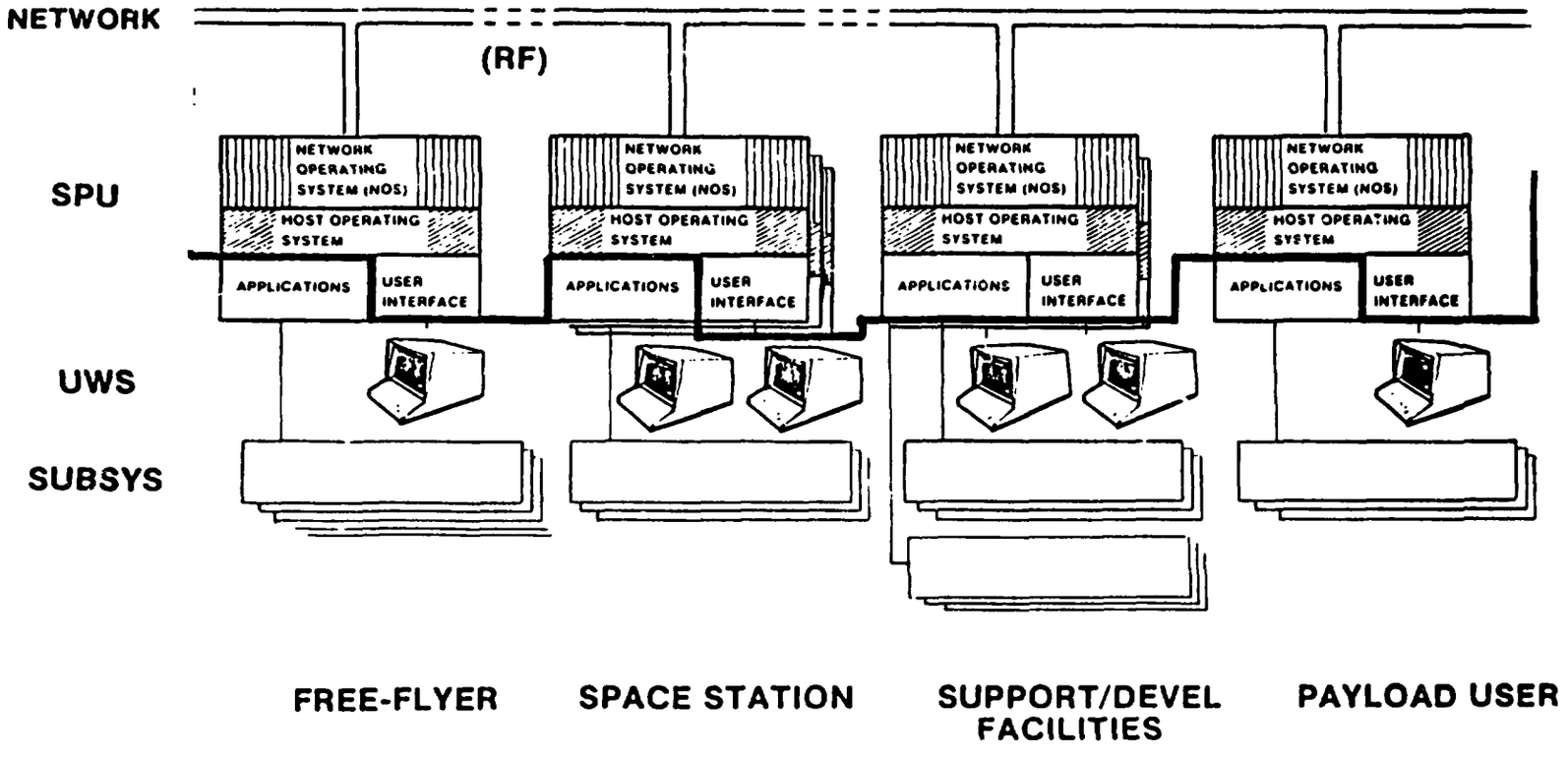


Figure 1. Space Station Information System - common "core."

3. WORKING GROUPS

The workshop participants divided into three working groups to deliberate on specific topics.

3.1. Management and Acquisition of Software

Group One was called Management and Acquisition of Software. John McLeod, JPL, was the leader and members were Joe Hennessy, John Wolfsberger, Ken Wallgren, Bill Wilson, Joe Bishop, and Bob Carlson. They were tasked to consider issues related to a software management plan, contract requirements, NASA control, integration and testing, government staffing, and the role of the SWWG.

3.2. Software Development Environment

Group Two was called the Software Development Environment Group. Frank McGarry, GSFC, was the leader and members were Bob Nelson, Dave Callender, Rhoda Hornstein, Tom Purer, Art Zygielbaum, and Susan Voigt. They were tasked to consider issues related to the definition and provision of a standard environment for space station software development, the appropriate language(s), and what (or how much) the government should provide to the contractors.

3.3. Methodology and Technology

Group Three was called the Methodology and Technology Group. Dana Hall, Headquarters Code D, was the leader and members were Dolly Perkins, Jim Raney, Ai Fang, Joel Wakeland, and Sharon Beskenis. They were tasked to consider the issues related to standards, new technology, quality assurance, verification and validation, configuration management, terminology, training, and methods.

The deliberations of the three working groups resulted in the major recommendations given in the Introduction as well as in some more specific issues and recommendations which have been subsequently refined by workshop participants and comprise the remainder of these proceedings. Associated with each issue statement is the name of the individual who was responsible for the text explaining the issue and the recommendations.

4. SOFTWARE MANAGEMENT ISSUES

4.1. Software Management Planning

BACKGROUND:

NMI 2410.6, which establishes software management policy for NASA flight projects, is currently inappropriate for an undertaking of the Space Station's scope and duration. Revisions to make NMI 2410.6 more

comprehensive and provide planning policy appropriate to Headquarters programs as well as Center projects are being performed by the NASA Software Management and Assurance Program. Unfortunately, this revised policy will not be approved early enough to officially impact space station activities in FY85. JPL has been developing a space station software management plan, but that plan has no official status.

The recommended Space Station Software Management Plan will have to guide and control a wide variety of activities involving organizations at various levels of management hierarchy. Many of the management issues addressed by the plan and many of the mechanisms and procedures that will have to be implemented to execute the plan will be difficult to resolve and will require lengthy coordination and trial periods.

ISSUE: (Bill Wilson)

HOW IS THE SPACE STATION SOFTWARE MANAGEMENT PLAN TO BE DEVELOPED?

ESSENTIAL CONSIDERATIONS:

The scope, duration, and number of organizations that will be involved in space station software activities necessitate the early availability of policy, standards, and management procedures to ensure acquisition of compatible and maintainable software.

There now is no individual or organization with the direct responsibility to produce the needed plan.

RECOMMENDATIONS:

(1) An individual within the Space Station Program Office should be appointed to be responsible for ensuring that a software acquisition management plan is developed to provide guidance to all participating organizations.

(2) An intercenter working group should be established immediately to develop the needed plan in time for it to be implemented in conjunction with phase B contracts.

(3) The planning activity should use existing NASA software management and acquisition guidance and the products of related ongoing activities as a basis for developing the needed plan.

(4) The plan and associated planning materials should be captured and maintained within an automated information management capability.

ISSUE: (Bill Wilson)

HOW IS THE SPACE STATION SOFTWARE MANAGEMENT PLAN TO BE IMPLEMENTED AT EACH MANAGEMENT LEVEL AND BY CONTRACTORS?

ESSENTIAL CONSIDERATIONS:

Software management planning must begin immediately so that the needed management capability will be in place to affect the initial software acquisition activities. Procrast'nation will result in a situation extremely difficult to reconcile.

RECOMMENDATIONS:

The following software management considerations should be immediately addressed by one or more working groups in parallel and in close coordination with the recommended planning group:

* International and DoD participation
* Definition/identification of center SMP controls/contact points
* Definition of TMIS/MCDS capabilities and schedules
* Hierarchy of configuration management and controls
* Software safety, reliability and quality management & assurance
* Total system integration planning, management and control
* Definition of the role and function of the software development environment

ISSUE: (James L. Raney)

HOW CAN THE WHOLE NASA STAFF BE BROUGHT UP TO DATE ON THE STATE OF TECHNOLOGY OF SOFTWARE AND SOFTWARE MANAGEMENT PRACTICES? MORE SPECIFICALLY, HOW CAN THE FOLLOWING BE ACCOMPLISHED:

(1) EDUCATING UPPER MANAGEMENT?

(2) IDENTIFICATION OF MINIMAL CURRICULUM FOR TRAINING AT ALL LEVELS, FROM THE WORKING LEVEL TO HIGHEST NASA MANAGEMENT?

(3) TRAINING FOR THE SDE (SOFTWARE DEVELOPMENT ENVIRONMENT)?

ESSENTIAL CONSIDERATIONS:

The NASA staff must realize the importance of embedded software in future programs and develop an understanding of software management and acquisition. NASA must establish an environment for software development and stick with it, improving it universally as the state of the art permits. Recognizing the importance of documenting software projects from their inception to their retirement will enable active projects to identify and satisfy real requirements and allow following projects to profit from past "lessons learned".

NASA, as a whole, must adopt improved practices for identifying and tracking the state of the art in software and software management. NASA management must be educated and persuaded by an aggressive training and advocacy program to adopt this position. Efforts currently in progress in the Chief Engineer's Office and other areas of NASA will contribute to solving this problem.

RECOMMENDATIONS:

(1) A NASA-sponsored software management tools and practices database and information retrieval system is being designed and constructed under the auspices of the Chief Engineer's Office, namely the NSIS (NASA Software Information System). There should be established within the charter of that system the responsibility to provide NASA-wide training and advocacy programs for software and management of software. There should be a NASA-wide mandate to support and participate in the NSIS, particularly in its role in providing a means of keeping NASA management and staff up to date on the state of the art in software technology and management practices. Each Center should establish a function to specifically interface with the NSIS both for receipt of current software-related information and for input of "lessons learned" into the NSIS. This same function can interact with the NSIS training program and local Center employee development functions to make the necessary training available and useful.

(2) The Space Station Program should establish procedures and identify key points (such as formal reviews) for capturing appropriate "lessons learned" data for inclusion in the NSIS database.

4.2. Independent Verification and Validation (IV&V)

BACKGROUND:

Experience has proven that an IV&V team, appropriately supported with standards, guidelines, and tools, can perform valuable services during the requirements, design, and testing phases, ensuring that the as-built software satisfies the requirements. This second engineering staff, an entity separate from the main development staff, is almost a necessity given situations of new technology, high risk, and engineering not done before. Each of these criteria fits the Space Station.

ISSUE: (Dana Hall)

USES OF INDEPENDENT VERIFICATION AND VALIDATION (IV&V) IN SPACE STATION SOFTWARE ACQUISITION

(1) WHAT ELEMENTS OF SPACE STATION SOFTWARE WILL BENEFIT FROM AN INDEPENDENT SECOND ENGINEERING OPINION?

(2) HOW SHOULD THAT IV&V SUPPORT BE MECHANIZED, I.E., CONTRACTING METHOD, TOOL SUPPORT, NASA ROLE, ETC.?

(3) WHAT SHOULD BE THE RELATIONSHIP OF THE IV&V SUPPORT EFFORT TO THE CREATION AND THEN USE OF THE SOFTWARE DEVELOPMENT ENVIRONMENT?

ESSENTIAL CONSIDERATIONS:

During the requirements process, the IV&V team can reevaluate requirements for completeness and clarity before development involving those requirements proceeds. Changing requirements, manifested during later stages of the life cycle, are one of the major drivers that increase development cost and delay completion. The IV&V team can demonstrate traceability of each software requirement back to a system and user requirement. Another important early life cycle task is to closely examine test plans to ensure that a practical way has been identified for testing every requirement.

During the design phase, the IV&V team can use the results of prototypes and simulations to help identify sound design approaches and verify that the evolving design satisfies the requirements. Prototypes of the GSFC Transportable Applications Executive, for example, enabled resolution of user interface and portability issues which yielded a cleaner implementation. Each design document should be subjected to independent review prior to release for coding.

Prior to and during testing, the IV&V team, as an independent group, can validate the system via test plan development, test designs, conduct of tests, and evaluation of test results. The IV&V staff can participate in requirements, design, and code reviews and inspections. These procedures have been found to be very effective and inexpensive in early detection of errors. The inspection process used on several ARC projects, for example, has increased productivity 2.5 times, decreased testing time 60%, and reduced maintenance problems 40 per cent. Perhaps most importantly, experience shows that acceptance tests should be performed by an independent dedicated test group.

