

AD-A073 357

RAYTHEON CO WAYLAND MA EQUIPMENT DIV

F/G 9/2

PAVE PAWS MODERN PROGRAMMING DATA COLLECTION SYSTEM.(U)

JUN 79 B H SCHEFF, W B VODGES, N R HALL

F30602-77-C-0141

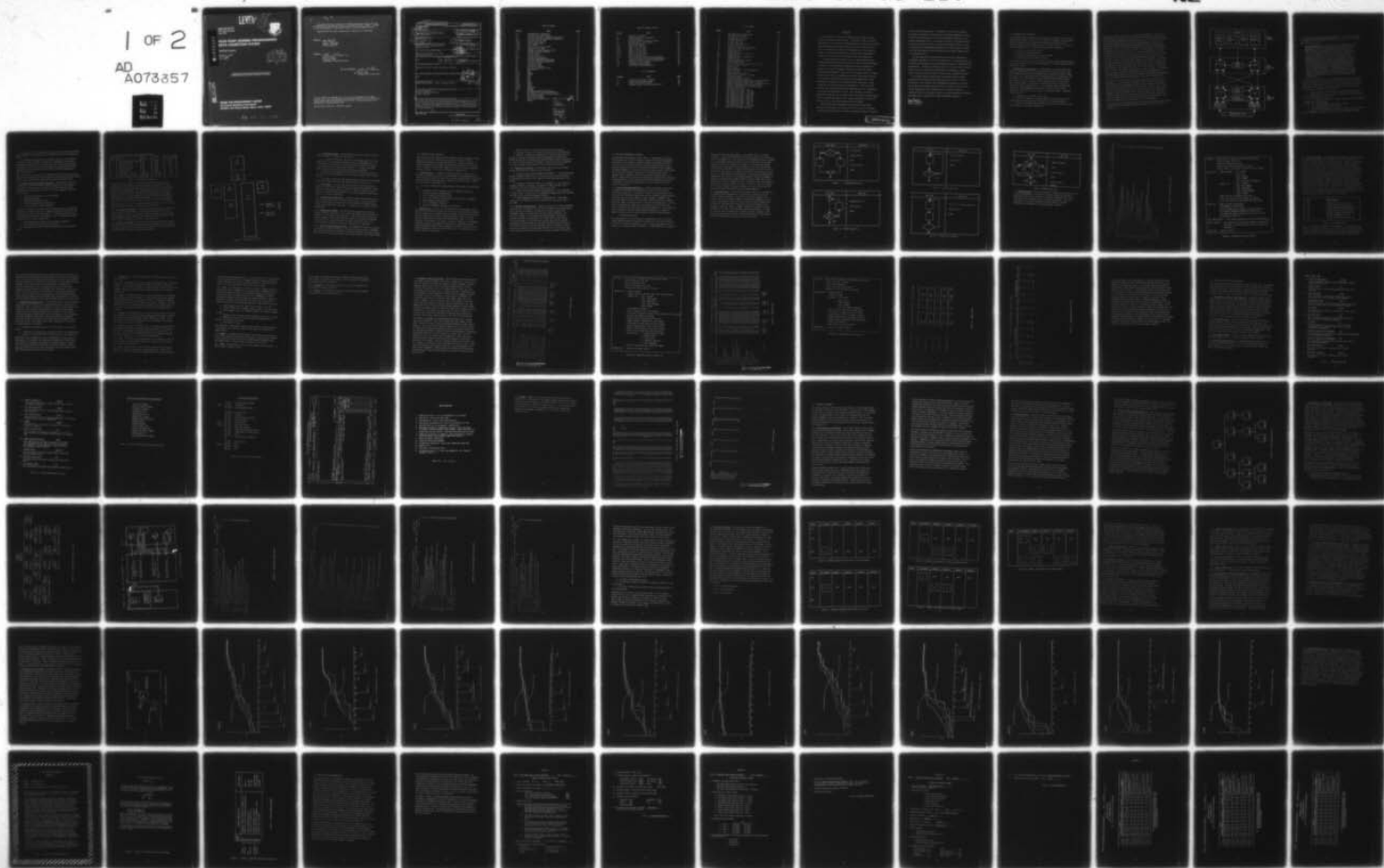
UNCLASSIFIED

RADC-TR-79-137

NL

1 OF 2

AD
A073857



LEVEL *II*

12



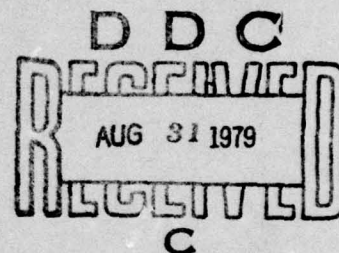
RADC-TR-79-137
Final Technical Report
June 1979

AD A 073357

PAVE PAWS MODERN PROGRAMMING DATA COLLECTION SYSTEM

Raytheon Company

Benson H. Scheff
W. B. Vogdes
N. R. Hall



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

79 08 31 009

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-137 has been reviewed and is approved for publication.

APPROVED:

Deane F. Bergstrom

DEANE F. BERGSTROM
Project Engineer

APPROVED:

Alan R. Barnum

ALAN R. BARNUM
Assistant Chief
Information Sciences Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADCTR-79-137	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PAVE PAWS MODERN PROGRAMMING DATA COLLECTION SYSTEM	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Jul 77 - Feb 79	6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR Benson H./Scheff W. B./Vogdes N. R./Hall	8. CONTRACT OR GRANT NUMBER(s) F30602-77-C-0141	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Raytheon Company, Equipment Division Wayland MA 01778	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 25280301	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441	12. REPORT DATE June 1979	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	13. NUMBER OF PAGES 91	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADCR Project Engineer: Deane F. Bergstrom (ISIE)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Engineering Modern Programming Techniques Computer Software		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes the software development technologies which were utilized on the PAVE PAWS project and the techniques which were implemented to collect data to support on-going independent technology studies. At the request of the contracting agency, the emphasis of this report is on describing the software technology used on PAVE PAWS and providing an assessment of the effectiveness of those techniques.		

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

388201

Gue

DDC
RECEIVED
AUG 31 1979
C

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1.0	BACKGROUND AND INTRODUCTION	1
1.1	PAVE PAWS System Description	1
1.2	Software Development Technology Requirements	4
1.3	Software Hierarchy (CPCI/CPCG Formulation)	5
1.3.1	Real Time Monitor (RTM)	6
1.3.2	Mission Control (MCTL)	6
1.3.3	Radar Manager (RAM)	8
1.3.4	Track (TRCK)	8
1.3.5	Displays and Controls (DISP)	8
1.3.6	Communications (COMM)	8
1.3.7	Satellite Catalog Management (SCM)	8
2.0	CONTRACT FOR DATA COLLECTION	9
2.1	Contract Purpose	9
2.2	Contract Scope	9
2.2.1	Manual Data Collection	10
2.2.2	Automatic Data Collection	10
3.0	PAVE PAWS PROGRAMMING ENVIRONMENT	11
3.1	Top-Down Programming/Segmentation	11
3.2	Structured Coding	12
3.3	Indented Listings	15
3.4	Hierarchical Library	18
3.5	Authorization Checking in PSL	19
3.6	PSL Directives	20
3.6.1	ADD	20
3.6.2	MODIFY	20
3.6.3	COMPILE	20
3.6.4	LOAD	20
3.6.5	COPY	20
3.6.6	XMIT	20
3.6.7	LIST	21
3.6.8	REPORT	21
3.6.9	PURGE	21
3.6.10	PUNCH	22
3.6.11	CHECKPOINT	22
3.6.12	RESTORE	22
3.7	Management Statistics Reporting	23
4.0	PAVE PAWS DATA COLLECTION ENVIRONMENT	31
4.1	PSL Changes in Support of Data Collection	31
4.2	TR Data Base Reports	31
4.3	Data Collection CPCIs	31
4.4	Manual Data Collection Form	31
4.5	Products	38

Accession For	
NTIS Grant	<input checked="" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	

TABLE OF CONTENTS (Cont'd)

<u>Section</u>	<u>Title</u>	<u>Page</u>
5.0	TECHNOLOGY ASSESSMENT	41
5.1	Top-Down Design and Development	41
5.2	Structured Coding	43
5.3	Indented Segment and Program Listings	44
5.4	Program Design: HIPO and PDL	46
5.5	Hierarchical Library	54
5.5.1	Usage of the PRG Level	58
5.5.2	Usage of the CPT Level	58
5.5.3	Usage of the INT Level	58
5.5.4	Usage of the FIX Level	59
5.5.5	Usage of the TST Level	59
5.5.6	Usage of the FRZ Level	59
5.5.7	Usage of the DEL Level	59
5.6	Chief Programmer Team/Librarian Operations	59
5.7	Structured Design/Structured Code Walkthroughs	60
5.8	Management Statistics Collection/Reporting	60
5.9	Qualification Test Program	61
5.10	Programming Communications	74
6.0	CONCLUSIONS AND RECOMMENDATIONS	78

LIST OF APPENDICES

<u>Appendix</u>	<u>Title</u>	<u>Page</u>
I	General Contract/Project Summary	80
II	Management Methodology Summary	82
III	Design and Processor Summary	84
IV	Personnel Profile (Chief Programmer Team)	86

LIST OF FIGURES

<u>Number</u>	<u>Title</u>	<u>Page</u>
1	PAVE PAWS System Block Diagram	3
2	PAVE PAWS CPCI Breakout	6
3	CPCG Structure for CPCI 2	7
4	IF_THEN_ELSE Logic Form	13
5a	DO WHILE Logic Form	13
5b	DO UNTIL Logic Form	14
6	DO LOGIC FORM (Indexing)	14
7	CASENTRY Logic Form	15
8	Example of Indented Segment Listing	16
8a	Explanatory Notes for Figure 8	17
9	PSL Library Levels	18
10	SEGMENT Summary	24
10a	Explanatory Notes for Figure 10	25
11	PROGRAM Summary	26
11a	Explanatory Notes for Figure 11	27
12	LIBRARY Summary	28
13	Progression/Durability Report	29
14	COMPILE REASON CODES	32
15	Data on PSL Data Collection Statistics File	34
16	PSL Data Collection CPCGs	35
17a	Sample TR Form	36
17b	Error Categories	37
18	Sample PSL Report - Compiler Summaries	39
19	Sample TR Report	40
20	Program Segment Structure	45
21	Visual Table of Contents - Example	47
22	HIPO Chart Example	48
23	Indented PDL Program Listing	49
24	Program Configuration After Entry of Initial Segment	55
25	Program Configuration After XMIT to CPT Level	55
26	Program Configuration After Entry of Segments A and B	56
27	Program Configuration After Subsequent XMITs	56
28	Program Configuration After Further Changes	57
29	Code Development Curves	62
30	Code Progression Chart - CPCI 2	63
31	Code Progression Chart - COMM CPCG	64
32	Code Progression Chart - DISP CPCG	65
33	Code Progression Chart - MCTL CPCG	66
34	Code Progression Chart - RAM CPCG	67
35	Code Progression Chart - RTM CPCG	68
36	Code Progression Chart - SCM CPCG	69
37	Code Progression Chart - TRCK CPCG	70
38	Code Progression Chart - CPCI 3	71
39	Code Progression Chart - RTSM CPCG	72
40	Code Progression Chart - TSG CPCG	73
41	Example of PAVE PAWS Green Sheet	75

EVALUATION

The PAVE PAWS is a phased array warning system designed to detect submarine launched ballistic missiles. In addition to real time mission requirements for the detection and characterization of SLBM's, the PAVE PAWS system implementation provides capabilities for simulating the mission functions, generation of scenarios for simulation, and a data reduction system. All the above system functions necessitated the development of computer software for both real time and non-real time capabilities.

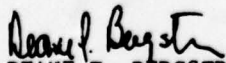
A system requirement called for the use of modern programming and software engineering tools and methods for all system software development. In response to this requirement, Raytheon/IBM selected and employed a complete set of modern programming techniques for PAVE PAWS software development and management. These tools and procedures included a Program Support Library (PSL), pre-compilers to translate structured source code, use of graphic design methods and Program Design Language (PDL), Chief Programmer Team operations, structured design and code reviews, coding conventions, and top down design and implementation. The PSL provided extensive data collection and reporting capabilities for use by management in making timely assessments of status. This complement of software engineering techniques will be utilized during the operation and maintenance phase of the PAVE PAWS system. Thus, software maintenance personnel will utilize the tools and methods employed during development.

The PAVE PAWS Modern Programming Data Collection System effort described in this report was initiated as part of an effort to determine the utility and effectiveness of software engineering technology as applied

to large system implementations. Furthermore, the PAVE PAWS programming environment was examined to obtain data on PSL software management functions and how the PSL reporting function affected management visibility into the software development process. A combination of manual and automated methods were used for the data collection. Manual data collection forms were used to characterize the programming environment and a software module was added to the PSL which gathered error and change data and produced summarizations of the change activity.

The data collection effort described herein has been supplemented by a technology assessment of the tools and methods used. The modern programming techniques and development tools won widespread acceptance by programmers and managers alike. Although the technology does not, in and of itself, guarantee success it must be credited with establishing an environment to support project success and the early identification of real or potential problems.

This report supports ongoing efforts under RADC technical program objectives under the Software Cost Reduction thrust of TPO 5, C³ System Availability. The conclusions and recommendations contained in the report and the data obtained under this effort will be utilized by efforts in the Software Engineering Tools and Methods area. The report should be of significant value to all personnel involved in system acquisition and software development and the application of modern programming techniques.


DEANE F. BERGSTROM
Project Engineer

1.0 BACKGROUND AND INTRODUCTION

The PAVE PAWS system acquisition is a fixed-price acquisition by the Electronics Systems Division (ESD) of the Air Force to Raytheon's Equipment System Division requiring system design, development, and integration leading to Initial Operating Capability (IOC) within three years of contract award. It includes several different types of software system development, among them -

- a. A real-time early warning system.
- b. A real-time simulation system.
- c. A non-real-time simulation scenario generator.
- d. A non-real-time data reduction system.

This section describes the tactical system, the software development technologies required, and the allocation of system requirements to Computer Program Configuration Items (CPCIs).

1.1 PAVE PAWS System Description. The PAVE PAWS is a fixed base Phased Array Warning System utilized for the detection and attack characterization of Submarine Launched Ballistic Missiles (SLBM's) which penetrate the radar coverage. It consists of two Phased Array Warning Sensors located at Otis AFB, Mass. and Beale AFB, Calif. The primary mission of PAVE PAWS includes SLBM detection and tracking in order to provide the NORAD Cheyenne Mountain Complex (NCMC) with credible warning of SLBM attacks, including estimation of Launch and Impact (L&I) points, and times of L&I. As a secondary mission the PAVE PAWS supports the USAF SPACETRACK System with Earth Satellite Vehicle (ESV) surveillance, tracking, and data collection as requested by NCMC. SPACETRACK functions include:

- a. Maintenance of a catalog of known ESVs.
- b. Detection, recognition, and data reporting (either cross-section or position data) for ESVs specified by NCMC or by local system operators.
- c. Detection, tracking, and data reporting (cross-section, position, and orbital element set data) for unknown ESVs.

Message communication, both to and from NCMC/SAC/NMCC/ANMCC, is performed in accordance with the American National Standard for Advanced Data Communication Control Procedures (ADCCP) over Government data links. The system also includes six display consoles which are used for Systems Operations, Monitoring and Control, Missile Warning Operations, SPACETRACK Operations, Training, and Maintenance Control. Over thirty different display formats are independently selectable at the display consoles in order to provide complete flexibility in monitoring and controlling the system. Because the PAVE PAWS is an on-line system which is intended to be operational 7 days per week, 52 weeks per year, the data processing system contains redundant hardware throughout. In the event of a hardware or software fault, hardware is automatically reconfigured to eliminate the fault and resume the primary mission within 8 seconds. The data processor (duplex CDC CYBER 174's) communicates with one of two MODCOMP mini-computer which interface directly with the radar hardware (signal processor, et al). The hardware configuration is shown in Figure 1. The MODCOMP computer controls and directs reconfiguration of the radar hardware, the real-time system resident in the on-line CYBER controls MODCOMP reconfiguration, and the PAVE PAWS Operating System (CYBER) directs CYBER reconfiguration.

In addition to the software to perform the primary and secondary missions of PAVE PAWS, the system includes a simulation facility capable of operating concurrently with the operational software and providing the full range of mission, threat, communications, and radar stimuli to that software. Object trajectories, radar cross sections, launch and impact points, communications messages, radar environmental effects, and event timing can be simulated under user specification. The system also records real-time data pertinent to the performance of the primary and secondary missions and provides data reduction capabilities for a wide variety of recording formats.

The structuring of these requirements into Computer Program Configuration Items (CPCIs) and Computer Program Configuration Groups (CPCGs) is discussed in Section 1.3.

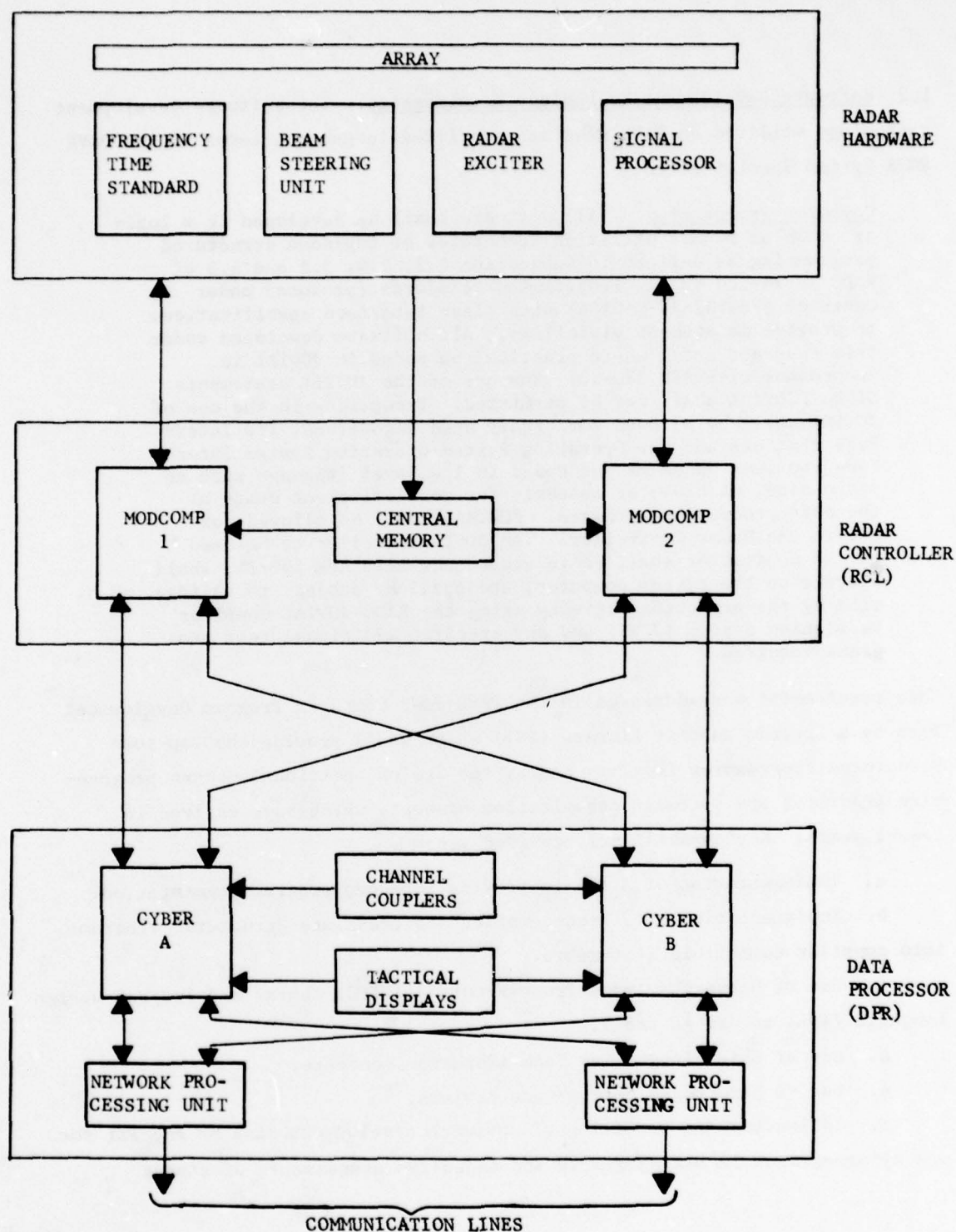


Figure 1. PAVE PAWS System Block Diagram

1.2 Software Development Technology Requirements. The software development technology utilized on PAVE PAWS was specified in general terms in the PAVE PAWS System Specification:

Computer Programming. "All software shall be developed in a logical modular manner utilizing techniques of top-down structured programming as defined in Subsection 2.2, 2.4, 3.2 and 4.3 of RADC TR-74-300 Vol 1, Programming Standards (produced under Contract #F30602-74-C-0186) with clear interface specifications to provide management visibility. All software developed under this contract shall where practical be coded in JOVIAL in accordance with AFR 300-10. The use of the JOVIAL statements DIRECT/JOVIAL shall not be permitted. Exceptions in the use of JOVIAL shall be allowed for highly used algorithms, I/O Interface routines and the Operating System/Operating System Interface routines which may be coded in low level language such as micro code, machine, or assembly for more efficient usage of the data processing hardware. FORTRAN shall be allowed for use in the Radar Controller. The JOVIAL compiler to be used by the contractor shall be in accordance with AFM 100-24, shall operate on the system computer, and shall be subject to validation by the procuring activity using the RADC JOVIAL Compiler Validation System (JCVS) and any specific additional test programs required."

This requirement was addressed in the PAVE PAWS Computer Program Development Plan by a Program Support Library (PSL) which would provide the Top-Down Structured Programming facility and by the use of additional modern programming practices and software organization concepts which have evolved in recent years. Key capabilities provided are:

- a. Implementation of a PSL to provide Top-Down program segmentation.
- b. Implementation of a "pre-compiler" to translate Structured Programs into compiler compatible statements.
- c. Use of Hierarchy Input-Process-Output (HIPO) charts and Program Design Language (PDL) as design tools.
- d. Use of Chief-Programmer Team/Librarian concepts.
- e. Use of Structured Design/Code Reviews.
- f. Collection and reporting of software development data by the PSL for use by management in making timely and objective assessments of status.

g. Creation of a Test organization separate from the software development group responsible for developing all test documentation and for conducting the tests.

h. Organizational separation of the group responsible for developing the Quality Assurance Program, including the establishment of project-wide procedures, implementation of a Trouble Report system, and providing regular assessments of status and forecasts for management consideration and action, from the software development group within each implementing organization (IBM, CDC, Raytheon).

The technical scope and content of the PAVE PAWS PSL is discussed in Section 3. Section 5, the Technology Assessment, addresses key elements of the PSL together with the other procedural and organizational practices mentioned above.

1.3 Software Hierarchy (CPCI/CPCG Formulation). The allocation of system requirements to individual Computer Program Configuration Items (CPCIs) is an important function because from that point forward each CPCI will be managed with a certain degree of autonomy. The term "managed" in this context includes -

- a. estimating and planning the effort involved,
- b. allocating resources,
- c. assessing and reporting status,
- d. financial management and reporting, and
- e. the resolution of technical problems.

Clearly it is important that these functions provide control and visibility below the total system level, but the danger of subdividing too much is that "all the pieces work but the system doesn't." A number of guidelines were developed for defining CPCIs on PAVE PAWS in order to establish an effective subdivision of the total software effort:

- f. CPCI responsibility should not cross corporate boundaries.
- g. CPCIs should not cross computer boundaries.
- h. Software systems which are executed separately should be separate CPCIs.

The resultant CPCI definitions are presented in Figure 2.

CPCI	Title	Corp.	Comp.	Size (Lines)
1	PAVE PAWS Operating System	CDC	CYBER	N/A
2	Tactical Software	IBM	CYBER	139K
3	Simulation Software	IBM	CYBER	29K
4	Support Software	IBM	CYBER	16K
5	Data Reduction	IBM	CYBER	27K
6	Radar Control Software	RAYTHEON	MODCOMP	N/A
7	Signal Processor Software	RAYTHEON	Sig. Proc	N/A

Figure 2. PAVE PAWS CPCI Breakout

Below the CPCI level, software is next broken down into Computer Program Configuration Groups (CPCGs) and Computer Program Components (CPCs). CPCGs are generally structured along major functional lines within a CPCI while CPCs represent individual programs. This structuring of the software is important because it forms the basis for allocating system requirements to software, identifying interface control definitions, subdividing design and development responsibilities, and making personnel assignments. In short, a well understood software structure allows a software project to be effectively managed.

