RADC-TR-77-188
Final Technical Report
June 1977

SOFTWARE SYSTEMS RELIABILITY: A RAYTHEON PROJECT HISTORY

Raytheon Company

Approved for public release; distribution unlimited.

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

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.

This report has been reviewed and is approved for publication.

APPROVED: James V. Cellini, Jr

JAMES V. CELLINI, JR.
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Colonel USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

294

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-188	2. GOVT ACCESSION NO.
	3. RECIPIENT'S CATALOG NUMBER: AD A040-992
4. TITLE (and Subtitle): SOFTWARE SYSTEMS RELIABILITY: A RAYTHEON PROJECT HISTORY	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report Feb 76 - Nov 76
	6. PERFORMING ORG. REPORT NUMBER: BR-9568
7. AUTHOR(s): H.E. Willman, Jr., T.A. James, A.A. Beaureguard, P. Hilcoff	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0140
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Raytheon Company, Bedford Laboratories, Hartwell Road, Bedford MA 01730	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F 55811405
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS) Griffiss AFB NY 13441	12. REPORT DATE: June 1977
	13. NUMBER OF PAGES: 84
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	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

RADC Project Engineer: James V. Cellini, Jr. (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Data Collection
Software Fault Taxonomy
Software Development
Software Reliability Modeling
Software Tools
Modeling and simulation tools

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents results of a project to collect software data from the records of development of a large Department of Defense ground-based system. A description of the subject systems software development process, characteristics, tools, and test methods are presented. Qualitative and quantitative data gathered from configuration management files are included as well as statistical summaries of this data. A detailed description of the data base files is included as well as portions of the actual data base. Recommendations are made

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

for the use of the data as well for the future collection of such data.

The data consists of three files, viz:

1) Module Description File (109 entries)

2) Software Problem Report File (2165 entries)

3) Error Category File (193 entries)

Each problem report was assigned an error category from the fault taxonomy and the data was cross correlated and summarized. The most frequent problems were in the categories of:

a) User Requested Changes (35%)

b) Data Handling (19%)

c) Logic (18%)

PREFACE

This report is the final technical report (CDRL Item A003) for the Software Data Acquisition contract, Number F30602-76-C-0140. It presents results of a project to collect historical software development data from the records of development of a large Department of Defense ground-based system. It includes a general description of the subject systems software characteristics, the software development approach and the software tools that were used. Qualitative and quantitative data gathered from configuration management files are presented. Software reliability model development and evaluation is expected to be a primary use of this data and therefore, a summary of project characteristics useful to the modeling task is also included.

The following personnel participated in this project:

A. Beaureguard	R. Leary
C. Braun	W. Polak
N. Goddard	A. Shores
P. Hatton	I. Wescott
P. Hilcoff	H. Willman, Jr.
T. James	

Acknowledged for their contributions in establishing the procedures and collecting the original data are G.J. Kacek, W.R. Murphy, and J.J. Shanley.

TABLE OF CONTENTS

Page

PREFACE iii

1. INTRODUCTION 1-1

2. SOFTWARE DEVELOPMENT PROCESS 2-1

3. OPERATIONAL SOFTWARE CHARACTERISTICS 3-1

3.1 Object Computer Description. 3-1

3.2 Data Base Structures 3-2

3.3 Control Structures and Mechanisms. 3-4

3.3.1 Task Management 3-4

3.3.2 Memory Management 3-5

3.3.3 I/O Management 3-5

3.3.4 System Auditing 3-5

3.3.5 Centralized Error Processing. 3-5

3.3.6 System Service Routines 3-6

3.4 Build Characteristics. 3-8

4. SUPPORT SOFTWARE CHARACTERISTICS 4-1

4.1 Cross Compiler 4-1

4.2 Compiler Support Software. 4-2

4.3 Cross Assembler. 4-4

4.4 Digital Simulator. 4-5

4.5 Operating System 4-5

4.6 Digital System Simulator 4-6

4.7 Data Collection/Data Reduction 4-6

5. TEST METHODS 5-1

5.1 Unit Testing 5-1

5.2 Integration Testing. 5-2

5.3 Operational Testing. 5-3

TABLE OF CONTENTS (Cont.)

	Page
6. DATA BASE	6-1
6.1 Data Base Development Task	6-1
6.2 Data Base Contents	6-2
6.2.1 Software Module Descriptions	6-2
6.2.2 Software Problem Report File	6-5
6.2.3 Error Category File	6-8
6.3 Supplementary Information	6-10
6.3.1 Build Analysis	6-10
6.3.2 Acceptance Test Data	6-20
6.3.3 Operational Data	6-21
7. RECOMMENDATIONS	7-1
7.1 Subject Project Characteristics That May Affect Modeling	7-1
7.2 Data Collection	7-3
7.3 Use of Fresh Data	7-4

TABLE OF CONTENTS (Cont.)

	Page
APPENDIX A DATA BASE DESCRIPTION FILE FORMATS	A-1
APPENDIX B SOFTWARE MODULE DESCRIPTIONS FILE NO. 1 LISTING	B-1
APPENDIX C SOFTWARE PROBLEM REPORTS SAMPLE OF FILE NO. 2 LISTING	C-1
APPENDIX D ERROR CATEGORIES (FAULT TAXONOMY) FILE NO. 3 LISTING	D-1
APPENDIX E STATIC STATISTICS FOR JOVIAL SOURCE MODULES	E-1
APPENDIX F CONSTITUENT PROGRAM MODULES OF BUILDS "F" AND "G"	F-1
REFERENCES.	R-1
BIBLIOGRAPHY.	BG-1

LIST OF ILLUSTRATIONS

Figure		Page
2-1	Software Development Process	2-2
2-2	Program Unit Release Notice	2-3
2-3	Build Release Notice	2-4
2-4	Software Problem Report.	2-5
2-5	Software Modification Notice	2-7
3-1	Data Accessing Techniques.	3-3
3-2	State Control Table Structure.	3-7
4-1	JOVIAL Compiler System	4-3
6-1	Distribution of SPRs	6-7
6-2	Build "F" Problem Reports by Month and Error Category. . .	6-13
6-3	Build "G" Problem Reports by Month and Error Category. . .	6-18

LIST OF TABLES

TABLE		PAGE
6-1	MODULE SIZE DISTRIBUTION	6-3
6-2	DISTRIBUTION OF SPRs BY MODULE TYPE	6-4
6-3	SPRs NORMALIZED TO 1000 LINES OF SOURCE	6-4
6-4	SERIOUSNESS OF SPRs	6-5
6-5	OCCURRENCE OF SPRs	6-6
6-6	SPRs BY CATEGORY GROUP	6-9
6-7	BUILD "F" PROBLEM CATEGORY DATA	6-14
6-8	COMPUTER TIME FOR SOFTWARE INTEGRATION IN WALL CLOCK HOURS	6-16
6-9	BUILD "G" PROBLEM CATEGORY DATA	6-19
6-10	ACCEPTANCE TEST ERRORS BY CATEGORY	6-20
6-11	OPERATIONAL ERRORS BY CATEGORY	6-21
6-12	EXECUTION LOADING BY MODULE TYPE	6-22

EVALUATION

The mandate for producing reliable, maintainable and quality software, has been expressed in various "studies" and "working groups," that have been generated by different departments of DOD. In addition, there have been other meetings held concerning the same topics, with participation of individuals from concerned DOD organizations. As a result, the requirement for devising methods to analyze software error data to attain these goals, has continually surfaced as a need that has to be dealt with. However, recent error data analysis has been deterred by the lack of ample data from large software developments, that can be utilized for analysis as well as in software model testing.

This effort was undertaken in response to these needs and lack of software error data. It fits into the goals of RADC TPO No. 5, Software Cost Reduction (formerly RADC TPO No. 11, Software Sciences Technology); specifically in the area of Software Quality (Software Data). The report presents results of collecting software error data from the records of a large DOD ground-based software development project. The significance of obtaining this data, is that it will be used to support current software model development projects as well as be analyzed with the goal of developing software measurements. By utilizing this data as stated, it is expected that we will be better able to determine the causes of software errors and develop means to predict and possibly prevent them. Additionally, this data will be used along with other acquired software error data, to aid in establishing a baseline for ground-based software projects in quantitative terms. This type of information will, in the future, lead to better methods of developing ground-based software projects.

James V. Cellini, Jr

JAMES V. CELLINI, Jr.
Project Engineer

1. INTRODUCTION

This is the final report of a task which provided a software error data base to be used in support of further research in software error analysis and software error prediction model analysis. The effort provided a complete error history from a large Department of Defense software development project. The subject project was the development of software for a large, ground-based, radar data processing dominated system. The error data base was extracted from 2165 Software Problem Reports (SPRs) written against 109 operational software modules. The data base developed by this task consists of three files, viz:

1) Module Description File (109 entries)
2) Software Problem Report File (2165 entries)
3) Error Category File (193 entries)

The task included assigning each of the SPRs to one of the error types contained in the error category file. This fault taxonomy is a modification of one developed by TRW as reported in Reference 1. This report discusses the modifications made to the fault taxonomy and makes recommendations for further usage.

The subject project was an advanced development phase project whose purpose was to demonstrate new concepts. The software development was a formal process with full documentation required. Engineering change order (ECO) control was used for all software and its documentation from unit release to operational (demonstration) testing. Software Modification Notices (SMNs) were written to close out each opened SPR. This formality resulted in a very successful project and produced a wealth of documentation which formed the basis for this data base generation effort.