RECOMMENDATIONS:

(1) The space station definition efforts should address the role and means for independent verification and validation in the software acquisition process. That support should be planned to build up in parallel with the development team.

(2) The Software Development Environment must provide the standards, guidance, and tools needed to support the IV&V functions. These include inspection procedures, requirements analysis aids, requirements traceability tools, prototyping and simulation capabilities, data input generators, and automatic test data reduction tools. Note that aids such as these are identical to those needed by the main software development team.

4.3. Quality Assurance and Configuration Management

BACKGROUND:

NASA Quality Assurance organization's participation in software systems on previous space programs has been very minimal due to lack of role definition and allocation of experienced manpower resources. To provide effective organization participation, role definition must be established early in the Space Station Program with prime consideration to establishing responsibilities and tasks that do not duplicate other NASA organizations or contractor participation. Manpower with software system experience must be made av lable and training programs instituted at each NASA Center.

In space station software development, many people will be developing a series of products, each of which builds on previous products, and all of which are constantly undergoing change. A change to any one product must be evaluated for its impact on the software developed thus far. Multiple versions of these products must be maintained. The configuration management discipline must be rigorous in order to keep constant track of all the software products being developed and to ensure that all personnel are working towards the same goal. All of the associated documentation, test plans, and test suites should also be controlled as they change and grow with the individual products and the system as a whole.

ISSUES: (Kent Castle - JLR)

(1) WHAT IS THE ROLE OF NASA QUALITY ASSURANCE ORGANIZATIONS IN SPACE STATION SOFTWARE ACQUISITION MANAGEMENT?

(2) IF SKILLS DO NOT EXIST AT ALL LOCATIONS, WHAT IS THE TRAINING AND PREPARATION TASK?

ESSENTIAL CONSIDERATIONS:

NASA Quality Assurance organizations must provide product assurance tasks for software systems as integral parts of software requirements, design, implementation, test and operational phases. Activities and tasks are to ensure the safety, reliability and quality of the software through the establishment of requirements and criteria, analysis, reviews, audits, inspections and assessments. Organization personnel must be intimately knowledgeable of both the software and total system design and operations.

The Space Station Work Package structure has established software definition for each NASA Center and the product assurance requirement document defines related organizational software requirements. Specific software activities and tasks should be documented in NASA Center organization plans and exchanged for inter-Center review. Software system training programs must be made available for organizational personnel at each Center.

RECOMMENDATIONS:

(1) The role of each NASA Center Quality Assurance organization relative to software should be documented in organizational plans and submitted to the Center Space Station Project Office for approval. The plan should include the following activities and tasks as a minimum:

(a) requirements development participation
(b) development plan evaluation
(c) requirements/change request evaluation
(d) preliminary hazards analyses
(e) hardware/software hazards analyses
(f) hardware/ software interaction analyses
(g) requirements traceability
(h) independent code assurance analyses
(i) milestone review support
(j) test program evaluation/test witnessing
(k) code evaluation/walk-throughs
(l) discrepancy reports/waivers/user note assessment
(m) end item delivery coverage

(2) Each Center Quality Assurance organization should acquire manpower with required software skills and institute software systems training programs. The training courses under development by the NASA Office of the Chief Engineer should be given serious consideration as aids in proper staff preparation.

ISSUES: (Sharon Beskenis)

WHAT LEVEL OF CONFIGURATION CONTROL IS NEEDED? HOW DO WE GUARANTEE WE HAVE CONFIGURATION CONTROL AND TO WHAT LEVEL IS NASA INVOLVED?

ESSENTIAL CONSIDERATIONS:

Since the space station software suppliers will be spread across the country, a very stringent configuration control methodology will need to be enforced so that configuration control is guaranteed. The system must be flexible enough, however, to permit developers to correct and modify their programs in-house until they are satisfied that the programs work according to design specifications using their own test data. At the point when the module first enters a system build, it should be baselined for control. When a problem is found in a module included in a system build, the programmer should be given the flexibility to correct and test the code informally until the problem is solved. At that point, the software configuration should be updated for the next build.

Several questions arise such as who is responsible for providing and maintaining the configuration control system. Where will the configuration control system reside and how will suppliers access it? What mechanism will be used to enter or update code and documents into the system? How will management monitor what is going on? These important questions must be answered IMMEDIATELY.

RECOMMENDATIONS:

(1) The core space station data system and SDE should utilize the most stringent configuration control discipline, and the automated configuration control system should reside in TMIS/SDE under the control of a single Configuration Control manager. All developers should have access via networks to the configuration controlled code, documentation, and test suites impacting their tasks.

(2) The JSC Space Station Program Office should generate the configuration control plan before the work of the separate contractors or producers is begun. All of the products to be controlled should be identified at this point, i.e., major subsystem names, and provisions for unique identification and control of each source module and each system created for test and integration must be made. The products that will serve as a baseline for the next phase of development must be determined.

(3) Configuration control training must be provided to managers and developers so that everyone will understand the mechanisms for admitting, accessing and modifying the documents and code that will be administered by the configuration control system. The training time must be included in the scheduling.

(4) A proposed mechanism for initially entering documentation or code into the configuration control (CC) system would be the following:

- Documentation should follow the format guide specified for all documents. A review of the document to be submitted to the configuration control system must be held with the NASA technical monitor and representatives of all products affected by the product whose documentation is being reviewed. Once the document is approved by the review committee and signed off by the appropriate technical task or test leader, the configuration control manager can approve inclusion of the document in the CC system.

- The developer must have a requirements document for his subsystem resident in the CC library before the interface document can be written. The interface document must reside in the CC library before a design document can be written, reviewed and included in the library. The interface and design documents should be living documents that become more detailed with the functional decomposition of the product into modules.

- Once a developer's design document has been approved and included in the CC library, the source code for the product is eligible for admission in the library. Once the developer has performed in-house testing and has had code walkthroughs that have been signed off by the technical task leader, the CC manager can place the code under the configuration control system.

The bottom line is that good software engineering practices should be a vital part of the configuration control process.

(5) A Configuration Control Board (CCB) composed of the software project manager, the configuration control manager, software technical task leaders, software test leaders, and QA representatives should control how changes to products are handled. The software project manager as chairman of the CCB has the final approval authority. A typical scenario might be the following:

> A Modification Request (MR) is filled out by a developer or tester. The affected subsystem/document is identified, the nature of the problem is detailed, the priority is assigned, and the impact of the problem and the consequence if not corrected are determined. The CCB reviews the MR to concur with or adjust the priority, etc., and select the due date for resolution of the MR. These decisions are based on feedback from the affected CCB members who may have had to check with the actual developers and/or testers to determine the scope of the problem. Next the MR is assigned to the appropriate developer or tester for a proposed solution. If the solution is approved by the CCB, the affected code or document can be checked out by the assigned developer or tester for modification and checked back into the configuration control system when the necessary changes are complete. If the change is not complete a week before the due date, a warning flag should be issued by the configuration control system and a notice should be sent to the CCB members and the person assigned to fix the problem so that an alternate strategy can be chosen, if the deadline won't be met. A red flag should be raised if the MR is unresolved by the due date.

(6) The configuration control system should provide planning and contingency options. What happens if a piece of software is not delivered to the CC library in time for the next phase or build ... what alternate strategy can be used, if any? Flags and scheduling outputs should be provided to indicate the project status.

(7) The builds or makes should be designed so that system dependencies can be indicated. In this manner, all modules affected by a change in a particular piece of software will automatically be rebuilt. Builds for different versions or test cases can also be specified. This eliminates human errors such as not recompiling a module using the changed code or forgetting to include the appropriate data in a test suite. Recompilation of the entire system is also prevented

with proper build capabilities such as the UNIX makes and builds.

4.4. Avoiding Major Software Problems on the Space Station

BACKGROUND:

A classical software disaster is one where the time to develop the software was at least twice what was originally estimated and the cost at least a factor of three over the original estimate. The term "at least" is stressed.

Software is more important to Space Station than to any previous, major NASA project. As Jack Garman (JSC) stated, "software is the glue that holds Space Station together." There are many factors in the embryonic Space Station Project that indicate that the current Space Station Project mode will lead to a classical software disaster in the next five to ten years.

ISSUE: (E. D. Callender)

SPACE STATION SOFTWARE - A CLASSICAL DISASTER IN THE MAKING

ESSENTIAL CONSIDERATIONS:

The primary technical indicator of a classical software disaster is the attempt to use new technology to do new things. For Space Station NASA is proposing to:

(1) Do the SE&I. NASA did not do that on Shuttle.

(2) Define and enforce use of a Software Development Environment.

(3) Use a new language - Ada. There is no backup language defined.

(4) Create a long-term commercially viable environment for industry. Space Station is not just a "put it together and fly it once project."

(5) Develop software for use/evolution over a projected 30 year life span. This is longer than any existing software product has ever operated!

The technical indicators are compounded by the following classical software management errors:

(1) Lack of software requirements,

(2) No software management plan,

(3) Little project management attention to software,

(4) An imposed project schedule and budget based upon no firm software requirements.

Following the current NASA approach to software for Space Station, the only question is the extent of the disaster. Will the schedule overrun be a factor of 2 or 3 or ? Will the cost overrun be a factor of 3 or 4 or ?

RECOMMENDATIONS:

Formulate a software management plan immediately and establish software management responsibility in the level A program office.

5. SOFTWARE DEVELOPMENT ENVIRONMENT ISSUES

BACKGROUND:

Good software engineering practice includes the use of support tools and a methodology that together form the essence of a software development environment (SDE). Although the generation of a SDE for space station software seems like a logical step to take, there may be many logical, valid reasons for not having NASA itself attempt to define such a functional entity. There may also be valid reasons for not defining a single (or small set) SDE at all, even by major vendors supplying space station software to NASA.

ISSUE: (Frank McGarry)

SHOULD A UNIFORM NASA SDE FOR SPACE STATION BE DEFINED AND DEVELOPED?

ESSENTIAL CONSIDERATIONS:

(1) The development of space station software will be highly distributed. There will not be a localized, single contractor or group responsible for the complete set of required software. Major portions of the software will be managed by various centers as opposed to being localized at a single NASA center.

(2) There will be major functional differences between major components of the software which may call for completely separate types of support development environments. The types of software to be developed include:

1. real-time flight software
2. ground data processing software (non real-time)
3. ground command and control software
4. integration and test software
5. simulation/modeling software

6. customer (user) application (data reduction) software

(3) Classically, NASA has told contractors what software products are needed, not how to develop them. NASA normally does not supply tools or define components of development environments. Possibly a NASA defined software environment should be designed to support NASA managers/developers only--not contractors. Vendors may do a much more effective job of defining their own software development environments optimally tuned to support their own specific efforts.

(4) The lifetime of the space station support will potentially exceed 30 years. If a single NASA SDE is defined, there would be a possibility of stifling the infusion of new technology to address specific concerns in the later years of support. (I.e., a standard environment would have the potential of growing stagnant.)

RECOMMENDATIONS:

Although there are potentially many difficulties with defining a standard NASA software environment, the working group feels the potential advantages far outweigh the difficulties.

NASA should define, design, and generate two well-defined software development environments:

(1) A SDE consisting of the tools, languages, data bases, etc., to support the software developers and their managers; subsets of the environment could be used to address the specific functional software being developed (i.e., flight vs. ground, etc.)

(2) A SDE management environment that will support the NASA software managers who are responsible primarily for acquiring software from vendors

The first environment would consist of the following functional capabilities:

1. Mail, telecommunications support
   (e.g., editors, file systems, communications aids,...)

2. Technical management/control aids
   (e.g., cost models, project management systems, build plans)

3. Data base support
   (file management, retrieval, control, etc.)

4. Modeling/simulator aids
   (architecture models, testing aids)

5. Prototyping aids
   (for requirements, specs, man/machine interface studies)

6. Documentation preparation aids

7. Requirements specification validation and analysis aid

8. Design specifications aids
   (PDL analyzers, data dictionary, etc.)

9. Code construction and control aids
   (compilers, cross compilers, link editors, change control, ...)

10. Program analysis/testing and integration
    (path coverage/test generators, etc.)

11. Metrics
    (quality measures, complexity measures, cost and reliability measures)

12. Man-machine interface support
    (interface and use of the environment, help, tutorial, etc.)

The second environment would contain those aids required only by the NASA manager responsible for requirements/acquisition/acceptance. The capabilities would include such functional areas as outlined above:

1. Same
2. More heavily directed toward schedules/planning/pms/pert
3. Same
4. Minimal or none
5. Minimal or none
6. Same
7. Same
8. Minimal or none
9. Minimal or none
10. Minimal or none
11. Same
12. Same functional need

ISSUE: (Susan Voigt)

HOW MUCH OF THE SPACE STATION SOFTWARE DEVELOPMENT ENVIRONMENT SHOULD BE FURNISHED BY NASA?