The CPCG structure for CPCI 2 is shown graphically in Figure 3. In this figure each CPCG is scaled to show its relative size (source cards).

1.3.1 Real Time Monitor (RTM). The Real Time Monitor (RTM) acts as the single interface between the PAVE PAWS Operating System (PPOS) and the tactical or mission software of CPCI 2. It performs interrupt handling, cyclic and demand task schedulings, task dispatching in accordance with system priorities, Input/Output resource management, and dynamic storage management.

1.3.2 Mission Control (MCTL). Mission Control performs the high level control functions of CPCI 2, including initialization, reconfiguration, and termination. It also provides disk file, recording, and errorlog services, and checkpointing of tactical data in support of system reconfiguration.

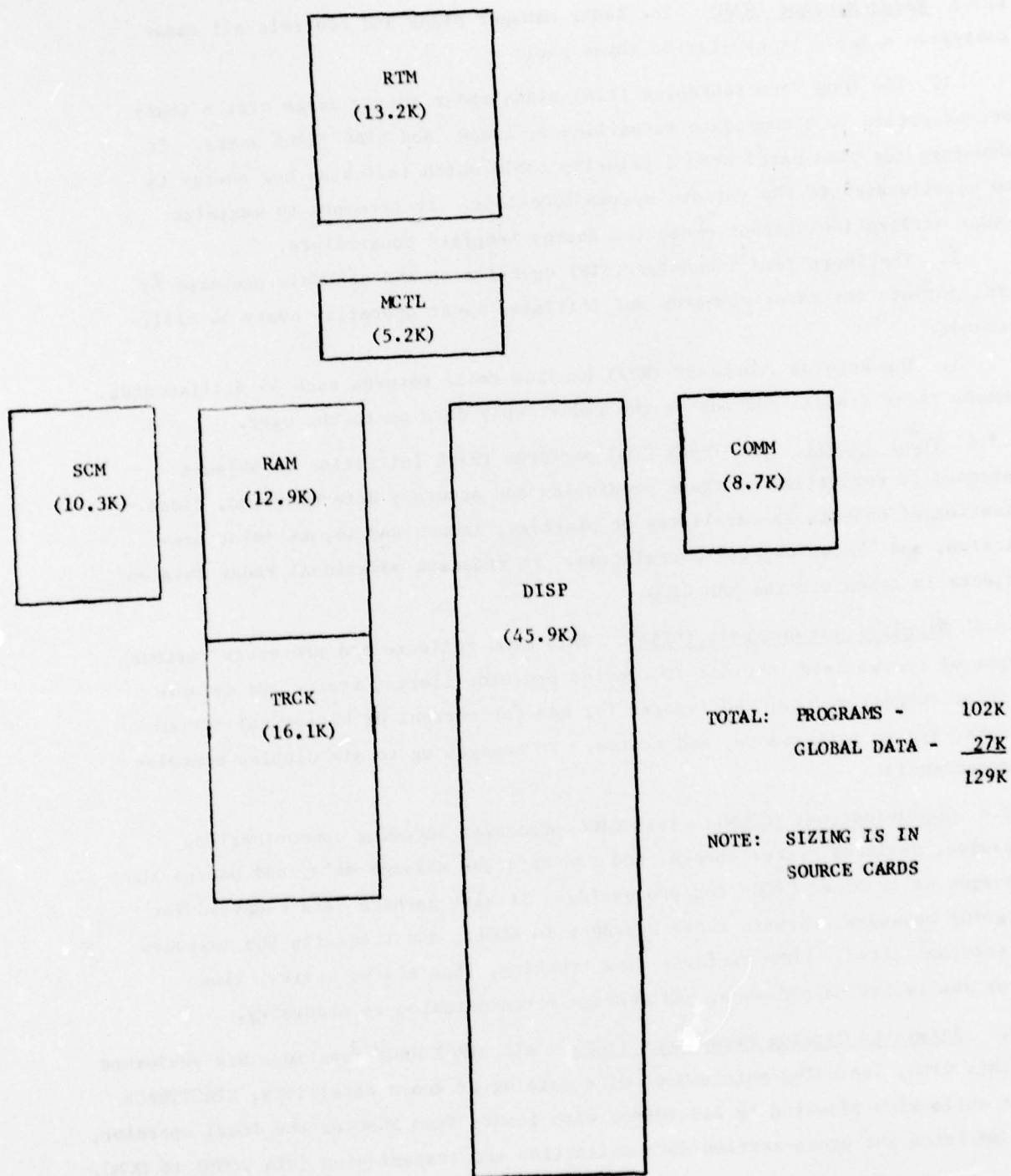


Figure 3. CPGG Structure for CPCI 2

1.3.3 Radar Manager (RAM). The Radar Manager plans and controls all radar subsystem usage. It consist of three parts -

1. The Long Term Scheduler (LTS) plans radar energy usage over a four-second period to accommodate surveillance, track, and SPACETRACK users. It develops the plan based upon a priority table which indicates how energy is to be allocated to the various system functions. It attempts to maximize radar utilization without exceeding energy template constraints.

2. The Short Term Scheduler (STS) operates on the schedule prepared by LTS, formats the radar commands and initiates radar operation every 54 milliseconds.

3. The Returns Processor (RTP) handles radar returns each 54 milliseconds, checks radar status, and passes the radar reply data on to the user.

1.3.4 Track (TRCK). The Track CPG performs track initiation on objects detected in surveillance, track prediction and accuracy determination, classification of objects as satellites or missiles, launch and impact point prediction, and "known object" correlation. It requests additional radar data on objects in track via the RAM CPG.

1.3.5 Displays and Controls (DISP). This CPG collects and processes various types of system data in order to provide operator alerts, static and dynamic display images, and printed reports for man for current or historical system events, system performance, and status. It manages up to six display consoles independently.

1.3.6 Communications (COMM). This CPG processes incoming communications messages, unblocks, error checks, and converts the message data, and passes the messages on to other CPGs for processing. It also gathers data required for outgoing messages, formats those messages in ASCII, and transmits the messages to external sites. COMM performs line trunking, line status review, line error statistics maintenance, and message retransmission as necessary.

1.3.7 Satellite Catalog Management (SCM). All SPACETRACK functions are performed in this CPG, including maintenance of a catalog of known satellites, SPACETRACK data collection planning in accordance with inputs from NCMC or the local operator, and position and cross-section data collection and transmission (via COMM) to NCMC.

2.0 CONTRACT FOR DATA COLLECTION

Realizing that the PAVE PAWS software development effort represented a unique opportunity to collect information and experience relative to "modern programming technology", Raytheon/IBM submitted an unsolicited proposal to Rome Air Development Center (RADC) proposing that such data be collected for CPCI's 2 through 5 and provided to RADC for their use in on-going technology evaluation studies. This contract was awarded in August 1977.

2.1 Contract Purpose. The purpose of the Data Collection contract was to validate the special tools used, to provide guidance on programming environments for large system acquisitions, and to provide insights into new experiences in software engineering using modern programming tools and methods. Specific areas of interest on PAVE PAWS were:

- a. Use of a comprehensive Program Support Library system (the PAVE PAWS PSL).
- b. Structured coding, including the use of language precompilers.
- c. Top-down design and implementation.
- d. Use of Program Design Language (PDL).
- e. The use of transaction data collection and reporting to management.
- f. Chief-Programmer Team Operations.
- g. Use of a Programmer Librarian.
- h. Effective programming standards and conventions.

2.2 Contract Scope. The intent of the Data Collection effort was to provide data which characterized the nature and environment of the software development activity together with information about the reasons underlying software change. This would allow ongoing software technology studies at RADC to correlate software change activities with project characteristics such as the size, complexity, and schedule of the project, the type of contract, the programming technology utilized, the management organization and methodology, the programming language utilized, the data processor availability and capacity, the system documentation structure and availability, etc. The collection of this data was effected in three ways:

- a. Manual collection of project and personnel characteristics.
- b. Automatic collection of software change data by the PAVE PAWS PSL.
- c. Automatic recording and summarization of software change activity as part of a project-wide Trouble Report/Change Request (TR/CR) system.

Because the bulk of the software design and development had been completed by the time of contract award, the "automatic" collection of data was augmented by a one-time manual reconstruction of the existing TR/CR data base.

2.2.1 Manual Data Collection. The following types of data were provided through the completion of forms by project personnel:

- a. General Contract/Project Summary (see Appendix I). This form provides general information about the size of the project (cost, people, software, and documentation) together with a high level technical description of the project.

- b. Management Methodology Summary (see Appendix II). This identifies management procedures utilized, the schedule for PDR's and CDR's and an enumeration of the AF and Military Standards which apply.

- c. Design and Processor Summary (see Appendix III). This identifies the data processor configuration, the programming languages used, the standards followed, and the software technology utilized.

- d. Chief Programmer Team Profiles (see Appendix IV). These forms characterize the educational and work experiences of each of the teams on PAVE PAWS.

2.2.2 Automatic Data Collection. Changes were made to two existing PAVE PAWS systems in order to automate the collection and reporting of software change data. The first of these was an extension to the PSL to require that programmers specify a "reason code" for each program compilation. The second was a change to the Trouble Report/Change Request system which similarly required the specification of a "reason code" at the time the TR or CR was closed. It should be noted that the PSL data will include programming effort which does not fall under the TR/CR system and that one TR (or CR) may result in many PSL operations before the problem is solved. Thus the two systems collect data which overlaps but is in no way the same. These systems and the data they collect are further described in Section 4.0.

3.0 PAVE PAWS PROGRAMMING ENVIRONMENT

The PAVE PAWS Program Support Library (PSL) is a programming system specifically designed to support and enforce Top-Down and Structured Programming technologies. This requires a program storage and maintenance capability which is oriented toward a high degree of program segmentation and a pre-compiler which has the effect of extending the commercial JOVIAL, COMPASS, and IFTRAN languages to include the necessary structured forms. Additionally, the PSL has been designed to accommodate a structured Program Design Language (PDL). Although similar to most compiler languages, PDL is completely unconstrained in syntax, thus allowing natural English-like description of program design. This section describes the PSL implemented and utilized on PAVE PAWS. A subjective evaluation of its most effective features is provided in Section 5.

3.1 Top-Down Programming/Segmentation. Top-Down programming is based upon a technique of designing (and implementing) software by specifying the top level functions first. The details of each of those functions and the specification of additional subfunctions are then developed through successive iterations until the entire problem is fully developed. Throughout this process the amount of design (or code) which is being developed is purposely kept fairly small in order to allow it to be dealt with effectively. This can only be accomplished by referring to total functions or sub-functions as "black box" modules with known input and output requirements. This modularization is reflected in the PSL through program segmentation. A segment of program code can identify a needed function by using an INCLUDE statement:

```
INCLUDE function name
```

This named function can then be dealt with independently, and it may itself utilize INCLUDE statements to identify and define even lower level functions. In this way a program is developed as a set of single page segments which fit together in a program structure or hierarchy. The PAVE PAWS PSL is designed to

handle such highly segmented programs. The Top-Down aspect of software development is enforced by identifying each segment placed in the library as either a top-segment (i.e., the top-level of an independently compiled program) or as an INCLUDE'd segment (one which is simply a lower-level part of some program). As top-level segments are entered into the library and INCLUDE statements are encountered, stubs are generated to act as position holders until real-code is provided. A program stub identifies the need for code to perform the named function, it reserves the name for that function, and since it is part of some already existing program, it specifies the implementation language for that function. The Top-Down ordering of software development is enforced by requiring that INCLUDE'd segments cannot be added into the PSL library unless they are replacing a stub. In addition, since stubs represent unimplemented software segments, the number of stubs in a CPG or a program can be used as a measure of status or progress. Section 3.6 describes the PSL tools available for dealing with these program segments.

3.2 Structured Coding. Structured Coding requires the use of a standard set of program control statements and at the same time precludes the use of explicit branching statements. In order to provide the standard set of control statements for JOVIAL, COMPASS, IFTRAN and PDL programmers, the PSL includes a pre-compiler which accepts the structured source statements and converts them into traditional control forms which are processed by the appropriate compiler. Figures 4 through 7 show both the logical form and the coded form of each of the PAVE PAWS standard control forms. It should be noted here that the requirement to provide a separate statement to end each of the forms provides an ideal closure mechanism for the generation of indented listings which are discussed in the next section.

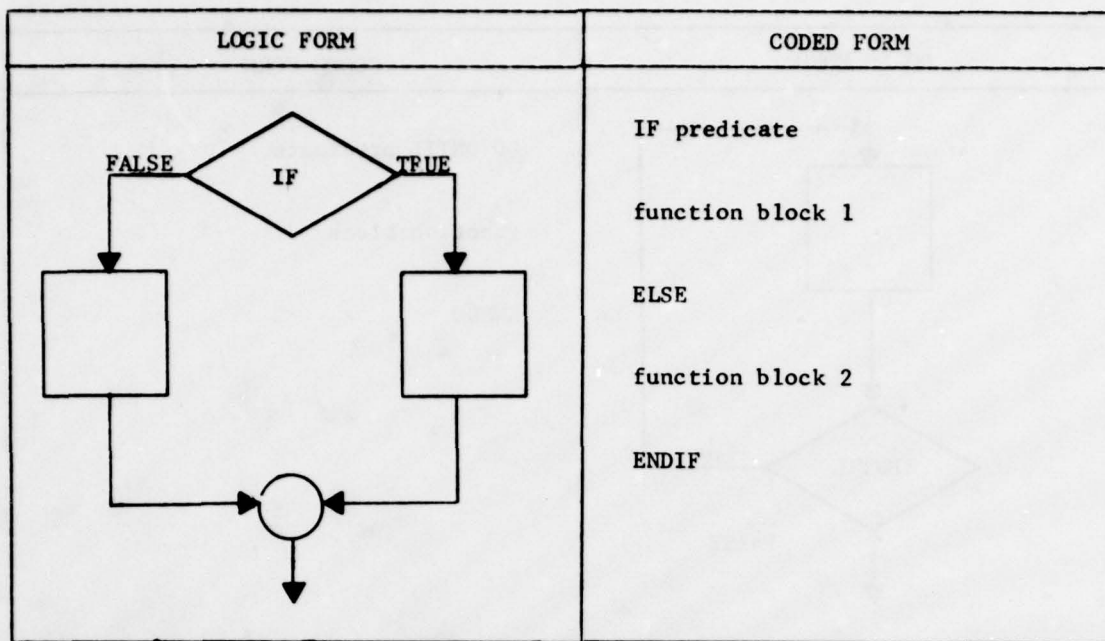


Figure 4. IF_THEN_ELSE Logic Form

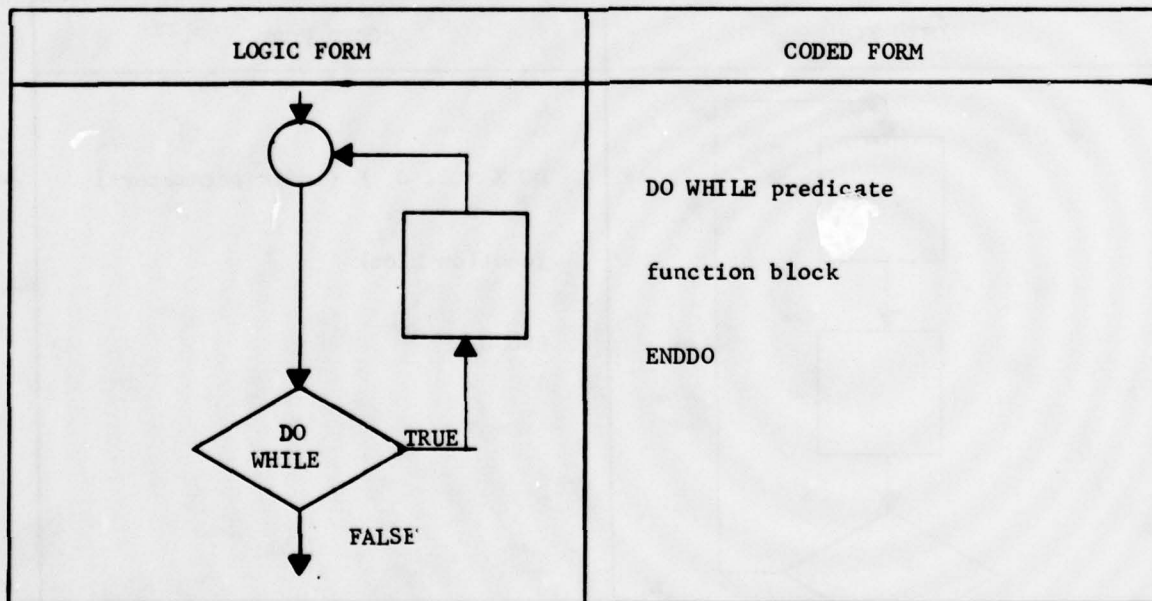


Figure 5a. DO WHILE Logic Form

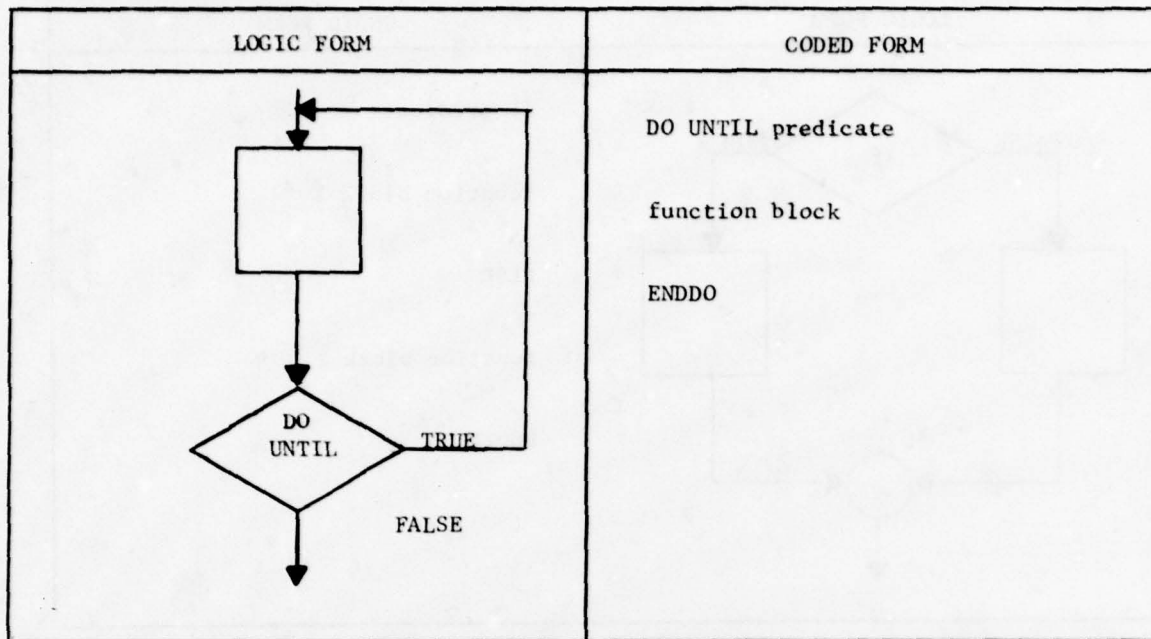


Figure 5b. DO UNTIL Logic Form

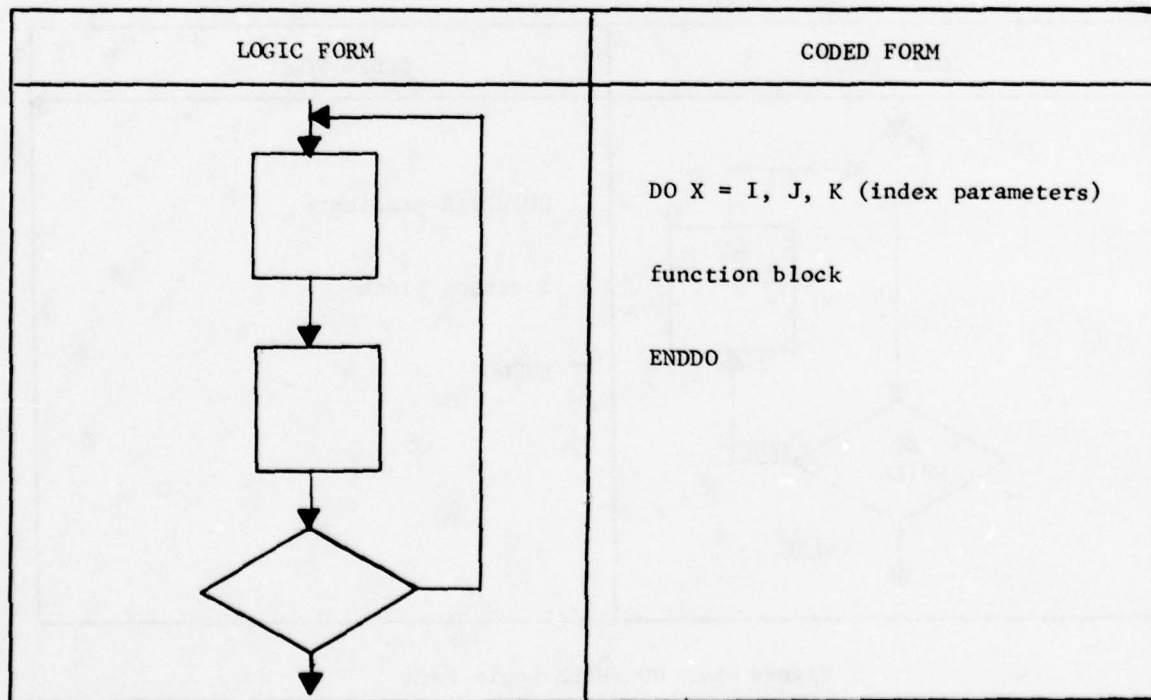


Figure 6 - DO Logic Form (Indexing)

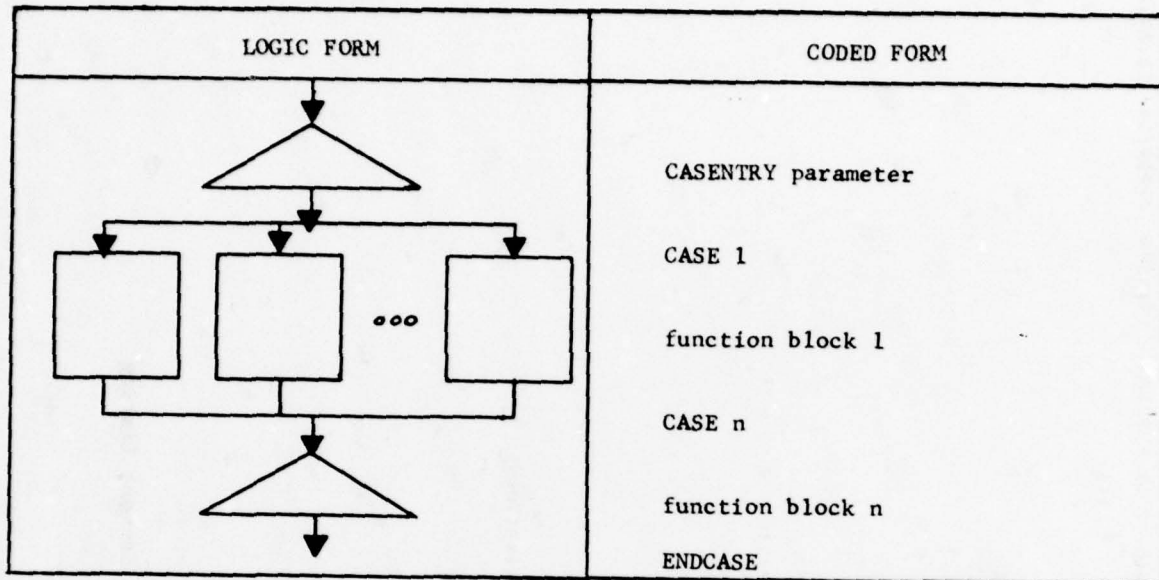


Figure 7 - CASEENTRY Logic Form

3.3 Indented Listings. One of the principal advantages accruing from top-down structured programming is the ability to generate program listings which physically identify logic structure by pairing the statements which open and close a logic form and indenting all intervening statements. Figure 8 is illustrative of an indented segment listing as prepared by the PSL. Figure 8a is an explanation of the data displayed in Figure 8.

```

79/02/02 15.18.22 PAVE PADS VERSION-79/02/28. SEGMENT = PSL.SSOSAR.PURGE.ELEMENT LVL=1ST EDN= 9
LANGUAGE=JUV TYPE=INCL VERSION=40 CREATED 77/01/10 15.36.40 BY PHBDAD CHANGED 78/09/14 12.28.23 BY 18MFIX
LIST BY SEGMENT FOR PSL.SSOSAR.PURGE.ELEMENT PPG
1 SEGMENT
2 CALL PSL.SSOSAR.GET.CPCG.PAGE
3 IF STATUS=0 GO 1
4 DO A = 0,1,NENT(CPCG.PAGE) - 1
5 IF CPG.UNIT.LONGNAME(S) EQ CPG.UNIT.LONGNAME(SAS)
6 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
7 STATUS=0
8 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
9 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
10 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
11 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
12 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
13 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
14 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
15 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
16 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
17 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
18 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
19 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
20 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
21 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
22 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
23 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
24 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
25 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
26 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
27 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
28 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
29 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
30 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
31 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
32 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
33 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
34 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
35 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
36 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)
37 IF CPG.REQUEST.LEVEL(S) EQ CPG.UNIT.LEVEL(SAS)

```

Figure 8. Example of Indented Segment Listing

Top Line - Date and Time of computer run producing this listing
- Version ID (date) of the PSL
- Name of the segment being listed
- Library level at which the segment was found
- Edition number of the segment (incremented for each change)

Second Line - Segment Language - JOV = JOVIAL
- PDL = PDL
- COMP = COMPASS
- IFTR = IFTRAN
- LEL = LEL (loader statements)
- Segment Type - INCL = INCLUDE
- MAIN = MAIN PROGRAM
- SUBR = SUBROUTINE
- LOCL = LOCAL PROCEDURE
- COMP = COMPOOL
- Segment Version (established by the user)
- Date, Time, and User ID when segment was created
- Date, Time, and User ID when segment was last changed

Third Line - Request type and library level
(LIST PROGRAM or LIST SEGMENT)
(For the example shown, a segment listing was requested
for PSL.DATA.STORAGE.AND.RETRIEVAL at the PRG level;
this particular segment was drawn down from the TST level,
as indicated on Line One.)