Because one of the problems of software reliability modeling is the simplistic assumptions made about the software development and testing process, this report includes discussions which are intended to assist the model users and developers in placing the error data base in context of the software development process (Section 2), the type of operational software and its modularity (Section 3), the tools used (Section 4), and the testing process (Section 5). The data base section (Section 6) discusses the data collected and provides additional summary and statistical information. Recommendations (Section 7) are made with respect to the data collection process, the fault taxonomy, and the modeling process.

2. SOFTWARE DEVELOPMENT PROCESS

Figure 2-1, the Software Development Process, provides an overview of the process followed during the development of the software for the subject project. All activity flowed from the system requirements. These were developed by a System Engineering group who also developed the software requirements with the aid of senior software engineers. Software requirements were developed and released for design in several functional packages over a two year period. This lengthy "requirements phase" resulted in considerable redesign which contributed to the high percentage (35 percent) of SPRs prompted by changes in requirements.

Following the release of a set of requirements, the software functional specification would be updated to reflect the new requirements and software modules would be identified and described functionally. Next, a design specification for each software module was developed and the "module" or "program unit" was then tested and released for integration. Figure 2-2 is the release notice that is filed when such a release takes place. The module then enters build integration testing. This integration phase was responsible for the largest number (1984) of SPRs of any of the test phases. Integration testing is the testing of program modules with the system executive and the system data base. This constitutes a build. Following successful integration testing, the build was then released (see Figure 2-3) for acceptance testing. This took place at the hybrid test facility or at the demonstration site. Acceptance testing accounted for a very small number of SPRs (19). Following acceptance testing the build was released for operational demonstrations. SPRs were filed for any problems, changes, or suspected problems to a program unit after that unit had been released for integration testing.

Figure 2-4 is the SPR form. It may be filled by anyone, e.g., systems analyst, programmer, or user of the software. The program unit author may issue SPRs against his own program unit to alert others to deficiencies under correction.

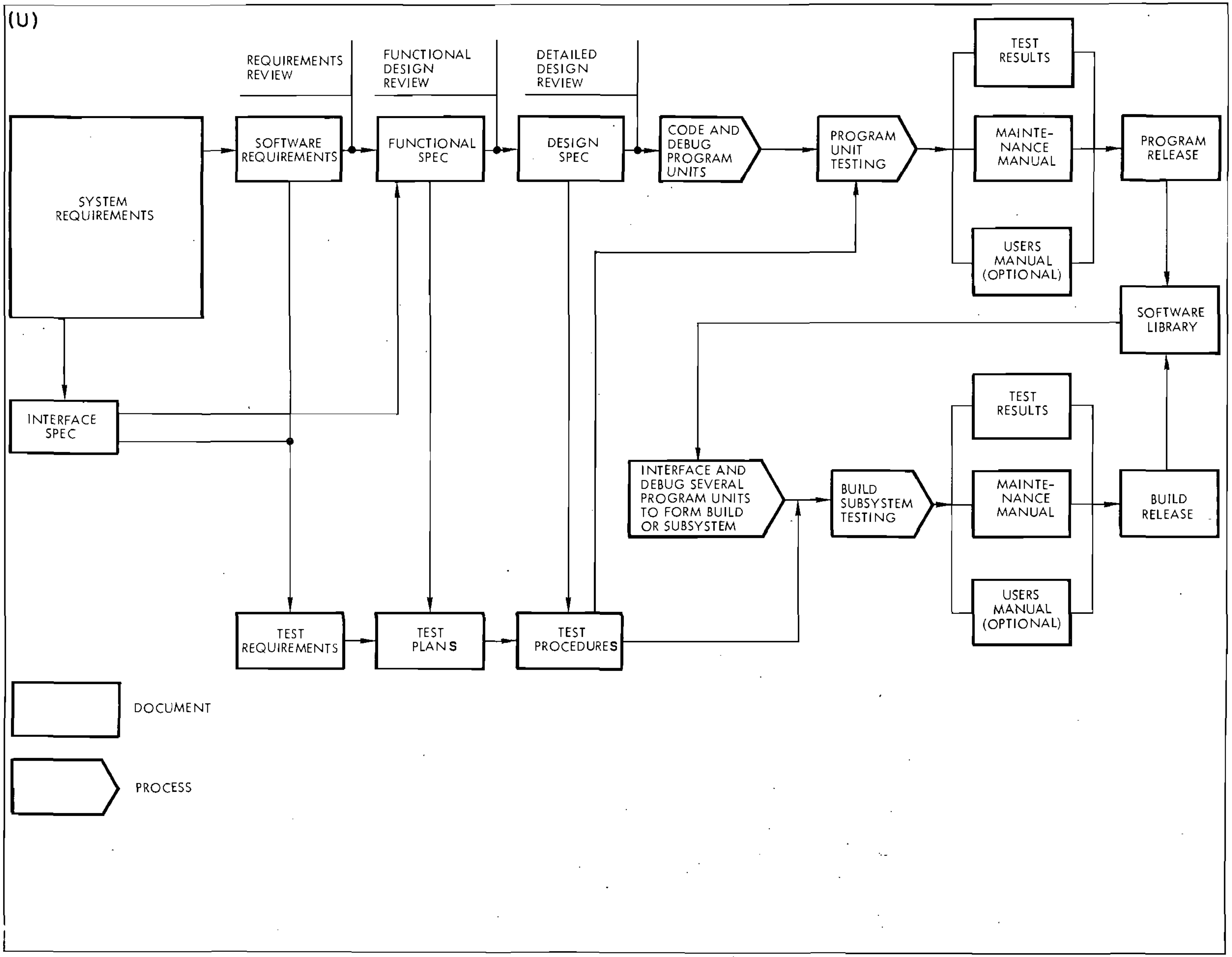

Figure 2-1 - Software Development Process

PROGRAM UNIT RELEASE NOTICE

I. IDENTIFICATION DATE: ____________

ACRONYM ____________ VERSION ____________ MOD ____________

TITLE ____________ PHASE DM(__) ED(__)

MACHINE AREA ____________ RELEASE: INITIAL ________ FINAL ________

CONTRACT/PROJECT ____________ PROGRAMMER ____________

CUSTOMER ____________ BUILD ____________

II. DOCUMENTATION

	DOCUMENT NO.		DOCUMENT NO.
REQUIREMENTS	________	ACCEPTANCE TEST RESULTS	________
FUNCTIONAL DESIGN SPEC	________	TEST RESULTS DATA	________
DETAILED DESIGN SPEC	________	MAINTENANCE MANUAL	________
ACCEPTANCE TEST PLAN	________	USERS MANUAL	________
ACCEPTANCE TEST PROC	________	LISTING	________

III. PROGRAM MEDIA

ASSOCIATED COMPOOL ____________

ASSOCIATED INITIAL CONDITIONS ____________ TAPE NO./FILE NO. ____________

SOURCE TAPE NO./FILE NO. ____________ OBJECT TAPE NO./FILE NO. ____________

CARD DECK (DATE) ____________ CURRENT LISTING (DATE) ____________

ED JOVIAL KEYWORDS (if appropriate) ____________ ____________

____________ ____________

____________ ____________

IV. CAPABILITY

A) DESCRIPTION:

B) CHANGES FROM PRIOR VERSION/MOD

C) GOVERNING DOCUMENTS (MEMOS)

D) SPR/SMN CORRECTION NO's.

E) STATUS OF UNIT ACCEPTANCE TESTING (CIRCLE ONE)

FULLY PARTIALLY * NONE *

F) TESTED WITH ALL REQUIRED HARDWARE (CIRCLE ONE)

YES NO

DATE ________ INITIALS ________

V. RELEASE TYPE

SECTION APPROVAL ____________ DATE ________

BUILD LEADER APPROVAL * ____________ DATE ________

CARDS/TAPE ON MASTERS ____________ DATE ________

DOCUMENTATION COMPLETE ____________ DATE ________

INITIAL RELEASE ☐

FINAL RELEASE ☐

Figure 2-2 - Program Unit Release Notice

ATTACHMENT D

BUILD RELEASE NOTICE

I. BUILD ____________ VERSION/MOD V ____________ M ________

SYSTEM BUILD TITLE ________________________

BUILD LEADER ____________ RELEASE: INITIAL ______ FINAL ______

CUSTOMER ________________________

II. DOCUMENTATION

DOCUMENTATION	DOCUMENT NO.	ER CONTROL NO.
FUNCTIONAL SPEC.		
REQUIREMENTS		
BUILD PLAN		
TEST PLAN		
TEST PROCEDURE		
TEST REQUIREMENTS SPEC.		
TEST RESULTS		
TEST DATA		
USERS MANUAL		
MAINTENANCE MANUAL		

III. BUILD COMPONENTS

A. ASSOCIATED COMPOOL ________________________

B. PRECEDING BUILD(S) ________________________

C. THIS BUILD CONSISTS OF FOLLOWING PROGRAM UNITS (See Below)

D. BUILD CORE IMAGE TAPE NO. ________________________

	ACRONYM	V/M	FILE NO.		ACRONYM	V/M	FILE NO.
1.				11.			
2.				12.			
3.				13.			
4.				14.			
5.				15.			
6.				16.			
7.				17.			
8.				18.			
9.				19.			
10.				20.			