ESSENTIAL CONSIDERATIONS:

Many (if not all) major aerospace companies and software houses have developed one or more SDEs for use in-house. Recent reviews of IR&D activities in several companies certainly bear this out. There are some software engineering workstation products now on the market which can be considered to be turn-key software development environments.

Within NASA, no SDE has evolved, although elements exist at several NASA Centers. No system has been identified as appropriate for NASA or Space Station use. Howe· ·r, given the premise that a space station SDE is desirable, the question is, how much of it should NASA provide.

The benefits of a single SDE for Space Station include savings in time and money during the development phase, as well as commonality of the software which greatly enhances the ability to maintain the system over a long lifetime. The SDE would provide common editors, compilers. and utilities. The format of the software could be standard, making reviews and integration much easier. For example, test repeatability would enhance system integration activities. A NASA-furnished language processor, for use by all software developers, increases the probability of having a common language used at manufacturing, experiment and payl· facilities as well as at the launch site and on orbit. The transfer · application programs and SDE tools from one location to another wil greatly reduce the overall cost of development.

A major drawback to requiring the use of the NASA official SDE is that comp·nies have invested much in their own environments, they have their peop.e trained to use them (not the NASA SDE), and they have tools that fit their methodology and way of creating software. Permitting companies to use their own development environment would provide the best schedule/cost (e.g., planning cycles would be shorter, training minimized, and the risk for integration and test schedules would be reduced within the specific subsystem involved).

A NASA SDE would probably include vendor proprietary products, since it would be costly and time-consuming to develop an entire SDE for space station. This implies legal arrangements to permit citation of vendor products in specifications and provision of software to contractors for use under the space st.·tion development contracts. Arrangements would need to be made with the vendors (or suppliers) of SDE software to establish who has responsibility for support and maintenance. If changes ar· necessary, who is responsible? If delivery of a space station software product is late, can the blame be placed on the poor performance of the SDE supplied by the government?

RECOMMENDATIONS:

(1) NASA should perform SDE requirements analysis, design, and development, and make these requirements and the SDE software available to contractors and NASA managers.

(2) Negotiations with software tool vendors should begin soon after the NASA space station SDE is specified, so that development need only proceed on the elements of the environment that do not exist. These negotiations should clarify the responsibilities for software correction, maintenance and enhancement, and also establish provision for distribution of the SDE with these vendor-supplied elements to NASA sites and space station contractor sites as well.

(3) The use of the official NASA SDE by contractors is optional at their own sites, but mandatory at NASA installations (for software acceptance and for integration and test).

(4) Space station contractor-developed software products must be compatible with and reside within the NASA SDE. This requirement should be included in each contract for space station software, and demonstrated upon delivery of the software as part of the acceptance criteria.

(5) The NASA SDE should be provided to every NASA Center with space station software management responsibility and should be used by the NASA managers in support of the software development, integration, test, and control.

ISSUE: (E. D. Callender)

THE IMPACT OF SDE ON CONTRACTORS

ESSENTIAL CONSIDERATIONS:

Given that the Space Station Project mandates that all contractor-developed software either shall be developed within the software development environment (SDE) or shall, as a first step in integration and software acceptance, execute and be documented within the SDE, there will clearly be an impact upon space station contractors. Ultimately the impact will be assessed by those contractors who wish to bid on space station software. In the material below, a few of the obvious issues relative to this topic are outlined. It is assumed that the SDE has a very positive impact upon software development and test.

The potential positive impacts are:

(1) The time and costs for development and subsequent maintenance of a particular piece of software will be reduced when compared to a nonsupportive contractor environment.

(2) The contractor will have a good set of requirements for the format and content of the documentation for the software.

The potential negative impacts are:

(1) Contractor training costs.

(2) Problems with acceptance of the SDE by contractor personnel.

(3) Lack of ability to use contractor unique methods and tools.

(4) Possibility of contractor blaming integration or acceptance test problems on SDE, particularly if configuration control of SDE is lax, causing potential legal and contracting problems.

RECOMMENDATION:

This is an issue on which we should request major feedback from the industry representatives. We should also ask for industry experience on the use of particular SDEs.

ISSUE: (R. W. Nelson)

WHAT IS THE DEVELOPMENT APPROACH TO THE SOFTWARE DEVELOPMENT ENVIRONMENT?

ESSENTIAL CONSIDERATIONS:

Software Development Environments for major software development efforts such as the Space Shuttle on-board software system have taken many years to develop in order to provide fully functional support. SDEs for possible space station languages such as Ada are currently in their infancy. Many technologically advanced tools using graphical user interfaces and decision support systems are being developed especially in prototype form. These tools are generally addressing portions of the software development environment rather than total life-cycle support. Totally integrated software development environments will be necessary to realize the highest productivity goals of the Space Station Program.

RECOMMENDATIONS:

The inception of the design and development of the Software Development Environment for Space Station must begin as soon as possible in order to achieve a totally integrated environment. Since all functions probably will not be required to the same degree of urgency, it is recommended that a list of desired functional capabilities be generated and prioritized. In order to quickly demonstrate the concept of a totally integrated software development environment and to evaluate alternative technologies, a prototype (or model) should be developed. The prototype system will insure that only proven and fully evaluated support tools will be made available in the space station Software Development Environment. Based upon experience with the prototype, the Software Development Environment can be fully specified. The full environment can be developed incrementally according to the list of prioritized functional capabilities.

ISSUE: (Tom Purer)

HOW SHOULD THE MAINTENANCE AND EVOLUTION OF THE SOFTWARE DEVELOPMENT ENVIRONMENT BE CONTROLLED OVER THE 30 YEAR LIFETIME?

ESSENTIAL CONSIDERATIONS:

Maintenance is one of the major software costs, and as such, it should be a prime design driver for the Space Station Information System (SSIS). The ability to upgrade hardware technology without a major

impact to existing software systems is referred to as technology transparency. Software maintenance is directly related to technology transparency, since the software organization or structure that promotes easy maintenance also enhances the degree of software independence from the hardware. With the magnitude of the Space Station Program and its expected long life, it is hard to visualize the start of this program without having technology transparency as a major design goal for SSIS software.

Technology transparency as it relates to maintenance is found in the overall software structure. The ease of software maintenance is related to the type of structure used to develop the system. If the development approach is one that includes both layers and small modules with clearly defined interfaces, the problems associated with maintenance are kept to a minimum. A layer approach is not new; it was born with the first generation of machines. However, these early machines generally had only one layer, and we referred to that layer as the machine language level. Modern computers usually have 5 layers or levels. The five layers are:

First layer ..... Microprogramming layer

Second layer..... Machine language

Third layer ..... Operating system level

Fourth layer..... Higher Order Languages

Fifth layer ..... User Interface Languages

The selection of the five levels is somewhat arbitrary, but it serves a purpose in presenting the concept of technology transparency and its relationship to software. The first level consists of microprograms that are executed directly by the hardware. The microprograms in the first level are used to decode the machine language instructions in the second level. The third level, the operating system, is generally written in an assembly language and translated by an assembler to a set of machine language instructions. The operating system is a collection of program modules that control the resources of the machine. These resources include main storage, secondary storage, I/O devices, and files. The fourth level consists of programs written in languages designed to be used by application programmers. Such languages are called by many names including high level languages and problem-oriented languages. Literally hundreds of different ones exist. A few of the better known ones are: FORTRAN, COBOL, PASCAL, BASIC, HAL/S, and ADA. The fifth level, the user interface level, is oriented to the user and defines how the user interfaces with the machine. This may be done with a set of cryptic codes, commands, menus, prompts, or a mouse-type device. The user interface could also include a very high level language and even a voice recognition device to control the lower levels within the machine. With a clearly defined interface between the layers, it is possible to replace layers when the technology of those layers becomes obsolete. In other words, the user interface programs, operating

systems, or the microprocessor could be replaced with minimal impact on the rest of the system.

RECOMMENDATIONS:

The selection of the microprocessor and operating system is of critical importance. A microprocessor should not be selected because of unique requirements, but rather it should be based on a widely used "industrial standard" microprocessor. When a manufacturer upgrades a widely used microprocessor, it is far less likely that the next generation will be incompatible with the old. The interface between the next generation microprocessors and level one, i.e., the microcode, should remain relatively constant. If the next generation microprocessors are not upward compatible, changes to the microcode could be implemented to maintain the compatibility with the upper layers.

The selection of the operating system presents different types of problems. With the selection of an "industrial standard" or one supplied by the computer manufacturer, one generally does not buy the source code or the right to maintain the operating system. Without the source code or maintenance agreement, one can be locked into an operating system that cannot be modified to support unique requirements. Also, the computer manufacturer may decide to upgrade his system in a manner that is incompatible with your existing system, leaving you with an old, outdated, and non-maintained operating system. The decision to build a unique operating system is also full of problems; for example, the cost of building and maintaining an operating system over the lifetime of the Space Station will be enormous.

To maintain an operating system or other levels of software for twenty or more years has never been done. If this is to be a goal of the Space Station Information System, then maintainability must be built into the system from the start. One way to help insure that the software can be maintained is to create software using a modular and structured approach. The modules must be well documented and small, with the interfaces clearly defined.

Application programs located in the fourth and fifth layers must be insulated from the effects of changes made to the data, their organization, and the physical devices on which they are stored. The term 'data independence' is often used as one of the attributes of a data base. One must consider the use of a data base and its relationship to the application programs or be faced with numerous changes to the application programs because of changes to the hardware.

N85-20692

6. SOFTWARE STANDARDS ISSUES

6.1. Need for Common Terminology

BACKGROUND:

Terminology has always been a problem when discussing software with people from different work situations and backgrounds. This problem is very likely due to the relatively young age of the computer/software industry. People in different parts of the country (and various companies) began to put "tags" on certain practices that became common terminology to that group, but certainly not common throughout the software community. As literature about the industry began to be published, terminology began to become more common. However, even today there exist considerable differences of opinion as to what many "common" terms really mean. Working definitions of some common terms still mean different things to different software people.

ISSUES: (Joel Wakeland)

NEED FOR COMMON SOFTWARE TERMINOLOGY

(1) DOES THE EXISTING SPACE STATION LEXICON COVER SOFTWARE?

(2) IS THE COVERAGE ADEQUATE?

(3) SHOULD THERE BE A SPECIAL SOFTWARE LEXICON?

(4) WHO SHOULD BE RESPONSIBLE FOR A SOFTWARE LEXICON?

ESSENTIAL CONSIDERATIONS:

This phenomenon of common terms having different meanings to different individuals has been observed numerous times in software meetings that have taken place to date. It is, of course, most prevalent wher software people from different Centers meet to discuss software issues because of their diverse backgrounds. The only real solutions to this problem are time and continued communication within the software community. However, until this happens, a concerted effort should be made to standardize the key software terminology.

RECOMMENDATIONS:

A concerted effort should be made to standardize key software terminology as it relates to the Space Station. The document containing this terminology should be included as part of all software RFP packages. Potential contractors should be required to use this terminology in their proposals as well as in subsequent documentation.

The existing Space Station Lexicon (Issue 1, September 14, 1984) does not address the issue of software terminology. It should! There are two possible approaches. One is to have a set of software terminology definitions as a separate section. The other is to have the software definitions mixed in with the other definitions. The former is considered to be the preferred method.

The responsibility of developing a space station software lexicon would best reside with a group of individuals that make up a composite of the software community within NASA. The obvious candidate with the correct set of qualifications is the SWWG. However, it is unlikely that those on the SWWG have the time to take on this task. The most viable alternative could be a support contractor under the direction of someone from Headquarters. Here the most likely candidate is probably someone in the Office of the Chief Engineer.

6.2. Project Directives

BACKGROUND:

In previous NASA space programs, not only have the various NASA Centers developed their own software project practices/standards, but individual efforts have also been made at each Center on the same NASA program. To control cost of ownership for NASA, the Space Station Program must establish and enforce a single set of software directives across the project.

Software for the Space Station Program will originate in three ways, namely: (1) from adaptation of existing owned software, (2) from acquisition of existing commercial software, and (3) from development of new software. A minimum set of software engineering standards/practices must be established and enforced across the entire project to ensure that the end result of integrating all this software is a cost-effective, easy-to-operate space exploration/utilization system.

