N88-15610

59-61

116711

43P

1987

NASA/ASEE SUMMER FACULTY RESEARCH FELLOWSHIP PROGRAM

Marshall Flight Space Center
The University of Alabama in Huntsville

ENHANCEMENT OF COMPUTER SYSTEM FOR APPLICATIONS SOFTWARE BRANCH

Prepared by:	Alex Bykat
Academic Rank:	Professor
Affiliation:	University of Tennessee at Chattanooga Center of Excellence - Computer Applications
NASA/MSFC:	
Laboratory:	Information and Electronic Systems
Division:	Software and Data Management
Branch:	Applications Software
NASA Colleague:	Delano R. Hyter
Date:	August 5, 1987
Contract No:	The University of Alabama in Huntsville NGT-01-008-021

ENHANCEMENT OF COMPUTER SYSTEM FOR APPLICATIONS SOFTWARE BRANCH

Alex Bykat
Professor of Computer Science
Center of Excellence - Computer Applications
University of Tennessee
Chattanooga, TN

ABSTRACT

This report presents a compilation of the history of a two month project concerned with a survey, evaluation and specification of a new computer system for the Applications Software Branch of the Software and Data Management Division of Information and Electronic Systems Laboratory of Marshall Space Flight Center, NASA.

Information gathering consisted of discussions and survey of branch activities, evaluation computer manufacturers literature, and presentations by vendors. Information gathering was followed by evaluation of their systems. The criteria for the latter were: the (tentative) architecture selected for the new system, type of network architecture supported, software tools, and to some extent the price.

The information received from the vendors, as well as additional research, lead to detailed design of a suitable system. This design included considerations of hardware and software environments as well as personnel issues such as training. Design of the system culminated in recommendation for a new computing system for the Branch.

ACKNOWLEDGEMENTS

I would like to express my appreciation of the hospitality extended during my tenure at Marshall Flight Space Center. I am quite impressed with the efforts of both, Ernestine Cothran, NASA/University Relations and Gerald Karr - NASA-ASEE Program Coordinator, to provide a meaningful and all-round experience during the program.

My thanks and appreciation go also to Mr. Ray H. Craft and Mr. Delano R. Hyter for, not only providing a cordial and productive atmosphere, but also for their efforts in providing the opportunity to participate in their program.

Finally, my thanks go to Dr. G.R. Wallace, Mr. J.H. Lucas, and Mr. D.J. Aichele, for supporting and helping to identify sources for further cooperation with MFSC.

CONTENTS

1. Introduction. 6

2. Planning. 8
   2.1. Project activities. 8
   2.2. Resource allocation. 9
   2.3. Resources available and needed. 9

3. Project specification. 10
   3.1. General discussion. 10
   3.2. Project specification. 14

4. System survey and analysis. 14
   4.1. Current system. 14
   4.2. Survey. 15
   4.3. Analysis. 16

5. Baseline design. 17
   5.1. Local Area Network (LAN). 17
   5.2. Towards the future. 19
   5.3. Application development software. 21
   5.4. Software for embedded systems. 21

6. Information gathering. 22
   6.1. Operating system. 22
      6.1.1. VMS. 22
      6.1.2. UNIX. 23
   6.2. Systems reviewed. 25

7. Detailed design. 27
   7.1. Personnel. 27
   7.2. Hardware. 27
   7.3. Software. 30
      7.3.1. General management. 30
      7.3.2. Project management. 30
      7.3.3. Software development. 30
      7.3.4. Document preparation. 32
      7.3.5. Training tools. 32

8. Conclusion. 33

9. Appendix 1: Software development life cycle. 35

10. References. 37

11. Glossary. 38

12. Index. 41

FIGURES

Fig. 2.1.1. Project stages: Define, Design. 8
Fig. 2.1.2. Project stages: Accept, Install. 8
Fig. 3.1.2.1. Software Engineering functions. 12
Fig. 5.1.1. Applications Software Branch computing system architecture. 19
Fig. 6.1.2.1. Unix structure. 24
Fig. 7.2.1. Applications Software Branch computing system architecture. 29

1. Introduction.

This report presents a compilation of the history of a two month project concerned with a survey, evaluation and specification of a new computer system for the Applications Software Branch of the Software and Data Management Division of Information and Electronic Systems Laboratory of Marshall Space Flight Center, NASA. The project was allocated to me within the framework of the Summer Research Faculty Program conducted between NASA and various universities.

The general philosophy that I adopted for execution of this project follows a typical project management pattern; the main difference is in human resources involved. In this project both aspects, the project management and project development are executed by one and the same person, namely me.

Thus, one constraint in this project is the size of personnel: one person. Another definite constraint was related to the framework of the Summer Program and to the first constraint: time available was somewhat less than two man-months.

As it should be with any project, I started the project by planning the general path that the project would follow. This planning phase included identification of project activities, resources available, and resource allocation.

The planning phase was followed by project analysis and culminated with detailed specification of the requirements of the project.

The first two phases were then followed with a survey and analysis of the existing computer system. This phase consisted mainly of interviews of the branch personnel, discussions of their past and present projects, design of survey forms and collection and evaluation of the survey information. (The survey form is included in [Bykat, 1987].)

The next stage, system analysis - analysis of information gathered in the two prior stages - resulted in some interesting data concerning the resources used in past projects. For example, the survey revealed that whereas 92% write and compile their code, only 62% use debugging tools and only 23% use software monitors to evaluate the efficiency of the code. (The survey results are presented in section 4.2.)

The problem definition stage yielded sufficient information to formulate the base design of the system. Essentially, it became clear that a distributed system based on networked workstations, and supported with centralized file system would suit and benefit development of projects dealt with by the branch.

Thus the design stage of the project was initiated. I started by collection and selection of information of some leading commercial system vendors which support computing functions prevalent in the branch. This stage consisted mainly of collection and review of the vendors data describing their offerings. A number of system were reviewed - they are listed in section 6.2.

Information gathering was followed by information consolidation. Here, the vendors offerings were re-evaluated and only a few were "short listed". The criteria were: the (tentative) architecture selected for the new system, type of LAN supported, software tools, and to some extent the price. From this short list, five vendors were contacted. The requirements of the system were presented to the vendors (see section 3.2) and detailed discussion followed. The vendors were requested to prepare a presentation of their proposed systems.

The information received from the vendors, as well as additional research, lead to detailed design stage of the system. This design included considerations of hardware and software environments as well as personnel issues such as training. A draft of this paper was prepared and distributed to branch management and selected technical personnel for comments.

The input from reactions to the draft paper was then reviewed, evaluated and incorporated. The final recommendation was then prepared and the project was concluded. The recommendation calls for a distributed heterogeneous system of multivendor hardware, interconnected with a local area network, and providing an integrated software development environment.

The following sections provide detailed account of the various stages of the project, and specify some of the considerations that have lead to the final recommendation.