IV. OPERATIONS OFFICE CONCURRENCE

I CERTIFY THAT THE OPERATIONS OFFICE FILES CONTAIN CARD DECKS, MAGNETIC TAPES, UP-TO-DAT LISTINGS FOR EACH OF THE BUILD COMPONENTS LISTED IN III ABOVE.

DATE ____________ OPERATIONS OFFICE MANAGER ____________

V. INTEGRATION

I CERTIFY THAT ENTRIES IN I, II AND III ABOVE ARE CORRECT. THE SYSTEM BUILD DESCRIBED IN I ABOVE IS UP-TO-DATE, MEETS ALL KNOWN SPECIFICATIONS, AND IS READY FOR RELEASE AS OF THIS DATE.

DATE ____________ INTEGRATION/SECTION MANAGER ____________

Figure 2-3 - Build Release Notice

APPLICATIONS SOFTWARE DEPARTMENT
(SOFTWARE PROBLEM REPORT)

Log No. ________

SUBMITTED BY: ______________________ (Signature)

Associated Build: ____________ (If Applicable)

Date: ________

Program Unit: ________ Version/Mod: ________ Computer: ________

STATEMENT OF THE PROBLEM: (Type or Print Plainly)

(Describe the problem both in programming and operational terms. Indicate the manifestation and the significance of the problem.)

PROPOSED SOLUTION: (If Known)

PRIORITY: (Optional)

CLASSIFICATION

- Design Change ☐
- Improvement ☐
- Error ☐
- ECO No. ☐
- Special ☐

Figure 2-4 - Software Problem Report

SPRs are generated as soon as a problem is identified and are not delayed until a solution is devised and tested. Their purpose of to give technical and management personnel early visibility of problem areas for earliest solution and correction. They are submitted to the department control activity.

The department control activity logs in the Software Problem Report and routes copies of the SPR to the report originator, the appropriate program unit author and his immediate supervisor, integration manager (within one working day), Department Management, designated personnel in Systems Analysis, and other specified activities (within four working days).

The Software Modification Notice (SMN) shown in Figure 2-5 is used by the program author to log and correct a specific program problem which corresponds to a Software Problem Report. An SMN may be issued directly by a program author to correct an error even though no SPR has been filed. A total of 822 SMNs were filed to record such corrections. SMNs were submitted to the control activity, with the corrections properly sequenced to reflect their position in the original source. SMNs are distributed by the control activity in similar fashion to SPRs.

For each submitted Software Problem Report the control activity obtains a corresponding Software Modification Notice form. For example, a submitted Software Problem Report which does not identify a legitimate program problem still must be closed with a Software Modification Notice form. The control activity insures that the Modification form is correctly approved (signed by the program author, Section/Group Manager, and systems integration activity Manager) when the change in implemented. The control activity maintains the master file for both forms, issues a weekly log report, and maintains a historical file of SPR/SMN submissions and disposition.

SOFTWARE MODIFICATION NOTICE	Log No. ______	Date: ______	SPR, if any, Submitted by: ______

PROGRAM UNIT ______________ Version ______ Mod. ______ to which this SMN applies. Build ______

Description of Modification (or Disposition of Problem)	DISPOSITION
	Design Change ( )
	ECO No. ______
	Improvement ( )
	Error ( )
	No Error ( )
(If explanatory materials are required, please attach them to this form)	Special ( )

CORRECTION CARDS: If correction cards are required, enter them on the form below. If there are more than twelve cards, use additional SMN forms. NOTE: Columns 73 - 80 MAY BE USED EITHER FOR DECK SEQUENCE DATA OR TO INDICATE THE LINE NUMBER IN A LISTING.

APPROVED:

Program Author ______________ Group Leader ______________ System Integration Group Leader ______________

	1	10	20	30	40	50	60	70	73	80
0										
5										
10										
15										

Figure 2-5 - Modification Software Notice

3. OPERATIONAL SOFTWARE CHARACTERISTICS

The subject project is a real-time control system for a land-based radar system. The operational software was developed by Raytheon and executes in a multiprocessor computer built by Raytheon.

Operational software was developed in a modular fashion. Nearly all of the modules are written in JOVIAL/J3. The chief exception is the Executive program, which, along with a few other modules and subroutines, is written in assembly language.

3.1 Object Computer Description

The Raytheon computer consists of two identical processors and 81,920 words of 24-bit core memory. One of the processors is utilized as a Central Processing Unit (CPU) and the other as an I/O Control Unit (IOCU); either processor is physically capable of assuming either role without any special reconfiguration. Each processor has its own set of internal registers. Both processors have common access to all primary memory locations.

Each processor contains two accumulators, two accumulator extension registers, 16 index registers, 16 program counter registers, 16 pairs of I/O control registers and miscellaneous special-purpose registers. A repertoire of 61 instructions includes hardware square root and register-to-register operations. Add time is 2μs. All arithmetic is fixed-point.

Other features of interest include:

- Unlimited indirect addressing
- A "register-substitution mode," which allows registers other than the accumulators to be specified in arithmetic operations
- A linked-list "search within limits" capability which automatically stacks list elements successfully meeting the search criteria
- Special arithmetic instructions for evaluating nested polynomials
- Interprocessor communication capability

I/O is performed via 16 independently-programmable, bidirectional channels. The I/O channels operate in accordance with a multiplex scheme based on channel priority and channel mode of operation. A single channel may be connected to several individually-selectable devices. Data transfers can be performed in either block mode or single-word mode.

3.2 Data Base Structures

The subject system features a common data base, whose overall layout is defined by means of a COMPOOL. The JOVIAL compiler is COMPOOL-sensitive, and so it creates at compile time the linkages necessary for operational programs to gain access to the data base.

COMPOOL data is segmented into blocks, and the absolute location of a particular data item is defined in terms of the base address of the block containing the item and displacement of the item within the block.

In general, the compiler generates code to look up block base addresses in a directory (see Figure 3-1a). A limited subset of COMPOOL blocks, however, is accorded a special status: whenever the compiler determines that a data item resides in one of these so-called "special blocks," it assumes that block base address to be preset in a uniquely associated index register (see Figure 3-1b).

Data sets which are subject to heaviest use are assigned to the special blocks and significant reduction in accessing overhead results. It is the responsibility of the Executive program to maintain the special block base addresses in the associated index registers for use at run-time.

Initialization of COMPOOL data is accomplished by means of an Environment Generation program. Series of JOVIAL assignment statements are used to assign values to data items and thus create data sets which can subsequently be loaded into memory. All nonvolatile data is initialized in this fashion.

In addition to nonvolatile data, which consists of system parameters, constants and permanent files, there are two classes of volatile data -- "volatile data tables" and program working storage.

Volatile data tables are used to contain raw or processed data whose source is external to the system and whose life span is relatively short. Radar input data is an example. Application programs call system service routines to