ISSUE: (James L. Raney)

WHAT IS THE MINIMUM SET OF SOFTWARE PROJECT PRACTICES/STANDARDS?

ESSENTIAL CONSIDERATIONS:

The joint "lessons learned" of the SWWG must be combined into one common directive specifying the minimum set of software engineering practices/standards to be applied to all software, both developed in-house or by contract and purchased "off-the-shelf" for the Space Station Program. The complete set of such directives must be applied by each Center in the preparation and utilization of their Software Management Plan for the Space Station Program.

RECOMMENDATIONS:

The Space Station Program Office must prepare and enforce a software management directive, subject to SWWG approval, that provides the following:

(1) A common software engineering methodology, including:

a. Critical life-cycle events
b. Documentation standards (identity and contents)
c. Structured software only
d. A set of acceptable languages
e. Coding and naming standards
f. Common support software tool set
g. Software development plans
h. Transportable standard format data files

(Note: The Software Management and Assurance effort of the Chief Engineer's Office is responsible for items a, b & g. The Space Station Level B Program Office is responsible for all the others.)

(2) All practices/standards must be in place by end of FY85.

(3) The practices/standards should apply to all test beds.

(4) The practices/standards must address existing hardware/software systems, such as Space Shuttle and communications satellites that must interface with the space station.

(5) The practices/standards must address the requirements for security of data to be handled by the Space Station Information System, including such various aspects as international, inter-corporate, political, and defense requirements.

6.3. Software Technology and Portability

BACKGROUND:

The designers of space station software are faced with a plethora of existing software tools and technologies -- and ideas as yet unrealized -- from which to choose to support an internally compatible, consistent, integrated, and maintainable system. Past development efforts give recurring lessons: incompatible tools increase the complexity of the job; tools which work well in prototype (i.e., limited) environments do not transfer well to operational environments; a useful-looking tool may not be mature enough for widespread use; a tool may be tightly coupled with a particular environment and will not easily transfer; methodologies are not chosen early enough, or are not enforced uniformly; developers do not have adequate experience using chosen technologies.

ISSUES: (Dolly Perkins)

(1) WHAT CRITERIA SHOULD BE USED TO SELECT SSIS SOFTWARE TECHNOLOGY (E.G., SOFTWARE ENGINEERING METHODS AND PRACTICES; STANDARDS FOR PORTABILITY; PROGRAMMING LANGUAGE; WHETHER TO IMPOSE AN INSTRUCTION SET ARCHITECTURE; DATA DRIVEN VS. DATA EMBEDDED SOFTWARE)

(2) WHAT CRITERIA SHOULD BE USED FOR TECHNOLOGY CHANGEOVER?

(3) HOW CAN WE MAKE TECHNOLOGY CHANGE TRANSPARENT?

(4) HOW DO WE KEEP CURRENT IN TECHNOLOGY?

ESSENTIAL CONSIDERATIONS:

Experience has shown that upgrades to technology -- both hardware and software -- are generally difficult unless the system architecture was deliberately designed to accommodate them. Typically, the practical result of not planning for technology change is either high reengineering cost to upgrade the outdated system components or hard decisions to keep obsolete parts.

However technologies are chosen, valuable experience can be gained if prototyping of system elements is possible. Prototyping can provide insight into software interface architectures and design, data base issues, system operation, system performance and human-computer interactions. Rapid and inexpensive experimentation allows concepts and requirements to be investigated, validated, and reviewed before implementation.

RECOMMENDATIONS:

(1) The NASA Software Working Group should develop a position paper to define criteria for choosing appropriate technologies and for determining when technology change should take place.

(2) The primary means for accomplishing technology transparency should be through careful layering of system elements. The interfaces between the layers should be well-defined and rigorously maintained throughout the system's life. With such portable interfaces, the component parts (the internals of the layers) can be replaced or modified without affecting any of the surrounding layers.

    Choice of breakpoints between layers is critical -- they should be well-isolated with clean separations. Divisions between portable (in concept or code) and non-portable elements are obvious candidates for layer separations. (In any event, non-portable code should always be isolated.) Interfaces to operating system services are also candidates -- we should be able to change operating systems and machines.

(3) Standards and methodologies for software architecture should include procedures for creating portable and reusable software.

(4) Inputs from experienced and knowledgeable people are critical to making successful choices. In order to keep current with technology, people should be rotated among centers and/or each center should establish a technology tracking/infusion organization.

ISSUE: (Chuck Lawson - AZ)

APPLICABILITY AND METHODOLOGY OF PORTABILITY AND TRANSFERABILITY FOR THE SPACE STATION

ESSENTIAL CONSIDERATIONS:

In typical projects with about 10 year durations it is commonly assumed that software will not be moved across different machines during its lifetime and thus portability is commonly not considered. When it happens that software must be moved this is typically very disruptive and expensive.

With a projected time span of 30 years for the space station it must be assumed that software developed on one machine will likely be moved to other machines, with perhaps significantly different architectures, during the life of the project. As a result, consideration should be given to portability and transferability of all software developed as a part of the Space Station Project.

Portability concerns are not distinct from language and methodology questions. Portability is one issue in the choice of a programming language and data file structures. A software development methodology must take portability into account.

The basic principle of portability is to express the source description of a computing process at as high a level as possible subject to the availability of processors that can transform that high level description to executable code on the relevant target machines, some of which may not be known at the time the source description is being written. Data files must also be designed to facilitate their transfer between different systems.

Since transformation processors are complex and expensive, and generally more so when the source description level is higher, there are tradeoffs that must be faced.

RECOMMENDATIONS:

Tradeoffs should be made on a wide range of options available in choosing (or inventing) the source description language and its associated transformation processors. Some examples are:

(a) Use an ANSI standardized source language such as FORTRAN or Ada. Then processors (compilers) and programming environments are relatively economically available in the marketplace. A drawback is the fixed domain of expression allowed by an externally

standardized language.

(b) Use of an extension of an ANSI standardized language with a supplementary preprocessor to transform the extended language to the standard language. The extended language could, for example, be a design language or a specification language. Some of the extensions could be for the specific purpose of aiding portability.

(c) Creation and support of a unique NASA language and set of processors. This approach provides the greatest potential to achieve the language of one's dreams but is very expensive to support over the long haul.

Specific support must be provided to programmers who are expected to compose portable source descriptions of software. This support must include courses, manuals, software tools for checking portability, etc. It must be remembered that programmers are not accustomed to being told that portability is one of their goals.

6.4. Languages

BACKGROUND:

There are basically 4 groups of languages which the SDE should support. They are languages for requirements and specification (such as PSL/PSA), for design (such as PDL), for development (such as HAL/S) and for special applications (such as GOAL). Only the languages for development will be addressed here. On the languages for development, the following issues need be considered:

1. Requirements
2. Usage
3. Heritage or reusability
4. Evolution
5. General & special purpose languages
6. Standardization
7. Assembly language
8. Tools
9. Multilingual environments
10. Distributed processing
11. Transportability
12. Lessons learned

ISSUE: (Ed Ng and Irene Falkenstein - AZ)

LANGUAGES FOR SOFTWARE DEVELOPMENT

ESSENTIAL CONSIDERATIONS:

1. Requirements

The Space Station Operations Working Group sponsored a study on high order languages for space station and ground support operations. ("High Order Languages", 2nd Level White Paper, Space Station Operations Working Group, July 1983. Study leader: Audrey Dorofee, NASA KSC.) In the study high-level requirements for the space station operations have been compiled, comparison criteria have been defined, and candidate languages have been described. A similar analysis of requirements is necessary for the development side. Some of the findings of the referenced study are applicable to development.

2. Use of Languages

In 1979 the Software Standardization Subcommittee of the NASA Intercenter Committee on ADP issued a report which included a survey of language use within the NASA family. ("Report of the Software Standardization Committee", NASA Intercenter Committee on ADP, June 1979.) The findings, not surprisingly, indicated that COBOL, FORTRAN and HAL/S were the three main high-order languages of predominant use within NASA. During the 5 years since, the languages C, PASCAL and PL/1 have probably made some gains in NASA usage, but there is no sign of a significant gain.

3. Software Heritage and Reusability

Since the Space Station Program anticipates a 3-decade life cycle, with at least 1 decade of coexistence with the Space Transportation System (STS), software heritage is required or desired both within Space Station and between the two projects. The long life of the Program imposes significant challenges on the evolution of the software.

4. Evolution of Languages

High-order languages have been in existence for about 3 decades. During this period they evolved very slowly, with breakthroughs few and far between. The well-known breakthroughs include the abstraction of expressions in the 50's (exemplified by FORTRAN), the abstraction of control structures in the 60's (exemplified by ALGOL), and the abstraction of data structures in the 70's (exemplified by PASCAL and Ada). Languages also take a long time to be standardized (Ada the major exception) and to mature. Moreover, even after standardization, non-standard implementations abound in practice. The implication to Space Station is that flexibility is an important factor in the planning. This

flexibility includes the construction of a strategy and bridging technology for linking the past to the future. One immediate example may be an Ada transition strategy which assumes the present use of FORTRAN and HAL/S with proper preparation to adopt Ada at the appropriate time.

5. General and Special Purpose Languages

The selection between special purpose languages and general purpose languages is a very controversial issue as old as computer languages and will not be addressed in detail here. The relation of this topic to space station issues on languages is very relevant. The software needs of the space station cover a broad range of application areas, from real-time on-board to ground support to simulation modeling to special applications such as GOAL. Each of these applications areas has special data processing requirements and the need for particular language features. Can there be one language general enough to meet the requirements of not only the ground and on-board systems but also the special applications? If so, what inefficiency costs does this incur and how big and complex is this general purpose language? If not, then where can the boundary reasonably be drawn and how many special purpose languages should be supported (as initial compiler/tool costs and long-term maintenance costs may increase substantially with multiple languages and environments)? This issue interrelates with standardization, support tools, and multi-languages issues, and should be addressed in that context.

6. Standardization

The advantages of language standardization include transportability, uniform training, ease of communication, software sharing, and reduction in redundant effort. On the other hand, language standards tend to stifle growth and versatility, and tend to introduce considerable language control effort. NASA's HAL/S and DoD Ada efforts are experiencing both the advantages and disadvantages.

7. Assembly Language

Assembly language is still a necessary evil, but should be kept to an absolute minimum. For special cases it gives the programmer almost complete control over what instructions the computer will execute, and how and when the computer will execute an instruction. Disadvantages to assembly languages are well-known:

- reduced productivity (3 to 5 times less on average)
- many compilers generate more efficient machine language code than the average assembly language programmer can write

especially on large applications

- assembly language is detailed and errors are more frequent and harder to find
- assembly language is not self-documenting, and the documentation is left up to the programmer
- assembly language programs are hard to read and comprehend
- assembly language is a non-structured language
- maintenance is difficult
- few tools exist for detecting errors in assembly language programs
- assembly language programs are difficult in terms of attempting to do V & V
- assembly language is machine and operating system dependent, which makes it difficult to transport
- assembly language is costly to write, debug, and maintain

8. Tools

The use of tools can aid in the development and management of software for Space Station, by reducing the cost and time to develop and manage software. The selection of the tools to be used is important, since the selection of incompatible tools can add to the complexity of the software and the time needed to develop the software. Some tools are more language and environment independent, while others are dependent on a specific language and/or software development environment. Some tools can be used for developing software on the host, while others are designed primarily for developing software on target machines.

9. Multilingual Environments

In developing software for Space Station, most likely several programming languages will be used. Realistically, no one language is optimal for all applications on all machines (host and target). Also, some machines (and operating systems) are only compatible with certain languages or a given version of a language. In selecting the language for an application (or module) one needs to consider memory (space) and/or time (efficiency) constraints, maintainability (availability of programmers with knowledge in a given language and ease of programming/debugging), as well as which language(s) are compatible with

the software/hardware/network architecture to be used.

10. Distributed Processing

At present, four levels of distribution can be perceived, viz., at the Agency level, at the Center level, at the system level, and at the user level. It is anticipated that Space Station Project activities will involve national and local networking, will use distributed system architectures, both on board and for ground support, and will witness near-universal user ownership of intelligent workstations. All this has significant implications to the selection and evolution of programming languages.