Left, Right Margins - Line sequence numbers used for directing modifications

Body of Listing - Card images left justified then indented to show logical
structure. (Periods are used as a visual connector for
indentation.)

Bottom Line - Repeat of Top Line

Figure 8a: Explanatory Notes for Figure 8

3.4 Hierarchical Library. The PAVE PAWS PSL is designed to support an orderly and well controlled progression of software from a development environment through integration and test into a delivered status. This is implemented as a multi-level program support library or hierarchy. Software segments are entered into the library using a user-specified name (up to 40 characters long) at a user specified level. (By convention, the first four characters of each software element name represent the Computer Program Component Group (CPCG) to which it belongs; the remainder of the 40 character name may be constructed of multiple alphanumeric syllables separated by periods.) Since each level of the library is separate and distinct from all other levels, the same software element may appear in the library at several different levels. Thus, to completely identify an item in the library it is necessary to specify both the name and level. This provides a simple mechanism for parallelism in development, error correction, and version modification. Within the PSL seven library levels are defined in a progressive hierarchy. These levels are shown in Figure 9, starting with the highest.

<u>Level</u>	<u>Usage Convention</u>
DEL	software which is in the field
FRZ	software which has been qualified
TST	software undergoing qualification test
FIX	software corrections for TST level
INT	software undergoing integration test
CPT	software undergoing group test
PRG	software under development/unit test

Figure 9. PSL Library Levels

Basic to the PSL level hierarchy are the concepts of control level and the migration of program elements from one level to another. A program element is ready to change control level when it has satisfied a predefined qualification criteria and is to be placed under more stringent change control.

This is effected within the PSL by use of an XMIT directive (see Section 3.6). All segments of a program which is being XMIT'ed will be moved to the specified level. In order to facilitate changes to segments once they have been XMIT'ed from one library level to another, the PSL includes a feature called "automatic drawdown". This feature allows library operations to be addressed to a specific library level and if the element does not exist at that level, successively higher levels will be searched until the element is found. Once found, it will be treated as if it were found at the originally requested level. This is based upon the upward migration of software through library levels and the recognition that all elements above the requested level have of necessity already satisfied the functional benchmark associated with that level.

3.5 Authorization Checking in PSL. The hierarchical nature of the PSL library system readily lends itself to the systematic application of change control procedures. Since the migration of programs from level to level requires that successively more stringent benchmarks have been satisfied, the software stability (and the corresponding authorization required to effect change) continually increases from the lowest level to the highest. This is addressed in the PSL through an authorization verification scheme which recognizes users (by an input ID) and restricts the operations and the library levels which they may use. The scheme is based upon a combination of user identity and organization and it disallows:

- a. Operations on software which is not in the province of that organization.
- b. Transactions at library levels at which the user is not authorized and,
- c. Execution of special PSL verbs for which the user is not authorized.

Among other things, implementation of this authorization check may prevent a programmer in one department from changing code belonging to another department, inhibit the Development organization from making changes to software which has been delivered to Test, prevent Test from accessing any software which has not been delivered to them, and disallow any source change activity (ADD, MODIFY) above the INT level of the library.

3.6 PSL Directives. This section provides a brief description of each of the PSL directives.

3.6.1 ADD. The ADD directive is used to add a new segment of code to the PSL library. It must specify the segment longname and the level at which the segment is to be added, in addition to a number of other items which define the segment. Following a successful ADD an indented segment list is produced automatically.

3.6.2 MODIFY. The MODIFY directive is used to make updates to code segments which are already in the PSL library. It must specify the segment longname, level, and edition number. The MODIFY directive is immediately followed by sub-directives which describe the changes to be made. Following a successful MODIFY operation the segment edition is incremented, the updated segment is written into the requested library level, and an indented segment list is generated.

3.6.3 COMPILE. The COMPILE directive initiates the pre-compiler of the PSL which performs source segment merging and forms translation before invoking the appropriate commercial compiler (JOVIAL, COMPASS, or IFTRAN). At the completion of this step the program statistics are updated in the library and a compilation listing is printed.

3.6.4 LOAD. This directive specifies that a user program consisting of NOS and LOADER CONTROL cards be precompiled and then executed. The directive must specify the longname of the user program and the library level. This function is identical to the COMPILE directive with the exception that the pre-compiled program will be executed rather than compiled.

3.6.5 COPY. The COPY directive specifies that a code segment at a specific level be copied to another segment and level. The names of the "from" and "to" segments may be different.

3.6.6 XMIT. This directive is used to deliver programs from one level to another. It specifies a program name (top-segment name), a "from" level, and a "to" level. XMIT will use the drawdown feature of the PSL to construct

the entire source program hierarchy. It will then move all of those segments up to the specified "to" level. (Segments which were drawn down from that level or above are not moved unnecessarily, however.) At the same time a full set of source listings for the program will be printed.

3.6.7 LIST. The LIST directive is used to request an indented listing. It must specify either a SEGMENT list (one segment only), a PROGRAM list (the full set of segment listings for the specified program plus a hierarchy listing which shows the program structure), or a HIERARCHY list (which generates the program structure without any segment listings). During list processing a number of error conditions are tested and if detected the segment statistics will be updated appropriately. These conditions include:

- a. source segment exceeds one page limit (56 lines - an F flag),
- b. protocol errors due to improper coding of control forms (a P flag),
- c. mixed language error if an INCLUDE'd segment is a different language (M flag)
- d. branching error if explicit branch statements are detected (B flag),
- e. COMPOOL access error if the stated ACCESS requirements do not match the design access (a C flag),
- f. syntax errors (S flag).

The indentation of each segment listing shows the direct relationship between control forms. Each line also contains a line number for reference when making MODIFY's.

3.6.8 REPORT. The REPORT directive requests that summary data be extracted from the PSL library and prepared in report format. There are three different types of reports which may be selected - SEGMENT summary, PROGRAM summary, and LIBRARY summary. (These reports are discussed in Section 3.7).

3.6.9 PURGE. This directive is used to delete segments from the library. It does not use the drawdown feature.

3.6.10 PUNCH. This directive provides a mechanism for getting card image representation of a segment out of the PSL. It is a convenient mechanism for maintaining procedure or data files.

3.6.11 CHECKPOINT. This directive causes PSL to create a checkpoint file containing every segment in the PSL.

3.6.12 RESTORE. The RESTORE directive allows the user to restore elements to the PSL library from a checkpoint file.

3.7 Management Statistics Reporting. The PSL maintains statistical data for each segment and each program in the library. Segment data is derived from the user specified values when the segment was ADD'ed (longname, shortname, language, segment type, version) or computed automatically by the PSL (creation date, date and time of last change, number of lines, ID of the user making the last change, etc.). Program data, which is associated with the top segment of each program but is distinct from its segment statistics, is computed at the time the program is either LIST'ed or COMPILE'd. It includes the date and time of the most recent segment change, the total number of segments, lines of code, and stubs in the program, the date and time at which the program was compiled, and the program object size. The REPORT directive may be used to prepare tabular summaries of either SEGMENT statistics or program statistics, examples of which are in Figures 10 and 11. (Descriptions of the contents of these reports are given in Figures 10a and 11a.) These reports are subdivided by CPCG and then by library level. Each level also contains totals as shown at the bottom of these examples.

In addition to the SEGMENT and PROGRAM REPORTS mentioned above, a LIBRARY report may be requested. This report provides very basic summary data as shown in Figure 12 as well as the Code Progression/Durability report shown in Figure 13. This latter report addresses "effective code" in the PSL library by eliminating the double-accounting which arises from multiple versions of the same segment appearing at different levels and simultaneously accommodating the drawdown feature for code which exists at a higher level. The Code Progression part of the report, which is organized as a CPCG/level matrix, indicates how much effective code exists (using drawdown as necessary) at each level of the library. Thus code (segments) which exist at the INT level of the library, "effectively" exist at the PRG and CPT levels as well. Since each of the library levels represents some sort of testing benchmark, this report allows management to answer questions like "How much code has reached functional test?", "How much code has been integrated?," "How much code has been written?"

UNIT NAME	SHATNM	LANG	TYPE	CREATED	NET GRS	DATE/TIME	CHANGED	VN	EDN	CHG	VN	OWNER	FLAGS	CHG	TOT NBR	NBR	LAST
					SZ	SZ									CHG	LN	PGR
REP- CARRIAGE-CONTROL-PLAN	SSK01	JOV	LOCL	77/02/08	7	7	77/02/08	11.10.05	A0	0	0	0	0	0	0	0	PHBLAH
REP- CARRIAGE-MINUS	SSK03	JOV	LOCL	77/02/08	7	7	77/02/08	11.10.05	A0	0	0	0	0	0	0	0	PHBLAH
REP- CARRIAGE	JOV	JOV	INCL	77/02/08	10	10	77/02/08	11.10.05	A0	0	0	0	0	0	0	0	PHBLAH
REP- JAR-INITIALIZATION	JOV	JOV	INCL	77/02/08	57	114	77/07/14	01.09.40	B0	1	4	2	13	PHBLAH	F	13	PHBLAH
REP- MEASURING-SUMMARY-OF-CLIPRAY	SSHLB	JOV	LOCL	77/02/08	11	13	77/03/17	13.16.36	A0	1	1	1	3	PHBLAH		3	PHBLAH
REP- MEASURING-SUMMARY-OF-PROGRAMS	SSHPRG	JOV	LOCL	77/02/08	13	23	77/05/23	13.28.37	B0	1	3	2	7	PHBLAH		7	PHBLAH
REP- MEASURING-SUMMARY-OF-SEGMENTS	SSHSEG	JOV	LOCL	77/02/08	17	20	77/03/17	13.16.36	A0	1	1	1	1	PHBLAH		1	PHBLAH
REP- LIBRARY-INDEX-PROCESSOR	MRPLP	JOV	LOCL	77/02/08	50	109	77/07/21	12.17.45	B0	4	6	5	50	PHBLAH		50	PHBLAH
REP- LIBRARY-PAGE-FORMATTER	MRPLF	JOV	LOCL	77/02/08	34	48	77/05/09	14.05.39	B0	1	4	2	12	PHBLAH		12	PHBLAH
REP- LIBRARY-PROGRAM-INDEX-REPORT	MRPLG	JOV	LOCL	77/04/13	22	24	77/07/21	12.17.45	B0	3	3	3	4	PH8VCG		4	PH8VCG
REP- LIBRARY-REPORT-GENERATOR	MRPLRPT	JOV	LOCL	77/02/08	34	52	77/07/14	13.55.13	A0	3	3	3	3	PHBLAH		3	PHBLAH
REP- PROGRAM-INDEX-OF-CLIPRAY	MRPLPP	JOV	LOCL	77/02/08	47	62	77/05/17	12.48.28	A0	6	6	6	17	PHBLAH		17	PHBLAH
REP- PROGRAM-INDEX-OF-PROGRAMS	MRPLPP	JOV	LOCL	77/02/08	66	121	77/08/10	13.38.14	A0	9	9	9	52	PHBLAH	F	52	PHBLAH
REP- PROGRAM-INDEX-OF-SEGMENTS	MRPLPP	JOV	LOCL	77/02/08	45	103	77/08/11	11.06.33	A0	7	7	7	39	PHBLAH		39	PHBLAH
REP- REQUEST-INITIALIZATION	JOV	JOV	INCL	77/02/08	42	92	77/08/11	11.06.33	A0	3	3	3	45	PHBLAH		45	PHBLAH
REP- SEGMENT-INDEX-OF-CLIPRAY	MRPLSP	JOV	LOCL	77/02/08	46	60	77/05/16	12.04.28	A0	6	6	6	16	PHBLAH		16	PHBLAH
REP- SEGMENT-INDEX-OF-PROGRAMS	MRPLSP	JOV	LOCL	77/02/08	42	55	77/07/14	01.09.40	B0	3	6	4	2	PHBLAH		2	PHBLAH
REP- SEGMENT-INDEX-OF-SEGMENTS	MRPLSP	JOV	LOCL	77/02/08	42	74	77/08/26	11.23.22	A0	5	5	5	6	PHBLAH		6	PHBLAH
REP- SEGMENT-PAGE-FORMATTER	MRPLSS	JOV	LOCL	77/02/08	46	105	77/08/11	11.06.33	A0	7	7	7	39	PHBLAH		39	PHBLAH
REP- SEGMENT-PAGE-FORMATTER	MRPLSS	JOV	LOCL	77/02/08	20	26	77/04/13	10.50.07	B0	0	4	1	3	PHBLAH		3	PHBLAH
REP- TABLE-ITEMS	HEAD	JOV	LOCL	77/02/08	20	22	77/04/14	13.55.13	B0	0	2	1	0	PHBLAH		0	PHBLAH
REP- TOTALS-SUMMARY-OF-CLIPRAY	SSTPRG	JOV	LOCL	77/02/08	14	28	77/05/05	16.37.22	A0	3	3	3	8	PHBLAH		8	PHBLAH
REP- TOTALS-SUMMARY-OF-PROGRAMS	SSSTSEG	JOV	LOCL	77/02/08	15	33	77/05/06	11.46.34	A0	4	4	4	10	PHBLAH		10	PHBLAH

TOTAL NUMBER SEGMENTS	NET SIZE	GRDSS SIZE	TOTAL NUMBER CHANGES	TOTAL NUMBER CHANGES - VERSION	TOTAL NUMBER LINES - VERSION
24	724	1236	90	78	332

Figure 10. SEGMENT Summary

Top Line - Date and Time of computer run producing this listing.

- Version ID (date) of the PSL
- Type of report requested
- CPGC for this page of the report
- Library Level for this page of the report

Tabular Data - Segment longname

- Segment shortname (for MAIN, SUBR, LOCL, and COMP types)
- Segment type
 - INCL = INCLUDE
 - MAIN = MAIN PROGRAM
 - SUBR = SUBROUTINE
 - LOCL = LOCAL PROCEDURE
 - COMP = COMPOOL
- Date segment was created
- Current number of lines in the segment
- Gross size of segment (includes all lines which have been deleted)
- Date and time segment was changed
- Segment Version (established by the user)
- Segment Edition (incremented for each change)
- Total number of times segment has been changed
- Number of changes made to the current version
- Number of lines (gross) for the current version
- User ID of the person who created the segment
- Special Flags
 - F = Segment exceeds one page
 - P = Protocol error
 - S = Syntax error
 - B = Branching Statements
 - M = Mixed languages
- User ID of person who last changed the segment

Summary Data - Totals for the above figures

Figure 10a: Explanatory Notes for Figure 10

74/03/02	16.18.52	PAVE PAWS PSL	VERSION 79/02/28.	SUMMARY BY PROGRAMS	CPCG= PSL.	LEVEL= TST	
UNIT NAME	SHRTNM	LANG	DATE/TIME CHANGED	NBR TOTL SEG SIZE	NBR STUB VN	EDN	INST DATE/TIME COMPILED SIZE
PSL-ADKT	PSLABT	JOV	78/01/16 15.49.20	1	7	0	AO 3 19 78/12/19 13.49.15 16
PSL-ADD	PSLADD	JOV	78/12/19 16.34.18	9	254	0	AO 13 9 78/12/19 16.34.18 290
PSL-ALPHA	PSLALF	JOV	78/12/19 16.34.18	1	51	0	AO 2 9 78/12/19 16.34.18 90
PSL-ALPH	PSLALF	COMP	78/12/12 17.27.14	1	11	0	AO 0 1 78/12/12 17.27.14 3
PSL-AUTHORIZATION-CHECKER	PSLAUT	JOV	78/11/01 22.12.07	17	521	0	AO 56 34 78/12/19 16.34.18 814
PSL-BINARY-TO-DECIMAL	PDEC	JOV	77/04/20 17.38.47	1	21	0	AO 2 10 78/12/19 13.49.15 26
PSL-CHECKPOINT	PSLCMK	JOV	78/08/09 08.28.17	12	295	0	AO 22 23 78/12/19 13.49.15 462
PSL-COMPILE	PSLCMP	JOV	78/12/11 19.39.00	8	333	0	AO 27 24 78/12/19 13.49.15 519
PSL-COMPILE	PSLC	JOV	78/12/19 16.34.18	67	925	0	AO 70 29 78/12/19 16.34.18 4979
PSL-COMPILE	PSLI	JOV	79/01/15 13.48.07	19	470	0	AO 61 38 79/01/17 17.47.45 2485
PSL-COPY	PSLCpy	JOV	78/12/19 16.34.18	4	142	0	AO 10 16 78/12/19 16.34.18 174
PSL-DATA STORAGE AND RETRIEVAL	PSLDAT	JOV	79/01/15 22.11.44	1	682	0	AO 82 13 79/01/15 22.11.44 1200
PSL-DECK-COUNTER	PSLDCR	COMP	79/02/14 17.19.44	52	1275	0	AO 52 4 79/02/14 23.41.49 4562
PSL-DIAGNOSE	PSLDIA	JOV	78/12/20 13.45.32	8	221	0	AO 31 20 78/12/20 13.45.32 619
PSL-DIAGNOSTIC-STATISTICS-POINTER	PSLSTA	JOV	77/05/26 12.43.15	11	174	0	AO 2 5 78/12/19 19.45.10 349
PSL-EFFECTIVE-CARD-PARSER	PARSER	JOV	79/01/15 13.48.07	18	497	0	AO 49 17 79/01/17 17.47.45 993
PSL-FIN-ASCII-KEYWORD	SPFIN	JOV	78/01/16 15.22.02	1	56	0	AO 4 10 78/12/19 19.45.10 70
PSL-FIN-ASCII	PSLFNI	JOV	78/12/20 13.45.32	4	133	0	AO 8 11 78/12/20 13.45.32 260
PSL-FIN-EDIT	PSLFBE	JOV	78/11/02 14.42.48	12	262	0	AO 18 20 78/12/19 19.45.10 536
PSL-FIN-EDIT	PSLFST	JOV	78/11/28 17.54.11	90	2168	0	AO 338 27 78/12/19 13.49.15 5627
PSL-MESSAGE	PSLEMR	COMP	78/12/13 22.15.58	13	657	1	AO 40 53 78/12/13 22.15.58 1180
PSL-MODIFY	PSLMOD	JOV	79/01/17 17.47.45	35	931	0	AO 119 50 79/01/17 17.47.45 2482
PSL-MOD-CONTROL-CARDS	PSLMCV	JOV	78/12/20 13.45.32	13	319	0	AO 36 36 78/12/20 13.45.32 831
PSL-PRODUCT-MANAGER	PSLMAN	JOV	77/03/15 20.59.12	3	60	1	AO 8 17 78/12/19 19.45.10 154
PSL-PRODUCT-MANAGER	PSLPCH	JOV	79/01/15 13.48.07	8	177	0	AO 24 16 79/01/17 17.47.45 224
PSL-PRODUCT-MANAGER	PSLPUR	JOV	77/12/16 19.57.26	6	118	0	AO 28 17 78/12/19 19.45.10 175
PSL-PRODUCT-MANAGER	PSLRDX	JOV	77/04/11 17.27.48	1	38	0	AO 0 8 78/12/19 19.45.10 42
PSL-PRODUCT-MANAGER	PSL5	JOV	78/08/30 12.53.51	16	211	0	AO 35 33 78/12/19 19.45.10 3699
PSL-PRODUCT-MANAGER	PSLRST	JOV	79/01/18 16.54.04	18	526	1	AO 44 24 79/01/18 16.54.04 822
PSL-PRODUCT-MANAGER	PSL4	COMP	78/01/09 14.01.30	7	234	0	AO 29 24 78/01/10 01.38.34 330
PSL-PRODUCT-MANAGER	PSL0	COMP	78/01/18 14.49.23	12	291	0	AO 35 22 78/01/18 14.49.23 299
PSL-PRODUCT-MANAGER	PSL2	JOV	78/09/26 09.38.08	38	741	0	AO 71 75 78/12/19 19.45.10 3743
PSL-PRODUCT-MANAGER	PSLXMT	JOV	77/12/12 11.59.25	2	76	0	AO 15 24 78/12/19 19.45.10 121

TOTAL NUMBER PROGRAMS	TOTAL NUMBER SEGMENTS	TOTAL SIZE	TOTAL OBJECT SIZE	TOTAL NUMBER STUBS
33	529	12877	38182	3

TOTALS

Figure 11. PROGRAM SUMMARY

Top Line	-	Date and Time of computer run producing this listing
	-	Version ID (date) of the PSL
	-	Type of report requested
	-	CPCG for this page of the report
	-	Library Level for this page of the report
Tabular Data	-	Program longname
	-	Program shortname
	-	Language - JOV = JOVIAL
	-	PDL = PDL
	-	COMP = COMPASS
	-	IFTR = IFTRAN
	-	LEL = LEL (loader statements)
	-	Date and Time of most recent segment change
	-	Total number of segments, lines, and stubs
	-	Program Version (max of all segment versions)
	-	Program Edition (sum of all segment editions)
	-	Program Instance (incremented for each compile)
	-	Date and Time Compiled
	-	Object module size (decimal words)
Summary Data	-	Totals for the above figures

Figure 11a: Explanatory Notes for Figure 11

79/03/02	16.18.52	PAVE PAWS PSL VERSION 79/02/28.	SUMMARY BY LIBRARY
LIBRARY LEVEL = PRG			
--LANGUAGE--SEGMENTS--STUBS--PROGRAMS--LINES--CHANGES--			
JOV	5	4	0 92 6
COMP	5	0	2 154 22
POL	1	0	0 1 0
LEL	5	0	5 102 27
LIBRARY LEVEL = INT			
--LANGUAGE--SEGMENTS--STUBS--PROGRAMS--LINES--CHANGES--			
JOV	154	3	16 3898 779
COMP	35	8	8 1213 140
LEL	8	1	8 119 19
LIBRARY LEVEL = FIX			
--LANGUAGE--SEGMENTS--STUBS--PROGRAMS--LINES--CHANGES--			
JOV	5	0	1 158 9
COMP	1	0	0 30 8
LIBRARY LEVEL = TST			
--LANGUAGE--SEGMENTS--STUBS--PROGRAMS--LINES--CHANGES--			
JOV	128	0	30 4500 572
COMP	63	0	5 1790 114
LEL	13	0	13 245 78
LIBRARY LEVEL = FRZ			
--LANGUAGE--SEGMENTS--STUBS--PROGRAMS--LINES--CHANGES--			
JOV	459	4	33 11907 1099
COMP	69	2	9 2781 205
LEL	4	0	2 58 8

Figure 12. LIBRARY Summary

79/03/02

16.14.52

PAVE PADS PSL VERSION 79/02/28.