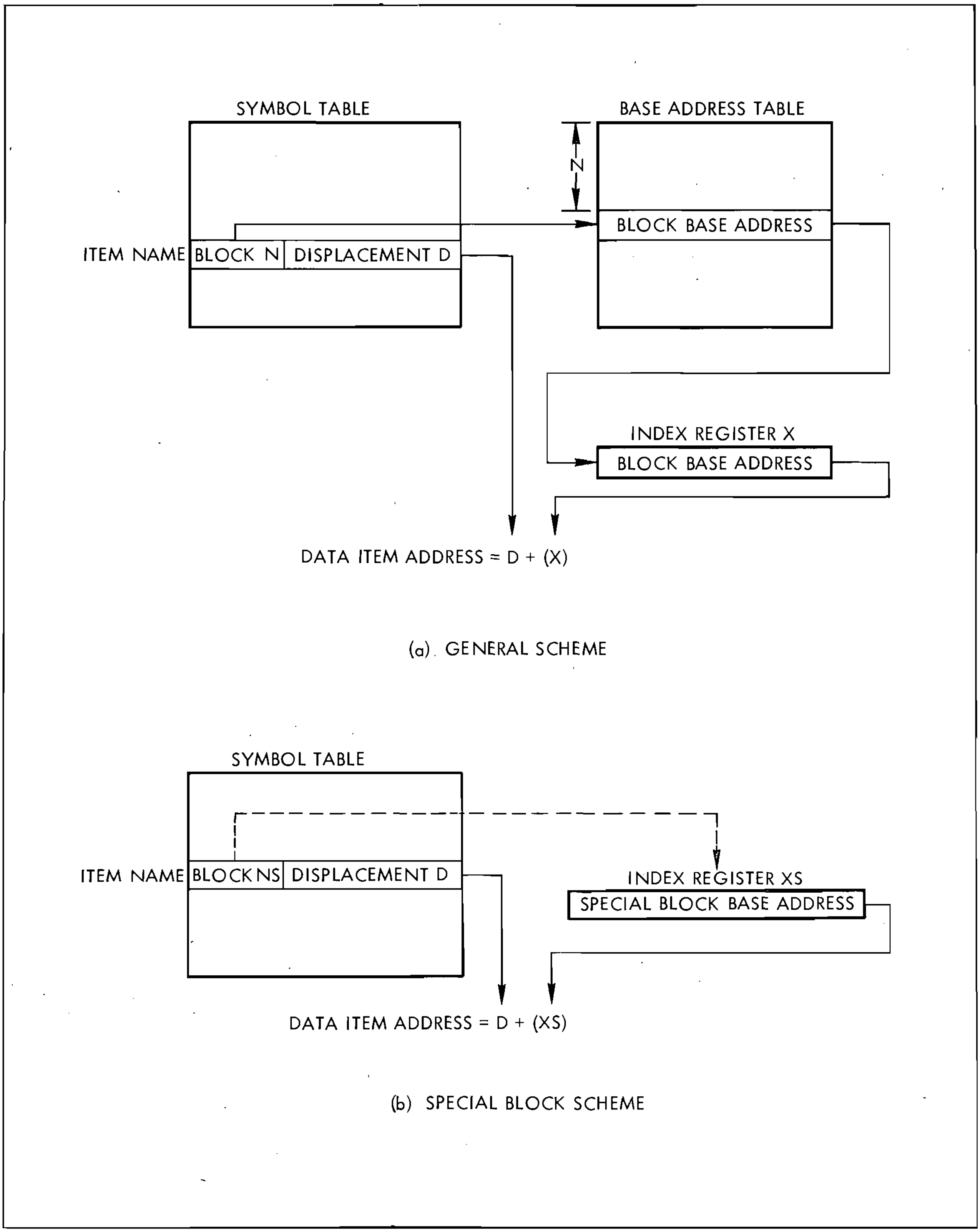

Figure 3-1 - Data Accessing Techniques

assign and deassign volatile data tables of various types as necessary. Unused tables of each type are held in free pools. Table structures are defined in the COMPOOL and allocated to special blocks. From the JOVIAL compiler's viewpoint, there is only a single table of each type defined. The Executive, however, updates the address in the special block index register to link an application program to a particular data table and thereby makes that table the "current" one of its type.

Program working storage is allocated and deallocated by the Executive and is intended strictly as a local scratch area, rather than a medium for passing data from program to program. In order to avoid usage conflict, two working storage areas are available -- one for interrupt programs and one for noninterrupt programs (only one level of interrupt program is possible). Each area consists of a chain of blocks, with the first block provided for main programs and successive blocks provided for successively nested subroutines. The JOVIAL compiler automatically generates code requesting working storage as part of the standard calling sequence for subroutines; the Executive responds to these requests by advancing the working storage index pointer to the next block in the chain. The procedure is reversed when exiting from a subroutine. This design allows reentrance.

3.3 Control Structures and Mechanisms

The subject system operates under the control of a highly centralized, modular Executive program which supervises all real-time activity on both the CPU and the IOCU. The functional units comprising the Executive and described in the subsections that follow.

3.3.1 Task Management

This unit regulates the scheduling, selection and sequencing of application program modules. Tasks are selected for execution on a priority basis in adherence to a limited multiprogramming philosophy: The limitation is that only a task of the maximum priority value can cause immediate preemption of the current program module; in the absence of such tasks, program modules are always allowed to run to completion. In order to assume timely execution of all program modules under this scheme, application functions are deliberately segmented into small, logically coherent program units. The Executive uses a device called the State Control Table (discussed below) to sequence from one module to the next to form processing threads. At the completion of each program unit in the thread, the Executive checks for higher-priority tasks, whose presence will result in temporary suspension of the current thread.

New tasks are scheduled either in response to the arrival of fresh input data or in response to an explicit request from a program module. Scheduled tasks are placed either in a "Run Queue," for execution as soon as resources become available, or in a "Delay Queue," to delay execution until a specified time interval has elapsed.

3.3.2 Memory Management

This unit is responsible for the allocation and deallocation of working storage and volatile data tables. All such memory areas are predefined; the Executive performs no dynamic carving of memory.

3.3.3 I/O Management

This unit governs IOCU activity, including coordination and activation of data transfers and processing of external interrupts. It also reports the arrival of new input data to the Task Manager.

3.3.4 System Auditing

This unit records information about program executions, service routine usage and error occurrences in a table in memory to assist in system performance analysis and debugging.

3.3.5 Centralized Error Processing

This unit processes errors detected by other software modules or by hardware error traps. Responses vary for different types of errors as dictated by an Error Response Table. This table, moreover, contains two sets of responses, one for the tactical environment and one for the test and development environment.

3.3.6 System Service Routines

A variety of system-level subroutines are collected within the Executive to eliminate programming redundancies and promote visibility. Functions provided include program queuing services, data management services, I/O device handlers, math routines and miscellaneous special-purpose services. (Some of these services fall within other Executive units as noted prior.)

Sequencing of application program modules, while carried out by the Executive, is prescribed by a "State Control Table." This table is broken down into a number of sections called "states." Each state corresponds to a single program module and consists of a group of entries representing all the various queuing and sequencing options for that module (see Figure 3-2).

Two indices are used to access State Control Table entries: a "current state" index is maintained by the Executive; a "condition" index is supplied by any program module that exits to the Executive or calls the Executive to queue a new program. These indices determine a unique table entry, from which the Executive retrieves the identity of the new program to call a queue, the new state associated with the program, and the priority of the program. The State Control Table entry may alternatively indicate that there is no new program (end-of-thread situation), in which case the Executive will select the next program module from the Run Queue.

The State Control Table may be viewed mathematically as a state-input device defining a function of such that, given a current state S and an input condition C, the new state is $S'=f(S,C)$.

The State Control Table enhances modularity by eliminating the need for program modules to call one another explicitly; program module control interfaces are under centralized management and can be modified without impacting the program modules. During the development phase of the subject system, the State Control Table facilitated substitution of dummy programs and driver modules, and also proved to be a convenient tool for tuning the system by adjusting program priorities.

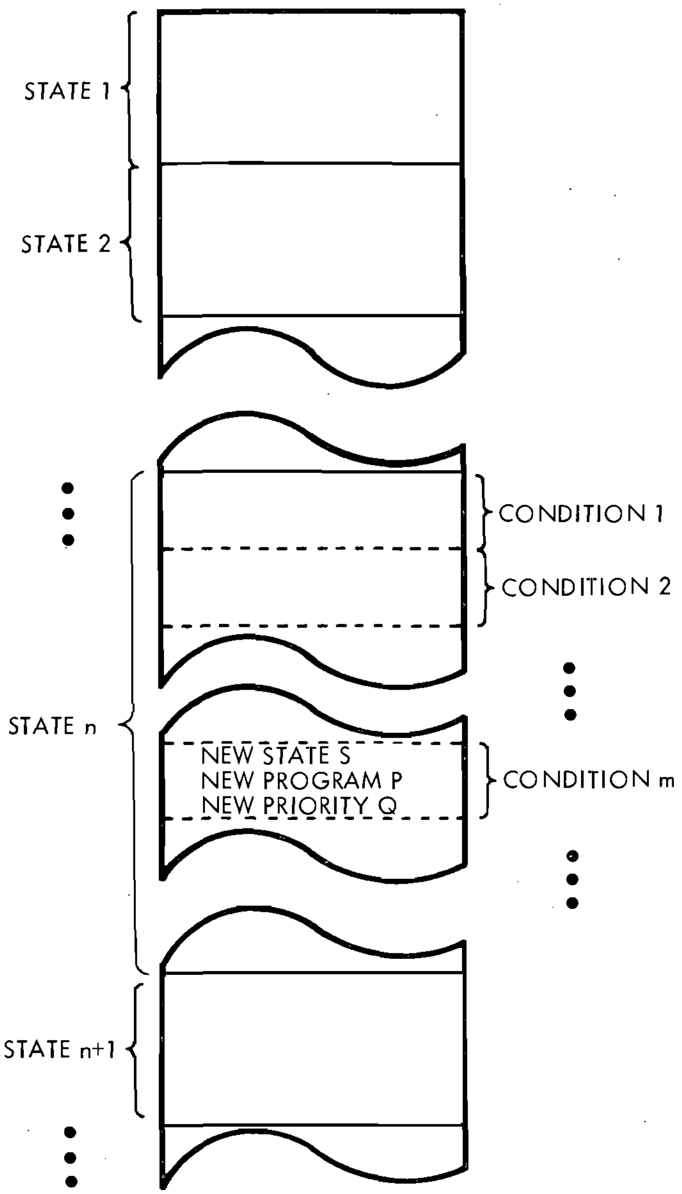

Figure 3-2 - State Control Table Structure

3.4 Build Characteristics

The method of construction of the subject system was a synthesis of top-down and bottom-up techniques. Program module specifications were derived from the top-down, beginning with system-level requirements and progressing through functional and detailed design specifications.

The highest level component of the system, the Executive, was the first program designed and the first to be up and running. Beyond providing the control functions and services described above, the Executive, in conjunction with the State Control Table, served in a broader sense as a development medium for the rest of the operational software.

Within the framework and ground rules established by the Executive, integration of the remainder of the system was performed in a rigorously controlled series of incremental steps called "builds." The initial builds consisted of groups of functionally related program modules. More advanced builds were formed by combining elementary builds and introducing additional new modules. The last build in the sequence was the fully integrated system.

Each build represented an increment in hardware capability as well as software capability. The purpose of a particular build was not only to check the interrelationships among the component software modules, but also to check program interfaces with new hardware (some of which was itself being tested for the first time under realistic conditions).

Program modules which were not part of a given build were replaced with dummy modules. Driver programs performed whatever functions were necessary to keep the system cycling smoothly. Owing to the modular nature of the system, early builds, such as the initial radar and display builds, were functionally independent to a significant degree and thus were able to be developed in parallel.