2. Planning.

2.1. Project activities.

```
         Define Phase                                  Design Phase
"                                "                                  "
"--------------------------------"                                  "
"project planning                "                                  "
"--.-----------.-----------------"                                  "
"  :project spec.:               "                                  "
"  :-----.-------:-----.         "                                  "
"        :system survey:         "                                  "
"        :-------.-----:---------"                                  "
"                :system analysis"                                  "
"                :----------.----"---------.                        "
"                .          : base design :                         "
"                .          :-----.-------:----.                    "
"                .                ": info gather:                   "
"                .                ":----------.-:-----------.       "
"                .                "         :info consolidate:      "
"                .                "         :---.-----------:-----"
"                .                "         .   :detailed design  "
"================.================"=========.===:=================.
                 .                          .                     .
milestones:     (1)                        (2)                   (3)
```

Fig. 2.1.1. Project stages: Define, Design.

```
         Accept Phase                                  Install Phase
"                                "                                  "
"----------------.---------------"                                  "
"  presentation  :               "                                  "
"------.---------:--.            "                                  "
"      : critique :              "                                  "
"      :-------.--:--------------"                                  "
"              : modification    "                                  "
"              :------------.----"-------------------.              "
"                           :      final report      :              "
"                           :-------------.----------:--------------"
"                                "        : further work            "
"================================.========:===========.=============="
                                 .                     .
milestones:                     (4)                   (5)
```

Fig. 2.1.2. Project stages: Accept, Install.

Milestones.

(1) specification written and accepted.
    document: "Specification statement".
(2) baseline design completed.
    document: "System design specification".
(3) detailed design completed.
    document: "Proposed system".
(4) detailed system design accepted.
    document: modified "Specification statement".
(5) summer project concluded.
    document: final report.
(6) further work considerations.

2.2. Resource allocation.

	weight	days
Definition		
Problem specification	1	1.5
Survey of current system	4	6
Analysis	2	3
Planning	1	1.5
Design		
Baseline design	1	1.5
Information gathering	4	6
Information consolidation	6	9
Detailed design	2	3
Acceptance		
Presentation	3	4.5
Critique	1	1.5
Modification	2	3
Installation		
Report	3	4.5
Follow-Up		
Further work	1	1.5
Continuation	?	?

2.3. Resources available and needed.

Time (9 wks)	45
seminars and other activities	-5
Total time available:	40

3. Project specification.

3.1. General discussion.

Initial definition of the project:

> Survey, evaluate and recommend systems and tools to support software engineering functions of the Applications Software Branch which includes requirements through sustaining engineering. [Hyter, 1987]

Discussion:

To evaluate the existing system and to propose a new system the above specification needs to be refined. In particular, the following questions need further explanation:

(1) What type of applications does the branch engage in?
(2) What software engineering functions does the branch engage in'
(3) What type of capabilities have to be supported?
(4) What type of tools are needed to support these capabilities?
(5) What type of system is suitable for this branch?

Discussion of each of these questions now follows.

3.1.1. Type of applications:

Applications can be classified into a number of general areas. These include information processing (scientific and commercial), process control, and real time processing. Below, I characterize briefly some of these areas:

Information processing

The salient features of this type of application are the formatting of input data by people and formatting of output for human 'consumption'.

Examples are   engineering calculations, information retrieval etc.

Various types of software fall under this class, and in particular:

System software
editors, compilers, interpreters..

Real-time software
interaction with 'real world' events:
data gathering, device monitoring, ...

Business software
payroll, inventory...

Scientific software
large amount of numerical computation:
engineering calculations, CAE,
system simulation...

Symbolic manipulation software
artificial intelligence,
pattern recognition...

Process control
The salient features of this type of application are the formatting of input data by machine (and possibly people) and formatting of output for machine consumption.

Examples are	instrumentation control, real time data gathering, automated manufacturing, etc.

Real-time software
The salient feature of this type of application is interaction with 'real world' events. Frequently these applications reside on processors packaged within a product.

Examples are	data gathering, device monitoring, on-board flight control; weapon systems, etc.

After discussions with management and members of the Applications Software Branch, the following supplementary information was received.

3.1.2. Software engineering functions:

Typical software engineering functions consist of the following stages [see also Miller, 1979]:

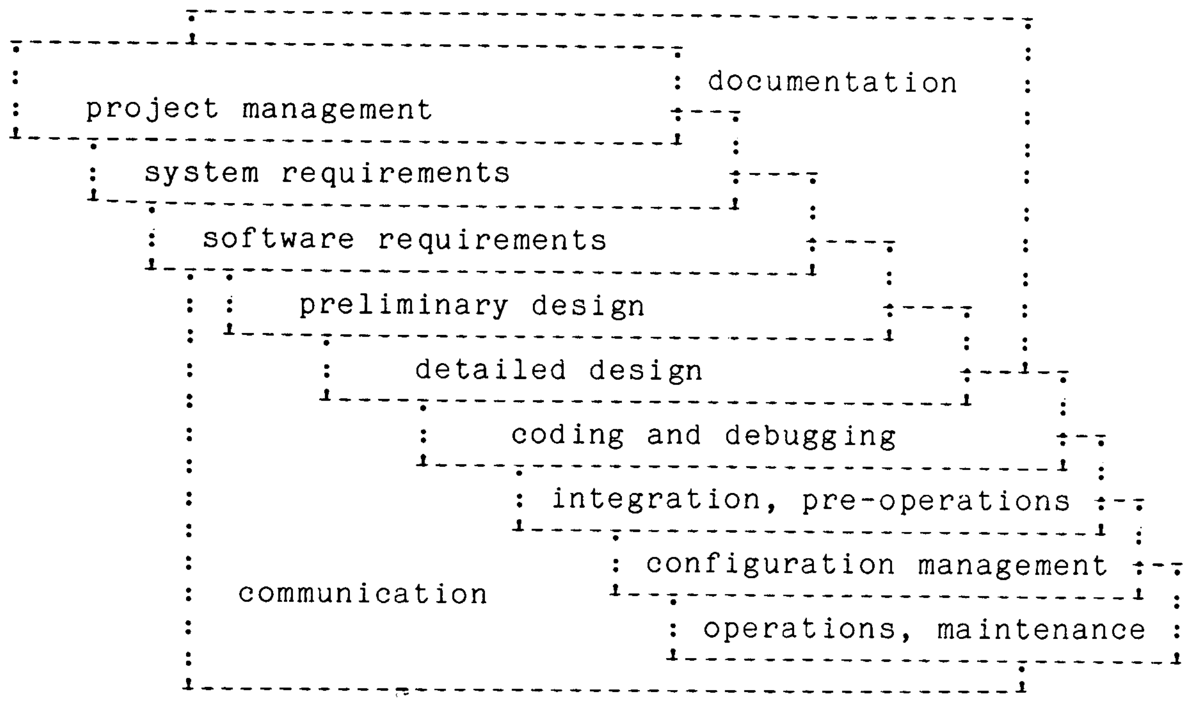

Fig. 3.1.2.1. Software Engineering functions.

The Applications Software Branch projects fall within these functions. Generally, the functions of this branch can be described as [Lucas, 1986]:

Application software development and sustaining engineering;
Support software management and development.

3.1.3. System capabilities required:

Efficiency in numerical computation;
Interactive graphics display: both input and output;
Fair amount of real time processing;
Data acquisition from sensors;
General purpose computation.

Languages used:

predominant:	Fortran
occasional:	assembly, Pascal, PL/M

In addition to above, it is perceived that in coming applications ADA will be used as the major language of this branch.

Operating systems used:

VMS, INTEL360, RT-11, RSX-11

3.1.4. Tools needed:

System capabilities/tools considered should include:

Simulators, emulators, logic analyzers, debug and diagnostic aids;
Computer aided requirements analysis, design, code generation, test generation and documentation;
Project management and configuration control;
Cost, training, productivity, quality, and flexibility to support a variety of projects;
Networks. [Hyter, 1987].

System architecture:

Two major types of computer system architecture that have to be considered are:
a) centralized, and
b) distributed.
Further, a decision is to be reached on the processor architecture:
a) uniprocessor,
b) multiprocessor, or
c) parallel processor.