SUMMARY BY LIBRARY

	*** CODE PROGRESSION ***						*** CODE DURABILITY ***					
	--PRG--	--CPT--	--INT--	--FIX--	--TST--	--DEL--	--PRG--	--CPT--	--INT--	--FIX--	--TST--	--DEL--
AJTH	158	158	158	158	0	0	0	0	0	158	0	0
BLD.	5558	5475	5475	245	245	0	0	172	0	5217	0	169
D.V.	15	0	0	0	0	0	0	15	0	0	0	0
LPC.	1710	1710	1710	1710	1710	1695	0	0	0	0	128	1582
MEEP	724	724	724	724	724	724	0	0	0	0	34	690
PSL.	14210	14176	14176	14196	14196	12327	0	162	0	0	6028	7990

Figure 13. Progression/Durability Report

The Code Durability report acknowledges the fact that segments which have already been changed at lower library levels represent a discount to the figures of the Progression report. The accounting mechanism employed in the Durability report ignores segments which have already undergone further change at a lower level, i.e., the Durability report shows management that it is dangerous to consider a segment as having been successfully integrated when it has passed the INT level of the library if it is simultaneously undergoing change at the PRG level. The value of this report lies in complementing the Progression report in allowing management to answer questions such as "How good is the code that has been developed?" and "How much effort remains to be done?". To consider an extreme example, if the library only contains ten unique segments and they have all progressed to the TST level but nine of them have new changes introduced at PRG, the code is clearly not very "durable" and the progression numbers are apparently (but not necessarily) misleading. These discrepancies can only be resolved by management understanding of the technical status of the software at the higher level and the reasons behind the changes at the lower level. To calculate "durable" lines of code, the PSL counts each unique segment only once, and that at the lowest level of the library at which it appears.

4.0 PAVE PAWS DATA COLLECTION ENVIRONMENT

The controls inherent in the Program Support Library (PSL), and in the automated Trouble Reporting (TR) System provided for ease of automatic data collection, with a minimal amount of manual effort. Program modifications were made to the Program Support Library and Trouble Reporting System to provide for data collection reports. These were provided to RADC on a periodic (monthly) basis.

4.1 PSL Changes in Support of Data Collection. The Program Support Library programs were modified to read the compiler list output and determine compiler detected errors. A special data file was added to the PSL for the purpose of saving compiler detected errors. The contents of this data file were used as inputs to a report program on a weekly basis to produce the PSL Error Reports which were provided to RADC as part of the Data Collection Effort. Impact on the PSL users was minimal, with one additional field required for compilation (compile reason code). The compile reason codes are described in Figure 14. A list of the PSL Report Data is shown in Figure 15.

4.2 TR Data Base Reports. Using the TR Data Base maintained for PAVE PAWS, special TR reports were written for the purpose of data collection. The modified TR form (described in section 4.4) was used to provide input data for these reports, which were produced on a weekly basis. There were three reports used for TR Data Collection: CPCI, CPCG, and originating organization. The number of errors by error category was provide in the TR reports.

4.3 Data Collection CPCIs. A subset of the PAVE PAWS CPCIs was used for Data Collection (CPCIs 2, 3, 4, 5). Specific CPCGs are listed in Figure 16.

4.4 Manual Data Collection Form. The Trouble Report/Change Request form was modified to support Data Collection. This was accomplished by adding the Error Category field to the form. Figure 17a shows the sample TR form, and Figure 17b the Error Categories.

COMPILE REASON CODES

- | | | |
|-----|---|---------|
| 1.1 | INITIAL PROGRAM COMPILE | INITIAL |
| | This code should be used until the program compiles without compiler detected error. | |
| 1.2 | KEYPUNCH ERROR | KEY |
| | This code should be used when keypunching errors are being corrected. | |
| 1.3 | DECK SETUP ERROR | SETUP |
| | This code should be used when the compile is to correct a deck setup error such as using the wrong COMPOOL. | |
| 2.1 | COMPUTATIONAL ERROR | COMP |
| | This code should be used when correcting computational errors such as the wrong sign or wrong trigonometric function. | |
| 2.2 | LOGIC ERROR | LOGIC |
| | This code should be used when correcting logic errors such as NQ instead of EQ. | |
| 2.3 | DATA BASE ERROR | DATA |
| | This code should be used when correcting data base errors such as tables not correctly initialized. | |
| 2.4 | I/O ERROR | IO |
| | This code should be used to correct errors in using the IO facilities such as changing reads to puts or adding necessary WAIT statements. | |
| 3.1 | SPECIFIED FUNCTION NOT IMPLEMENTED | SFNI |
| | This code should be used to insert functions whose implementation has been deliberately delayed. | |
| 3.2 | SPECIFIED INTERFACE NOT IMPLEMENTED | SINI |
| | This code should be used to insert interface code which has been deliberately deferred. | |
| 4.1 | UNSPECIFIED FUNCTION | FUNCHG |
| | This code should be used to implement new or changed functions. | |
| 4.2 | UNSPECIFIED INTERFACE | INTCHG |
| | This code should be used to implement new or changed interfaces. | |

Figure 14. COMPILE REASON CODES

- | | | |
|-----|--|----------|
| 5.1 | MEMORY OPTIMIZATION | MEMOPT |
| | This code should be used to compile changes made to improve core memory utilization. | |
| 5.2 | CPU TIME OPTIMIZATION | CPUOPT |
| | This code should be used to compile changes made to improve CPU utilization. | |
| 5.3 | LOGIC SIMPLIFICATION | LOGOPT |
| | This code should be used to compile changes made to the program to make the logic easier to understand. | |
| 6.1 | COMMENT | COMMENT |
| | This code should be used when the compile is to verify the legality of comments. | |
| 6.2 | EXTRA LISTING REQUIRED | LIST |
| | This code should be used when the compile is to obtain an extra listing or an additional listing feature e.g., generated code. | |
| 6.3 | OBJECT MODULE VERIFICATION | VERIFY |
| | This code should be used when the purpose of the compile is to guarantee that the object and source code match. This code should also be used when a common include has been changed in another program. | |
| 7.1 | COMPILER ERROR | COMPILER |
| | This code should be used when investigating or correcting internal computer errors. | |
| 7.2 | OPERATING SYSTEM ERROR | PPOS |
| | This code should be used when correcting operating system errors. | |
| 7.3 | PSL INTERNAL ERRORS | PSL |
| | This code should be used when correcting PSL internal errors. | |

Figure 14. COMPILE REASON CODES (Continued)

DATA ON PSL DATA COLLECTION STATISTICS FILE

LONGNAME OF PROGRAM
FIRST TWO COMPILER ERRORS
SHORTNAME OF PROGRAM
COMPILER CPU TIME
PRECOMPILER CPU TIME
PROGRAM SIZE IN LINES
PROGRAM OBJECT MODULE SIZE
PROGRAM EDITION
COMPILE REQUESTOR
JULIAN DATE AND TIME
COMPILE TIME ERROR COUNT
PROGRAM (TOP SEGMENT) OWNER
PROGRAM LANGUAGE
USER PROVIDED COMPILE REASON

Figure 15. Data on PSL Data Collection Statistics File

PSL DATA COLLECTION CPGs

CPCI 4	{	PSL	-	Program Support Library
		LPC	-	PreCompiler
		MREP	-	PSL Management Reports
CPCIs 2 and 3	{	COMM	-	Communications
		DISP	-	Displays
		DPCS	-	Data Processing Data Base
		MCTL	-	Mission Control
		RAM	-	Radar Manager
		RTM	-	Real Time Monitor
		RTSM	-	Real Time Simulation
		SGDB	-	SIMEX Global Data Base (CPCI 3)
		TGDB	-	TIMEX Global Data Base (CPCI 2)
		TRCK	-	Track
CPCI 5	{	TSG	-	Target Scenario Generation
		DTRD	-	Data Reduction
		PRNT	-	Print
		STRP	-	Strip
	{	SORT	-	Sort

Figure 16. PSL Data Collection CPGs

PAVE PAWS PROGRAM TROUBLE REPORT/CHANGE REQUEST

TR <input type="checkbox"/>	ORIGINATOR		DATE	CPCI	FUNCTION/CPCG	PRIORITY <input type="checkbox"/> EMERG <input type="checkbox"/> URGENT <input type="checkbox"/> ROUTINE	TR/CR NO. <input type="text"/>
CR <input type="checkbox"/>	SYSTEM BUILD ID	LEVEL	PROGRAM/DOC'T MOST AFFECTED		REF CHANGE NO(S)		
PROBLEM/CHANGE DESCRIPTION:							
TEST IMPACTED: ATTACHMENTS: <input type="checkbox"/> ON-LINE <input type="checkbox"/> DUMP <input type="checkbox"/> LISTING <input type="checkbox"/> OTHER							
BRIEF DESCRIPTION:							
ACTION ASSIGNEE		DEPT	APP'L LEVEL	<input type="checkbox"/> LCB <input type="checkbox"/> PRB <input type="checkbox"/> CCB <input type="checkbox"/> ECP	CCG PROCESSED BY: DATE:	MGR APPROVAL DATE	
CORRECTIVE ACTION DESCRIPTION:							
✓ IF IMPACTED: <input type="checkbox"/> DOC'T <input type="checkbox"/> PDL <input type="checkbox"/> CODE							
ERROR CATEGORY <input type="checkbox"/> (See Reverse Side) REJECT REASON (✓): <input type="checkbox"/> NON-PROBLEM <input type="checkbox"/> INSUFFICIENT DATA <input type="checkbox"/> DUPLICATE (REF <input type="text"/>) <input type="checkbox"/> OTHER							
<input type="checkbox"/> APPROVE <input type="checkbox"/> DISAPPROVE		<input type="checkbox"/> APPROVE <input type="checkbox"/> DISAPPROVE	<input type="checkbox"/> APPROVE <input type="checkbox"/> DISAPPROVE	<input type="checkbox"/> APPROVE <input type="checkbox"/> DISAPPROVE	<input type="checkbox"/> APPROVE <input type="checkbox"/> DISAPPROVE		
CHIEF PRGMR	DATE	LCB	DATE	PRB	DATE	CCB	DATE

Figure 17a - Sample TR Form

ERROR CATEGORIES

1. Computational Error - Error in implementation of equations
2. Logic Error - Error in decision logic
3. Data Base Error - Error in data base definition
4. Input/Output Processing Error - Error in processing data items
5. Specified function not implemented - Missing code
6. Specified interface not implemented correctly - This could apply to hardware, operating system, other programs, common data areas, etc.
7. Unspecified function required - Additional problem definition needed
8. Unspecified interface not satisfied - This could apply to hardware, operating system, other programs, common data areas, etc.
9. Memory/throughput optimization
10. Design modification/enhancement
11. Documentation change only - type C spec change/user manual/PDL
12. Key punch error
13. Deck setup - JCL/Procedure error
14. Configuration Error - i.e. Build uses mismatched code, wrong IGS package in Build, etc.

Figure 17b. Error Categories

4.5 Products. Samples of the TR reports and PSL Reports are shown in Figure 18 and 19 respectively. The compiler summary report presents tabular information for each compilation, including the CPU time of the pre-compiler and the compiler, the number of compiler errors, etc. The TR report shows the number of TR's of each error category broken down by originating organization (development, RAYTHEON, etc.) or TR series (JOVIAL, data dictionary, etc.).

PGM NAME	LEV LANG	DATE AND TIME OF COMPILE	PGM OPTION	LINES	OBJECT SIZE	COMP ERRS	COMPILE REASON	ERROR ID	OWNER	COMPILER	CPU TIME COMPILE	LPC
PSLADI	FIX JCV	22SEP79 8:41:27	54	822	001452	0	LOGIC		PHSVGG	PH8JRL	10	5.126
SSDSAR	FIX JCV	22SEP79 8:41:27	77	1257	002232	0	FUNCHG		PH8JRL	PH8JRL	10	9.870
PSLADI	FIX JCV	22SEP79 8:41:27	32	837	001405	0	FUNCHG		PH8JRL	PH8JRL	11	5.635
SSDSAR	INT JCV	14NOV78 17:51:17	80	1261	002260	0	LOGIC		PH8JRL	PH8JRL	17	9.876
PSLADI	INT JCV	14NOV78 17:51:19	40	881	001443	0	FUNCHG	JV177	PH8JRL	PH8JRL	11	5.927
PSLADI	INT JCV	14NOV78 17:51:47	38	881	001446	2	FUNCHG		PH8JRL	PH8JRL	11	5.962
PARCEA	PGG JCV	4NOV78 14: 9:52	46	773	001746	0	LISTING		PH8JRL	PH8JRL	11	5.669
PSLADI	PGG JCV	3NOV78 15:35:27	35	517	001477	1	LOGIC	JV007	PH8JRL	PH8JRL	8	3.527
PSLADI	PGG JCV	2NOV78 20:18:40	66	1062	012430	0	COMPILE		PH8JRL	PH8JRL	0	4.462
PSLADI	PGG JCV	2NOV78 20:18:40	18	368	001030	0	LOGIC		PH8JRL	PH8JRL	6	2.978
PSLADI	INT JCV	1NOV78 22:12: 7	56	822	001452	0	SETUP		PH8JRL	PH8JRL	10	5.151
SSDSAR	INT JCV	1NOV78 22:12: 7	79	1261	002260	0	SETUP		PH8JRL	PH8JRL	17	10.033
PSLADI	INT JCV	1NOV78 22:12: 7	33	837	001405	0	SETUP		PH8JRL	PH8JRL	11	5.862
PSLADI	INT JCV	1NOV78 22:12: 7	128	2755	011653	0	LOGIC		PH8JRL	PH8JRL	40	20.841
PSLADI	INT JCV	27OCT78 21: 3:41	124	2745	011616	0	LOGIC		PH8JRL	PH8JRL	40	21.066
PSLADI	INT JCV	25OCT78 14:31:27	3	26	000023	0	LOGIC		PH8JRL	PH8JRL	0	.103
PSLADI	PGG JCV	20OCT78 17:29:13	3	26	000023	0	LOGIC		PH8JRL	PH8JRL	0	.104
PSLADI	PGG JCV	20OCT78 17:29:13	3	26	000023	0	LOGIC		PH8JRL	PH8JRL	0	.111
PSLADI	PGG JCV	20OCT78 16:37:16	3	26	000023	0	LOGIC		PH8JRL	PH8JRL	0	.359
PSLADI	PGG JCV	20OCT78 16:37:16	1	84	000106	0	LOGIC		PH8JRL	PH8JRL	0	.355
PSLADI	PGG JCV	20OCT78 17: 6:39	1	84	000106	0	LOGIC		PH8JRL	PH8JRL	0	.377
PSLADI	PGG JCV	20OCT78 17:29:13	20	264	000354	0	LOGIC		PH8JRL	PH8JRL	0	2.101
PSLADI	INT JCV	29SEP78 21:35:49	54	822	001452	0	LISTING		PH8JRL	PH8JRL	10	5.078
PSLADI	INT JCV	29SEP78 21:35:49	54	716	004604	0	LISTING		PH8JRL	PH8JRL	9	6.597
PSLADI	INT JCV	29SEP78 21:35:49	78	1261	002260	0	LISTING		PH8JRL	PH8JRL	17	9.864
PSLADI	INT JCV	29SEP78 21:35:49	32	837	001405	0	LISTING		PH8JRL	PH8JRL	11	5.593
PSLADI	INT JCV	29SEP78 17:36:36	74	1261	002260	0	FUNCHG		PH8JRL	PH8JRL	17	10.069
PSLADI	FIX JCV	21SEP78 14: 2:22	31	829	001317	0	FUNCHG		PH8JRL	PH8JRL	10	5.577
PSLADI	FIX JCV	20SEP78 12:13:26	52	714	004604	0	LOGIC		PH8JRL	PH8JRL	9	6.869
PSLADI	FIX JCV	20SEP78 12:13:26	75	1248	002232	0	FUNCHG		PH8JRL	PH8JRL	17	9.705
PSLADI	FIX JCV	20SEP78 8:41:27	74	1247	002232	0	FUNCHG		PH8JRL	PH8JRL	16	9.641
PSLADI	FIX JCV	11SEP78 10:46:22	51	714	004604	0	LOGIC		PH8JRL	PH8JRL	9	7.023
PSLADI	FIX JCV	11SEP78 14:21:53	30	809	001323	0	LOGIC		PH8JRL	PH8JRL	10	5.563
PSLADI	PGG JCV	6SEP78 14:21:53	127	2745	011646	0	SETUP		PH8JRL	PH8JRL	40	20.777
PSLADI	PGG JCV	6SEP78 10: 8:18	52	823	001454	0	VERIFY		PH8JRL	PH8JRL	10	5.074
PSLADI	FIX JCV	6SEP78 10: 8:18	52	823	001454	0	LISTING		PH8JRL	PH8JRL	10	5.188
PSLADI	PGG JCV	6SEP78 14:21:53	52	715	004602	0	SETUP		PH8JRL	PH8JRL	9	6.814
PSLADI	PGG JCV	6SEP78 14:21:53	62	982	001642	0	SETUP		PH8JRL	PH8JRL	13	7.173
PSLADI	FIX JCV	31AUG78 14:26: 6	62	983	001642	0	LOGIC		PH8JRL	PH8JRL	0	6.573
PSLADI	PGG JCV	30AUG78 12:53:51	51	820	001415	0	SETUP		PH8JRL	PH8JRL	0	4.982
PSLADI	PGG JCV	30AUG78 9:55:28	48	792	001417	0	SETUP		PH8JRL	PH8JRL	11	4.864
PSLADI	FIX JCV	30AUG78 10:10:59	51	820	001445	0	FUNCHG		PH8JRL	PH8JRL	0	5.034
PSLADI	PGG JCV	30AUG78 9:55:34	22	418	000716	0	SETUP		PH8JRL	PH8JRL	8	3.025
PSLADI	PGG JCV	30AUG78 12:53:51	22	418	000713	0	SETUP		PH8JRL	PH8JRL	0	3.012
PSLADI	PGG JCV	30AUG78 15:38: 3	46	649	004524	0	SETUP		PH8JRL	PH8JRL	8	5.785
PSLADI	PGG JCV	30AUG78 15:22:16	52	715	004602	0	SETUP		PH8JRL	PH8JRL	9	6.454

Figure 18. Sample PSL Report - Compiler Summaries

THIS PAGE IS BEST QUALITY FRAGMENT
FROM COPY FURNISHED TO DDC

00 UNKNOWN	DATA DICT	DATA RED	DISPLAY	JOVIAL	PROG DEV	RAYTHEON	RCL
01 COMPUTAIN	2	13	1	36	240	14	9
02 LOGIC	0	1	0	0	13	0	0
03 I/C	0	9	11	0	219	3	17
04 DATA	12	7	1	0	155	0	0
05 SFNI	4	1	0	0	3	0	0
06 SINI	0	5	2	0	67	0	0
07 NEWFUNC	0	1	2	0	57	0	4
08 NEW INT	0	0	0	0	13	0	0
09 OPTIMIZ	0	0	0	0	9	1	0
10 REDESIN	0	0	0	0	29	0	3
11 DECOMPT	0	6	0	0	154	0	1
12 KEYPNCH	0	0	0	0	41	0	2
13 SETUP	0	0	0	0	1	0	0
14 COMPIG	0	0	1	0	7	0	0
15 OPEN	1	0	0	0	20	0	0
16 REJ D	6	12	0	0	78	0	2
17 REJ I	0	1	5	8	66	0	3
18 REJ N	0	0	0	2	1	0	2
19 REJ X	0	9	0	0	65	0	12
TOTAL	26	74	25	53	1275	19	67

Figure 19. Sample TR Report

5.0 TECHNOLOGY ASSESSMENT

This section discusses the utility and effectiveness of the development tools and techniques which were used on PAVE PAWS. For the most part, the assessment of these tools is subjective, for although PAVE PAWS has been a very successful project, the apportionment of that success to the programming team, the project management, and the technology is very imprecise. Each of the major software engineering tools which was employed is discussed separately and includes an assessment of the acceptance of that tool by the software development organization.

5.1 Top-Down Design and Development. The top-down discipline, which really becomes established during the project design stage, requires that all thought processes start by addressing system-wide issues first and then flow downward from that point. This in turn requires that system designers do their work and make their decisions before work proceeds at the subsystem level or lower. Consequently, the total system design gains visibility and credibility right from the start; all subsequent design work is viewed as refinement, clarification, or addition of detail to what has gone before. By adhering to this discipline throughout subsystem and program development, global questions are resolved first, structure and interfaces are established, and additional detail is added through a natural process of step-wise refinement. (In traditional, or bottom-up thought processes, global questions are addressed much later in time and tend to be resolved in keeping with the sum of all the micro-decisions which have already been made. Unfortunately this rarely turns out to be the best solution and some breakage of existing design or code is likely.)

Because top-down development uses a "macro" perspective and functions are initially identified by reference (an INCLUDE statement which names the desired function), a high degree of design and program code segmentation is required. In general, it is desirable to restrict each segment of code to a single page. In this way programs grow through the inclusion of new pages in an already established structure. Design updates may thus be more readily communicated and understood while program development proceeds in similar steps.

A major advantage of top-down program implementation is that the program can be compiled and executed on the day that the first segment is written!

Although this segment by itself may not actually do much more than initialize program variables, name the functions which are to be performed in that program, and exit, the program can be debugged of errors in syntax and compiler control statements immediately. It can be integrated with other programs in the system to test their interaction as well. Since the subsequent development of lower-level segments is only a refinement to an existing structure program testing can be accomplished continually, providing a regression test of existing code and incremental testing of new segments. Additionally, since control sections and data paths will be established early in the top-down approach, much less emphasis is placed on test driver programs.

The system perspective afforded by top-down techniques was very advantageous throughout the design phase of PAVE PAWS. Not only is this the proper perspective for software designers, but it is probably the single most effective perspective from which to present design to systems engineers, management, and the customer. Furthermore, since successive levels of design represent greater and greater degrees of detail, design reviews or presentations may be quite readily tailored to suit the needs of the audience by eliminating those levels which are too detailed.

During code development on PAVE PAWS most programmers began a series of compilations as soon as the first two or three segments were coded. In addition to providing early identification of syntax and data usage errors, this provided a welcome diversion from endless hours of coding. The compiler cross-reference listings also provide a very convenient point of reference for data item utilization when coding additional segments. Unit testing was begun as soon as a complete function was coded and testing results thus began to accrue much earlier than in traditional projects.

One last and very significant advantage accruing from top-down development is the easing of software development schedule interdependencies. Since the top level of each program is written very early in the game, interface testing is begun immediately and individual programs may be fully developed and tested while using only rudimentary versions of related programs.

5.2 Structured Coding. Although structured coding has been a controversial subject in the past, it is currently well accepted by the programming community. The requirement to restrict program logic statements to a standardized set of control forms and the prohibition against programmer generated branch instructions is one of the most significant advances in recent years. Suddenly programs can be read, understood, and debugged by someone other than the author! Additionally, because the code must be straight-forward in its logic flow, there are not as many hiding places for program bugs as there once were! Both of these are very important advantages of structured coding although it is again very difficult to quantify their effect. The benefits of standardization are felt very strongly during the project design phase when non-programmers form a significant part of the audience, and again in the maintenance period of the project, when a small number of people are assigned to maintain a large amount of code. The improved software quality assurance which derives from a lower incidence of program bugs due to the use of structured coding is a phenomenon which begins with the software design and stays with the software throughout its lifetime. It should also be pointed out here that part of the value attributed to structured coding comes about from program segmentation and the use of indented segment listings, which together serve to make program logic very apparent to the reader.

As one last rejoinder to the standard argument against structured coding, it must be noted that PAVE PAWS successfully met stringent real-time memory and throughput criteria. Although this did require the use of assembly language coding for a few very highly used subroutines, in no case was an argument put forth to violate structured coding techniques in order to achieve better performance. It is suspected that unstructured programs which are

tricky in an attempt to improve performance are likely to incur a performance reduction because of the overhead involved in making the tricks work. It is widely acknowledged that such programs will be extremely difficult to debug and maintain by other than the original program author.

5.3 Indented Segment and Program Listings. Given a highly segmented program and the use of structured programming, indented listings which graphically show the logic of a program are a valuable addition. (Refer to Figure 8 in Section 3 for an example.) The primary virtue of these listings is the almost instant comprehension of program logic structure, particularly in "either - or" cases. Note that by limiting segment sizes to 56 lines (one page), the likelihood of nested indentation pushing a card image too far to the right to be printed is almost negligible (in fact, this has never occurred on PAVE PAWS).

Indented program listings are constructed by the PAVE PAWS PSL as an ordered collection of indented segment listings. Figure 20 represents a typical segment structure of a program where each block represents a segment and the order of printed segment listings is indicated. As an additional convenience, an indented "hierarchy" listing is printed in the front of each indented program listing. The hierarchy simply shows the relationship between the segments of the program and any subroutines which are called.