For the user level, advantages and disadvantages of distributed processing are:

Advantages --

- local user has access to own data, and can update data immediately
- better data security because local users do not have immediate access to data at other locations
- host computer has to do little editing of data which is mostly done by the local computers
- distributed processing system is often less expensive than a centralized system of similar power
- the local system can function even if the host machine goes down
- host has little or no effect on response time and throughput
- local users have more flexibility in using the local computer as they require
- improved level of service and response to local needs

Disadvantages --

- distributed processing is in its infancy, and there are not many experiences or experts
- higher risk that the data will be defined and used differently at each location, thus making a centralized data base not useful, and sharing of data meaningless
- local computers often have to be programmed in a lower-level language than the host machine because of time, memory, and performance constraints

- the local machine often has language constraints, e.g., must be programmed in a specific language, or the number of levels of subroutines (or programs) that can be performed (or called) within a given module (application) is limited

11. Transportability

This issue, discussed elsewhere in these proceedings, has significant implications on the choice of languages.

12. Lessons Learned

The NASA Office of Chief Engineer now sponsors an activity to gather and organize data on lessons learned about software management within the NASA family. Though the activity initially emphasizes only the management aspects, it may either grow to include more engineering information, or may catalyze a parallel activity to gather and organize engineering information, which is also important to managers.

RECOMMENDATIONS:

(1) On requirements, we should revisit the analysis presented by the Space Station Operations Working Group, and decide if we should generalize or extend that study. ("High Order Languages", 2nd Level White Paper, Space Station Operations Working Group, July 1983. Study leader: Audrey Dorofee, NASA KSC.)

(2) On standardization, NASA should promote the use of ANSI standards, avoiding dialectal proliferation.

(3) On language use, we should collect data about the development of STS to determine evolutionary applications At the same time, analysis should be performed to project revolutionary applications.

(4) On tools, we should establish the generic requirements of tools and determine availability. Guidelines need to be established for the selection and assessment of tools. Owing to the long life cycle of Space Station, tools cannot be entirely left up to the choice of the contractors because certain tools (such as configuration management) have impact on the long-term maintenance activity.

(5) NASA cannot afford to leave multi-language issues for contractors' choice. There is also a long-term maintenance implication. Fortunately here we are starting with a small list well-known to NASA, viz., FORTRAN, COBOL, HAL/S, C, PASCAL, PL/1, and Ada. As for Ada, it is too big a movement for NASA to ignore. JSC is currently performing an evaluation of Ada for space station applicability. It is desirable to have another (or more) organization(s) perform an assurance function and to plan a transition strategy as a complementary function to JSC.

(6) Assembly language should only be used if the application (or module) cannot be written in a recommended high-level language. Justification must be documented before assembly language can be used. The justification must contain the software/hardware/network and/or performance or other technical constraint(s) existing that warrant the use of assembly language. Such constraints include:

- critical time/space constraints
- efficiency constraints within a given module(s)
- disk controller interface
- special I/O interfaces
- block search and transfer
- task dispatching
- context switching, etc.

(7) A list of languages used on recent NASA space missions needs to be compiled. Each language listed needs to be evaluated and its advantages and disadvantages described. Special consideration should be placed on the following criteria:

- productivity
- maintainability (amount of errors, types of errors, time and cost of maintenance, etc.)
- self-documenting
- use of computer resources (time and cost)
- performance
- transportability
- reliability
- applications used on
- hardware used on (compatibility)
- problems with the language
- compatibility with a variety of environments
- compatibility with other languages

(8) An evaluation of distributed processing machines should be made. The evaluation should include the language and tools that can be used on that machine to develop software. Limitations of each language and operating system need to be identified. A list of computer languages and desirable language characteristics needs to be made available. The types of computers and operating systems should allow for the use of structured languages, preferably high-level languages. The types of computers should have minimum space (memory) and performance constraints. Applications should be evaluated to determine how many applications would be better performed on a distributed processing system.

(9) Three other papers are needed to address the other types of languages: i.e., the languages for requirements and specification, for design, and for special applications.

6.5. Documentation

BACKGROUND:

Software is defined as documentation plus code. In line with this definition, experience shows that a minimum set of documentation should be required for each piece of software to be developed in space station related projects. The level of detail for each required document may be based on project needs, project size, project cost, extent of effort, future usage and application, or other factors. Documentation requirements should be specified in the software management plan and in the statements of work signed by the contractor.

ISSUES: (Ai Fang)

(1) WHAT IS THE CRITICAL, MINIMAL SET OF DOCUMENTATION AND WHAT LEVEL OF DETAIL SHOULD BE SPECIFIED?

(2) DO THE CRITICAL SET OF DOCUMENTS AND LEVEL OF DETAIL VARY WITH SOFTWARE CATEGORY?

(3) WHAT ACCEPTANCE CRITERIA ARE NEEDED?

ESSENTIAL CONSIDERATIONS:

The number of required documents varies with the category (i.e., criticality) of the software. An excess will raise cost and may not be necessary. However, to prevent the occurrence of missing or inadequate documentation, the project manager should identify, by title and function, basic documents in the software management plan to govern the software specification including requirements, design, development, testing (methods, test data, test cases, test results), interface, validation, integration, verification, maintenance, operation, etc.

Emphasis must be placed on keeping documentation current with the code. Documents generated should be easy to read, understandable, and convenient to use. Major acceptance criteria may be:

a) Accuracy: The generated documents must contain no significant error and must meet the requirements and
b) Adequacy: The generated documents must present adequate description of the needed information.

Both accuracy and adequacy should be checked and confirmed. Any modification of documentation must be approved. It is convenient to generate documents on electronic media, and it is easier to maintain control and to update if all significant documentation data are stored in a data base.

RECOMMENDATIONS:

The minimum set of documentation for each major component of software is:

Operations Concept Document
Software Acquisition Management Plan
Software Requirements Spec
General Design Specs (includes architecture and high level interfaces)
Requirements Traceability Matrices
Test Plans
Test Specs (includes procedures, data, and analysis)
Data Structure Specs
User's Manual
Interface Specs
Detailed Component Specs
Test Results

7. INFORMATION SYSTEMS ISSUES

BACKGROUND:

A Space Station Project-wide mechanism to document, control, and disseminate program design data required by subsystem implementation efforts is needed. In evaluating software requirements at the subsystem implementation level, each software effort should be required to develop and maintain a list and schedule of supporting data needs to be provided by other elements in the Space Station Project. A project-level scheme to coordinate and track these needs is essential to the success of these contributing subsystems. A project-wide information system should provide the response information via a computerized mechanism, providing a single controlled source for all such data. Such information may range in content from documentation to actual data base sets used directly as

an input to the subsystem software (i.e., from design documentation to telemetry definition files, etc.) This information system will likely consist of a large body of general use space station design data as well as a range of specific data developed in direct response to identified subsystem needs.

ISSUE: (Joe Hennessy)

HOW IS SOFTWARE INFORMATION FLOW TO BE SUPPORTED, CONTROLLED, AND MANAGED?

RECOMMENDATION:

A centralized computer data base should be established to log implementation-level data needs and the formal response data to those needs. The centralized data base should include the following features and capabilities:

--what information is needed, and when, to implement each subsystem element
--automated index and information library
--accessibility to permit technology transfer to industry
--capability to integrate and incorporate related data bases and service

ISSUE: (Susan Voigt)

WHAT IS THE RELATIONSHIP BETWEEN THE SDE AND THE MANAGEMENT AND COMMUNICATION DATA SYSTEM?

BACKGROUND:

The Technical and Management Information System (TMIS), formerly known as the Management and Communication Data System (MCDS), will be implemented by NASA to support its Space Station Program. TMIS will be a program-wide electronic information system for SSP management and systems engineering, serving both NASA organizations and contractors.

TMIS will be implemented primarily with off-the-shelf hardware and software, as a distributed network of data processing nodes and intelligent work stations scattered throughout NASA. The first phase will provide an early basic operational capability; phase 2, ready for use in mid-1986, will increase the functionality and integration.

The objectives of TMIS are:

a) Provide an electronic means of data communications (project management data, engineering data, CAD drawings, text, engineering and business graphics, presentations, conferencing, mail, etc) between all elements of the Space Station Program (SSP).

b) Implement controlled data bases and data interchange mechanisms which will establish an effective/reduced paper environment for the management and integration of the program.

c) Provide program management, configuration control, security and other management/analysis and support tools required by the program.

d) Provide these services during all phases of the SSP so that they support the system design, integration, test, and operations, and so that duplication of capability between technical and management activities is minimized.

The major functions of TMIS will be:

a) Management of Program Information (budgets, schedules, resource tracking, configuration control, documentation, problem and action-item management)

b) Communications (documentation dissemination and interchange, electronic mail, office automation, interchange of engineering drawings and data, conference support and presentation interchange)

c) Engineering Data and Technical Computation Support (SE&I, CAD/CAM/CAE, engineering databases, engineering drawings library, engineering analysis tools)

ESSENTIAL CONSIDERATIONS:

The TMIS is intended to support the engineering and management of the Space Station Program. TMIS requirements are being developed by the Level B Space Station Office and they will sponsor its implementation. The software development environment (SDE), described elsewhere in this document, will support the specific needs of the software developers. Many of the facilities to be in the TMIS will be required by the software development teams, and wherever possible, the SDE users should use the TMIS for communication and software project management. Details of the interface and cooperation between the TMIS and the SDE should be determined and documented prior to selection of an SDE development contractor.

RECOMMENDATION:

Once the SDE has been better defined, it should be discussed with the Level B TMIS developers to establish the interface between SDE and TMIS and the support facilities to be provided by TMIS.

Some of the TMIS facilities are also called for in the SDE (and hence should be provided by the TMIS to avoid duplication of effort). These include

Data Management of software information, documentation, code, test data, etc.

Document Management and Distribution

Management Analysis

Teleconferencing

Technical Workstations

Communications

8. FUTURE OF THE SWWG ISSUE

ISSUE: (John McLeod)

WHAT SHOULD BE THE FUTURE ROLE OF THE SPACE STATION SOFTWARE WORKING GROUP?

BACKGROUND:

The Space Station Software Working Group (SWWG) was begun in 1983 as a subcommittee of the Data Management Working Group. Its purpose has been to identify software issues related to Space Station and provide advice to the Space Station Technology Steering Committee on software technical and management concerns. Now that the Space Station Program has been formally started, many of the other advisory committees that were similar in function to the SWWG have phased down their activities or have been absorbed into the formal program structure. The SWWG, in contrast, has become more active and has served to highlight software issues such as software management, programming languages, development environments, user control languages, etc.

The SWWG is the only group specifically addressing software issues related to the Space Station. It brings together software expertise and experience from all NASA centers. The members presently serve on an informal basis; their SWWG activities are not part of their job responsibilities. As the Space Station Program develops, it will take more and more effort on the part of the SWWG members to adequately evaluate software issues and make recommendations to Program management. The time has come to evaluate what role the SWWG should have in the Space Station Program, what resources will be required to fulfill the role, and how these resources can be supplied.

RECOMMENDATIONS:

(1) The SWWG should have a formal role in the Space Station Program. Software is so critical to the success of the Program that we cannot afford to ignore software issues until we have problems.

(2) The SWWG could serve as the primary technical resource for the Level A and B software managers.

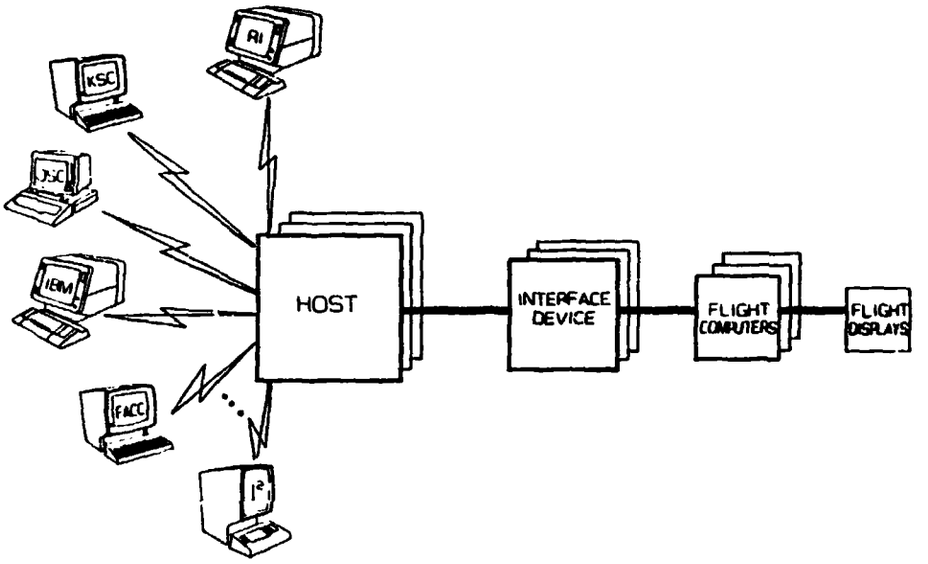
(3) The membership in the SWWG should be stabilized so that long-term software issues can be considered in light of a shared experience base and to avoid having to constantly familiarize new members with past history before issues can be addressed.