Scope of the system:

budget:	$600,000
personnel size:	25

Typical hardware engineering questions need to be answered:
class of hardware that best addresses functions to be supported,
commercial availability of such hardware and its cost,
type of interfacing required,
what constraints are present (size, environment...).

Information collected in answering the above questions lead to a refinement of the project specification presented in the next section.

3.2. Project specification.

Narrative.

Survey, evaluate and recommend a computing system consisting of appropriate hardware and software tools to support software engineering functions of the Applications Software Branch.

The software tools will form
(a) integrated project management environment
(b) integrated program development environment.

The system is to be capable of sustaining development of applications which exhibit:
- high numeric computation component,
- interactive color graphics manipulations,
- real time data acquisition from sensors,
- real time control, and
- general purpose computation.

The goal of the system is to provide a software environment for activities which span software design and implementation life cycle, and in particular:

project planning,	specification design,
design implementation,	verification, and
installation	

of software products.

Languages supported by the system will include FORTRAN and ADA, and possibly C and a good Prolog environment for use in rapid prototyping of projects.

An immediate expected benefit of the selected system is to support and promote high productivity both in software project development activities and implementation of these projects.

Computer characteristics.

A general commercial system is required. The system must be able to support at least 25 users running a mix of document preparation, telecommunications, project management, and code development software.

Hardware interfaces.

The system must support standard serial and parallel ports, and a LAN interface (Ethernet).

Software functions.

Software should include a project management environment, code development environment, document preparation tools and telecommunication tools.

4. System survey and analysis.

4.1. Current system.

Currently the computer system serving the Applications Software Branch is organized into a number of stand alone and time sharing centralized computing systems. These systems include:

INTEL 8080 -	A stand alone computer operating under ISIS II operating system. Languages supported are Assembler, FORTRAN-80, PL/M. Also supported is ICE-80 product development tool for 8080 CPU. This computer is utilized mainly for the MEA project.
PDP-11/23, PDP-11/73 -	A stand alone computer operating under RT-11 and RSX-11 operating system. Languages supported are FORTRAN. The PDP-11/23 is utilized mainly for the BATSE project.
VAX 11/7xx, mVAX II -	A number of timeshared dedicated computers operating under VMS operating system. Languages supported are FORTRAN, Pascal, PL/1.These computers are used for the HOSC project, the POCC project and the Space Telescope project.

The mode of operation of these computers is predominantly development of the software in a high level language, compilation and subsequent transfer (if necessary) of the object code to the target computer. Some projects were written in an assembly language and subsequently assembled and ported to a target computer.

Project groups within the branch are, typically, small and consist of 3-8 programmers. Although groups work on independent projects, some groups are divided into a number of teams. Further, project groups frequently work on programs which are targeted for different types of hardware. Majority of software is developed in FORTRAN.

4.2. Survey.

To get a better understanding of the capabilities used in the past projects developed by the branch, I have conducted a survey of the personnel. Out of 20 technical personnel, 13 have responded to the survey. The information is presented below.

Capabilities used:

graphics:	6 (46%)	port interfacing:	4 (31%)	
device drivers:	5 (38%)	communications:	7 (54%)	sound: 0

Languages used:

FORTRAN: 12 (92%)   PL/M: 1 (8%)   C: 1 (8%)   ADA: 5 (38%)
PASCAL: 2 (15%)

OS used:

VMS: 11 (85%)   UNIX: 1 (8%)   INTEL360: 4 (31%)   RSX: 1 (8%)

Software development tools used:

compiler: 12 (92%)	interpreter: 1 (8%)	editor: 12 (92%)
emulator: 1 (8%)	file manager: 3 (23%)	code manager: 0
debugger: 8 (62%)		

Evaluation tools:

software monitor: 3 (23%)   hardware monitor: 0

Project management tools used:

GANTT: 0	HIPO: 0	Milestone: 3 (23%)	PERT: 0
Flow charts: 5 (38%)		Costing: 0	Data Flow: 3 (23%)
Storage maps: 0	Coverage matrix: 0		

Document preparation tools used:

screen editor: 10 (77%)   wp: 3 (23%)

Networking tools used:

Ethernet: 4 (31%)

Hardware used:

VAX: 11 (85%)   PDP: 2 (15%)   IBM: 3 (23%)   INTEL: 7 (56%)

Programming background:

Average Programming experience: 8 yrs

BS Degree: 11 (85%)	MS: 2 (15%)	
Math: 8 (62%)	CS: 4 (31%)	Phys: 1 (8%)

4.3. Analysis.

Current system consists of a number of stand alone computing systems with no communication lines between them. Some of these systems are outdated and should be replaced. In particular the INTEL 8080 is an old 8 bit architecture. The PDP 11/23 is similarly in its advanced years.

The majority of projects are small to medium size, and should be supported by a system suited for this mode of operation. The particular advantage of software development in small sized projects is little inertia that needs to be considered when making source modifications and design changes. This makes it suitable for an approach to software development in terms of reusable, interchangeable, and configurable tools.

I have noted above, that the hardware currently in use shows signs of aging. However, the age of the hardware is not the major disadvantage; in fact it is the general architecture of the departmental system, or rather a lack of it.

Given that the majority of the Applications Software Branch' projects are developed in high languages, it would be conducive both to the improvement in productivity and to improvement in job satisfaction to provide an integrated programming environment.

An integrated software project requires frequent use of common information and data, eg. global files, cross-compilation tools etc. When the project is developed on a number of stand-alone systems, this requirement creates a problem of information integrity, decreases productivity, and magnifies the probability of problems during system integration.

Thus, the programming environment should be based on a locally distributed system architecture, linked together via a local network.

The need for porting the developed software to target systems can be resolved by provision of appropriate cross-compilers or cross-assemblers.

The "interchangeable, reusable and reconfigurable tools" paradigm of software development is well supported by Unix environments. In fact, Unix thrives on creation of tools with clean interfaces, and subsequent creation of larger programs by building interfaces which use those tools as communicating cooperating processes.

Unix environment also possesses facilities for "compile source to object and download object to (non-Unix) target" operations which are important in the projects developed in the Applications Software Branch (eg. MEA project).

5. Baseline design.

Analysis of typical branch activities indicates that the programming environment should be based on a locally distributed system architecture, linked together via a local network. Such a system may consist of a number of networked general-purpose workstations supported by compute and file servers.

Each of these workstations would be capable of functioning in a stand-alone mode thus off-loading the system servers; for compute intensive tasks the compute server would be used.

The file server will provide a central file system (with file security) to ensure the integrity of official versions of the project files, while the individual node storage capacity would be used for single user development, back up and other administrative chores.

5.1. Local Area Network (LAN).

LAN allows configuration of a locally distributed computing system. Among the major advantages of such a system one can count:

architecture	-	can be used as a number of stand-alone systems or as a distributed system.
fault tolerance	-	failure of one part of the network has only a limited effect on the operation of the network.
incremental	-	new nodes can be added when needed, allowing the system to grow together with the organization
common database	-	supports development and maintenance and access to common departmental databases of programs, files and other information.
data integrity	-	file server can regulate access to the official version of the information.
data security	-	file server can permit or deny access to information.
communication	-	allows communication and interchange of expertise; allows electronic mail communication.
reusability	-	promotes software reusability and sharing thus increasing productivity by reducing development effort.
productivity	-	see above.
quick response	-	single user nodes can be dedicated to individual tasks.
cost	-	allows sharing of hardware devices and software products.
fun	-	brings the computing system to programmers office and consequently increases job satisfaction (and productivity).