The physical structure of an indented program listing makes it an effective medium for design and code reviews. The limitation of segment size to a single page allows complete review of a single segment before selecting the path to be followed and essentially increasing the "magnification" being used. Surprisingly enough, these same features make indented program listings equally effective for debugging. Referring back to Figure 20, it can be seen that a bug in the lowest level segment in this structure can be reached from the top segment by going through no more than four segments. Assuming that the program is structured along functional lines, isolating a program logic bug to a single segment of code is usually a very straight forward procedure.

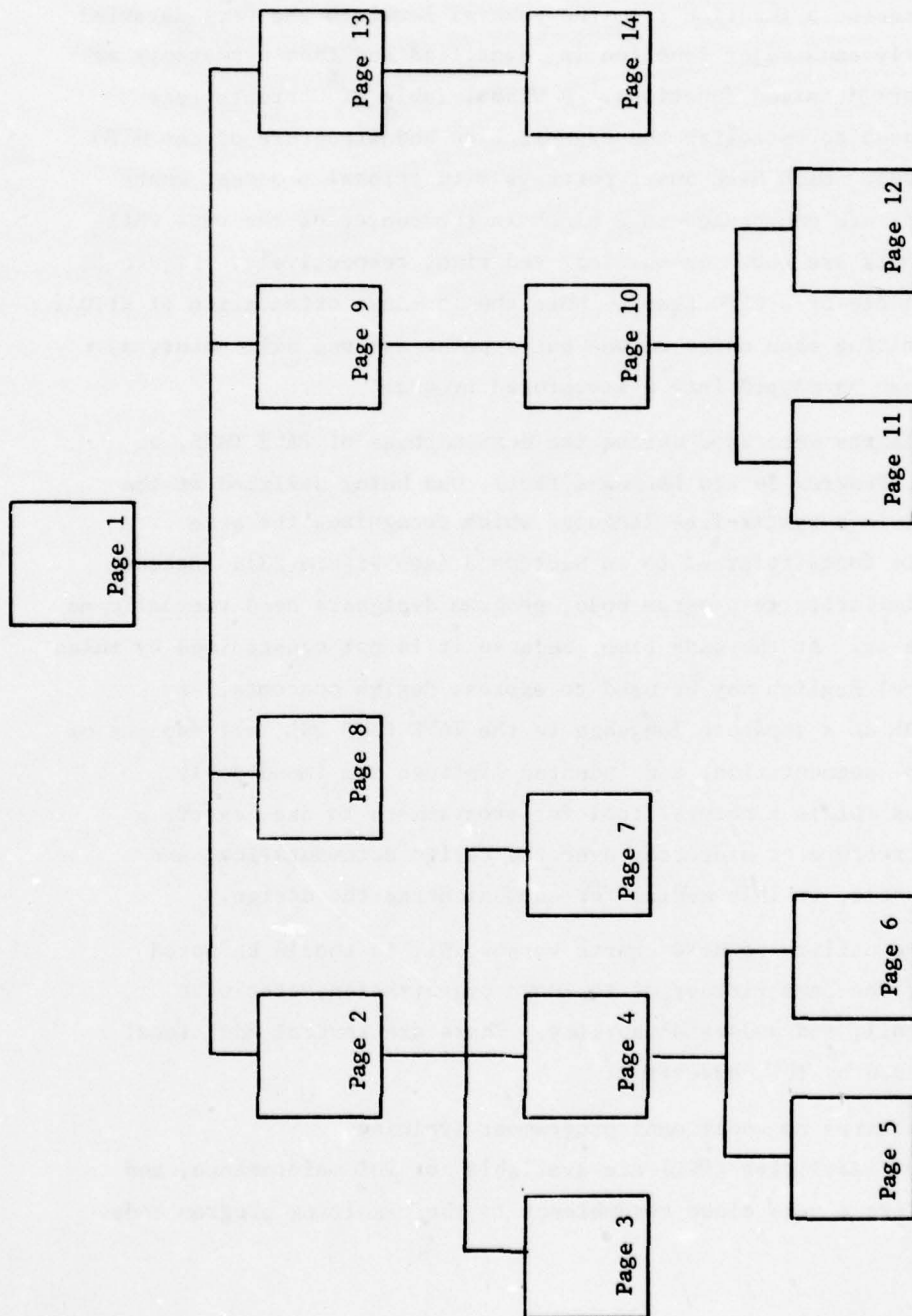


Figure 20. Program Segment Structure

5.4 Program Design: HIPO and PDL. Hierarchy plus Input-Process-Output (HIPO) is a documentation technique consisting of a set of diagrams which graphically describe a function from the general level to the very detailed level. Initially each major function is identified and then repeatedly subdivided into more detailed functions. A Visual Table of Contents (see Figure 21) is used to establish the organization and structure of the HIPO charts themselves. Each HIPO chart portrays a functional process, where processing steps are enumerated in a block in the center of the page while inputs and outputs are shown on the left and right respectively. Figure 22 provides an example of a HIPO chart. Note the top-down orientation of HIPO's and that by limiting each chart to one entry point and one exit point, a HIPO function can be mapped into a structured program!

Although HIPO charts were used during the design phase of PAVE PAWS, a companion tool, Program Design Language (PDL), was being utilized at the same time. PDL is a syntax-free language which recognizes the same structured logic forms referred to in Section 3 (see Figure 23). Because of its great similarity to program code, program designers need virtually no training to use it. At the same time, because it is not constrained by rules of syntax, normal English may be used to express design concepts. By implementing PDL as a separate language in the PAVE PAWS PSL, all aspects of top-down design, segmentation, and indented listings are immediately available. Thus PDL is a natural tool for programmers to use, exerts a well defined structure or hierarchy over the design documentation, and provides a readable, visible medium for communicating the design.

In comparing the utility of HIPO charts versus PDL, it should be noted that they share the same virtues of top-down organization, step-wise addition of detail, and understandability. There are several additional advantages offered by PDL, however -

- a. PDL requires no additional programmer training,
- b. Support facilities (PSL) are available for PDL maintenance, and
- c. PDL bears a very close resemblance to the resulting program code.

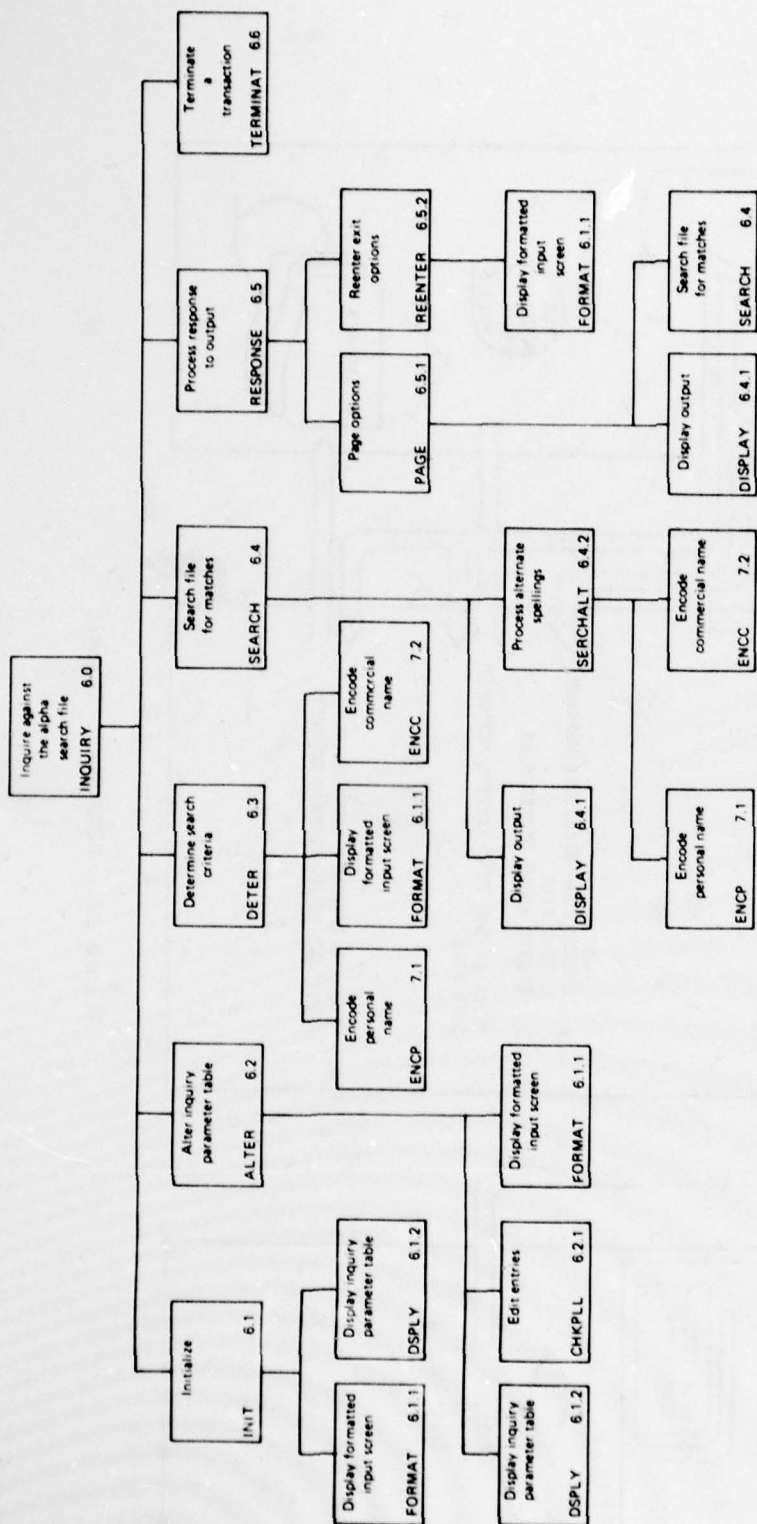


Figure 21. Visual Table of Contents - Example

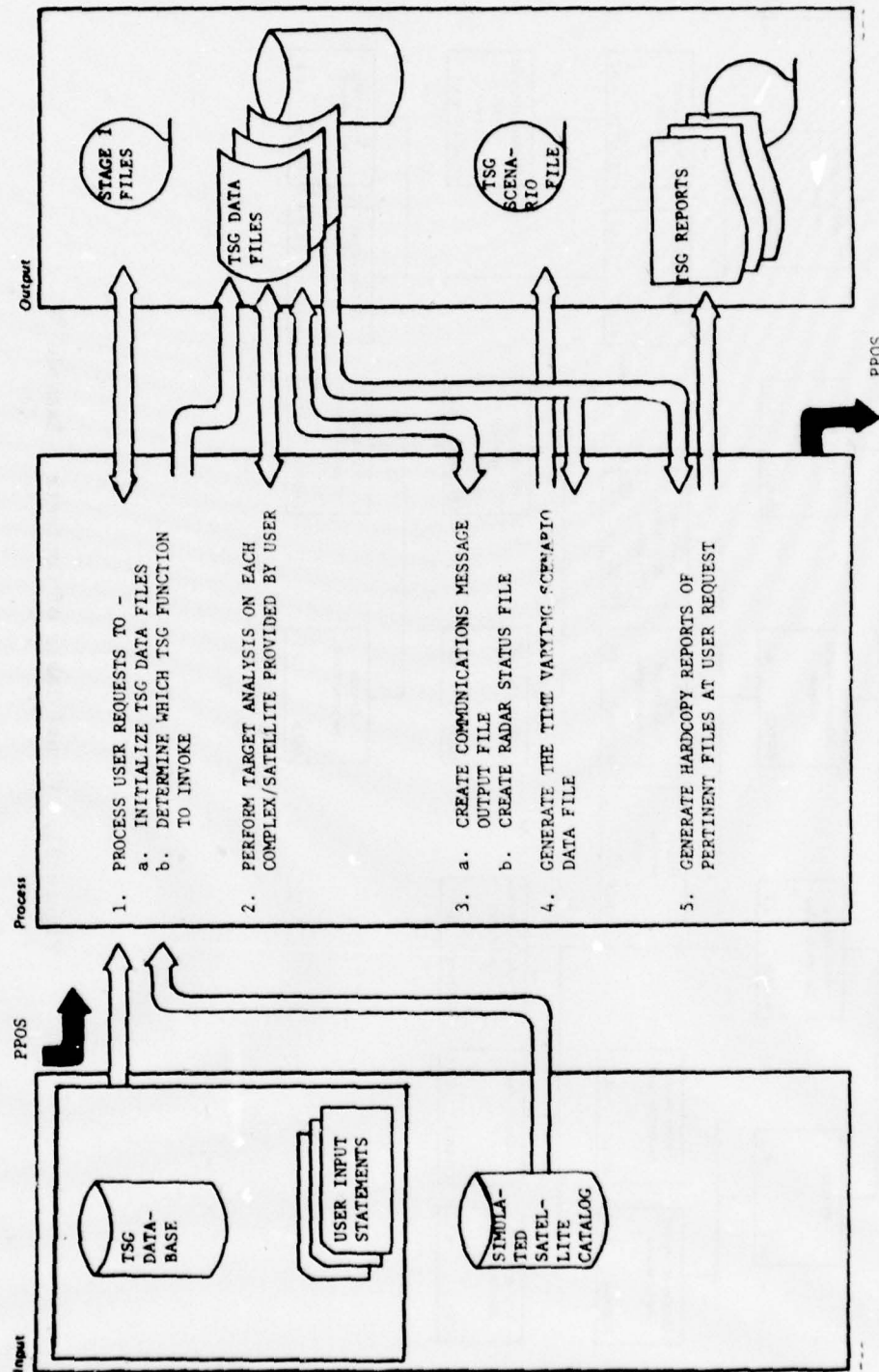


Figure 22. HIPO Chart Example

```

79/03/02 15.27.05 PAVE PAWS VERSION-79/02/28. SEGMENT = PSL-DIRECTIVE.CARD.PARSER LVL=FRZ EDN= 0 PARSER
LANGUAGE=PDL TYPE=SUBR VERSION=A0 CREATED 77/09/06 17.33.43 BY PH8WBV CHANGED 77/09/06 17.33.43 BY
LIST BY PROGRAM FOR PSL-DIRECTIVE.CARD.PARSER PRG PAGE = 1
1 SEGMENT PSL-DIRECTIVE.CARD.PARSER
2 * INITIALIZE OUTPUT ARRAYS (5 ELEMENTS FOR EACH PARAMETER TYPE - ALLOWS REPEATS)
3 * GET SCAN PARAMETERS FROM COMPOOL (SCAN INITIATED BY PSL.CONTROL)
4 * RETURN CODE = 63
5 *
6 * START WITH FIRST REQUESTED OPERAND IN CALLING SEQUENCE
7 * DO WHILE MORE OPERANDS ON INPUT CARDS)
8 * * OUTPUT INDEX = REPEAT COUNT FOR THIS OPERAND
9 * * IF OPERAND CHAR'S 5R 0 OR OPERAND IS REQUIRED
10 * * INCLUDE PSL-DIRECTIVE.PARSER
11 * * ENDIF
12 * * OPERAND NUMBER = OPERAND NUMBER + 1
13 * * REPEAT COUNT = REPEAT COUNT + 1 (FOR THIS OPERAND)
14 * * CALL PSL.CARD.SCAN TO EXTRACT NEXT FIELD
15 * * ENDOO - WHEN NO MORE OPERANDS
16 * * DO WHILE NOT END OF LIST (PROCESS REMAINING CALLING SEQUENCE)
17 * * * IF OPERAND IS REQUIRED
18 * * * * RETURN CODE = 63
19 * * * * CALL PSL.MESSAGE TO PRINT DIAGNOSTIC - MISSING PARAMETER
20 * * * * ENDIF
21 * * OPERAND NUMBER = OPERAND NUMBER + 1
22 * * ENDOO
23 * ENDOO PSL-DIRECTIVE.CARD.PARSER

```

Figure 23. Indented PDL Program Listing

```

77/03/02 15:27:02 PAVE PARS VERSION-77/02/28. SEGMENT * PSL-DIRECTIVE.PARSER
LANGUAGE=POL TYPE=INCL VERSION=AO CREATED 77/09/06 17:33:43 BY PH6ABV
LIST BY PROGRAM FOR PSL-DIRECTIVE.CARD.PARSER
PAGE 2
1 10000000 PSL-DIRECTIVE.PARSER
2 10000000 CASE=KEYWORD
3 10000000 CASE=KEYWORD
4 10000000 CASE=KEYWORD
5 10000000 CASE=KEYWORD
6 10000000 CASE=KEYWORD
7 10000000 CASE=KEYWORD
8 10000000 CASE=KEYWORD
9 10000000 CASE=KEYWORD
10 10000000 CASE=KEYWORD
11 10000000 CASE=KEYWORD
12 10000000 CASE=KEYWORD
13 10000000 CASE=KEYWORD
14 10000000 CASE=KEYWORD
15 10000000 CASE=KEYWORD
16 10000000 CASE=KEYWORD
17 10000000 CASE=KEYWORD
18 10000000 CASE=KEYWORD
19 10000000 CASE=KEYWORD
20 10000000 CASE=KEYWORD
21 10000000 CASE=KEYWORD
22 10000000 CASE=KEYWORD
23 10000000 CASE=KEYWORD
24 10000000 CASE=KEYWORD
25 10000000 CASE=KEYWORD
26 10000000 CASE=KEYWORD
27 10000000 CASE=KEYWORD
28 10000000 CASE=KEYWORD
29 10000000 CASE=KEYWORD
30 10000000 CASE=KEYWORD
31 10000000 CASE=KEYWORD
32 10000000 CASE=KEYWORD
33 10000000 CASE=KEYWORD
34 10000000 CASE=KEYWORD
35 10000000 CASE=KEYWORD
36 10000000 CASE=KEYWORD
37 10000000 CASE=KEYWORD
38 10000000 CASE=KEYWORD
39 10000000 CASE=KEYWORD
40 10000000 CASE=KEYWORD
41 10000000 CASE=KEYWORD
42 10000000 CASE=KEYWORD
43 10000000 CASE=KEYWORD
44 10000000 CASE=KEYWORD
45 10000000 CASE=KEYWORD
46 10000000 CASE=KEYWORD
47 10000000 CASE=KEYWORD
48 10000000 CASE=KEYWORD
49 10000000 CASE=KEYWORD
50 10000000 CASE=KEYWORD
51 10000000 CASE=KEYWORD
52 10000000 CASE=KEYWORD
53 10000000 CASE=KEYWORD
54 10000000 CASE=KEYWORD
55 10000000 CASE=KEYWORD

```

Figure 23. Indented PDL Program Listing (Cont'd)

```

79/03/02 15.27.05 PAVE PAWS VERSION-79/02/28. SEGMENT = PSL.PARSE.KEYWORD LVL=FRZ EDN= 0
LANGUAGE=POL TYPE=INCL VERSION=A0 CREATED 77/09/06 17.33.43 BY PH8BVB CHANGED 77/09/06 17.33.43 BY
LIST BY PROGRAM FOR PSL.DIRECTIVE.CARD.PARSER PRG PAGE = 10
1 SEGMENT PSL.PARSE.KEYWORD 1
2 IF NUMBER OF CHARACTERS IS BETWEEN 1 AND 10 2
3 KEYWORD(OUTPUT#INDEX) = FIELD 3
4 IF LAST DELIMITER WAS EQUALS (=) 4
5 INCLUDE PSL.CARD.SCAN TO EXTRACT KEYWORD#TEXT SEGMENT 5
6 IF NO MORE DATA OR PARENTHESES BALANCE OR RETURN#CODE = NOGO 6
7 SCAN CHARACTERS IN KEYWORD#TEXT SEGMENT 7
8 ADD ONE TO PAREN#COUNT FOR LEFT PARENTHESIS 8
9 SUBTRACT ONE FROM PAREN#COUNT FOR RIGHT PARENTHESIS 9
10 IF NO MORE THAN 40 CHARACTERS IN KEYWORD#TEXT TOTAL (ALL SEGMENTS) 10
11 MOVE THIS SEGMENT INTO KEYWORD#TEXT 11
12 UPDATE TOTAL CHARACTER COUNT 12
13 ELSE 13
14 CALL PSL.MESSAGE TO PRINT DIAGNOSTIC - INVALID KEYWORD TEXT STRING 14
15 RETURN#CODE = NOGO 15
16 ENDIF 16
17 CALL PSL.CARD.SCAN TO EXTRACT NEXT INPUT FIELD 17
18 ENDD 18
19 KEYWORD#LENGTH(OUTPUT#INDEX) = TOTAL CHARACTER COUNT 19
20 IF PAREN#COUNT IS NOT ZERO OR TOTAL CHARACTER COUNT IS ZERO 20
21 CALL PSL.MESSAGE TO PRINT DIAGNOSTIC - FIELD IMPROPERLY TERMINATED 21
22 RETURN#CODE = NOGO 22
23 ENDD 23
24 ELSE 24
25 CALL PSL.MESSAGE TO PRINT DIAGNOSTIC - INVALID KEYWORD 25
26 RETURN#CODE = NOGO 26
27 ENDD 27
28 ENDSEG PSL.PARSE.KEYWORD 28

```

Figure 23. Indented FDL Program Listing (Cont'd)

77/03/02 15.27.05 HAVE PAWS VERSION-77/02/28. SEGMENT = PSL.CARD.SCAN

LANGUAGE=POL TYPE=LOCL VERSION=AO CREATED 77/09/06 17.33.43 BY PH8WBV

LIST BY PROGRAM FOR PSL-DIRECTIVE.CARD-PARSER

```

1 SEQUENT PSL.CARD.SCAN
2 * IF ANOTHER OPERAND IS INDICATED (FIRST DELIMITER = COMMA)
3 * IF NEXT CHARACTER IS BLANK OR END-OF-CARD
4 * CALL PSL-HEAD.INPUT.CARD - DIRECTIVE IS CONTINUED ON NEXT CARD
5 * IF THE CARD CONTAINS $ IN COLUMN 1
6 * CALL PSL-MESSAGE - INVALID CONTINUATION CARD
7 * RETURN CODE = NOGO
8 * ELSE
9 * CALL PSL-MESSAGE TO PRINT CONTINUATION CARD
10 * CALL PSL-SCAN.ASCII.STRING TO LOCATE FIRST NON-BLANK CHARACTER
11 * SET SCAN PARAMETERS TO POINT TO FIRST NON-BLANK CHARACTER
12 * ENDIF
13 * ENDIF
14 * CALL PSL-SCAN.ASCII.READ TO EXTRACT NEXT DATA FIELD (MAY BE NULL)
15 * ELSE
16 * SET NUMAORC OPERANDS CONDITION TO TERMINATE PROCESSING
17 * ENDIF
18 ENDOF PSL.CARD.SCAN

```

Figure 23. Indented PDL Program Listing (Cont'd)

Although this suggests that PDL be used exclusively instead of HIPO, a more temperate conclusion is appropriate - don't use PDL and HIPO to meet the same objectives. Experience on PAVE PAWS indicates that HIPO charts can be used effectively at the system and subsystem level but become cumbersome and redundant with PDL when taken more than the first few levels deep.

It is also appropriate to comment on the maintenance of design documentation. Experience on PAVE PAWS indicates that HIPO's and PDL (or their equivalent) are not only useful but necessary for the software design, development, management, and procurement communities during the design phase of the project. It provides the technical foundation for the entire development period while simultaneously serving as the means by which technical direction and scope are communicated and understood throughout the project. As the implementation cycle begins, however, questions and changes arise which require deviation from the documented design. This is a natural phenomenon which should not cause undue concern as long as the basic design intent is still intact. Under these circumstances there is no immediate need to update the design documentation - the procuring agency and the project management understand the design on a conceptual level while the programmers reflect design variations directly in the code. When and why, then, is the software design documentation ever updated? The only apparent reason to update and reissue software design documentation are -

- a. To correct the documents of record.
- b. To establish an effective mechanism to communicate design to a project newcomer.
- c. To provide a bridge between system concepts and implementation for a maintenance group.

Assuming that one or more of these conditions holds, it is the author's opinion that the cost of updating design documentation is minimized by performing that function as seldom as is necessary to satisfy the users. This includes a "hands-off" approach while the software is developed or changed, followed by a periodic review, update, and republication to bring the design and the product back together again.