4. SUPPORT SOFTWARE CHARACTERISTICS

A modest array of software development tools were used in the production of the subject project's operational software:

- Cross Compiler
- Compiler Support Software
- Cross Assembler
- Digital Simulator of the Object Computer
- Operating System with a Debugging Package
- Digital System Simulator
- Data Collection/Data Reduction Software

Much of the software was developed at a dedicated software development facility using a UNIVAC 1108 as the host computer. All of the above mentioned software, except for the operating system, executed on the 1108. Software development and maintenance statistics for these software development tools are not included in the software reliability data base, but brief descriptions of each of these tools follow to provide a more complete understanding of the software development process of the subject project.

4.1 Cross Compiler

The Higher Order Language specified for use in the subject project was JOVIAL/J3. JOVIAL/J3 is the standard programming language for Air Force Command and Control Applications (Reference 3). As a general purpose procedure oriented language, JOVIAL has been widely used for many other types of applications. It has been used by all three services. A cross compiler for JOVIAL/J3 was implemented on the host computer to produce binary code for the object machine. The computer implemented the full J3 standard except for the features listed on the following page.

- Boolean Items
- Dual Items
- Exchange Operator
- Alternative Statement
- Input/Output Commands

The compiler does allow embedded direct code and this feature was used extensively in eight of the subject programs. These programs have been identified as DIRECT (rather than JOVIAL or ASSEMBLER) and consist of at least 50 percent assembly language embedded in a JOVIAL program. (See Appendix B.)

All system input/output was centralized in the executive program, thus relieving the JOVIAL programmer of this aspect of coding.

The average processing rate of this compiler is 33 source statements per second, including the use of the COMPOOL (central data base definition) and the generation of Set/Used information.

Appendix E contains statistics about the static occurrence of various elements of the JOVIAL language taken from a sample of 9 programs from the subject project.

4.2 Compiler Support Software

The JOVIAL Compiler Support Software consists of the following:

- (Communications Pool) COMPOOL
- COMPOOL Assembler
- COMPOOL Disassembler
- Data Base Picture Generator
- Environment Generator
- Source Library
- Source Reformatting Program
- Set/Used Program

Figure 4-1 depicts the relationships of these support programs. The COMPOOL Assembler is used to create and maintain the COMPOOL. The COMPOOL is the system data base description and contains the global data item definitions,

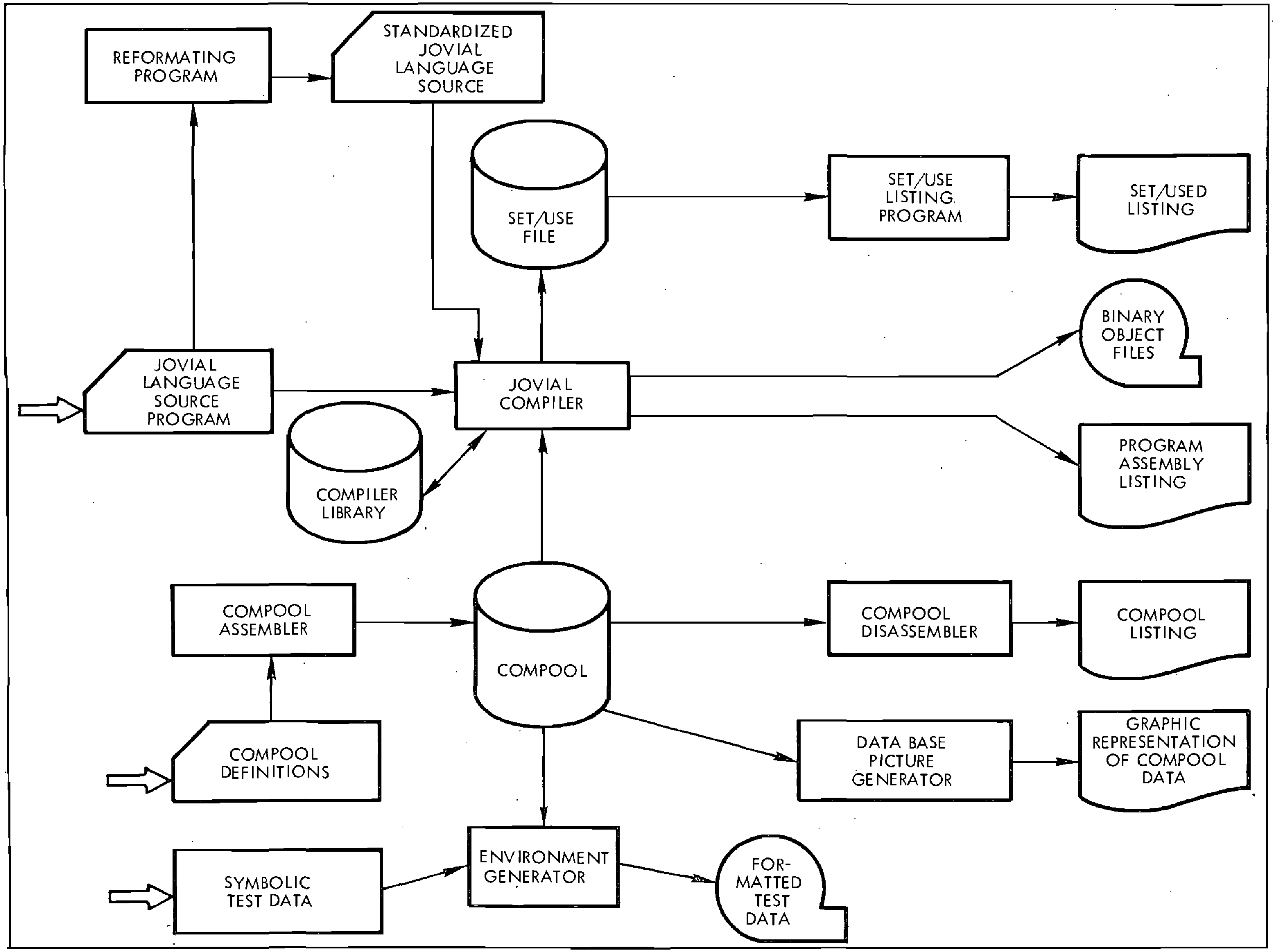

Figure 4-1 - JOVIAL Compiler System

primary memory mapping information, and parameter information for system subroutines. It is used by the JOVIAL Compiler and also used by environment generation and data reduction software. The COMPOOL Disassembler produces formatted listings and summaries of the COMPOOL contents to aid in the manual housekeeping of the data base. The Data Base Picture Generator provides a two-dimensional graphic listing of the data base and is useful in maintaining densely packed or overlayed data.

Data may be generated for initial conditions or for testing by the Environment Generator software which accepts symbolic test data, converts it to object code using the COMPOOL, and creates a load file ready for use.

The Source Library contains subroutines for inclusion directly in a source module prior to compilation.

The Source Reformatting Program produces well formatted, indented listings and will optionally resequence the source file.

The Set/Used Program is actually an optional pass of the JOVIAL Compiler and provides information on which data items are set (updated) and/or referenced (used) by the compiled program.

4.3 Cross Assembler

To provide the capability for generating programs at the instruction level, a cross assembler was developed. Since the JOVIAL Compiler produced no code to support input-output processing, multiprocessing control, diagnostic code sequences, and special instructions*, assembly language was used in these instances. The cross assembler was created by utilizing the PROC statement of the UNIVAC assembler to develop a macro for each object computer instruction. Thus, the cross assembler was a simple extension of the UNIVAC Assembler with a format conversion added to provide the proper binary formatted output for loading into the object machine. The advantage of this approach is a rapidly and inexpensively developed, highly reliable assembler. The disadvantage is that the macro processing of instructions is relatively slow, yielding an

*e.g., a linked list search/compare instruction was used for rapid correlation of track data.

assembler that averages 11 lines of source input processed per second. This is one-third the rate of the JOVIAL compiler; less if object instructions are compared.

4.4 Digital Simulator

Unit testing of individual program modules was not generally done on the object machine, but via a digital simulator of it, which executed on the UNIVAC 1108. The simulator was more accessible to the individual programmer because of the limited availability of the object computers. In addition, the fidelity of simulation was excellent and extensive debugging capabilities were provided. All instructions were simulated except for Input/Output and Multi-processor Control instructions. This exception did have an impact, as the highest incidence of SPRs were written for problems relating to Input/Output.

The job control language for the digital simulator was syntactically identical to the object machine operating system control language and most of the commands were provided. This allowed most unit tests developed on the simulator to be executed without alteration on the object machine. The effect of this on testing was not measured but was believed to be highly beneficial.

4.5 Operating System

The operating system which supported software development for the object machine was not primarily resident on the object machine, but instead resided on a Honeywell DDP-124. The DDP-124 was linked via direct memory access to the object machine. This support computer provided an early test bed capable of supporting the development of a new object machine. The DDP-124 was also used as a real time Input/Output satellite processor for the object machine. The DDP-124 Operating System also provided a program load capability for the object machine and was used to host a variety of debugging aids.

The DDP-124 included the following peripheral devices:

- Magnetic Tape Drives (2)
- Line Printer
- Paper Tape Reader/Punch
- Typewriter
- Disc Drive

4.6 Digital System Simulator

Integration of software modules into builds was accomplished with the use of a large digital system simulator as the test bed. The test facility included the object computer with its peripherals and operator stations. The object computer was linked via an interface device to a UNIVAC 1108. The 1108 based digital system simulation software provided a real time model of both the radar and the environment against which the object machine was exercised.