Given the preceding discussion, we no longer can ask whether the computer systems for the Applications Software Branch should be linked via a LAN, but merely what LAN.

Besides the usual considerations of speed, bandwidth, reliability, simplicity of use and cost, a major concern when choosing a LAN is compatibility with other network existing in the organization. Since MFSC/NASA has a number of branches and divisions with computer systems connected via Ethernet, it seems that a natural choice for this branch is therefore Ethernet LAN.

In fact, the existence of Ethernet in MFSC is a bonus. Ethernet is a baseband channel, optimized for LAN communications, which offers high-bandwidth and high transmission speeds for interconnection of intelligent nodes. Further, Ethernet interfaces are commercially available for many workstations and computers. Network architectures based on Ethernet offer typically the following functions (via application layer of OSI):

resource sharing,		
file transfers,		
remote file access	-	read, write, and update on files residing on remote systems,
database management	-	manipulations of databases distributed throughout the network,
network management.		

Thus having decided on the general approach to the Applications Software Branch computer system "architecture" we exhibit this architecture, a locally distributed system communicating over Ethernet, in the following diagram. This diagram shows only the basic configuration. Other nodes and servers may be needed; in particular addition of a print server and a terminal server may be desired. (All these servers provide interfaces between compatible devices on the LAN, and allow sharing of these devices among the network's nodes.) Also, connection of the existing computer systems may be specified.

```
                        other MFSC and/or public networks
=============================================================
                                  :
                                  :
                                  :
.----------.                 .----:-----.
: Routing  :-------/         : Routing  :
: server   :      /----------: server   :
:----.-----:                 :----------:
     :
     :
     :                          Applications Software Network
=============================================================
            :           :           :             :
            :           :           :             :
            :           :           :             :
        .---:----.  .---:---.  .---:---.      .---:---.
        : File   :  : Node  :  : Node  :......: Node  :
        : server :  :  1    :  :  2    :      :  25   :
        :--------:  :-------:  :-------:      :-------:
```

Fig. 5.1.1. Applications Software Branch computing system architecture.

The file server can be a minicomputer or a powerful workstation with sufficient mass storage to serve the database requirements of the branch. The nodes can be disk-less workstations, or workstations with local storage facilities (eg. hard disks). In the latter case, the local storage may be configured to be a part of the main file system (yielding a distributed file system), or it may be used for local storage individual to each node. In either case, the network software must provide transparent file system operations.

Similar considerations apply to distributed/central compute power of the system.

5.2. Towards the future.

Networked workstation concept advocated in this paper is an adaptation of a distributed system architecture. Such a system has a growth potential due to the ease of exchange, addition and removal of member nodes (see also section 5.1). A set of nodes in the system may consist of multi-vendor devices and computers, with the particular mix to be tuned to prevailing project requirements. Thus, a distributed system allows creation of a heterogeneous system.

Supported with a good network management software, this distributed system will offer total transparency of architecture to individual clients. At the same time, it will allow maximal utilization of systems resources.

The transparency of the system will manifest itself in promotion of perception of the resources of the system as belonging totally to the individual user. These resources can be as diverse as uniprocessors, specialized graphics devices, supercomputers, array processors, dedicated AI machines, general scientific computers, specialized finite-element machines, special storage devices, various supporting peripheral devices, etc.

To achieve such transparency, the system must be supported by appropriate network management software. This software must provide data independence as well computation independence of existing or future computer architectures.

Data independence (eg. based on the eXtended Data Representation (XDR)) provided in a multi-vendor system allows for ease of integration of new computer architectures without unwarranted commotion due to physical incompatibility with existing components of the system. Computation independence (eg. based on Remote Procedure Call (RPC)) supports integration of the computational power of the new computer, much as data independence supports data exchange.

Some such software tools are commercially available today, some will become available in the near future, yet others are still "flights of fancy".

In particular, the Network File System (NFS) by Sun Microsystems Inc. provides data sharing transparency, remote login facilities etc. NFS is a component of Sun's Open Network Computing (ONC) which provides an environment consolidating resources of multi-vendor computers operating under a variety of operating systems. NFS translates client file system into appropriate destination system commands, eliminating the need to learn new command languages. File sharing as well as file locking, and record locking is supported. The NFS system is based on Unix environment.

A particular strength of NFS is its ability to handle heterogeneous computing systems, its portability, as well as its independence of transport protocols (the latter is achieved through the RPC layer). For example, NFS is reported to have been implemented on machines such as Apollo, Cray, DG, HP, IBM PCs, mVax-II, Vax 11, Wang, etc, and has been ported to run under operating systems such as DEC Ultrix, VMS (Wollongong), Berkeley 4.2, System V.2, MS-DOS, DG MV 4000 and, ofcourse, SUN-OS.

The recently announced Network Computing System (NCS) by Apollo Computers Inc. promotes distributed computing via remote procedure calls to network resources and originating from a client process. NCS provides software (written in C) which runs on Unix systems in a heterogeneous networked environment. When ported on these machines, this software allows "packets of computation" belonging to the same process to be distributed and executed on various machines in accordance with availability of currently unused computational power.

Both of the above network systems coexist with the standard transfer protocol TCP/IP.

5.3. Application development software.

Software development tools required are:

operating system, command languages, programming languages, symbolic debuggers, linkers, library managers, source control systems, full screen multiple window editors, other editors.

Databases and file systems:

Databases are developed and serve groups of people. In a program development environment, the group is a project team. The personnel accesses and modifies the database, and therefore data compatibility, distributed processing, integrity and security are of major importance.

Project management:

Project management software should support methods for requirement formulation, performance evaluation, quality and reliability testing, planning, costing. This software should include:

planning tools:

milestone charts, PERT chart generator, GANTT chart generator, costing chart generator.

tracking tools:
software version control, modification control, object module librarian, bug report generators.

design tools:
flowcharts, HIPO charts, storage maps, coverage matrix (system functions v. program names).

documentation aids:
PDL, structured charts, data flow diagrams.

performance and evaluation:
software monitor, hardware monitors.

5.4. Software for embedded systems.

To improve productivity, software for embedded systems may be developed in high level language on modern workstations. Object code can then be generated by a cross-compiler and ported run on the target system. Testing and evaluation of the run efficiency of this software (on the target system) should also be performed in a high productivity environment. To afford this approach, emulators and logic analyzers are needed.

An emulator allows the developer to see the "role" that the software would be playing on the target system. The emulator frequently allows a good measure of control over the execution of the software, so that the software can be tuned and various tests conducted to compare the run properties of the new version versus the old one.

In addition to cross-compilers, disassemblers may be used to convert object code to assembly code. The latter may then be used to run "regression tests" to compare the object codes of an old version against that of a new version.

6. Information gathering.

6.1. Operating system.

The operating system used predominantly in the Applications Software Branch projects is the VAX/VMS running on various DEC machines. It is evident, therefore, that there is some existing expertise of this system among current personnel. However, it should be realized that a good application development environment would insulate the user from the operating system. Consequently, the existing expertise with VMS is not taken as a constraint in the choice of a new system.

A major alternative to VMS is the operating system Unix. A particular advantage of Unix is its pervasiveness on a majority of hardware. Concomitant with this is, of course, transportability of programs developed under Unix, as well as a very large software base.

Further, Unix is fast becoming a standard in federal government (Unix is specified in 70% of government procurement [GCN, 1987a]) and industry - it has been the OS of choice in Universities for a number of years. The armed forces have also recognized the importance of Unix. For example, both Army and Air Force are procuring Unix based systems; each has requested 20,000 32 bit systems. (The Army contract is estimated at $600 Million, while the Air Force contract -- Project 251 -- is estimated at $3 Billion [GCN, 1987b].)