5.5 Hierarchical Library. The hierarchical library implemented in the PAVE PAWS PSL (see Section 3.4) was extremely useful throughout the development and test phases of the project. The separation afforded by the various levels provided stability at the upper levels with complete freedom of change at lower levels. Figures 24 thru 28 give an example of the progress of a single program through the lowest three levels of the library. In Figure 24, the program top segment has been coded and entered into the library at the PRG level. In the example shown, this segment references (via INCLUDE) four other segments, for which stubs (placeholders) are created. Following successful compilation of this program it was XMIT'ed to the CPT level where it was to undergo group testing under the control of the Chief Programmer. This is reflected in Figure 25. Figure 26 portrays the ongoing code development being done by the programmer at the PRG level. Note that this in no way affects the group testing being done at a higher level. Figure 27 indicates that the Chief Programmer was able to perform satisfactory group testing despite the fact that the majority of the function of this program was not yet implemented. With the concurrence of the integration team the program has been moved from CPT to INT and subsequently the program at PRG was moved to CPT. Figure 28 now shows the entry of two new segments at the PRG level, but more ominously, also shows changes to existing segments. Happily enough, these changes are still segregated from users of higher levels - they retain full control over the program configuration for the library level at which they are working. At this point it is helpful to point out the effective configuration at each level of the library -

at INT - T_0 /stub/stub/stub/stub

at CPT - $T_0/A_0/B_0$ /stub/stub

at PRG - $T_1/A_1/B_0/C_0/D_0$

LEVEL	TOP SEGMENT	SEGMENT A	SEGMENT B	SEGMENT C	SEGMENT D
INT					
CPT					
PRG	<div style="border: 1px solid black; padding: 5px; display: inline-block;">T₀</div>	STUB	STUB	STUB	STUB

Figure 24. Program Configuration After Entry of Initial Segment

LEVEL	TOP SEGMENT	SEGMENT A	SEGMENT B	SEGMENT C	SEGMENT D
INT					
CPT	<div style="border: 1px solid black; padding: 5px; display: inline-block;">T₀</div>	STUB	STUB	STUB	STUB
PRG					

Figure 25. Program Configuration After XMIT to CPT Level

LEVEL	TOP SEGMENT	SEGMENT A	SEGMENT B	SEGMENT C	SEGMENT D
	<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">T_o</div>	STUB	STUB	STUB	STUB
		<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">A_o</div>	<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">B_o</div>		

Figure 26. Program Configuration After Entry of Segments A and B

LEVEL	TOP SEGMENT	SEGMENT A	SEGMENT B	SEGMENT C	SEGMENT D
	<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">T_o</div>	STUB	STUB	STUB	STUB
		<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">A_o</div>	<div style="border: 1px solid black; width: 50px; height: 50px; margin: 10px auto; text-align: center; line-height: 50px;">B_o</div>		

Figure 27. Program Configuration After Subsequent XMITs

LEVEL	TOP SEGMENT	SEGMENT A	SEGMENT B	SEGMENT C	SEGMENT D
	<div>T₀</div>	STUB <div>A₀</div>	STUB <div>B₀</div>	STUB <div>C₀</div>	STUB <div>D₀</div>
	<div>T₁</div>	<div>A₁</div>			

Figure 28. Program Configuration After Further Changes

The concept of library levels and their usage ties in very closely with change control authorizations. Note in the example above, that neither the programmer (sender) nor the Chief Programmer (receiver) can unilaterally decide to do an XMIT - this must be a joint decision where the sender offers a product (together with assurances and disclaimers) and the receiver agrees to forego the stability (or instability) of his current product and accept a new one. This need to establish change authorization by level is effectively carried out as described in Section 3.5. The following sections describe how each of the seven library levels is utilized on PAVE PAWS.

5.5.1 Usage of the PRG Level. This level of the library is essentially used for program development. No special authorization is required either to enter new segments of code or to make changes to existing segments. Code in this level is subject to both frequent and extensive change, consequently this is the least stable level of the library.

5.5.2 Usage of the CPT Level. The CPT level is under control of the Chief Programmer and is generally used to provide more stability than is offered at the PRG level. It may be used as the first point of program integration or it may be used to make high confidence or localized changes separately from the code at PRG. The authorization scheme in the PSL allows each Chief Programmer to perform XMIT's to the CPT level. No additional procedural constraints are placed upon this transaction due to the close working relationships within a Chief Programmer Team.

5.5.3 Usage of the INT Level. A separate integration team was utilized on PAVE PAWS to perform basic integration testing at the system level. Their responsibility was to establish stable and rational system operation in order to allow the functional test team to begin their testing. Although the integrators were authorized to XMIT code to the INT level, a formal procedure was followed to ensure documentation of software deliveries, including a list of all problems which had been corrected. This procedure required that the Chief Programmer list all the programs to be XMIT'ed together with a list of all problems corrected on a Software Change Release Notice (SCRN). The SCRNs were then signed by the manager (leader) of the integration team before the delivery was performed.

5.5.4 Usage of the FIX Level. This level, which is controlled by the Functional Test Group, is a low-traffic level containing specific corrections for software at the next higher level (TST). Changes can be made directly at this level if necessary to fix specific urgent problems. XMIT's of individual programs may also be performed following the SCRN procedure with the concurrence of the Functional Test manager. This level is separate from the TST level to avoid those situations where "the cure is worse than the disease".

5.5.5 Usage of the TST Level. This is the primary level of interest for the Functional Test group. It is highly stable and is the level from which the Qualification Tests are normally run. The emphasis at this level is to push the entire system to its next functional performance benchmark.

5.5.6 Usage of the FRZ Level. The FRZ level, which on PAVE PAWS is under control of the prime contractor, is used for deliveries from TST following successful completion of Qualification Testing. Software at this level is under ECO/ECP control.

5.5.7 Usage of the DEL Level. This level contains the software configuration which is operational. It is controlled by the acquiring agency.

5.6 Chief Programmer Team/Librarian Operations. As implemented on PAVE PAWS, Chief Programmer Teams require a very heavy technical involvement on the part of the Chief Programmer in software design, implementation of key programs, and review of the design and code of other members of the team. In general this included one or two key Backup Programmers who developed their own areas of specialization. Although the management responsibilities of the Chief Programmers detracted somewhat from their technical efforts, it seems important that the person making schedule and resource decisions (the manager) be as technically involved as possible. This makeup of a Chief Programmer Team was successful on PAVE PAWS and would be recommended for use on other projects.

Although Programmer Librarians were used on PAVE PAWS, they were not used in the classical role. Current literature calls for the Librarian to perform all the keypunching, job submittal, and filing of listings for a single Chief Programmer Team. The Librarian's role is to act as the central clearing house for all these operations. On PAVE PAWS although

the Librarian performed all of these services they did not act as the single focal point. This came about in part because the number of librarians could not keep up with heavy keypunch demands and as second and third shift operations increased, programmers were left more and more to their own devices. Contrary to popular opinion, it is not a total waste for a programmer to perform his own keypunching (within reason). It gives him the chance to simultaneously review his coding and correct coding or logic errors on the spot.

5.7 Structured Design/Structured Code Walkthroughs. Structured Walkthroughs were used extensively on PAVE PAWS with great success. Segmented, structured code with indented listings are an excellent vehicle for communicating design or implementation. An important distinction should be made, however, between the purpose of a design review and the purpose of a code review. A design review should be oriented toward presenting program design to a team of people (including systems engineers, customer personnel, and testers) and soliciting their comments on its validity, completeness, etc. A code review on the other hand should involve at most two people other than the author and should be done with a great deal of attention to detail, even going so far as to detect syntax and data usage errors. Done in this fashion, code reviews are not only informative but highly productive. In both types of reviews, indented listings are provided for each member of the audience and the author acts as a moderator in explaining the design or code. The author should maintain a record of all unanswered questions and discrepancies which then becomes an action item list at the conclusion of the review.

5.8 Management Statistics Collection/Reporting. The reporting capabilities of the PAVE PAWS PSL as described in Section 3.7 were of limited use to the programmers and Chief Programmers. Reports were used as a confirmation following a major delivery but were only rarely referred to in other instances. Middle and upper management made religious use of the Progression and Durability reports however. This is a natural phenomenon

when you recognize that programmers view progress in terms of solving technical problems while management is less concerned with "the problem of the day" and is more interested in demonstrated rates of progress. Coding and testing rates can be realistically measured by plotting the outputs of these reports. Figure 29 shows prototype software development curves for the theoretical case and for phased deliveries. Figures 30 through 37 present actual data for CPCI 2 as experienced on PAVE PAWS. Figures 38 through 40 similarly provide data for CPCI-3. The major Qualification Test dates have been added to these figures and clarifying foot notes have been added wherever possible.

5.9 Qualification Test Program. The Qualification Test program for PAVE PAWS followed Military Standards for Preliminary Qualification Tests (PQT's) and Formal Qualification Tests (FQT's) and was carried out by a separate Functional Test organization. Each CPCI had one or more PQTs and an FQT. Performance requirements were mapped from the Part I Development Specification into Test Procedures for each test and then test scripts were developed to guide the conductance of each test. One early mistake on PAVE PAWS was to structure the PQTs along CPCG lines. This was not practical for several reasons: CPCGs don't normally execute all by themselves, and software development plans call for parallel development, which would result in PQTs being bunched at the end of the program. This approach was corrected by using the software development plan to determine what functions would be completed at various times and then defining a Test Procedure to address those functions. This allowed fairly even spacing of fully integrated functional tests.

The advantage offered by having a separate test organization is considerable. A comprehensive test program requires a considerable amount of planning, organization, and documentation as well as the tasks involved in actually running the tests and performing post-test analysis. These efforts can thus be accomplished without detracting from the programmer's day to day activities while at the same time a separate organization provides an independent outlook with respect to test plans and results. It is clear from PAVE PAWS experience that this is a key ingredient to a successful program.