Test scenarios were developed by hand and processed by an environment preprocessor. This data was then used by the real time simulation to provide realistic test conditions for the object computer. The vast majority of SPRs were generated during the integration phase which occurred in this digital simulation environment.

4.7 Data Collection/Data Reduction

The data collection and data reduction software provided the capability for selective recording of data in real time and selective postprocessing of this collected data. This process was aided by the use of the previously discussed COMPOOL which provided data structure and location information for the collection process, and data format and content information for the post-process reduction.

The data collector executed under control of the real time executive module and selectively recorded data before and/or after program module execution. The data was recorded on magnetic tape for later reduction on the 1108.

5. TEST METHODS

Testing of the subject system was performed in conformance with a meticulously planned and structured regimen. The overall approach to testing closely paralleled the combined top-down/bottom-up approach described in Subsection 3.4 for system integration.

Testing proceeded in three phases: unit testing of individual program modules, including the Executive program; integration (build) testing; and operational testing of the system in the field.

5.1 Unit Testing

The first stage of testing was unit testing of individual program modules. In accordance with the Software Management Plan for the subject system, a Test Plan was conceived for each program module as it was being developed. The purpose of the Test Plan was to outline the tests necessary to demonstrate that the module fulfilled its functional requirements and to verify the module's logical integrity.

When the design of a particular program module was completed, a detailed Test Procedure was produced. Based on the parent Test Plan, the Test Procedure spelled out the specific techniques to be used in the tests, and included lists of input and output data as well as step-by-step instructions for performing the tests. The Test Procedure also described test driver program functions; such functions typically included interfacing with the test operator, simulating interfaces with other modules, and data base reinitialization between test cases.

Unit testing was carried out on the Digital Simulator (see Section 4) rather than the live computer in order to take advantage of the simulator's extensive repertoire of debugging tools, including a full instruction trace capability. An additional benefit of this approach was to conserve live machine time, which became an increasingly precious commodity as system development progressed. The Simulator not only proved entirely adequate for unit testing of application program modules, but was also utilized successfully in later stages of testing to help debug system problems.

Unit testing of the Executive program deviated slightly from the standard pattern in that it was further subdivided into testing stages of its own, and was performed on the live computer as well as the simulator. Due to its complexity, the Executive was tested at the individual routine level, and at the fully interactive level, where it operated as a skeletal version of the system. Because system I/O is one of the Executive's principal functions, and because the simulator was weak in the I/O area, the Executive unit tests performed on the simulator were repeated on the actual computer. This dual testing approach also provided an opportunity to use the Executive as a benchmark to evaluate the accuracy with which the simulator modeled the computer's behavior.

In most cases, unit testing of program modules was performed by the program authors. After a module had successfullly passed its unit tests, it was formally released to an integration team for incorporation into a software build.

5.2 Integration Testing

Integration was performed in a series of "builds" as described in Subsection 3.4. Each build was tested separately in a manner specified by its associated Test Plan and Test Procedure (counterparts to the program module Test Plan and Test Procedure). Because of the complex hardware interfaces required (whether actual or simulated), all build testing took place on a real machine.

Several facilities, each with a computer but otherwise featuring different hardware complements, were provided to support integration testing. All builds were initially tested at a software facility which contained a minimum hardware configuration (computer, peripherals, display unit) supplemented by a large scale simulation program to take the place of the remaining hardware and simulate the physical environment. The simulation program ran in a separate computer, which was connected to the tactical computer by means of a special interface device.

The chief purpose of integration testing at the software facility was to check out control and data interfaces among the program modules comprising the build. A special Executive service allowed temporary suspension of real time processing in order to return control to a build test driver program for varying test parameters or interacting with the operator. Test driver modules and dummy modules were also employed to fill processing gaps left by programs which were not included in the build.

After successful completion of integration testing at the software facility, a build was released to a facility which contained the actual hardware of central interest to the build; other hardware, where needed, was simulated by various means. The integration tests were repeated at the hardware facility, this time to check out interfaces between build software and pertinent hardware components. Acceptance testing was done at this facility.

5.3 Operational Testing

Following successful integration testing, the more advanced builds, including the full-scale system, were released as integrated hardware/software packages for operational testing in the field.

Operational testing consisted of a series of increasingly demanding missions designed to exercise the system and evaluate its response under various loads and in different physical environments. Operational missions were first rehearsed in conjunction with a Mission Simulator, then performed with a full hardware complement under actual field conditions.

6. DATA BASE

This section describes the subject project data base development task, discusses the data base contents, and supplies supplementary information useful in interpreting the data.

6.1 Data Base Development Task

The Application Software Department at the Bedford Laboratories has collected a file of approximately 10,000 SPR/SMNs. The format and use of these was discussed in Section 2. The first task was to extract each of the SPR/SMNs belonging to the subject project from the central file and reproduce it for use in the categorization task. Two files were then defined to constitute the data base (the third was added later). The SPR file was defined based on a format used by TRW for the Project 3 data. Changes were required because additional data was being collected and some data items were deleted. The second file defined was the software module file which was to contain the characteristics of the software modules against which the SPRs were written. See Appendix A for a detailed format of each of these files. Each SPR/SMN was then reviewed by a programmer who had worked on the subject projects integration task, and an error category was assigned using the TRW fault taxonomy as presented in Table 4-1 of Reference 1. Several programmers worked at this task which required about seven man/months to complete. Over 2400 SPR/SMNs were reviewed. Other historical documentation, some on microfilm files, were then reviewed and data on module characteristics were extracted. At this point the data was keypunched and placed on a computer for editing. A program was written to match the module description file against the SPR/SMN file to correlate program names. This program also presented formatted output and did some editing of the data (see Appendices B and C). At this point a third file was developed which contained the error categories.

This file was used to verify that the error category codes on the SPR/SMN file were valid (see Appendix D). Later code was added to accumulate the number of SPRs written against each program module and against each error category. Statistical routines were then added to produce summary statistics. Finally a fourth file was developed and a code was added to translate the subject project's program module names into innocuous names to preserve project anonymity.

6.2 Data Base Contents

The resulting data base as delivered to RADC consisted of the three files whose formats appear in Appendix A. Each will be briefly discussed in this section. Those data items requiring interpretation are specifically discussed.

6.2.1 Software Module Descriptions (Refer to Appendices A and B)

This file consists of 109 entires, each containing the characteristics of an individual program module. Ther version identification shown is that of the last released version/modification of that particular program. The version number represents a major functional release of the program. Thus version 2 indicates that three major functional releases had been made. The modification letter represents the number of modification releases (minor functional changes or error corrections) within the version. E represents the fourth modification release. PROG027 A0 would be the initial release of PROG027. PROG036 4J indicates that the program has had five major functional releases and the current version has had nine modification releases. This data is generally inadequate to allow determination of the total number of releases since each version may have from no modification releases to many.

The next field indicates the generic function of the module and is somewhat subjective although few programs were difficult to assign to a generic function. The complexity characteristic was also assigned in a subjective fashion, although again no difficulty was encountered in assigning complex or simple to a module. Mode of construction was limited to modular or unstructured, as top-down or structured development was not used. Appendix B contains a complete listing of the module description file.

Table 6-1 contains the distribution of modules by number of source statements and by object size (in memory words).

TABLE 6-1

MODULE SIZE DISTRIBUTION

MODULE SOURCE SIZE (No. of Statements)	NO.	MODULE OBJECT SIZE (No. of Words)	NO.
00000-00049	1	00000-00199	15
00050-00099	11	00020-00399	14
00100-00149	7	00400-00599	15
00150-00199	5	00600-00799	6
00200-00249	6	00800-00999	10
00250-00299	6	01000-01199	4
00300-00349	8	01200-01399	3
00350-00399	2	01400-01599	4
00400-00449	5	01600-01799	1
00450-00499	5	01800-01999	2
00500-00549	1	02000-02199	2
00550-00599	5	02200-02399	2
00600-00649	3	02600-02799	1
00650-00699	1	02800-02999	1
00700-00749	1	03200-03399	1
00800-00849	1	03400-03599	4
00850-00899	3	03600-03799	1
00900-00949	1	03800-03999	1
01000-01049	1	04000-04199	1
01050-01099	1	04400-04599	1
01100-01149	2	04600-04799	6
01150-01199	2	04800-04999	1
01250-01299	1	05200-05399	1
01350-01399	1	05400-05599	1
01400-01449	1	05800-05999	1
01450-01499	1	06000-06199	2
01500-01549	2	06600-06799	1
01650-01699	3	08000-08199	1
01700-01749	1	08200-08399	1
01750-01799	1	08600-08799	1
01900-01949	3	10000-10199	1
02000-02049	2	13200-13399	1
02050-02099	1	18000-18199	1
02100-02149	1	19800-19999	1
02150-02199	1		
02250-02299	1		
02500-02549	1		
03400-03449	1		
05100-05149	2		
05600-05649	1		
05900-05949	1		
06350-06399	1		
07750-07799	1		
08850-08899	1		
08900-08949	1		
10050-10099	1		

Table 6-2 contains the distribution of SPRs by module type and also gives the distribution of module types.

TABLE 6-2
DISTRIBUTION OF SPRs BY MODULE TYPE

Module Type	Percent of Total	Percent SPRs
Logical	20.2	9.6
Control	8.3	9.5
Mathematical	19.3	18.7
I/O	5.5	5.0
DATA BASE	8.3	17.5
Microcode	0.9	1.3
COMPOOL	0.9	2.3
Data Manipulation	11.0	18.4
Test Driver	5.5	10.3

This table reveals that the DATA BASE modules should have been given more attention. The DATA BASE modules for the subject project are not data base definitions (that is the COMPOOL) but are initial conditions for a build. Perhaps better tools could have helped here. One problem with this table is that the size of the modules is not taken into consideration.