6.1.1. VMS.

VAX/VMS is a multiuser, timesharing operating system for DEC VAX line of hardware. The system supports demand paging and swapping to satisfy memory requirements of its processes. A VMS process has a limit of 4 GB of memory, with a 1 GB limit per program.

VMS provides good program development tools, and supports various languages such as Fortran, C, Pascal, Ada, Cobol etc. It sustains electronic mail, and supports networking (Ethernet, DECnet).

VMS can be supplemented with various specific application packages and environments. Of particular interest are the VAX ADA and VAXset.

VAX ADA can be integrated with VMS to provide the standard VMS tools such as: debugger, record management services, run-time library, Digital Command Language and, of course, VMS file system.

VAXset is an optional toolset which together with the standard VMS tools mentioned above represent the VAX/VMS software engineering environment. This tool set consists of:

Language-Sensitive Editor (LSE) -
: A multi-window screen editor with language syntax sensitivity provided by language-specific templates.

Code Management System (CMS) -
: A program library system used for source-code and documentation control of software projects.

Module Management System (MMS) -
: A set of procedures for identifying dependencies of a system, and automatic rebuilding of the system in accordance with changes in referred modules.

Test Manager (DTM) -
: Facilitates user-designed regression testing of the project software, generation of benchmarks and result review.

Performance and Coverage Analyzer (PCA).
: Automates collection and analysis of data for program performance evaluation and generation of appropriate reports.

Good tools notwithstanding, a main disadvantage of VAX/VMS is its proprietary nature. Though the installed base of DEC computers is large, the VMS base is restricted to the VAX series. Selection of a proprietary system implies smaller software base (compared to a that of an industry standard system), and limits future expansion and hardware changes.

6.1.2. UNIX.

Unix is fast becoming a standard in federal government and industry - it has been the OS of choice in Universities for a number of years.

For our purposes, the following general features of Unix are its particular strengths:

- multiuser,
- multitasking,
- portable to multivendor computing systems,
- good development environment:
  - numerous tools,
  - utilities,
  - powerful user interface,
  - easy linking of programs into large applications
- good software project tools:
  - source-code control system,
  - make utility,
  - documentation tools.

A major criticism of early versions of Unix pertained to its rather cryptic command names and its, rather obtuse, user interface. These 'blots on the Unix character' have been erased by the software interfaces provided on most workstation vendors. In fact, it is my opinion that the original criticism was mainly related to 'fear of the unknown' by the critics for it is unusually easy to shape and bend Unix interface to conform with user's personal wishes.

Unix can be perceived as having the following layered structure:

```
-------------------------------
:          Utilities          :
:      ------------------     :
:      :     Shell      :     :
:      :   ----------   :     :
:      :   :        :   :     :
:      :   : Kernel :   :     :
:      :   ----------   :     :
-------------------------------
```

Fig. 6.1.2.1. Unix structure.

with the layers comprised of:

Utilities:
networking, software development tools, file manipulation, languages

Shell:
command language interpreter, C language

Kernel:
operating system functions

Of the three layers, the programmer interacts normally with the utilities and possibly the shell; the primitive functions of the kernel are accessed only for special hardware manipulation.

UNIX provides a "Programming WorkBench" environment which supports large project development. This environment is suitable for development of programs which are to be compiled and run on non-Unix target systems, including target real-time systems which differ markedly from timesharing environment. The Programming WorkBench includes facilities such as [Mitze, 1981] :

flexible remote job entry
source-code control system
control over interface specification
control over changes in documentation, source and data
modification-request control system
easy recreation of older versions
control over official and test versions
make files for compile and link mixes
variety of documentation tools
text processing tools
electronic mail

Over 90% of Unix is written in the programming language C . Since the function of a programming language is to make system resources available to the program, C is particularly useful in a Unix environment. In fact, C is the "official" language in Unix systems. C is an "intermediate level" language, providing the syntax and control constructs of a high level language, as well as power and flexibility of a low level language. It affords access to Unix source, libraries and utilities modules (promoting code reusability) as well as access to machine resources at primitive level. It is suitable for complex small, medium and large program development.

In short, coming from IBM personnel:
"Unix provides considerable functional power to the individual user, provides multi-user capabilities where needed, is open-ended, and has a large user and application base." [Henry, 1986]

6.2. Systems reviewed.

The following is a brief summary of the offerings reviewed for the new computer system.

Apollo. (Domain 3000)

A 32 bit workstation based on 68020. 2/4/8 MB RAM with 86/170/380 MB disk. Fair resolution (1024x800) with 15" color display. Runs at 16 MHz. Supports 3 terminals. Ethernet. Good user interface.

Unix. Good software development environment provided by Teledyne (TAGS) and Apollo's DSEE. The latter caters for development, management and maintenance of software, and provides the following capabilities: inter-module dependency tracking, automatic change notification, source code history control, task management and system configuration management.

Languages supported are Fortran 77, Pascal, C, Lisp and ADA (the latter is expected within a month).

Sun Microsystems. (3/52M)

A 32 bit workstation based on 68020. 4/32 MB RAM with 72/140/575 MB disk. Good resolution (1152x900) with 19" monochrome display. Runs at 15 MHz, rated at 1.5 MIPS. Unix. Ethernet. Good software development environment. Good user interface.

HP. (318M)

A 32 bit workstation based on 68020. 4 MB RAM with 80/571 MB disk. Fair resolution (1024x768) with monochrome 17" display. Runs at 16.6 MHz. Unix (HP-UX). Insufficient information on software environment.

DEC. (Vaxstation 2000)

A 32 bit workstation. 4/6 MB RAM with 42 MB disk. Fair resolution (1024x864) with 19" color display. Rated at 1 MIPS. Ethernet. Ultrix. Good software management environment (CMS, MMS: similar to Unix sccs). Good user interface - VAX Workstation Software (VWS).

Apple. (MAC II)

A 32 bit workstation based on 68020. 1/8 MB RAM with 20/40/80 MB disk. Low resolution (640x480) with 13" color display. Runs at 16 MHz and is rated at 2 MIPS. Unix A/UX. New machine, inexpensive but software environment is in question. EtherTalk.

Compaq. (Deskpro 386/40)

A 32 bit workstation based on 80386. 1/4 MB RAM with 40/70/130 MB disk. Poor resolution (EGA). Runs at 16 MHz and is rated at 3 MIPS. Unix V.3. New machine, inexpensive but software environment is in question.

Rational. (R1000/200-20)

A uniprocessor ADA machine. 32 MB RAM with 2 GB disk.

Supports from 16 to 32 users. Ethernet. Good ADA environment but what about the rest? Young company (2yrs).

Convex. (C-1)
A supermini. 8/128 MB Ram with 414 MB disk. 64 bit words with speed rated at 60 MOPS. Can support 160 users. Unix. Insufficient information.

Integrated solutions. (Optimum V8S)
A 32 bit workstation based on 68020 with 2/10 MB RAM and 140/280 MB hard disk. High resolution (1280x1024) with a 19" color display. Runs at 16MHz. Supports 16 users. Has a proprietary Transparent Remote File System. Good user interface in the form of Desktop Manager. Unix 4.2. Ethernet.

Encore. (Multimax-320)
A 32 bit parallel processor supermini based on NS32032 with 4/128 MB RAM and 515/4120 MB hard disk. Speed rated at 1.5 MIPS per CPU. Basic configuration has 2 processors, cost $139,000. Can be configured to have 20 processors. Works out at $11,000 for each two additional processors. Can support from 20 to 250 users. Unix (UMAX). Insufficient information on software supported for this machine. Ethernet.