(4) Adequate funding should be provided so that members of the SWWG can spend the time required to evaluate the software issues that arise without compromising their normal job functions.

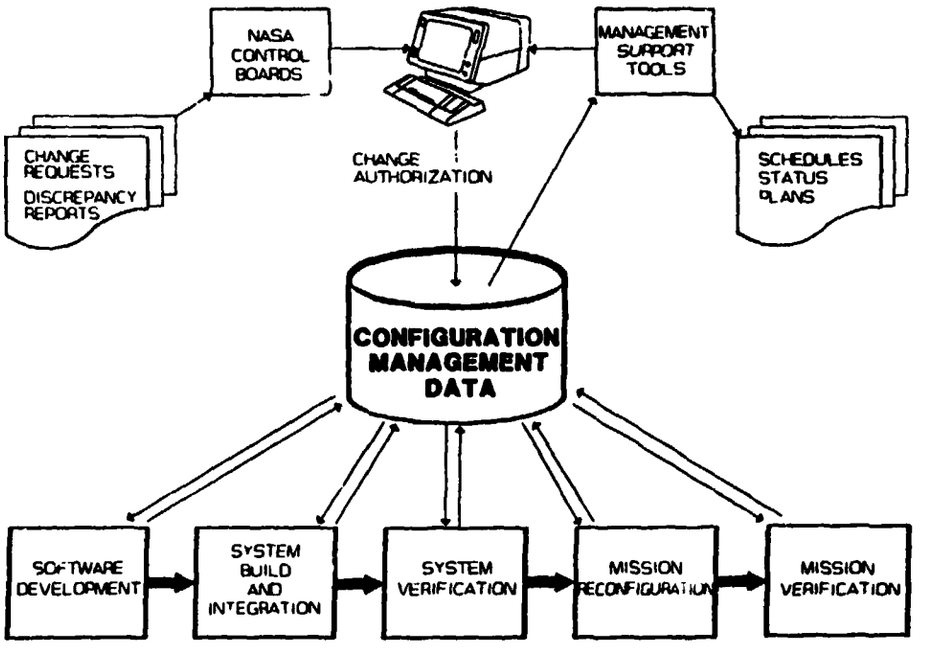
APPENDIX A

SHUTTLE SOFTWARE PRODUCTION FACILITY

Appendix A



SPF HARDWARE OVERVIEW



SPF PROCESS OVERVIEW

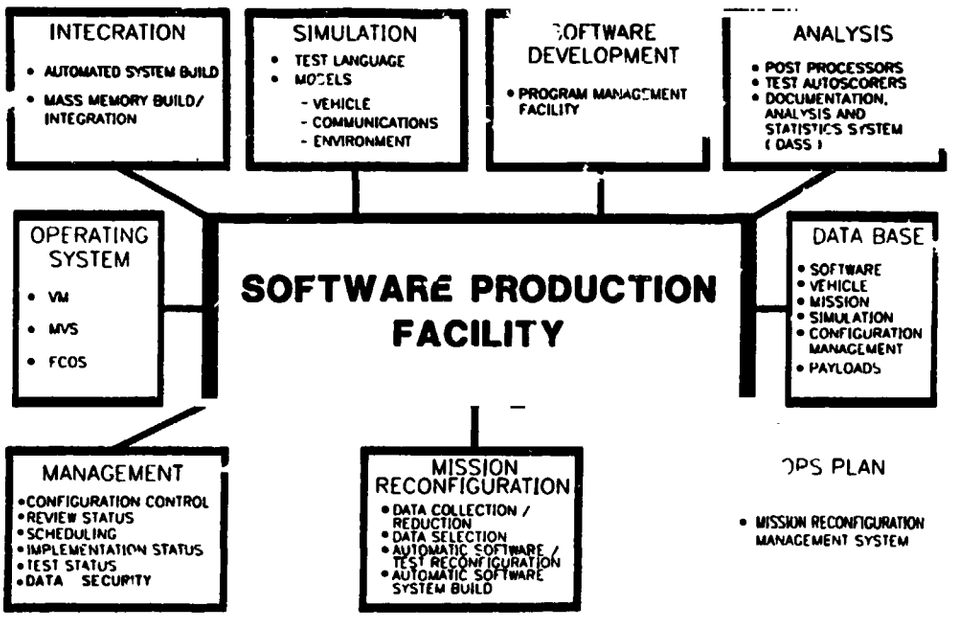
Appendix A

SPF HARDWARE

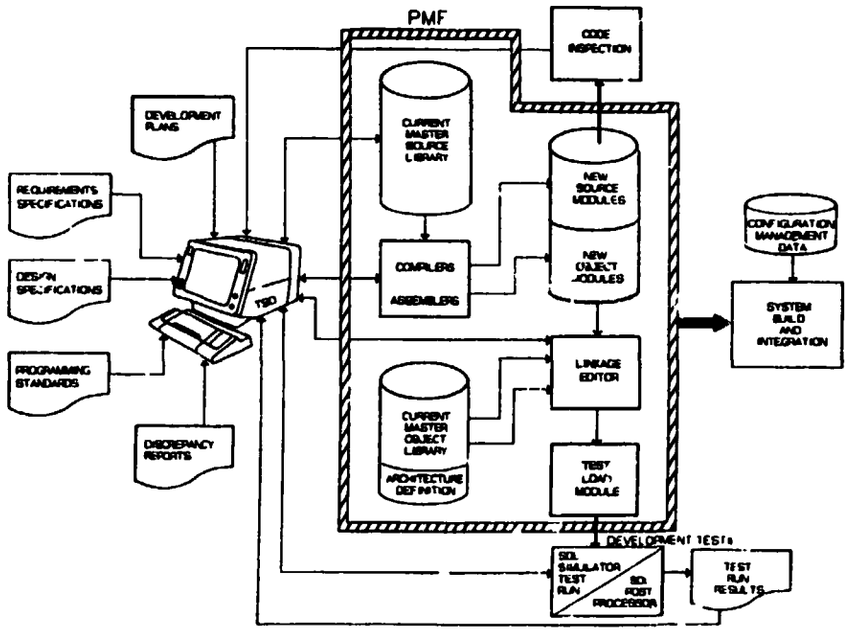
- SUMMARY
  - CPU: 28 MIPS, 64 MB
  - DASD: 152 GB
  - TAPE DRIVES: 43
  - PRINTERS: 47,000 LPM (+REMOTE)
  - TERMINALS: 58 LOCAL, 369 REMOTE
- TERMINALS
  - 54 LOCAL TERMINALS
    - "RED" ROOM (21)
    - "BLACK" ROOM (33)
  - 369 REMOTE TERMINALS
    - JSC (74)
    - IBM (139)
    - FORD AEROSPACE AND COMMUNICATIONS CORP. (62)
    - COMPUTER SCIENCE COPR. (15)
    - INTERMETRICS INCORPORATED (2)
    - KSC (13)
    - GODDARD (2)
    - MARSHALL (2)
    - BARRIOS (6)
    - MACDAC (11)
    - HUNTINGTON BEACH (4)
    - ROCKWELL INTERNATIONAL/DOWNEY (32)
    - MITRE (7)
- HOST PROCESSORS
  - 3033-U16 (1 RED ROOM, 1 BLACK ROOM)
    - 16 CHANNELS
    - 16 MEGABYTES
    - 6 MIPS
  - 3081-K32 (1 BLACK ROOM)
    - 16 CHANNELS
    - 32 MEGABYTES
    - 14 MIPS

SPF SOFTWARE

- SOFTWARE PRODUCTION FACILITY CONSISTS OF A NUMBER OF TOOLS (COMMERCIAL AND CUSTOMIZED)
- THESE TOOLS SUPPORT A NUMBER OF FUNCTIONS
  - SOFTWARE DEVELOPMENT
  - INTEGRATION
  - SIMULATION
  - ANALYSIS
  - OPERATING SYSTEM
  - DATA BASE
  - MISSION RECONFIGURATION
  - OPS PLANNING
  - MANAGEMENT
- DETAILED PRESENTATION WILL DISCUSS THESE SETS OF TOOLS EXCEPT FOR THOSE WHICH PERVADE ALL FUNCTIONS (OPERATING SYSTEM, DATA BASE, AND MANAGEMENT)



SOFTWARE OVERVIEW



SOFTWARE DEVELOPMENT (FSW/SPF)

APPENDIX B

SOFTWARE TECHNOLOGY WITHIN NASA

Appendix B

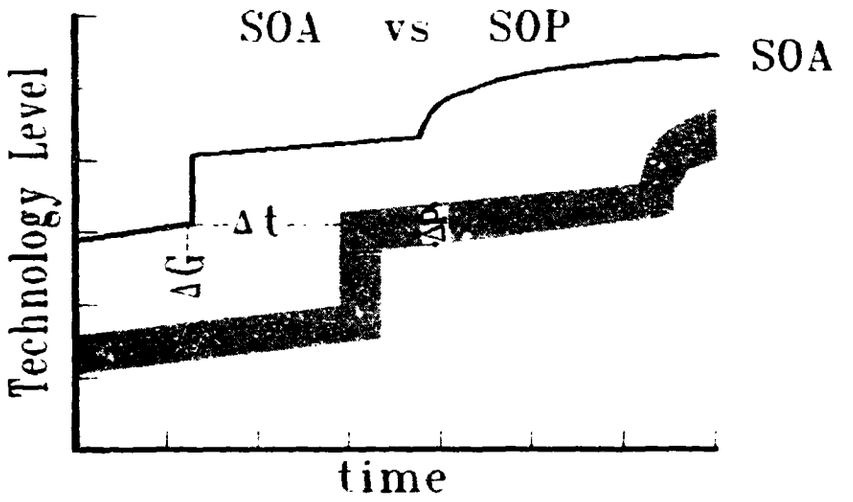
SOFTWARE TECHNOLOGY WITHIN NASA

- State Of Practice (SOP)

vs.

- State Of Art (SOA)

NASA SOFTWARE TECHNOLOGY



Δ t = time lag for technology transfer
Δ G = Gap between SOP and SOA
Δ p = Variance in *Practice* of Software Technology

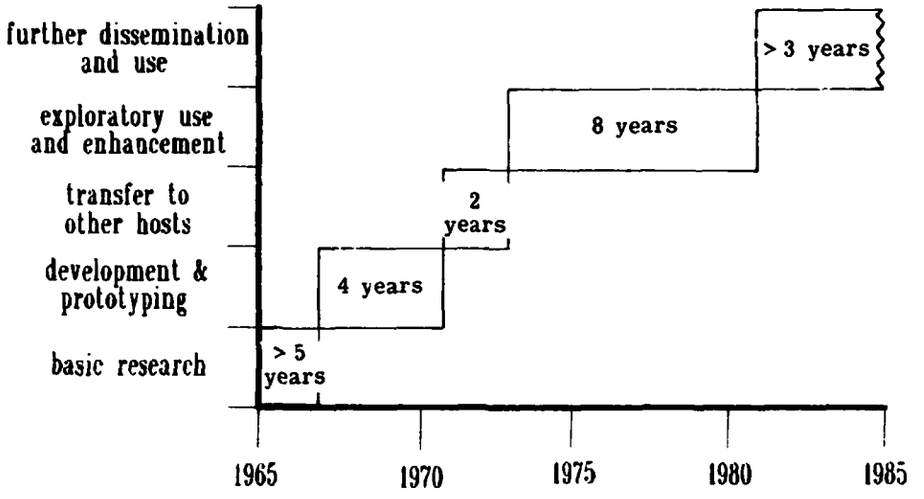
Appendix B

POINTS TO BE ADDRESSED

1. How big is Δt (and $\Delta G - \Delta p$) ?
2. For NASA what is SOP ?
3. What currently is SOA ?
4. Why is (SOA - SOP $\neq$ 0) ?
5. What is the ● ?