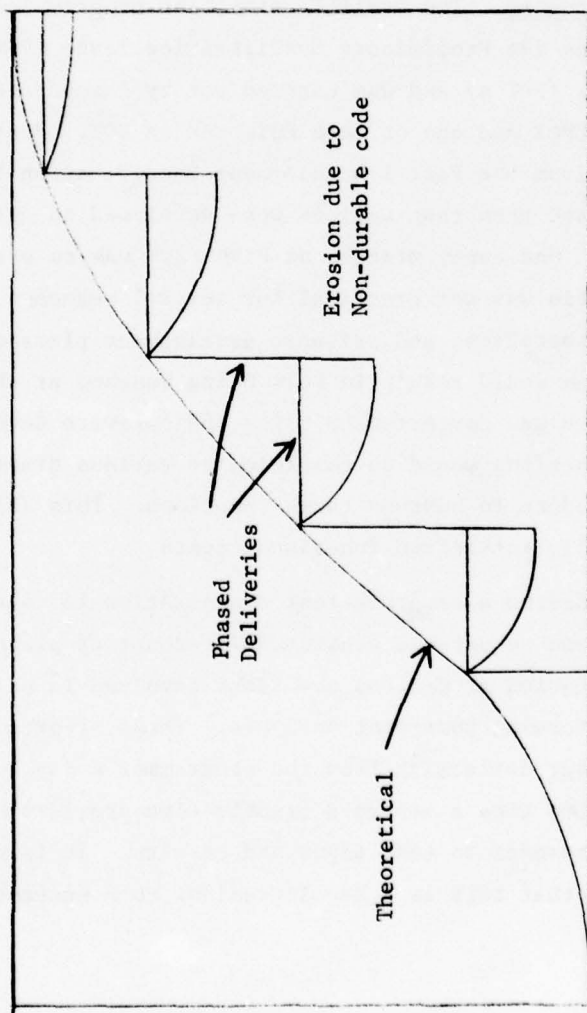


Figure 29. Code Development Curves

C P C I 2

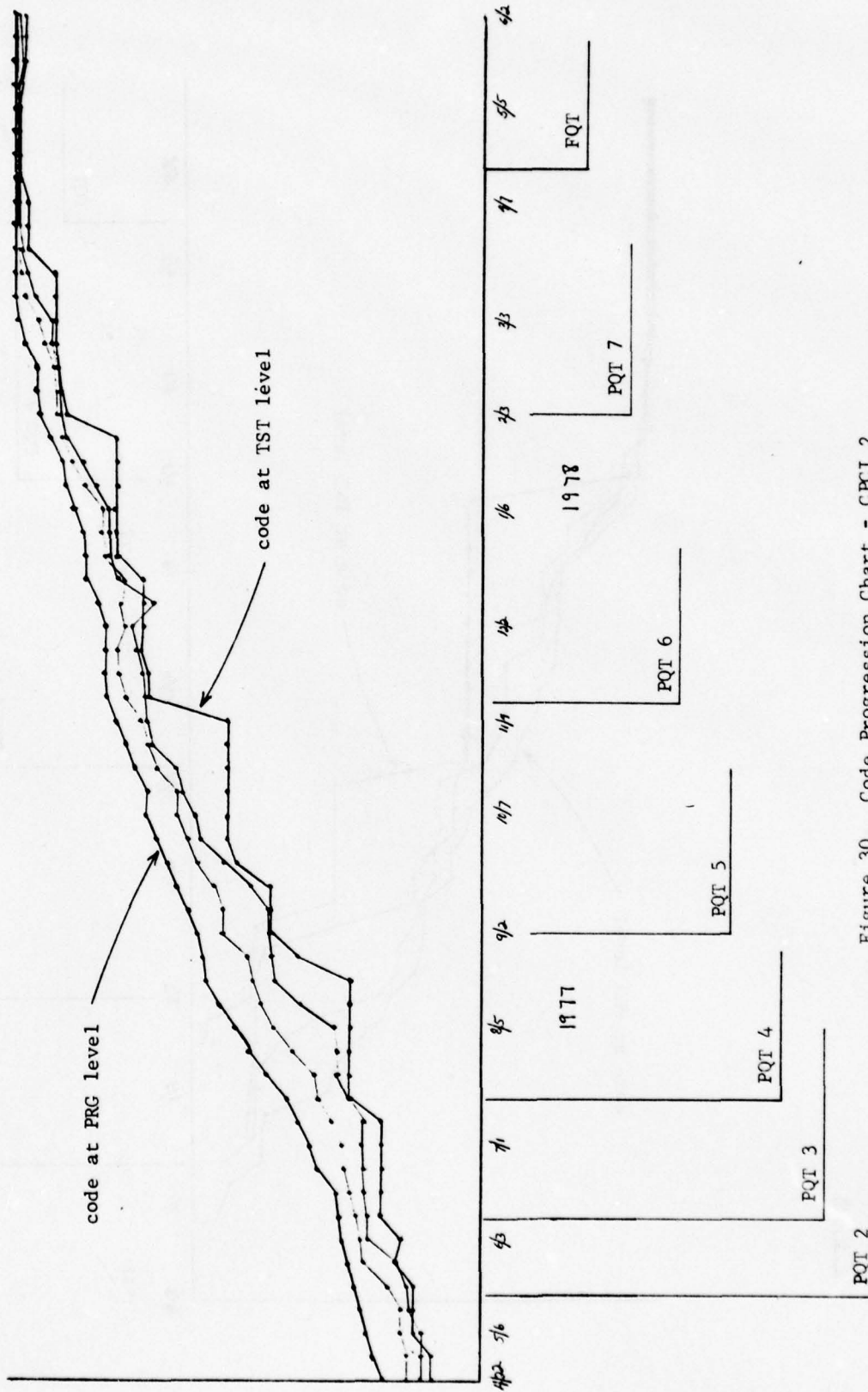


Figure 30. Code Progression Chart - CPCI 2

COMM

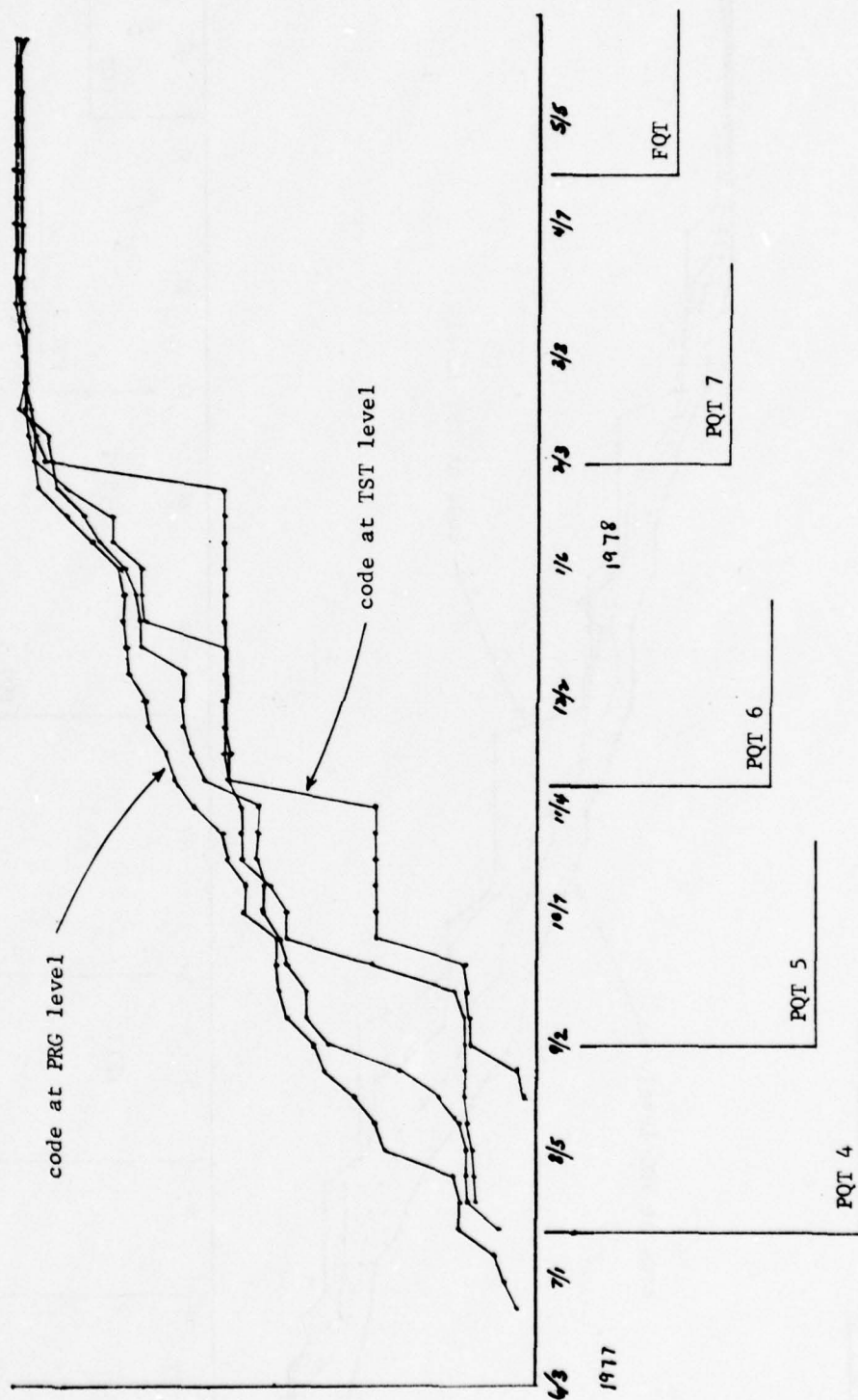


Figure 31. Code Progression Chart - COMM CPCG

DISP

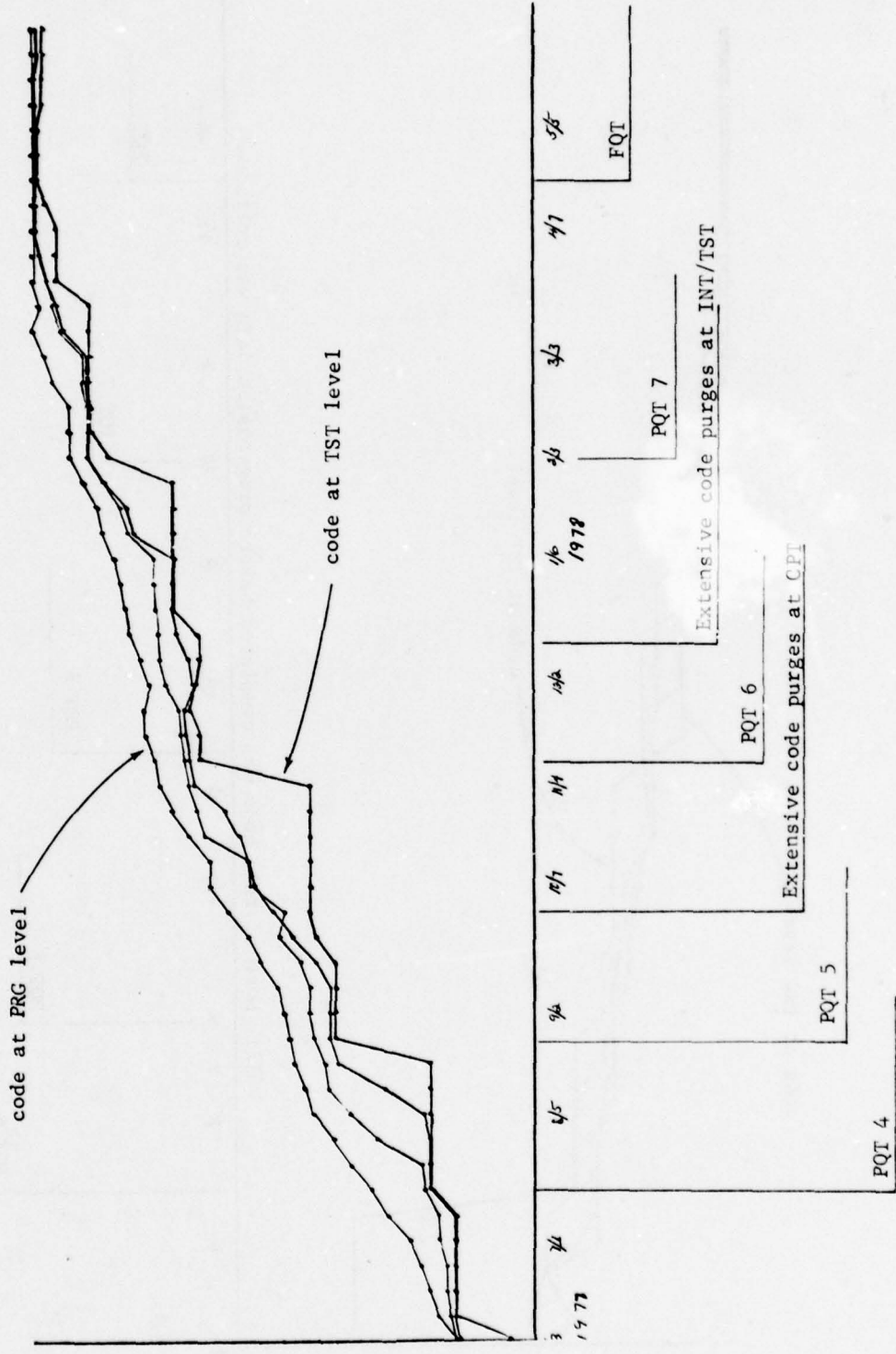


Figure 32. Code Progression Chart - DISP CPG

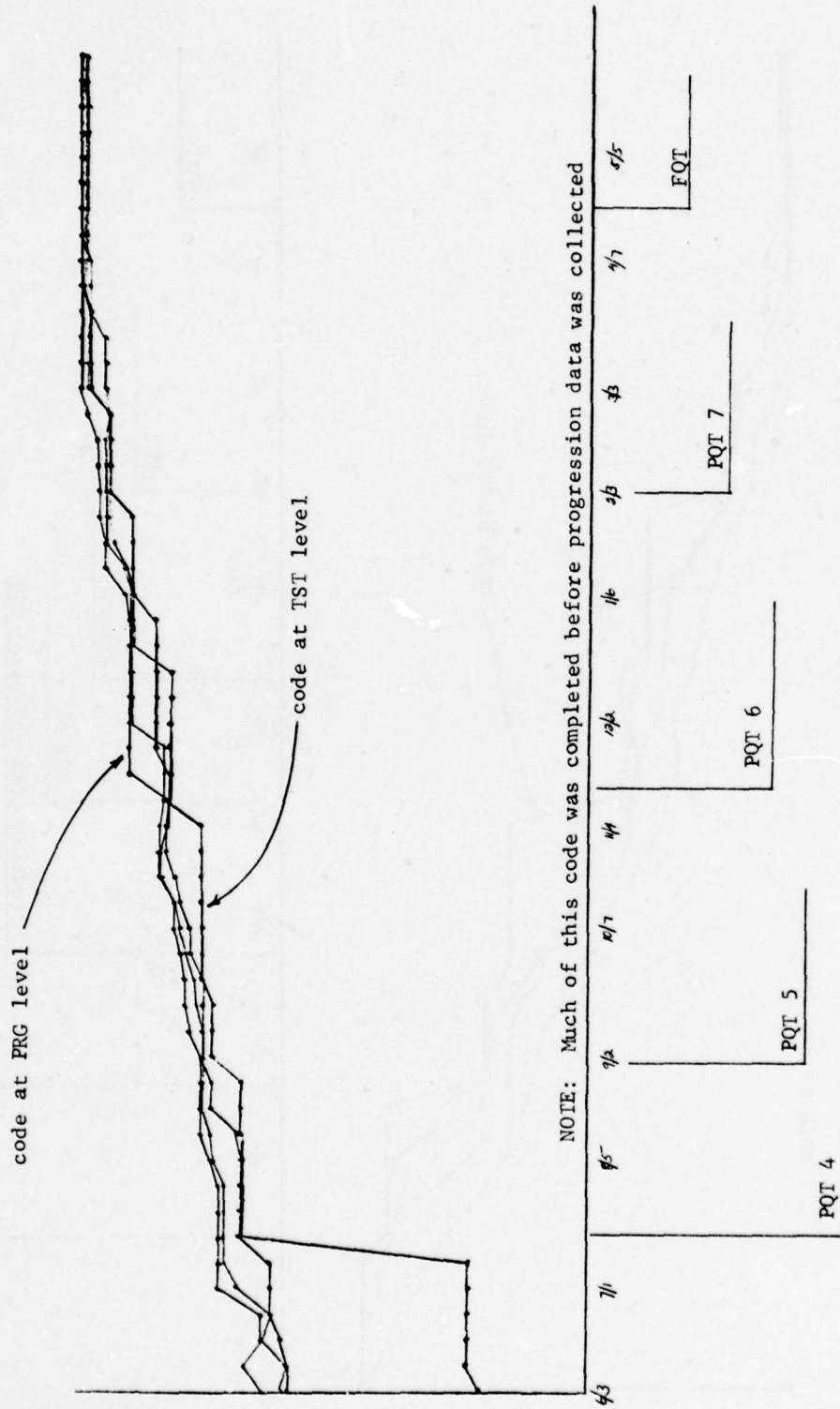


Figure 33. Code Progression Chart - MCTL CPG

RAM

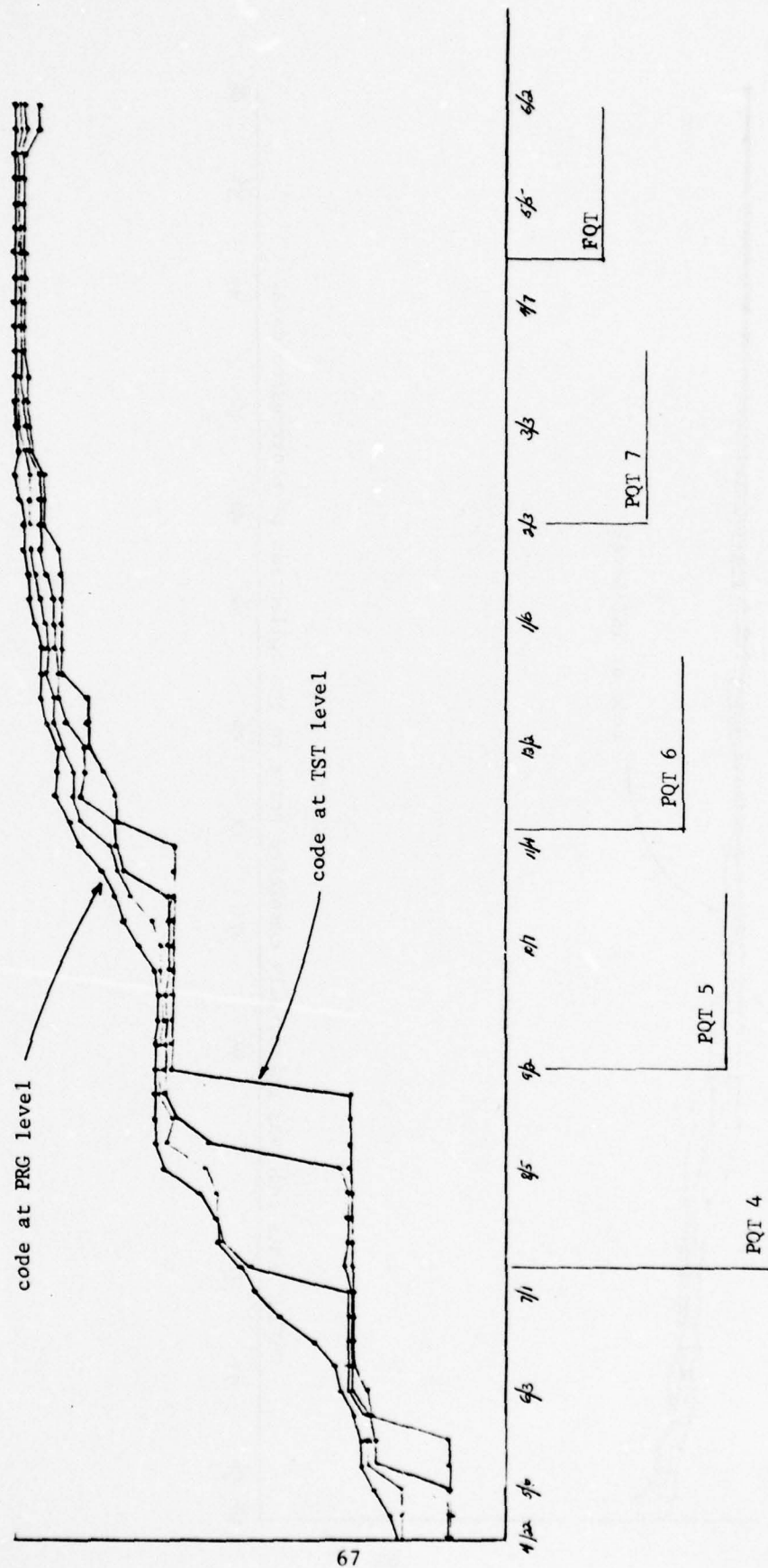


Figure 34. Code Progression Chart - RAM CPG

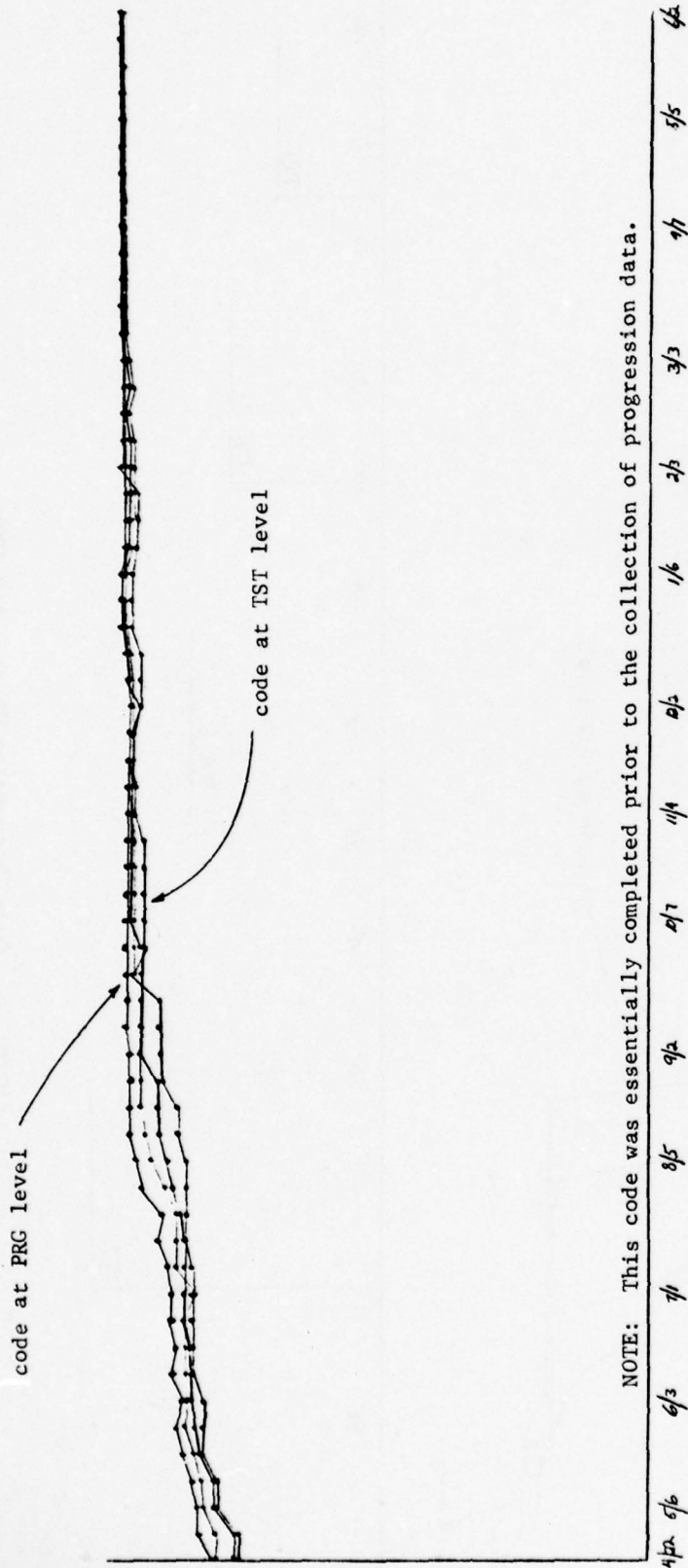


Figure 35. Code Progression Chart - RTM CPCG

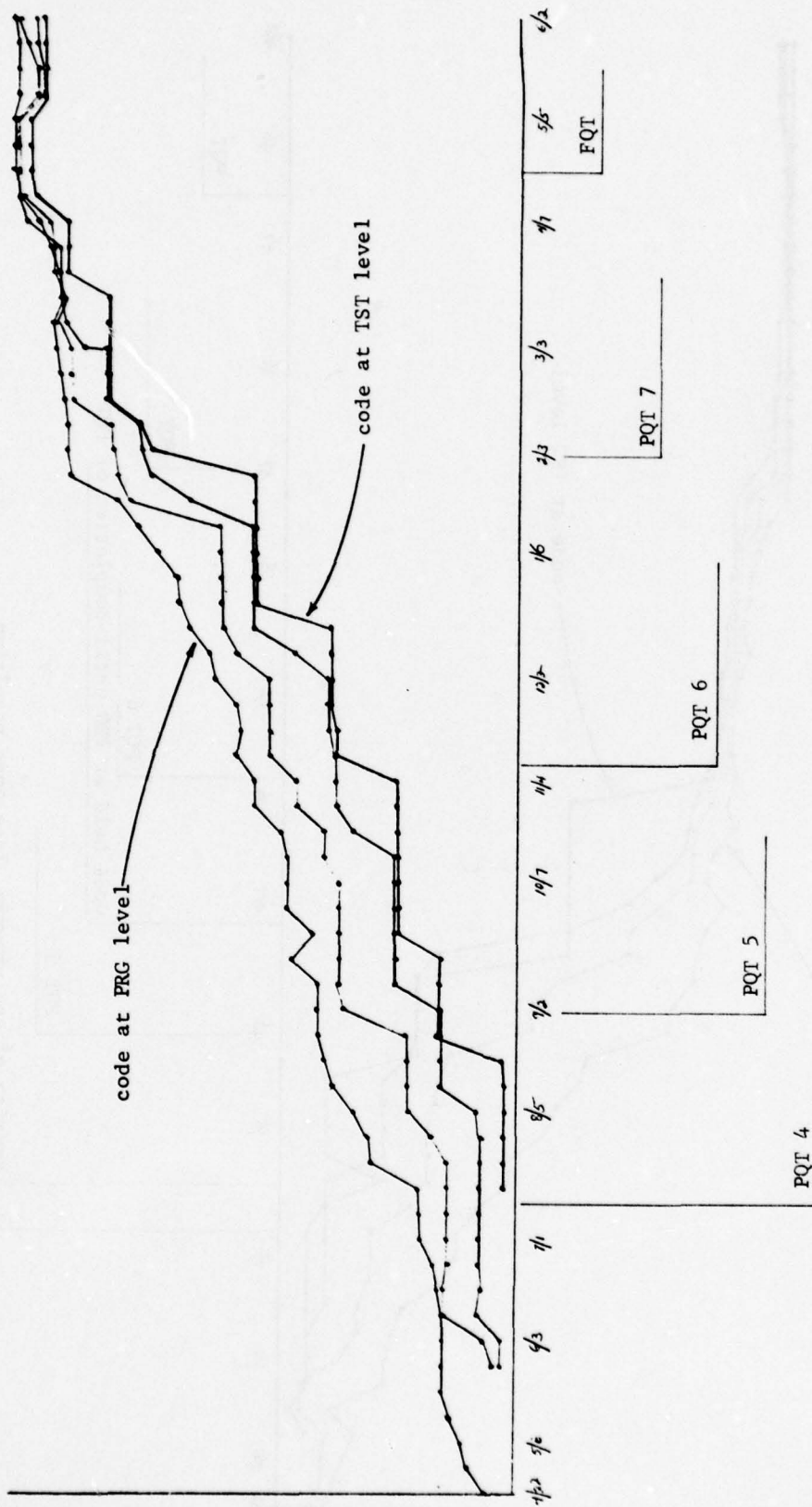


Figure 36. Code Progression Chart - SCM CPG

TRCK

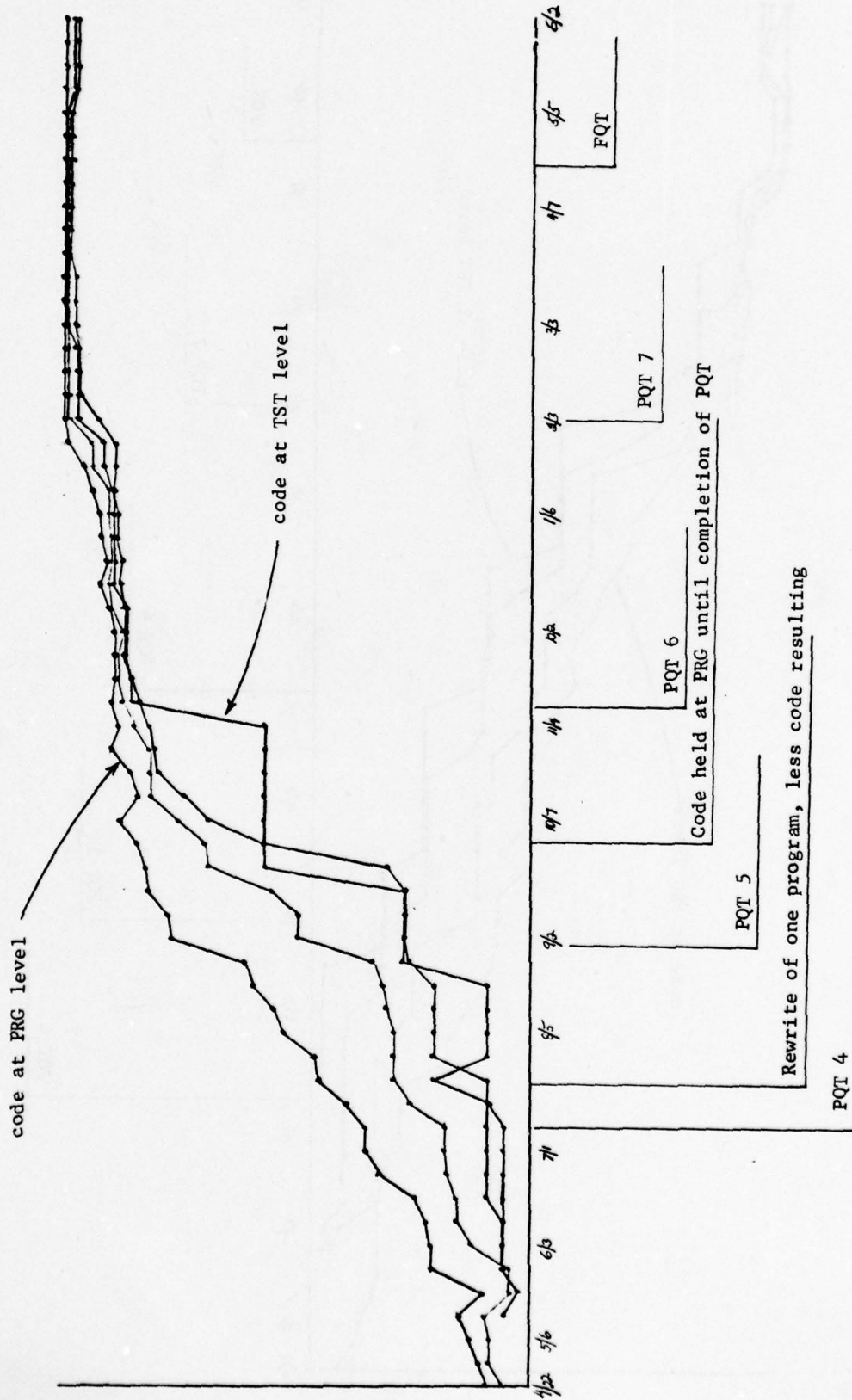


Figure 37. Code Progression Chart - TRCK CRPG

CPCI 3

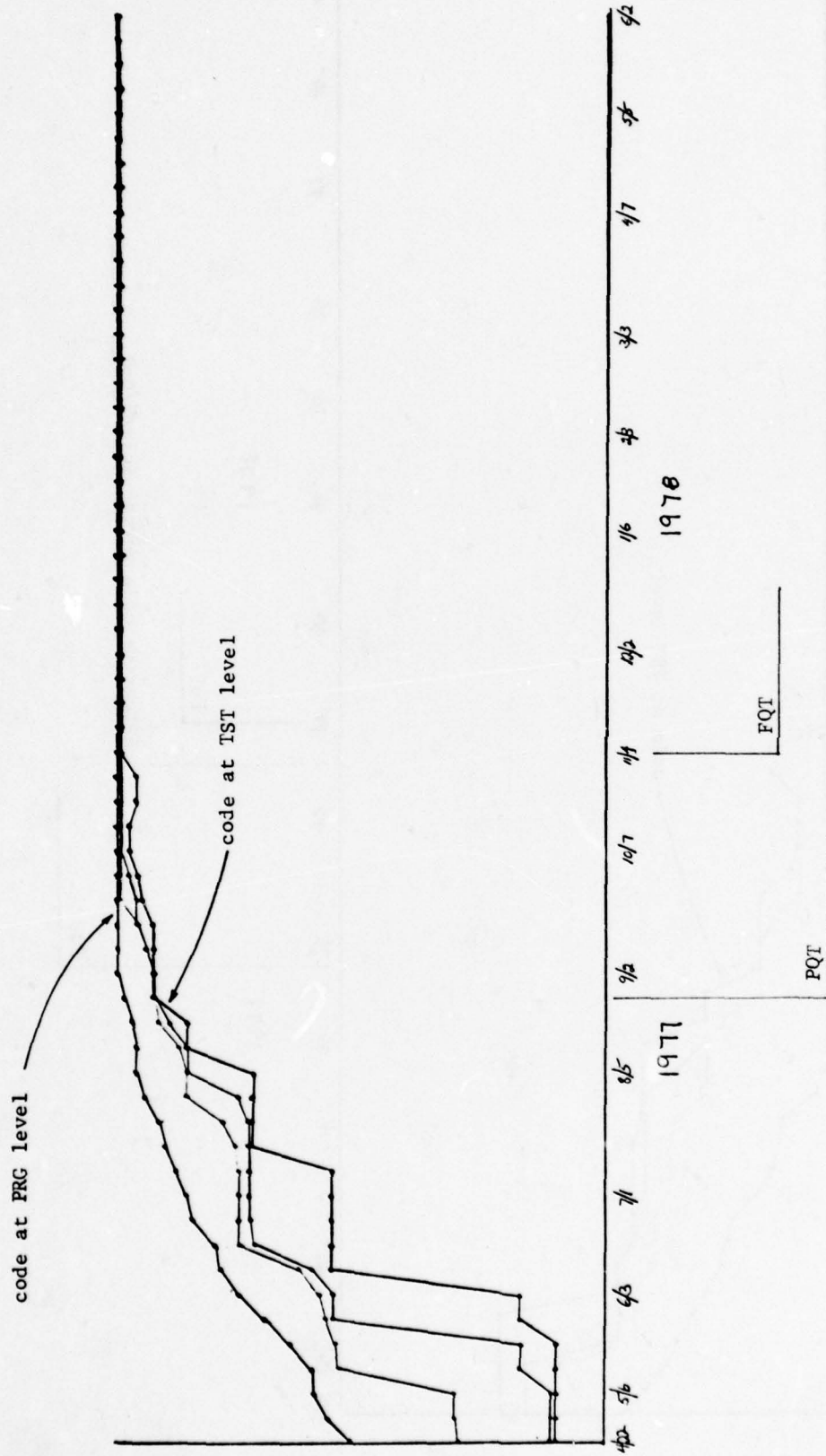


Figure 38. Code Progression Chart - CPCI 3

RTSM

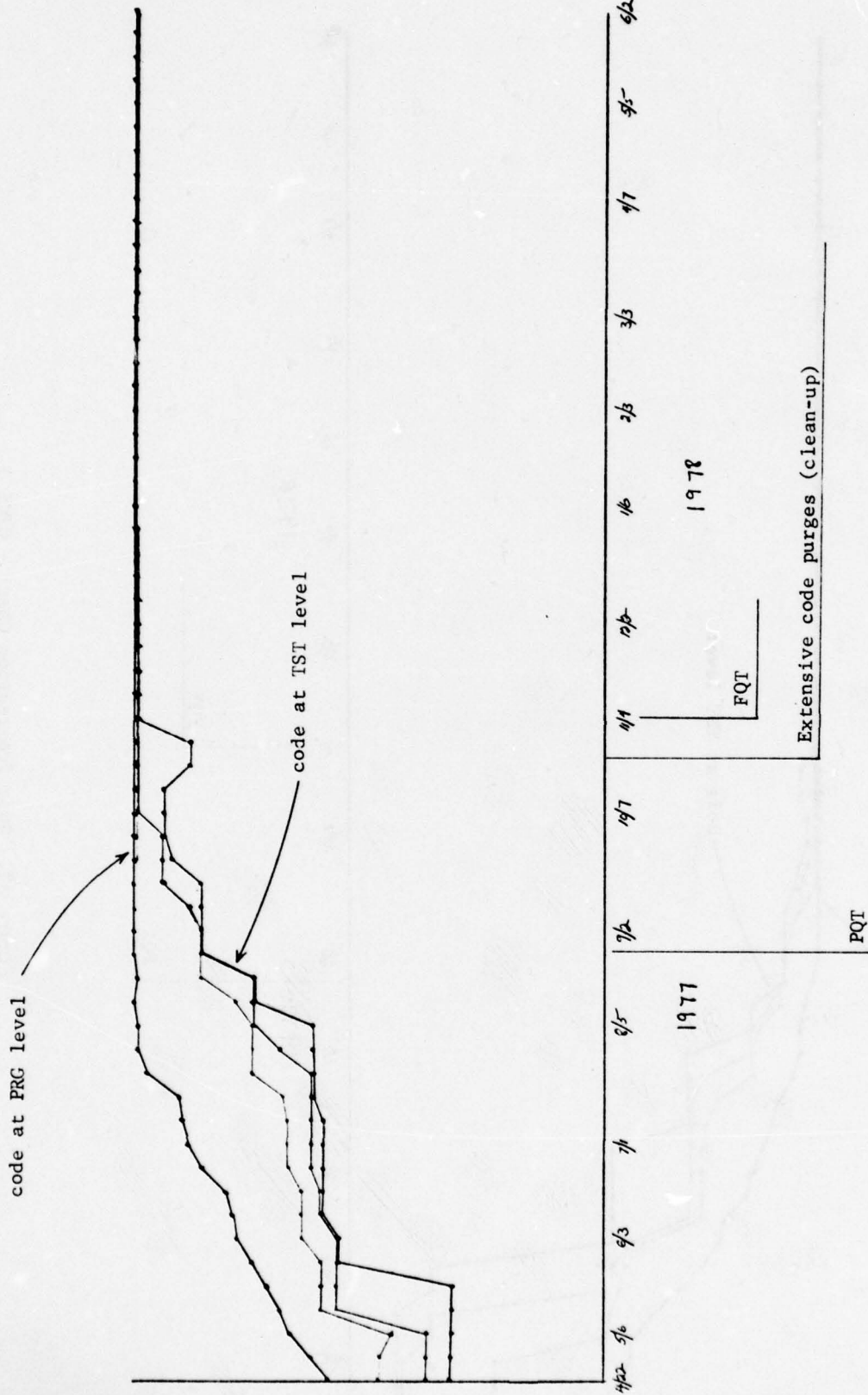


Figure 39. Code Progression Chart - RTSM CPCG

ISG

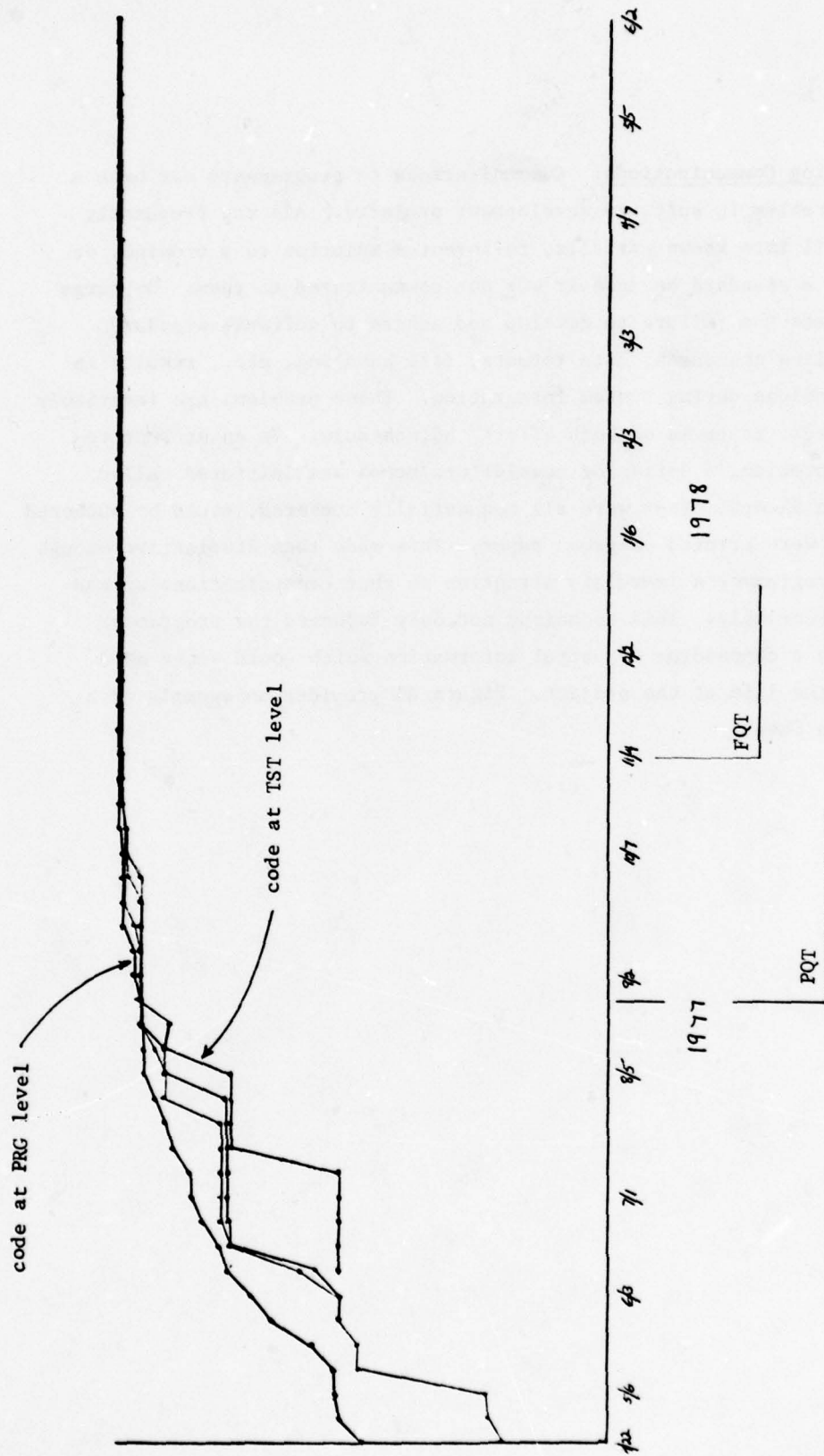


Figure 40. Code Progression Chart - TSG CRG

5.10 Programming Communications. Communications to programmers has been a longstanding problem in software development projects. All too frequently programmers fall into known pitfalls, re-invent a solution to a problem, or fail to follow a standard because it was not communicated to them. On large software projects the failure to develop and adhere to software standards for names, calling sequences, data formats, file handling, etc., results in significant problems during system integration. These problems are invariably costly to correct, in terms of both effort and schedule. In an attempt to overcome this problem, a series of newsletters/memos was initiated called PAVE PAWS Green Sheets. They were all sequentially numbered, could be authored by anyone, and were printed on green paper. This made them distinctive enough to attract a programmer's immediate attention so that communications spread quickly and effectively. This technique not only informed the programmer but resulted in a compendium of useful information which could serve as a reference for the life of the project. Figure 41 provides an example of a PAVE PAWS Green Sheet.

*** PAVE PAWS' GREEN SHEET ***

NUMBER 6

DATE: 30 June 1976

AUTHOR: W. B. Vogdes

SUBJECT: Software Standard for PAVE PAWS Library Usage

This Green Sheet defines the software library hierarchy for PAVE PAWS and establishes the standard to be followed in its usage. Examples are provided for clarity.

The PAVE PAWS program library hierarchy is designed to support an orderly and well controlled progression of software from a development environment through integration and test and into a delivered status. Basic to this hierarchy are the concepts of control level and the migration of program elements from one level to another. A program element is ready to change control level when it has completed a predefined qualification criteria and is to be placed under more stringent change control. It is common practice for such a control structure to be established and the PAVE PAWS PSL maps that approach into the library hierarchy.

Seven levels of software control are utilized for PAVE PAWS (although additional levels can be readily created). See Figure 1 for a definition of those levels and the change authority associated with each one.

With this hierarchy, the programmer is able to retain access to multiple versions of a software element without losing any stability. Because the same program element may exist at more than one control level, it is necessary to specify both LONG.NAME and control level when referencing any library element (e.g., COMPILE LONG.NAME, LVL).

When programs are ready to enter the next change level they are XMIT'ed to that level. This is effected within the library by simply changing the control level associated with the software. The change authorization of the software is automatically changed at the same time. The process of software migration through development, integration, and test can thus be conceived as a "bubble-up" occurrence.

In order to facilitate changes to software which has already been XMIT'ed from one level to another, the PAVE PAWS PSL provides a feature called "automatic drawdown". This feature allows references to be resolved in the library either at or above the specified level. Output requests are always performed at the specified level. An example is illustrative.

Figure 41. Example of PAVE PAWS Green Sheet

PAVE PAWS' GREEN SHEET (Page 2)

NUMBER 6

Consider program LONG.NAME which consists of a top segment (T) and two INCLUDE'd segments (LONG.NAME1(S1) and LONG.NAME2(S2)). This program was developed at the PRG level and then XMIT'ed to CPT. The PRG and CPT levels appear as follows:

(Contents)		
CPT	T	S1 S2
PRG		empty

Assume now that an error is detected in S1 which requires that it be corrected and undergo test at the PRG level. The segment may be updated and the program compiled using directives as follows (formats are for illustration only).

```
MODIFY LONG.NAME1,PRG
COMPILE LONG.NAME,PRG
```

During the MODIFY step, the PRG level will be searched to find S1. When it is not found, successively higher levels will be searched until it is found. In this case it is found at CPT and that will form the input source for the MODIFY. The updated source will be placed at the PRG level. Similarly, during the COMPILE step, both T and S2 will be "drawn-down" for input purposes. The object module output by the compiler will be stored at the PRG level.

The combination of control level hierarchy and automatic drawdown combine to make the PAVE PAWS PSL an easy to use yet highly controllable system.

Figure 41. Example of PAVE PAWS Green Sheet (Continued)

CONTROL LEVEL	SOFTWARE CHARACTERISTICS	CHANGE CONTROL
DEL	Completely tested, delivered by Raytheon to AF	AF
FRZ	Completely tested, delivered by IBM to Raytheon	Raytheon
TST	Software undergoing PQT/FQT	Test Dept.
FIX	Corrections to software in TST	Test Dept.
INT	Software undergoing integration	Integration Mgr.
CPT	Software under Chief Programmer's Control	Chief Programmer
PRG	Software under development	Programmer

{
 Delivery Libraries
 {
 Test Libraries
 {
 Development Libraries

Figure 1. PAVE PAWS LIBRARY HIERARCHY

Figure 41. Example of PAVE PAWS Green Sheet (Continued)

6.0 CONCLUSIONS AND RECOMMENDATIONS

The software engineering and modern programming technology employed on the PAVE PAWS consists of an integrated set of tools and techniques. Utilization of this technology does not, in and of itself, guarantee the success of any program development, but does establish an environment to support project success. Top down design and implementation is effective in assuring that all system functions have been accounted for in the software design and assists in the tracing of system requirements from the highest level of mission functions to the lowest component of code produced. Benefits from commonality and standardization of coding techniques, naming conventions, and uniform presentation by indented listings contribute to programmer understanding within and among the groups established to code major system functions. This commonality enhances design and code reviews by providing a common frame of reference for discussion and continuity. Thus, program concepts and structure can be communicated between programmers and offers the greatest improvements to efficiency and effectiveness. The disciplined programming environment embodied in the modern programming technology used on the PAVE PAWS has measurably improved the transition of software development from the mysterious and arty to the clear and cohesive world of software engineering.

The PAVE PAWS hierarchical program support library represents an important technological improvement. The PSL itself is used by the programmer to enter, store, manipulate and transition software from design through development, test and integration, and delivery. At the same time, the PSL provides reports to management with the necessary visibility into the process. Thus, commonality exists between management and software production and further improves the probability of successful program completion by providing an environment for software stability and unhampered software development.

Two of the reports produced from PSL data merit further discussion. The Code Progression and Durability reports are of significant value to management. By examining these figures over a period of a week or month, code generation, integration and testing rates can be measured. Thus, when faced with a problem and an estimate of the resources needed for the solution, management is armed with objective measures to assess program impact. The report is a direct indicator of software quality and can be used to pinpoint areas where code is progressing too slowly or quickly. As far as is known, these measures of software quality are unique in the industry.

In summary, a number of modern programming techniques were utilized on PAVE PAWS and supplemented by software development tools which won widespread acceptance by programmers and managers alike. Although it must be realized that availability and use of this technology does not, in and of itself, guarantee success, it must be credited with establishing the environment to support project success. The experience gained is being fed back into both Raytheon and IBM business areas for consideration and potential inclusion in all future efforts.

APPENDIX I

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

GENERAL CONTRACT/PROJECT SUMMARY

1. Type of Contract: FFP _____ CPFF _____ OTHER FPIF
2. Total Cost (Actual or Estimated) \$5.0M (CPCI's effort only)
3. Level of Subcontracting None
4. Project Environment

Dev. Team Collocated with User?	<u>No</u>
Dev. Team Collocated with Computer?	<u>Yes</u>
Dev. System Same as Operational System?	<u>Yes</u>