Harris. (HS-20)
A 32 bit workstation based on 68000 with 1MB RAM and 50MB hard disk. Relatively low resolution (832x600) with a 19" color display. Speed rated at .575 MIPS. Expensive for what it offers. Insufficient information on software supported for this machine. Unix. Ethernet.

Motorola. (M6600)
A 32 bit workstation based on 68020. Unix. Apparently supports 128 users. Insufficient information.

Prime.
Runs under operating system PRIMOS. Proprietary system. No Ada, no C, no Ethernet (?). Insufficient information.

7. Detailed design.

7.1. Personnel.

The personnel of the branch numbers 24 and consist of the following general categories:

a)	branch management	1
b)	project management	2
c)	technical (programming)	19
d)	clerical (secretaries)	2

Each of the above categories requires different software environments, though any category may very well use any of the software available. These requirements can be categorized into the following environments:

1.	general management	(category a)
2.	project management	(category a, b, c)
3.	software development	(category b, c)
4.	document preparation	(category a, b, c, d)

Though it would be very desirable to have all four of the above unified under one environment, it is, at present, rather difficult to find a commercial system providing this amount of integration. However, in a general workstation environment (such as Sun Microsystems, Apollo Computers, and similar) integration of at minimum of environments 3 and 4 is the norm.

Finally, another component should be available to the branch, namely:

5.	training tools	(category a, b, c, d)

On line help facilities are nowadays provided with (just about) all software. Whereas helpful - and at times indispensable - to an experienced user, on line help should be complemented with tutorials, computer assisted instruction (CAI) tools, as well as hot line facilities and on site courses. Of course, good hardcopy manuals are a must.

7.2. Hardware.

The programming environment should be based on a locally distributed system architecture, linked together via a local area network. This system should consist of a number of networked general-purpose high-performance workstations supported by compute and file servers. Interfaces must be provided to attach and/or to access existing computer systems. The latter may function as additional servers.

Each workstation should be configured with sufficient RAM memory (at least 6 MB) and with local storage facilities (hard disk and streamer tape). In such configuration each workstation will be capable of functioning in a stand-alone mode thus off-loading the system servers; for compute intensive tasks access to (existing or new) compute server should be available.

Servers will provide interfaces between compatible devices on the LAN, and allow sharing of these devices among the network's nodes.

File server can be an existing minicomputer or a powerful workstation with sufficient mass storage to serve the database requirements of the branch. The file server will provide a central file system (with file security) to ensure the integrity of official versions of the project files, while the individual node storage capacity would be used for single user development, back up and other administrative chores. In addition, as need arises, local storage should be configurable into a part or whole of the file system for the branch (yielding a distributed file system). This will provide facilities for close communication on smaller projects, while preserving access to the general information depositories for the branch (such as documentation aids, reusable libraries, etc.).

Similar considerations apply to distributed/central compute power of the system.

The computer system for the Applications Software Branch should be linked via a LAN. Besides the usual considerations of speed, bandwidth, reliability, simplicity of use and cost, a major concern when choosing a LAN is compatibility with other networks existing in the organization. Since IAN of MFSC is based on Ethernet, the latter should form the network backbone for this branch.

(Ethernet is a baseband channel, optimized for LAN communications, which offers high-bandwidth and high transmission speeds for interconnection of intelligent nodes. Further, Ethernet interfaces are commercially available for many workstations and computers.)

Network architectures based on Ethernet offer (via application layer of OSI) resource sharing, transparent file system operations, remote file and system access, distributed database management, network management, and so on.

A basic configuration for the new computer system is presented in the following diagram. Other nodes and servers may be needed; in particular addition of a print server and a terminal server may be desired.

Supported with a good network management software, the proposed heterogeneous distributed system will offer total transparency of architecture to individual clients. At the same time, it will allow maximal utilization of systems resources.

These resources can be as diverse as uniprocessors, specialized graphics devices, supercomputers, array processors, dedicated AI machines, general scientific computers, specialized finite-element machines, special storage devices, various supporting peripheral devices, etc.

To achieve such transparency, the system must be supported by appropriate network management software. This software must provide data independence as well computation independence of existing or future computer architectures.

Suitable network management system is exemplified by Network File System (NFS) by Sun Microsystems Inc., and the recently announced Network Computing System (NCS) by Apollo Computers Inc.(see section 5.2). Both NFS and NCS are open system. These systems seem to evolve towards parallel execution of computational activities of a client over the available network resources. (As a guide, this is a rather coarse grain parallelity at the moment.)

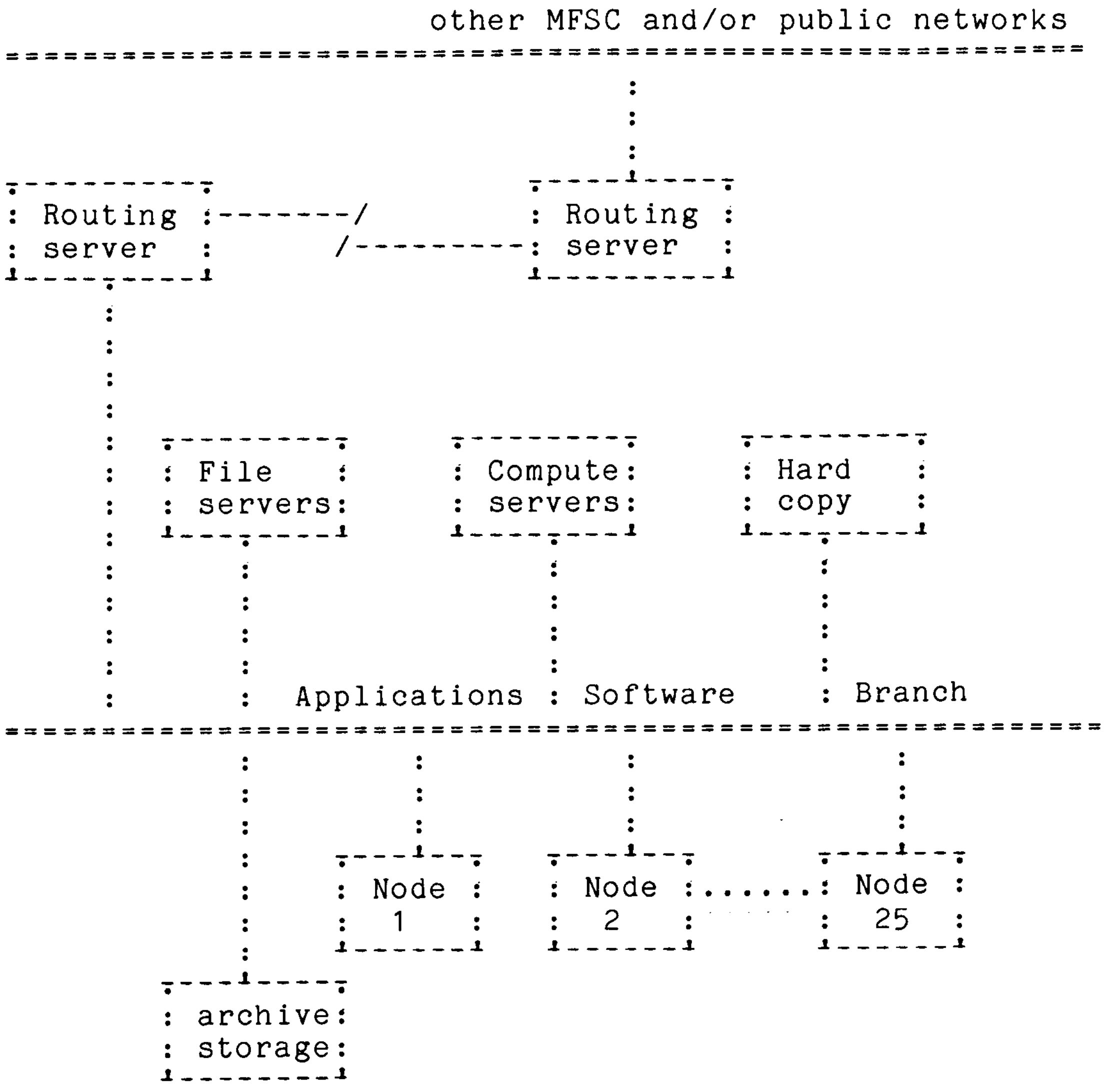

Fig. 7.2.1. Applications Software Branch computing system architecture.