UNIX Technology Maturation Time

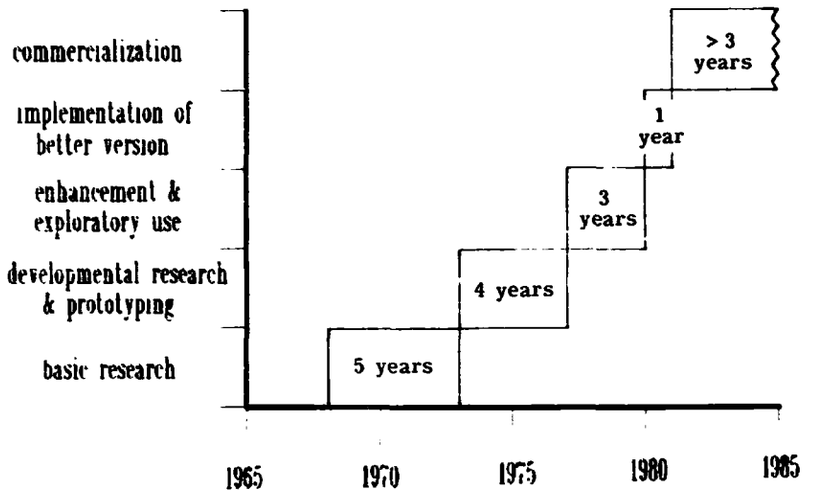
(See ref. 1)



Appendix B

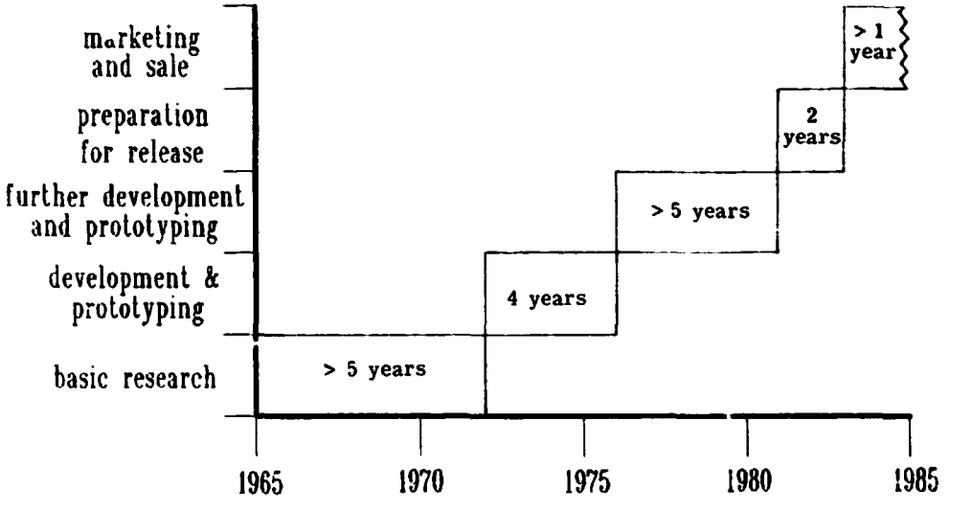
SREM Technology Maturation Time

(See ref. 1)



Smalltalk-80 Technology Maturation Time

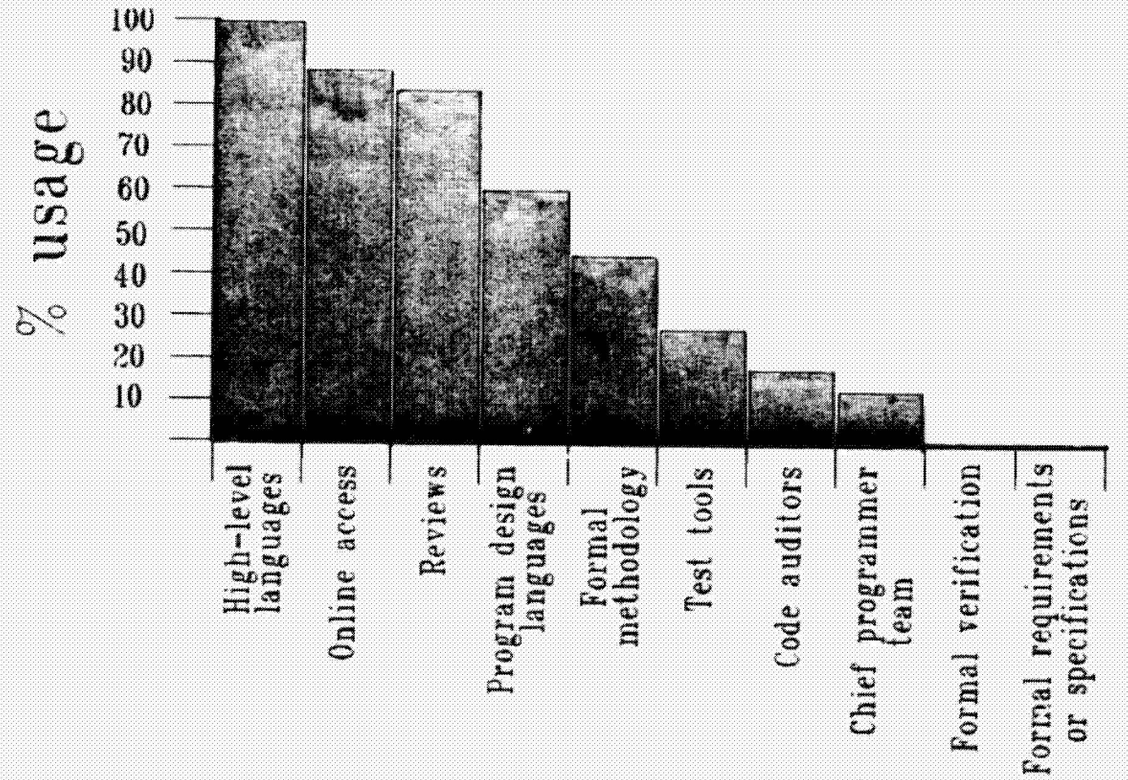
(See ref. 1)



What is Δp ?

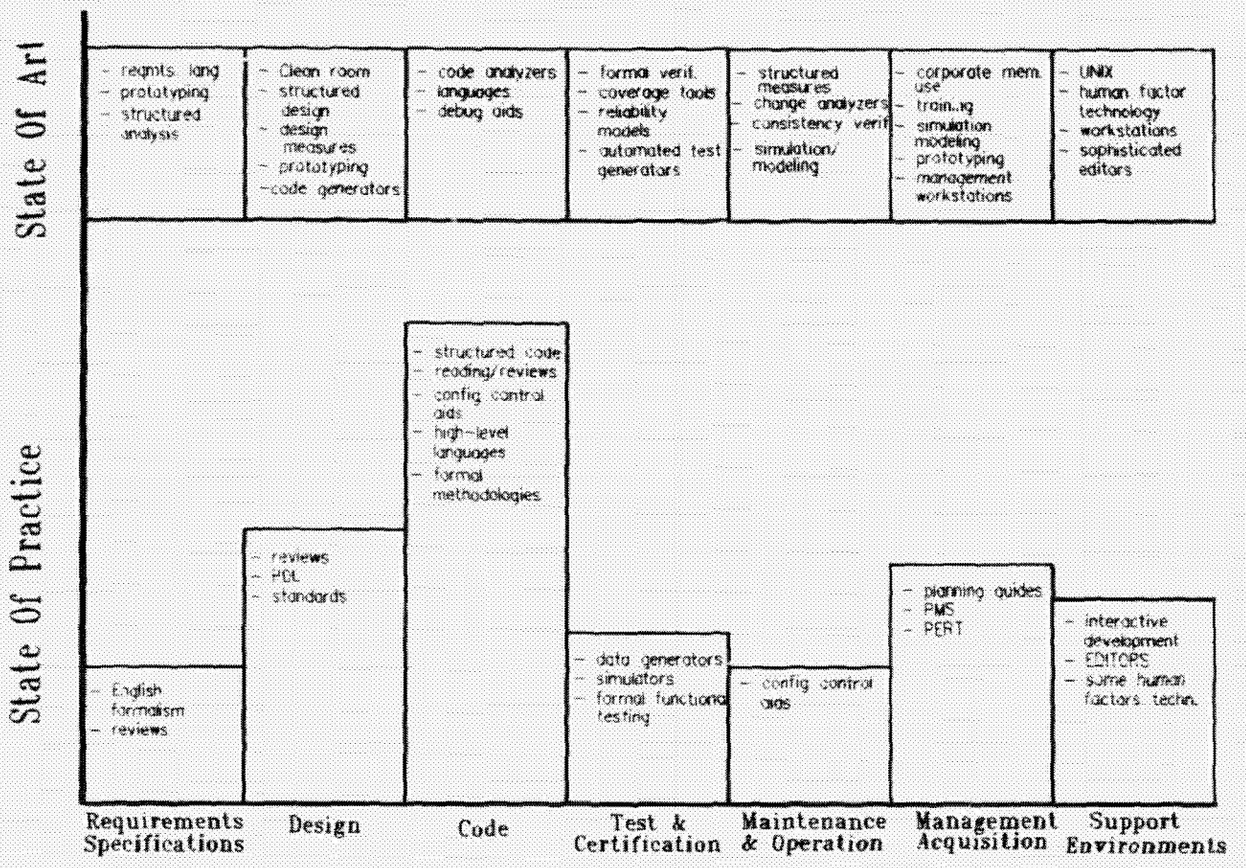
Some Indicators from Major Software Industry

Practices Currently Used *

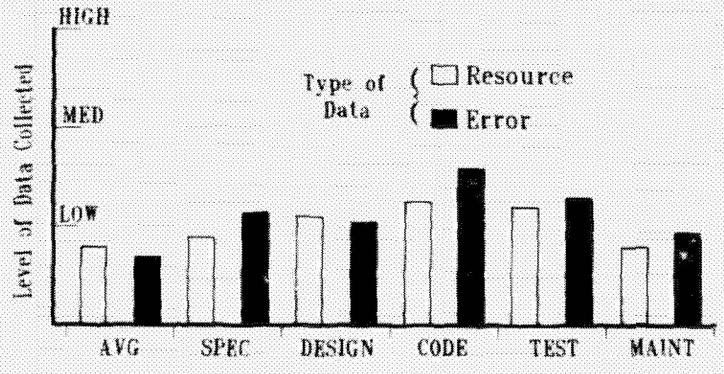


*Data from ref. 2.

Appendix B



Software Development Data Collection *



- o 'All installations collect some data'
- o 'Little data becomes part of corporate memory'
- o 'Data is rarely evaluated'
- o Cannot use data accross installations (terminology)

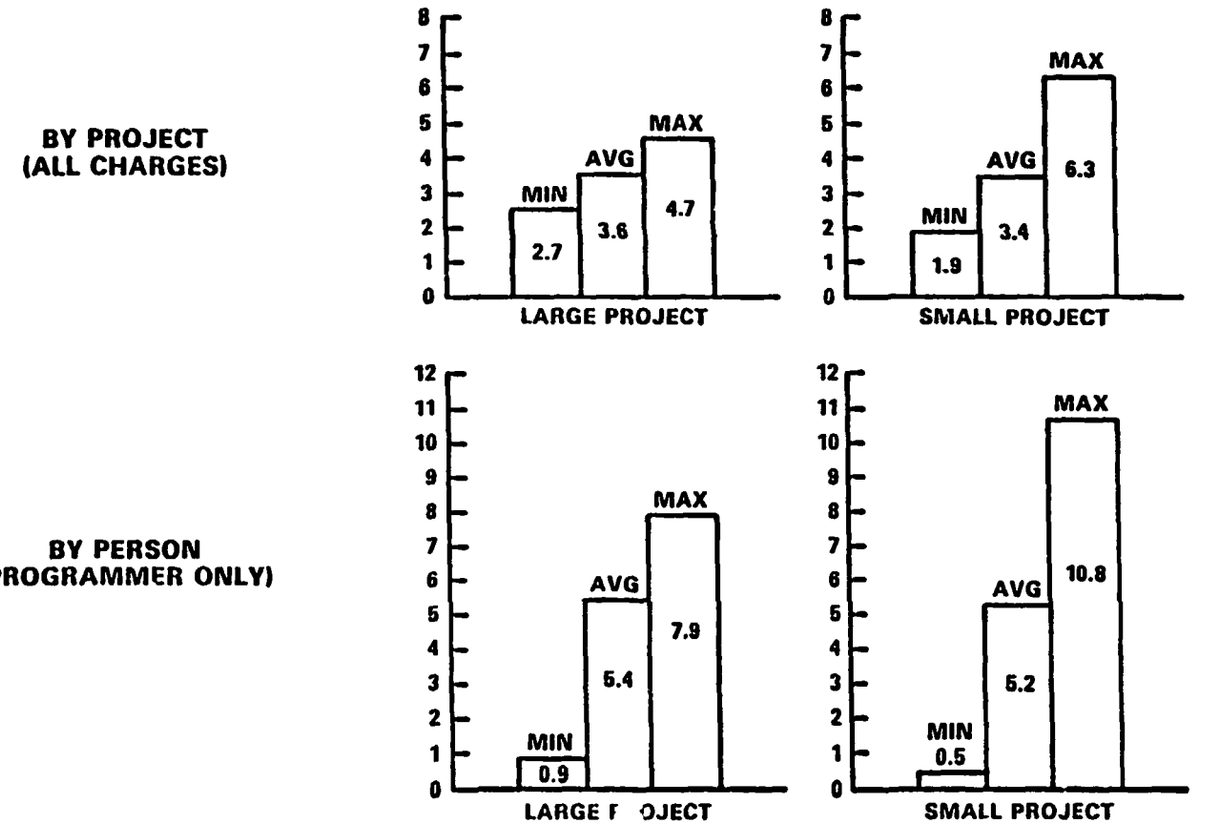
* Data from ref. 2.