Table 6-3 shows the number of SPRs normalized to 1000 lines of source code.

TABLE 6-3
SPRs NORMALIZED TO 1000 LINES OF SOURCE

	SPRs/1000 Lines of Source	Percent of Total Size
Control	18	25
Data Manipulation	29	31
Logical	34	14
I/O	36	7
Mathematical	40	23

The five module types represent the operational executable modules and were ratioed to 100 percent. The relatively low figure for the control module can be attributed to the fact that significant portions of the real time executive program were derived from a previous project.

6.2.2 Software Problem Report File (Refer to Appendices A and C)

The SPR file consists of 2165 entries each containing data on a single SPR/SMN pair or SMN only, if no SPR was filed. Note that the SPR numbers are not a dense set since they are not project specific. The termination code is "SOFTWARE" if an unexpected test termination attributed to a software problem was specifically mentioned on the SPR; similarly "hardware" for hardware problems which caused an unexpected test termination which was thought to be software (thus an SPR was filled out) but later attributed to hardware. Of the 2165 SPRs, 47 resulted in specifically identified unexpected software terminations and seven resulted in specifically identified unexpected hardware terminations originally though to be software problems. The seriousness of the error was determined to be CRITICAL if the discoverer indicated that it was impeding project development, LOW if it was not really necessary for a correction to be made for the current development to proceed, IMPROVEMENT if it was a suggestion for improvement but not necessary for satisfactory operation, and MEDIUM otherwise. Table 6-4 lists the occurrence of each of these levels of seriousness.

TABLE 6-4
SERIOUSNESS OF SPRs

Seriousness Type	Number	Percent of Total
Critical	134	6.2
Medium	1642	75.8
Low	105	4.9
Improvement	285	13.1

The test periods of concern to this data base are the Integration, Acceptance, and Operational periods. Integration occurs following unit development and formal release, and occurred at a software development facility. Acceptance tests were then run at a hybrid test facility. SPRs which specifically mentioned acceptance testing or were known to be found during acceptance testing by integration programmers were identified as Acceptance SPRs. All SPRs filed from the operational site were identified as Operational SPRs. Table 6-5 lists the occurence of SPRs during each of these periods.

TABLE 6-5
OCCURRENCE OF SPRs

Test Period	Number	Percent of Total
Integration	1984	91.6
Acceptance	19	0.9
Operational	162	7.5

The error category code is the code indicating the error category as listed in file 3 (see Appendix D).

The SMN number should in all cases be the same as the SPR number; except that some clerical errors were made during the original assignment of numbers. Cases of this are indicated by an * to the right of the SPR number. As mentioned in Section 2, some SMNs were filed without a corresponding SPR. These were usually the result of a programmer discovering the error, correcting it, and then issuing an SMN to release the correction. A total of 822 SMNs (38 percent) were filed without SPRs.

The Correction Type indicates the type of change or update made as a result of the SMN. Unfortunately this data was not generally captured and is insufficient for statistical use.

The Days Open data was extracted from the Raytheon Manufacturing Days calendar and represents the number of working days between the date open and date closed. SMNs filed without SPRs were set to 1 day opened.

The 2165 SPR/SMNs were opened for a total of 17,015 days, or an average of 7.9 days. This is distorted somewhat by the relatively high percentage of SMN-only reports. Removing the SMN-only reports yields 1343 SPR/SMNs opened for a total of 16,193 days or an average of 12.1 days.

Because of the file length only a small portion is included in Appendix C. RADC does, however, have the entire file.

File 6-1 shows the distribution of the SPR/SMNs by month opened during the 38 months of integration through operational testing.

The curve peaks at 133 SPRs opened during month 5 of the second year, and drops to a low of three opened during month 10 of the third year.

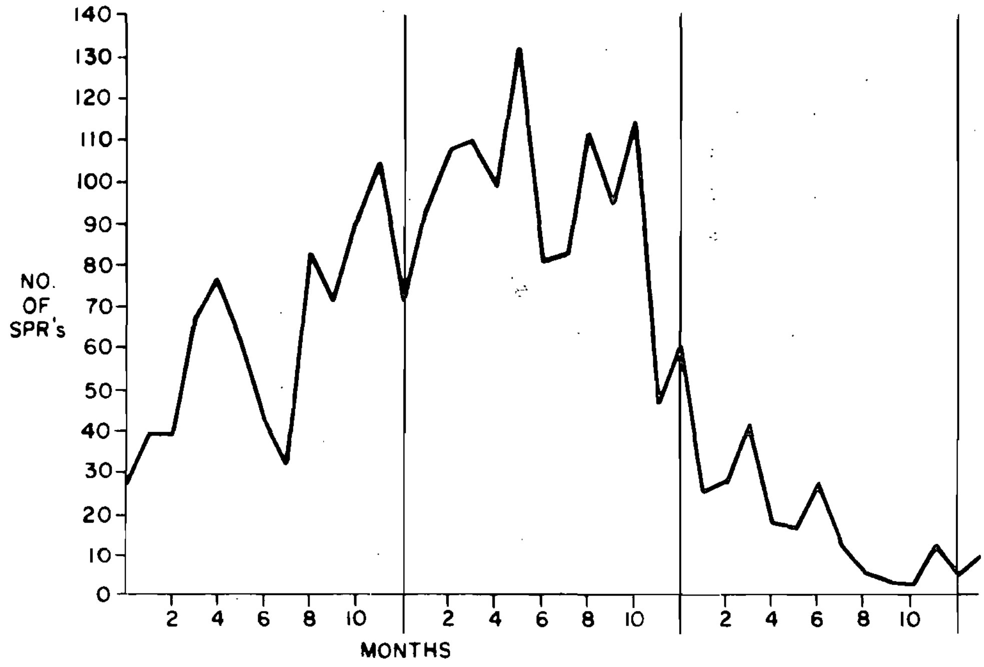

Figure 6-1 - Distribution of SPRs

6.2.3 Error Category File (Refer to Appendices A and D)

The error category file consists of 193 entires, one per error category. The error categories were based on the 184 as defined by TRW in Reference 1. Added categories are flagged with an asterisk to the right in Appendix D. Additions were made to categorize the following errors:

a) Scaling
b) New of enhanced function - display
c) Modifications for special test purposes
d) Unidentified hardware error
e) Nonrecurring problems
f) No error
g) Insufficient information for error analysis
h) Missing cards (source lines) in a compiled program
i) Inadequate/Inefficient requirements
j) Enhancement requirements

Table 6-6 contains the summary of SPRs by category group. Refer to Appendix D for the meaning of the category group code.

The most frequent errors by category group were the User Requested Changes (35.3 percent), with Data Handling Errors (18.9 percent) and Logic Errors (17.6 percent) making up the largest percentage of the remainder. The high incidence of user requested changes is most likely a characteristic of the evolutionary development approach.

TABLE 6-6

SPRs BY CATEGORY GROUP

Category Group		No. SPRs	Percent
AA	Computational	115	5.3
BB	Logic	382	17.6
CC	I/O	21	1.0
DD	Data Handling	409	18.9
EE	Operating System/Support Software	4	0.2
FF	Configuration	18	0.8
GG	Routine/Routine Interface	16	0.7
HH	Routine/System Interface	17	0.8
JJ	User Interface	10	0.5
KK	Data Base Interface	32	1.5
LL	User Requested Changes	764	35.3
MM	Preset Data Base	162	7.5
NN	COMPOOL Rejection	45	2.1
PP	Recurrent	39	1.8
QQ	Comments	15	0.7
RR	Requirements Compliance	10	0.5
SS	Unidentified	77	3.6
TT	Operator	15	0.7
UU	Questions	3	0.1
VV	Requirements Specification	11	0.5

6.3 Supplementary Information

This subsection contains supplementary information of possible use to modelers. It presents an analysis of build information, acceptance test data, and operational data.

6.3.1 Build Analysis

As mentioned previously, there were several builds implemented during the life of the project. As a final deliverable item, there were two builds delivered. These builds consisted of an Initialization Build (Build G) and an Operational Build (Build F). The Initialization Build performed hardware diagnostics, hardware and software confidence test, and initialized both hardware and software data bases. The Operational Build was comprised of 55 program modules which were implemented and tested in Builds A through E and then put together as a system. Appendix F contains the list of program modules for those two builds for possible use in further analysis.

During the life of the project, records were kept to be used for estimating new projects in the future. The types of data collected were:

- Record of all software problems by number and date
- Amount of computer time using wall clock time
- Manpower allocated to each build within the project

The following subsections discuss the software problems associated with each of the two delivered builds.

6.3.1.1 Build "F" Discussion

6.3.1.1.1 Background

Integration testing of Build F was performed over a 35 month period. Within this time frame, there were a total of 41 releases of the build reflecting error corrections, design changes and improvements. Months 1 through 7 were devoted to testing the build using the Digital System Simulator. During the next five months the build was tested at a test site with hardware and also in parallel on the Digital System Simulator.

It is appropriate here, to mention that the software was being tested on hardware that was not completely checked out, thus adding to the amount of time necessary to resolve problems. Hardware diagnostics were not sophisticated enough to diagnose all problems and many were found during operational software testing.