7.3. Software.

7.3.1. General management.

The following are software tools used in general management: scheduler, spreadsheet, access control, electronic mail, resource measurement, cost estimation, work breakdown structure (WBS).

For additional discussion see section 5.3.

7.3.2. Project management.

The following are software tools used in project management:
scheduler, project tracking, spreadsheet, access control, electronic mail, resource measurement, WBS.

For additional discussion see section 5.3.

7.3.3. Software development.

Software development tools should support all of the software engineering functions (see section 3.1.2). Thus, I shall classify the software under the following software engineering activities.

R - Requirements specification and analysis.
D - Design and analysis.
P - Program generation and testing.
S - System integration and testing.
C - Configuration management.

The following is a glossary of a minimal set of tools that I perceive as indispensable in a software engineering toolkit:

Consistency checker.
: verifies mutual consistency of requirements and design for specific functions.

Cross-referencer.
: provides cross-reference information of use and misuse of program components.

Design language processor.
: Creates automatically documentation of the program design, as it progresses through its various stages. Particularly useful for validation and evaluation of the system design.

Diagnostic and debugging tools.
: Tools for identification, isolation and eradications of program errors.

Emulator.
: A software tool allowing execution on a host computer of a program written for a target computer.

Program Logic analyzer.
: analyses and provides information about the logic structure of a specified program.

Program flow analyzer.
: A program that provides source code statement frequency and timing data in test executions of a system.

Program sequencer.
A program verification tool for forcing test executions of every branch and every accessible statement in a systems code.

Simulator.
A program that provides on the host computer the (precise) effect of execution of instructions native to a target computer.

Requirements language processor.
A language and its processor for unambiguous formal specification of requirements for a software project.

Software monitor.
A memory resident tool that captures and exhibits performance statistics of a software system.

Syntax directed editor.
An editor using knowledge of a language to prevent, diagnose and correct syntax errors.

Trace program.
A program that captures and exhibits chronological history of events in an execution of a software system.

The following table show the coverage of the software engineering functions by the above mentioned tools:

	R	D	P	S	C
Consistency checker.	X	X			
Cross-referencer.			X		
Design language processor.		X			
Diagnostic and debugging tools.			X		
Emulator.				X	
Logic analyser.		X	X		
Program flow analyser.	X		X	X	
Program sequencer.			X		
Simulator.		X		X	
Requirements language processor.	X				
Software monitor.			X	X	
Source code control system			X	X	X
Syntax directed editor.			X	X	
Trace program.	X	X	X	X	

In addition to the above, the branch will require 'specialized' tools for real time applications and target computer evaluations. These tools will include:

Hardware logic analyzer.
An electronic device that monitors the logic states of digital systems and stores the results for subsequent display and analysis.

In-circuit emulator.
An emulator allowing alteration of control paths, monitoring and changes of memory registers and memory locations, etc.

7.3.4. Document preparation.

The following are software tools used in document preparation:
Graphics editor, word processor with spelling checker, grammar checker, dictionary generator.

7.3.5. Training tools.

The following are software tools used for training:
Training manuals, reference manuals, tutorials, CAI tutorials, on-site training.

8. Conclusion.

This project is concerned with a survey, evaluation and specification of a new computer system for the Applications Software Branch of the Software and Data Management Division of Information and Electronic Systems Laboratory of Marshall Space Flight Center, NASA.

The computer system serving currently the Applications Software Branch is organized into a number of stand alone and time sharing centralized computing systems. These systems include INTEL 8080, PDP-11/23, PDP-11/73, and a number of VAX 11/7xx and mVAX II. The mode of operation of these computers is predominantly development of software in Fortran, compilation and subsequent transfer (if necessary) of the object code to the target computer.

The obsolescence of some of the above system hardware, lack of support of current software engineering technology, and the projected technological and software needs of future Space Station projects entail a need for a new system able to support current and future software engineering environments of the branch.

My recommendation to the Branch is to acquire a distributed, heterogeneous system consisting of current, multivendor hardware interconnected with a local area network, and supported by appropriate software to provide an integrated software design and development environment.

The hardware should consist of high-performance workstations running a Unix operating system. The workstations should be configured with sufficient memory (6-8 MB) to support software development activities and a hard disk storage of sufficient capacity (140 MB) to offload the LAN traffic by restricting the swapping, paging and other system related operations to the local disk. Suitable candidates are SUN, Apollo or mVax.

The network architecture is to be based on system which provides a presentation layer over and above a transport protocol such as TCP/IP. This presentation layer has to support a kaleidoscope of computer hardware and must provide transparency of file and compute operations to be performed in this heterogeneous environment. A suitable candidate would be NFS (SUN Microsystems) and possibly NCS (Apollo Computers).

No special interfaces are currently required beyond the serial and parallel provided as standard ports on most workstations. However, in addition to the general purpose workstations, the branch will require special purpose workstations such as, for example, mAnalyst 2000 by NWIS for hardware logic analysis and in-circuit emulation.

The software development environment is to consist of components such as general and project management, code design and development, configuration management, document preparation, and personnel training.

Such system, in my judgement, will offer immediate productivity gains, increase in job satisfaction, virtual "plug in" expandability, and a suitable platform for insertion of future technology, thus assuring not only continued currency but also longevity of the system without undue proclivity to obsolescence.

9. Appendix 1: Software development life cycle.

The following are various phases of a software development life cycle adopted by MFSC/NASA, [Aichele, 1983]. For a comparison, I follow this with the more generalized specification as understood in industrial organizations.

<u>Conceptual</u>
Initial definition of system requirements. Preliminary functional specification and software configuration.
Results in:
- Request For Proposal (RFP),
- Data Requirements Document (DRD),
- Work Breakdown Structure (WBS),
- Statement Of Work (SOW).

<u>Requirements</u>
Elaboration of requirements of DRD. Construction of software management and development plan.
Results in:
- Interface Control Documents (ICD), and
- final Software Requirements Document (SRD).

<u>Design</u>
Preliminary design leading to Preliminary Software Design Specification. Detailed design follows and results in:
- Software Design Specification (CODE-TO),
- Programmer's Handbook, and
- Software Test Specification.

<u>Code and Debug</u>
Module coding debugging.
Results in:
- Initial internal software delivery.

<u>Verification</u>
Verification of logic of the debugged software. Formal change control and problem report procedures start from this phase.
Results in:
- delivery of the software, and
- Functional Configuration Inspection (FCI) document which ascertains conformance with requirements.

<u>Validation</u>
Evaluation of system/software compatibility and software performance.
Results in:
- final software delivery and configuration inspection,
- software user manual, and
- validation report.