Test & Integration Separate Organization?	<u>Yes</u>

5. Project Description

Engineering support plus software design, fabrication, and test for

- (1) PAVE PAWS Tactical Software (CPCI 2) which is a real-time system including input and output interfaces with the PAVE PAWS Radar Controller (RCL-CPCI 6) via the PAVE PAWS Operating System (PPOS-CPCI 1). The system has strict storage and throughput goals.
- (2) PAVE PAWS Simulation Software (CPCI 3) which is a real-time system with the same interfacing requirements as above.
- (3) PAVE PAWS Tactical Scenario Generator (CPCI 3) which is a non-real-time data base maintenance tool used to prepare scenario files used to drive Simulation.
- (4) PAVE PAWS Data Reduction (CPCI 5) which is a non-real-time reduction system for a large variety of recording which is done by both CPCI 2 and CPCI 3.
- (5) PAVE PAWS Program Support Library (PSL-CPCI 4) which provides the basic software library services in a topdown structured environment.

6. Project Start Date 04/12/76 Est. End Date 04/12/78

7. Estimated Number of Project Personnel

Management _____	Systems Engineering _____
Chief Programmer _____	Functional Test _____
Support _____	Dev. Programming _____

8. Estimated Number of CPC's 48
9. Estimated Number of Pages of Documentation
- | | | | |
|--------------------------|-------------|--------------|-------------|
| Requirements (Part I) | <u>1460</u> | Test Reports | <u>1200</u> |
| Specifications (Part II) | <u>3400</u> | User Manuals | <u>900</u> |
| Test Specifications | <u>2000</u> | Other | <u>600</u> |
10. Estimated Total Number of Instructions N/A Cards 135K
11. Estimated Number of Different Input Formats N/A
12. Estimated Number of Different Output Formats N/A
13. Estimated Total Man/Months
- | | | | |
|-------------|------------|-------------|------------|
| Management | <u>85</u> | Programming | <u>630</u> |
| Support | <u>102</u> | Test | <u>170</u> |
| Engineering | <u>102</u> | | |
14. Estimated Total Computer Time (HRS) 7000 Hours
(wall clock on dedicated computer)

Contact B. Scheff (Raytheon)

APPENDIX II

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

MANAGEMENT METHODOLOGY SUMMARY

1. Management Procedures/Tools Used

PAVE PAWS Program Support Library (PSL) reporting
PAVE PAWS Trouble Report Procedures
Program Control Management System (PCMS - Financial)

2. Documentation Available at CDR:

- a. Development Specification (Part I) - CPCI 2
- b. Development Specification (Part I) - CPCI 3
- c. Development Specification (Part I) - CPCI 4
- d. Development Specification (Part I) - CPCI 5
- e. Product Specification (Part II) - CPCI 2
- f. Product Specification (Part II) - CPCI 3
- g. Product Specification (Part II) - CPCI 4
- h. Product Specification (Part II) - CPCI 5

Note: All above documents provided to customer.

3. Formal Reviews and Schedule

	<u>Date</u>	
a. CPCI 2	PDR <u>8/76</u>	CDR <u>1/77</u>
b. CPCI 3	PDR <u>8/76</u>	CDR <u>1/77</u>
c. CPCI 4	PDR <u>7/76</u>	CDR <u>9/77</u>
d. CPCI 5	PDR <u>8/76</u>	CDR <u>1/77</u>

4. AF Regulations, Manuals, and Military Standards Under Which Development Will Be Conducted

MIL-STD-483
MIL-STD-490
MIL-STD-1521

5. Description of Deliverable Software

Refer to GENERAL CONTRACT/PROJECT SUMMARY, Item 5, for an overview of the technical content of deliverable software. All software will be delivered in a PSL form (either disk or checkpoint tape).

6. Reference Measurement Gathering Procedures

Clarification required.

Contact B. Scheff (Raytheon)

APPENDIX III

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

DESIGN AND PROCESSOR SUMMARY

1. Target Computer(s) CDC CYBER 174-12
(same as development computer)
2. Processing Environment
 - 1 Card Reader (CDC 405)
 - 2 Line Printers (CDC 580-12)
 - 3 Disk Drives (CDC 844-21)
 - 6 CRT's (CDC 774-1)
 - 1 Plotter (Gould)
 - 6 Tape Drives (CDC 669-2)
3. Configuration: Hands on X Batch Remote On-line
4. Operating System(s) Version Nos. 1.0 as modified (PPOS)
5. Compiler Version(s) JOVIAL J3
6. Assembler(s) COMPASS
7. Est. Percent: JOVIAL 85 COMPASS 15
8. Automated Software Tools Used: PAVE PAWS PSL
9. Design Standards
 - MIL-STD-483, Appendix VI
 - IBM FSD Software Standards (33-09)
10. Programming Standards
 - PAVE PAWS Green Sheets
 - PAVE PAWS Computer Development Plan
11. Programming Techniques Employed:

Topdown Design	<u>X</u>	HIPO	<u>X</u>
Chief Programmer	<u>X</u>	Structured Code	<u>X</u>
Librarian	<u>X</u>	Structured Walk Thru	<u>X</u>
Topdown Test	<u>X</u>	Other - PDL	<u>X</u>

12. List Existing Programs/CPC's to be Used Standard commercial software
13. Estimated Turnaround Time (HRS): Batch 2 Hours

Contact B. Scheff (Raytheon)

PERSONNEL PROFILE

CHIEF PROGRAMMER TEAM #1

(See Table Below for Detail Breakdown)

TITLE	EDUCATION			TARGET LANGUAGE		EXPERIENCE			EXPERIENCE			EXPERIENCE LANGUAGES AND COMPUTERS		
	HIS	COLL	DEG	JOV	COMP	OPNS	FCMC	ANAL	OTIR	CYBER	NOS		JUN	SHLR
1 CHIEF	4	5	BS	x			14	1					14	FTN/FAF/SOS/MAL-S/360, 7090, 7094, CLC
2 BACKUP	4	4	AB	x			13	7	2			5	10	COROL/PLI/ASSEMBLER - S/360, CDC 160A, UNIVAC 1107, B5000
3 MEMBER	4	2	AA	x	x		2	2					3	PLI/MAL/CENTRAN, SNX - S/370, CLC
4 MEMBER	4	4	BA	x			3						3	PLI/CENTRAN/SNX/BAL - S/370, CLC
5 MEMBER	4	0		x			2						2	CENTRAN/SNX - CLC
6 MEMBER	4	4	BA	x			3	4					5	FTN, RTC, BASIC, CENTRAN, SNX - S/360, S/370, CLC
7 BACKUP	4	6	BS	x	x		12						12	ALC, FTN, PLI - S/360, S/370, 1401, 1410, 1620

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of IBM sponsored courses, including Structured Programming.

SYSTEM PAVE PAWS (Data Collected Again) DATE 10/07/77

PERSONNEL PROFILE
CHIEF PROGRAMMER TEAM #2

(See Table Below for Detail Breakdown)

	TITLZ	EDUCATION			TARGET LANGUAGE		EXPERIENCE				EXPERIENCE				EXPERIENCE LANGUAGES AND COMPUTERS
		HS	COLL	DEG	JOV	COMP	OPNS	PGM.	ANAL	OTHR	CYFR	NOS	JOV	SHLR	
1	CHIEF	4	4	BS	x			9	9			2	9	CENTRAM, SNX, PLI - S/360, CLC	
2	BACKUP	4	4	AB	x			13	7	2		5	10	COBOL, PLI, ASSEMBLER - S/360, CDC 160A, UNIVAC 1107, B500	
3	MEMBER	4	5.5	MA	x			8					5	PLI, BAL, FTM, CENTRAM, SNX - S/360, CLC, 4PI	
4	MEMBER	4	4	BA	x			3	4				5	FTM, RPG, BASIC, CENTRAM, SNX - S/360, S/370, CLC	
5	MEMBER	4			x		7	1.5					1.5	PLI, CENTRAM, SNX - S/360, CLC	
6	MEMBER	4	4	BS	x			1.2					1.2	FTM, PLI, COBOL, BAL, RPG - S/360, 1130, S/J	
7	MEMBER	4	4	BA	x			20		5			6	FTM, PLI, BAL - S/360, S/370, 9020, PHILCO 2000	

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of IBM sponsored courses, including Structured Programming.

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

PERSONNEL PROFILE

CHIEF PROGRAMMER TEAM #3

(See Table Below for Detail Breakdown)

	TITLE	EDUCATION			TARGET LANGUAGE			EXPERIENCE				EXPERIENCE			EXPERIENCE	
		HS	COLL	DEG	JOB	CONF	OPNS	HCNG	ANAL	OTHR	CYBER	HQS	JOV	SNLR	LANGUAGES AND COMPUTERS	
1	CHIEF	4	1.5					13						6	FTN, PLI, NIPS, CENTRAN, SNX - S/360, 1401, CLC, UNIVAC 490	
2	BACKUP	4	3					15						15	PLI, COBOL, FTM, APL - S/360, 1401, 1410	
3	MEMBER	4	4	BS				8						6	FTN, PLI, CENTRAN, SNX - S/360, S/370, CLC	
4	MEMBER	4	4	BS				9					6		FTN, PLI, PLS, APL, BAL - S/360, 1620, 1130, 7094, 9020	
5	MEMBER	4	2					2						5	BAL, PLI, CENTRAN, SNX - S/360, CLC	

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of IBM sponsored courses, including Structured Programming.

AD-A073 357

RAYTHEON CO WAYLAND MA EQUIPMENT DIV
PAVE PAWS MODERN PROGRAMMING DATA COLLECTION SYSTEM.(U)
JUN 79 B H SCHEFF, W B VODGES, N R HALL

F/G 9/2

F30602-77-C-0141

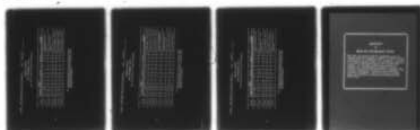
UNCLASSIFIED

RADC-TR-79-137

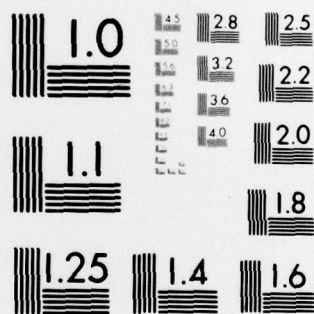
NL

2 OF 2

AD
A073357



END
DATE
FILMED
10-79
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

PERSONNEL PROFILE

CHIEF PROGRAMMER TEAM #4

(See Table Below for Detail Breakdown)

TITLE	EDUCATION			TARGET LANGUAGE		EXPERIENCE				EXPERIENCE			EXPERIENCE LANGUAGES AND COMPUTERS
	HS	COLL	DEG	JOB	COMP	OPNS	FGMC	ANAL	OTHR	CYBER	NOS	JOB	SHLA
1 CHIEF	4	4	MS	x	x		11						4
2 MEMBER	4	2	AA	x	x		5.5		2				4
3 MEMBER	4	1		x	x		3	2	10				3
4 BACKUP	4	4	BS	x	x		8.5						6
5 MEMBER	4	2		x	x	3	2						5

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of IBM sponsored courses, including Structured Programming.

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

PERSONNEL PROFILE

CHIEF PROGRAMMER TEAM #5

(See Table Below for Detail Breakdown)

TITLE	EDUCATION			TARGET LANGUAGE			EXPERIENCE			EXPERIENCE			LANGUAGES AND COMPUTERS		
	MS	COLL	INC	JOB	COMP	OTHER	ORNS	RCAC	ANAL	OTHER	CYBER	MOS		JOB	SNLR
1 CHIEF	4	4	BS	X				4	2	2				8	CENTRAN, SNK, BAL - S/360, S/370, CLC
2 BACKUP	4	5	MS	X	X			10						5	CENTRAN, SNK, PLI, BAL - S/370, CLC
3 MEMBER	4	8	MA	X				4	4					8	FTN, BAL, CENTRAN, SNK - S/360, CLC
4 MEMBER	4	5	BA	X				5						3	PLI, APL, FTN, COBOL, CENTRAN, SNK - S/360, CLC, CDC 6600, 1620
5 MEMBER	4	6	BS	X				16						16	FTN, BASIC, ASSEMBLER - 704, 7044, CDC 6000, UNIVAC 8300
6 MEMBER	4	4	BA	X				3						3	PLI, CENTRAN, SNK, BAL - S/370, CLC
7 MEMBER	4	6	MS	X				1.5	3.5					5	BAL, PLI, CENTRAN - S/360, CLC

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of IBM sponsored courses, including Structured Programming.

SYSTEM PAVE PAWS (Data Collected Against)DATE 10/07/77

PERSONNEL PROFILE

CHIEF PROGRAMMER TEAM #6

(See Table Below for Detail Breakdown)

TITLE	EDUCATION			TARGET LANGUAGE		EXPERIENCE			EXPERIENCE			EXPERIENCE LANGUAGES AND COMPUTERS		
	BS	COLL.	UG	JOB	CUM	OFMS	FCIC	ANAL	OTHR	CYBER	MOS	JOV	SHLA	
1 CHIEF	4	6	MA	X			9						5	BAL, FTM, PLI, CENTRAM, SNK - S/360, CLC
2 BACKUP	4	6	MS	X			5.5	0.5					3.5	FTM, PLI, CENTRAM, SNK - S/360, SIOGA 7, CLC
3 MEMBER	4	2	AA	X	X		2	2					3	PLI, BAL, CENTRAM, SNK - S/370, CLC
4 MEMBER	4	7	BS	X		1.5	1	1.5					2.5	CENTRAM, SNK, PLI - S/360, CLC
5 MEMBER	4	7	MS	X			6						4	FTM, COROL - S/370, DEC PDP-15, UNIVAC 1108

NOTE: Essentially all members of this team have completed a nine week Basic Programmer Training course after they were hired. They have additionally completed a variety of INM sponsored courses, including Structured Programming.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.