Appendix B

Measuring Impact Of Software Technology

Some Experiences
at
NASA/GSFC

PRODUCTIVITY VARIATION (SLOC/HOUR)[1]



PEOPLE ARE THE MOST IMPORTANT METHODOLOGY

[1] **A LARGE PROJECT IS GREATER THAN 20K SLOC.**

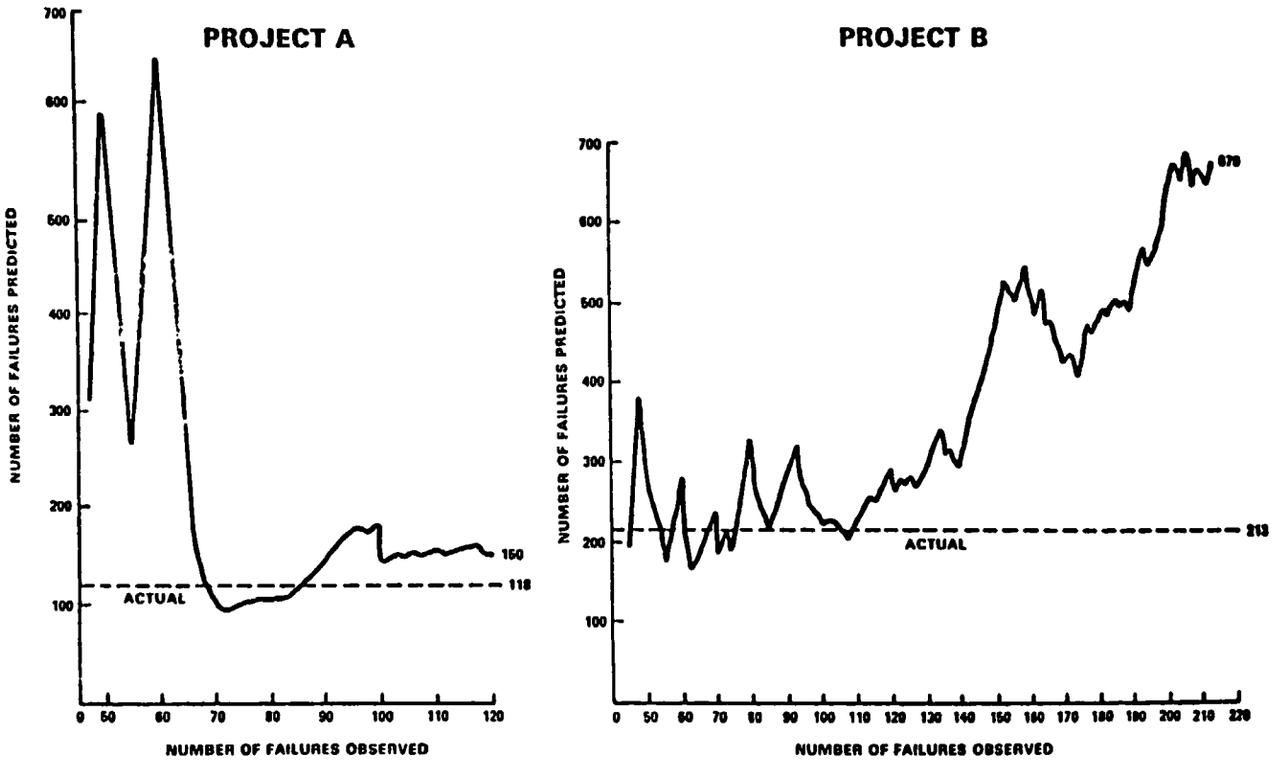
Appendix B

COMPARISON OF COST MODELS

PROJECT	ACTUAL EFFORT (MM)	PERCENTAGE OF ERROR IN PREDICTION				
		DOTY	PRICE S3	TECOLOTE	SEL	COCOMO
1	79	+65	+8	-4	-6	—
2	96	+30	+6	-25	-22	+1
3	40	+65	+6	-8	+93	—
5	98	+74	0	+3	-2	+2
6	116	+123	+36	+35	-3	—
7	91	+52	+14	-12	-14	—
8	99	+127	+7	+36	+14	+53
9	106	—	—	—	-24	+16

SOMETIMES, SOME MODELS WORK WELL

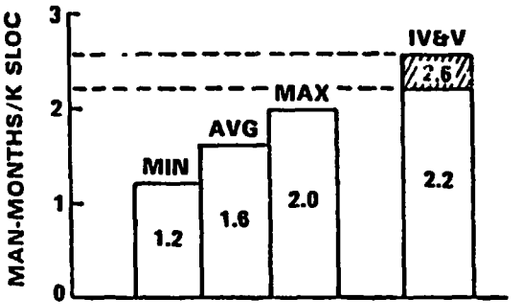
PREDICTING RELIABILITY
(MUSA MAXIMUM LIKELIHOOD METHOD)



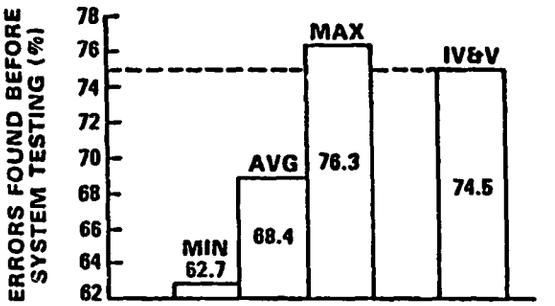
WE DON'T KNOW ENOUGH ABOUT RELIABILITY MODELS

A LOOK AT IV&V METHODOLOGY

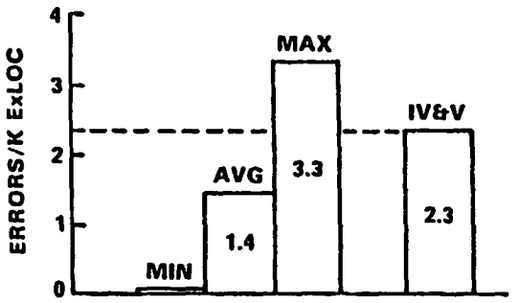
(BASED ON RESULTS FROM 3 EXPERIMENTS)



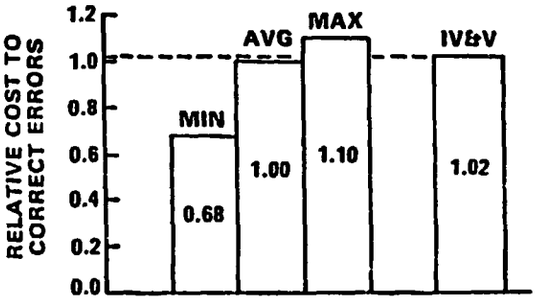
- COST INCREASED



- MORE ERRORS FOUND EARLY



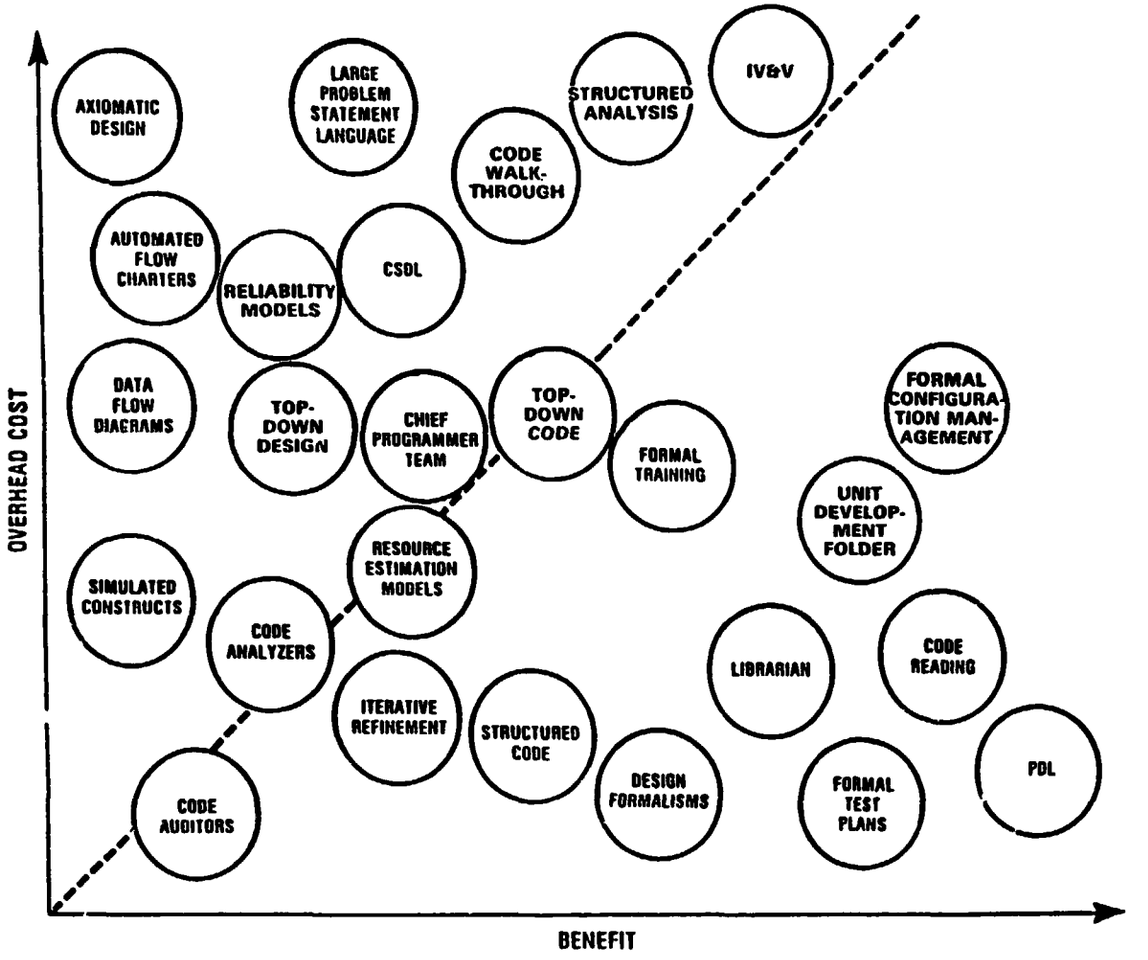
- RELIABILITY NOT IMPROVED



- ERROR CORRECTION COST NOT DIFFERENT

- IF YOU MULTIPLY ERRORS FOUND EARLY BY A LATENCY FACTOR, IV&V LOOKS GOOD
- IF YOU EXAMINE ALL MEASURES, IV&V LOOKS BAD

WHAT HAS BEEN SUCCESSFUL IN ONE ENVIRONMENT?



What's the o

1. Δt is BIG !
2. Define where we are (SOP) before going someplace new !
3. Not all 'New Technology' is 'SOA' !
4. Define issues/problems before choosing 'SOA' !
   (Right practices for right problems)

Appendix B

REFERENCES

1. Riddle, William E., The Magic Number Eighteen Plus or Minus Three: A Study of Software Technology Maturation. ACM SIGSOFT Software Engineering Notes, vol. 9, no. 2, Apr. 1984, pp. 21-37.

2. Zelkowitz, V. M., R. T. Yeh, R. G. Hamlet, J. D. Gannon, V.R. Basili, Software Engineering Practices in the US and Japan. Computer, vol. 17, no. 6, June 1984, pp. 57-66.