System Integration
Final test of software at the highest possible system level.
Results in:
Systems Acceptance Review (AR).

Operations and Maintenance
as indicated by the title.
Results in:
final and permanent MFSC record of the project.

For a comparison, the following is a specification of software life cycle practiced in industrial environments, [Shooman, 1983].

Conceptual
definition of system needs, and of system requirements

System specification
Hardware-software operational specification and System operational specification

Preliminary design
RFP, preparation of proposal followed by proposal modification and selection

Detailed design
module coding, software design and functional specification

Verification
module test, system integration, test simulation and acceptance testing

Field deployment and maintenance
minor design modifications, discovery and fixing of field bugs. Final check and data gathering, field release

10. References.

(1) Aichele, D.G. "MSFC Software management and development requirements", EB41, MA-001-006-ZE, Jan 1983

(2) Bykat, A. "Evaluation of Applications Software Branch' Computing System", EB43, June 1987

(3) Craft, R.H. "S&E Director's review: Applications Software Branch", EB43, March 1987

(4) Illingworth, V. "Dictionary of computing", Oxford U.P., 1983

(5) Government Computer News, June 19, 1987, p.53

(6) Government Computer News, July 17, 1987, p.1

(7) Henry, G.G. "IBM RT PC architecture and design decisions" IBM SA23-1057, 1986

(8) Hyter, D. "Task for Summer Faculty", letter to DX01/Ernestine Cothran, EB43 (87-14), 4/21/1987

(9) Lucas, J.H. "Software and data management division", EB41, October 1986

(10) Miller, E. "Automated tools for software engineering", IEEE 1979

(11) Mitze, R.W. "The Unix system as a software engineering environment", in Hunke (ed), "Software Engineering Environments", North-Holland, 1981

(12) NASA-MFSC, "Information and Electronic Systems Laboratory", U.S. Gov. 730-067/40100, 1987

(13) Rauch-Hindin, W. "Software tools: new ways to chip software into shape", Data Communications, Apr.1982, pp.83-113

(14) Shooman, M.L. "Software engineering", McGraw-Hill, 1983

(15) Weston, C.D., Stewart, G.A., Byte, Feb. 1987

11. Glossary.

(1) BATSE:
Burst And Transient Source Experiment.

(2) Client:
machine capable of accessing server's resources.

(3) DRD:
Data Requirements Document.

(4) Emulator:
A software tool allowing execution on a host computer of a program written for a target computer.

(5) Ethernet:
a baseband channel suitable for high speed transmission of data in LAN communications.

(6) File server:
provides a file system and services for a network of machines.

(7) GANTT chart:
a bar chart such as tasks v. time.

(8) Heterogeneous system:
A networked system consisting of a set of multi-vendor computers of possibly different architecture, integrated into a single system.

(9) HOSC:
Huntsville Operations System Center.

(10) ICD:
Interface Control Document.

(11) IMC:
Image Motion Compensation.

(12) Integrated Program Development Environment:
An integrated set of programs and tools designed to facilitate development of software. Typical components are: full screen (windowing) editor, help facilities, programming language processors, symbolic debuggers, monitoring tools, source control and management tools, document generation tools.

(13) Integrated Project Management Environment:
An integrated set of programs and tools designed to facilitate design and management of software projects. Typical components are: planning tools, charting tools, facilities, programming design language, monitoring tools, reporting management tools, document generation tools.

(14) LAN:
Local Area Network - a network linking devices scattered over a limited geographic area and providing for high speed high bandwidth communication between these devices.

(15) Milestone:
a measurable significant achievement in course of project development.

(16) MEA:
Materials Experiment Assembly.

(17) Modification-request control system:
manages request of modifications, error reports, and debugging progress.

(18) Monitor:
a device which inspects and collects data during execution to determine efficiency and utilization of a unit (software or hardware).

(19) Network:
a number of workstations (or computer systems) interconnected and capable of exchanging information using a common communication protocol.

(20) Node:
a single addressable unit on a network. A number of devices can be connected to a node.

(21) PERT:
Program Evaluation and Review Technique. An activity network which represents the project's progress. The nodes of the network represent milestones, while the branches represent activities which culminate in that milestone.

(22) POCC:
Payload Operations Control Center.

(23) RFP:
Request For Proposals.

(24) RPC:
Remote Procedure Call: a set of functions providing for machine and operating system independent access of components of a heterogeneous computing system.

(25) Server:
machine that provides a resource to a network.
Eg. file server - provides a file system and services for a network of machines.

(26) Simulator:
A program that provides on the host computer the (precise) effect of execution of instructions native to a target computer.

(27) Software Engineering:
an approach to engineering software, paying particular attention to software life cycle consisting of planning, design, implementation, verification and maintenance.

(28) Software Tool:
A computer program used to develop, test, analyze, or maintain another computer program or its documentation.

(29) Source-code control system:
preserves different versions of the source and provides a list of changes between the versions.

(30) SOW:
Statement Of Work.

(31) SRD:
System Requirements Document.

(32) System:
An assembly of interacting components, all functioning towards achieving a specified goal. Components of a computer system typically include personnel, hardware, and software.

(33) TCP/IP:
Transmission control protocol/internet protocol.

(34) User:
person logged in on a client.

(35) WBS:
Work Breakdown Structure.

(36) Workstation:
a single user machine; either a stand-alone or a client. The major characteristics of a workstation are defined [Weston, Stewart, 1987] as:
32 bit microprocessor, 1-2 MIPS, 20+ MB hard disk, high-resolution monitor with 1 MPIXELs, floating point coprocessor, graphics operations, array processing, multi-user operating system (eg.UNIX).

12. Index.

AI 20, 28
Application 10
AR 36
BATSE 15, 38
Benefit 14
Client 38
Consistency checker 30
Cross-compiler 21
Cross-referencer 30
Disassembler 22
DRD 35, 38
Emulator 22, 30, 38
    in-circuit 32, 33
Ethernet 18, 28, 38
File server 17, 27, 38
GANTT 21, 38
Hardware logic analyzer 32
Heterogeneous system 20, 38
HOSC 15, 38
ICD 35, 38
IMC 38
Integrated program development environment 14, 38
Integrated project management environment 14, 38
LAN 17, 38
    Ethernet 18, 28
Languages 14
logic analyzer 33
Management 14
MEA 15, 39
Milestone 9, 39
Modification-request control system 24, 39
Monitor 21, 39
NCS 20, 29
network 39
    Client 38
    File server 38
    LAN 38
    NCS 20, 29
    NFS 20, 29
    node 19, 28, 39
    ONC 20
    Server 39
    User 40
    Workstation 40
Network Computing System 20, 29
Network File System 20, 29
NFS 20, 29
ONC 20
PERT 21, 39
POCC 15, 39
print 19, 28
Program flow analyzer 30
Program Logic analyzer 30

Program sequencer 31
project
    benefit 14
    definition 10
    development 14
    management 14
    specification 14
Resource 9
RFP 35, 39
RPC 39
server 19, 28, 38, 39
    print server 19, 28
    terminal server 19, 28
Simulator 31, 39
Software 14
    tool 23, 40
Software Engineering 39, 40
Software monitor 31
Source-code control system 23, 24, 40
SOW 35, 40
SRD 35, 40
Syntax directed editor 31
system 12, 13, 40
    capabilities 12, 13
    functions 11
    Scope 13
TCP/IP 21, 40
Tool 40
Tools 12, 14
Trace program 31
Unix 17
User 40
WBS 35, 40
Workstation 40