Testing for the remaining 20 months was accomplished by first testing a particular release of the build on the Digital System Simulator and then shipping to a field site for operational testing on the hardware in a live environment.

During the entire integration period, a total of 136 man-months of effort was expended. There is no record for computer time used while testing with the hardware. The computer time (wall clock time) utilized for testing with the Digital System Simulator amounted to 1890 hrs and 47 min. See Table 6-8 for the monthly usage of computer time for the builds.

6.3.1.1.2 Discussion

In a 35 month period, there were 1198 problems reported, investigated, and resolved. Figure 6-2 depicts the number of problems reported each month. After investigating the file of problem reports, it was discovered that the peaks and valleys shown in Figure 6-2 tracked each major release of the build. The peaks represent the time of build release when several problems had been resolved. The valleys represent the end of testing particular functions and preparing to work on the next release, which is based on the results of the tests and addition of new functions of complicated test aimed at final checkout of the system.

Another factor which attibuted to the rise and fall in numbers of problems was the parallel effort of hardware integration and hardware downtime. When hardware is malfunctioning or down, the software problems are not readily found.

Months 12 through 15 reflect the period which had the largest number of problems reported. While reviewing the problem reports, it became visible that the build during this time period was being tested for the first time at the field site in preparation for the first mission. During

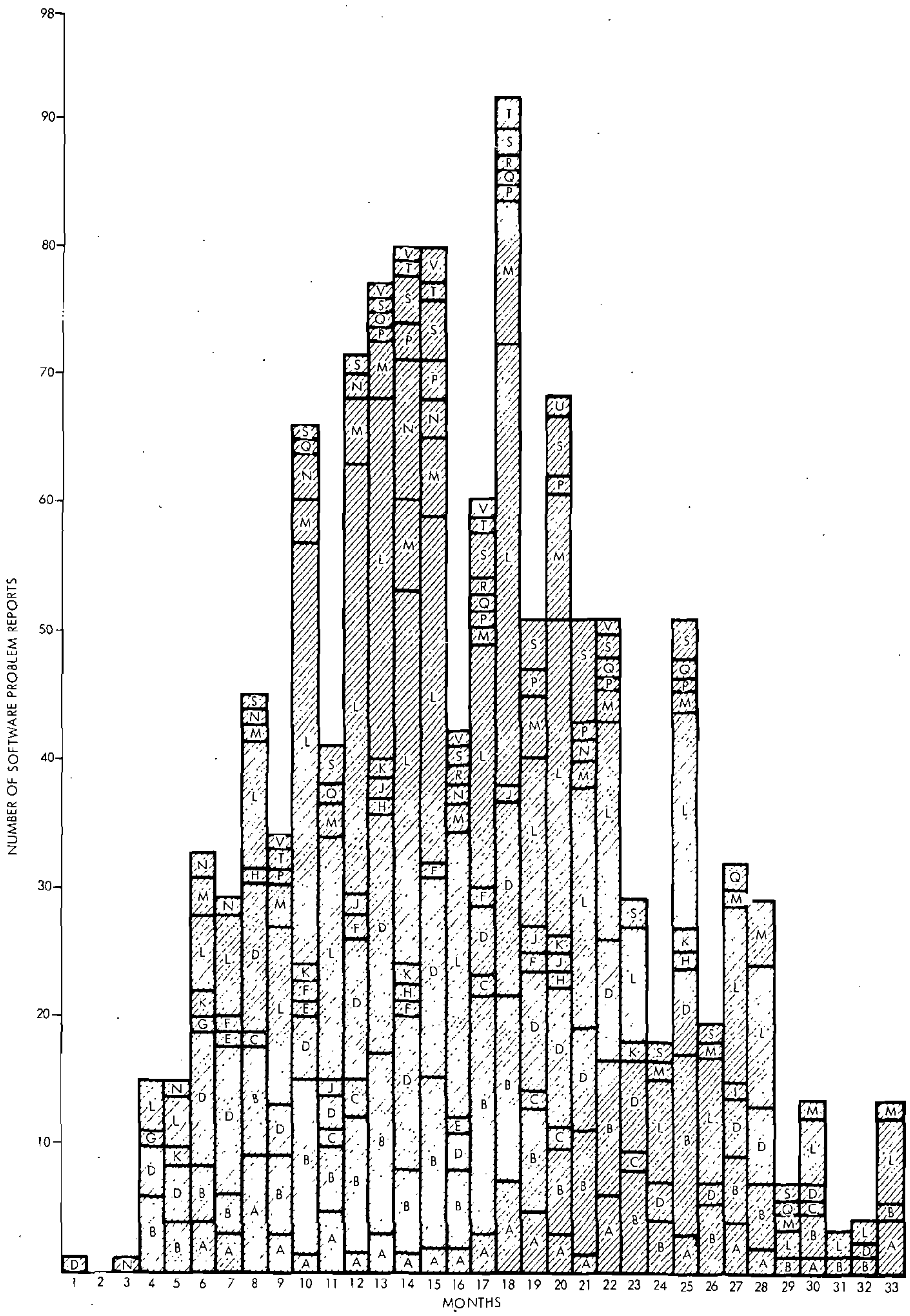

Figure 6-2 - Build "F" Problem Reports by Month and Error Category the testing, it became evident that some of the interfaces with site hardware, which could not be tested with simulation tools, and the environmental data, were different than had been anticipated. New software logic had to be added. Software was also modified to adapt to environmental interference (ground or weather clutter) which was overloading the system.

After the 15th month of integration testing the number of software problems decreased, which also resulted in a decrease of manpower levels. In essence, the remaining months were devoted to fine tuning the system. Software errors were found in areas that had not been completely tested using simulation. However, most of the problems were user requested changes, product improvements, and modifications to initial conditions due to environmental conditions.

Table 6-7 lists the number of total problems and the percentage of total problems reported for each problem category. It is readily observed that the majority of problems, in fact 38 percent, were due to design changes and improvements. Logic errors and data handling errors were 18 and 16 percent respectively. These three categories of problems constituted the major system problems.

It was rather difficult to collect data with respect to an individual build release. For example, Build F had 41 releases and the problem reports did not usually connect a problem to a build release. To generate this report, a great deal of time was devoted to correlating the problems and build releases using supervisor status reports and bracketing build release dates with problem report dates.

TABLE 6-7

BUILD "F" PROBLEM CATEGORY DATA

Problem Category	Number of Problems	Percentage of Total Problems
AA	72	6.01
BB	223	18.61
CC	10	0.83
DD	199	16.61
EE	3	0.25
FF	8	0.67
GG	3	0.25
HH	5	0.48
II	1	0.08
JJ	7	0.58
KK	11	0.92
LL	458	38.23
MM	80	6.68
NN	28	2.34
PP	15	1.25
QQ	11	0.92
RR	4	0.33
SS	45	3.75
TT	5	0.48
UU	1	0.08
VV	9	0.75
Total	1198	

6.3.1.2 Build "G" Discussion

6.3.1.2.1 Background

Build G had a 37 month span of integration testing. The Build was comprised of hardware diagnostics, hardware confidence tests, and hardware/software initialization programs. The diagnostics verified the operability of the computer while the confidence tests verified each subsystem within a radar system such as, receiver, transmitter, signal processor, etc.

In developing the programs, the majority of them could be tested individually on an off line computer, except for the actual I/O interfaces. The hardware interfaces had to be tested on the actual hardware as it became available. For Build G, the hardware and software development was being performed in parallel. A simulator was not available to test the I/O interfaces.

It should be pointed out that the programs in this Build at the start of the system were designed as independent programs. It was not until some time into system generation that a decision was made to automate the programs to operate sequentially without operator intervention as a Build. Therefore, testing of a majority of the programs had been completed independently. The Build testing basically consists of hardware integration testing.

Table 6-8 shows the monthly use of computer time (wall clock time) used to integrate the software before testing with actual hardware. Over the three year period, a total of 720 hours and 18 minutes were utilized.

TABLE 6-8

COMPUTER TIME FOR SOFTWARE INTEGRATION
IN WALL CLOCK HOURS

Month	Build F	Build G	Total Usage	Month	Build F	Build G	Total Usage
1	7:15	57:45	127:45	19	42:45	23:40	73:55
2	3:10	46:23	122:10	20	40:40	18:15	74:50
3	7:05	34:10	122:52	21	96:45	16:10	134:55
4	7:20	28:55	109:53	22	88:35	16:15	157:55
5	7:15	12:42	97:56	23	56:45	13:20	117:20
6	12:35	27:04	82:54	24	79:15	35:40	121:30
7	19:55	54:12	110:23	25	73:30	1:00	99:20
8	52:30	51:50	160:53	26	65:20	-	67:55
9	47:11	50:24	150:12	27	67:50	-	70:20
10	95:06	68:08	238:16	28	116:55	-	116:55
11	55:45	24:40	134:20	29	88:05	-	89:05
12	59:15	21:30	177:25	30	78:05	-	78:05
13	43:35	27:15	121:30	31	37:55	3:15	41:10
14	44:45	8:15	141:19	32	41:05	-	41:05
15	42:20	22:10	140:20	33	54:30	-	54:30
16	75:00	15:35	124:05	34	63:00	-	63:00
17	62:50	9:50	94:05	35	39:30	-	43:50
18	73:35	31:55	178:37	36	-	-	39:30

Note: Months without computer time indicate testing performed at acceptance test site or operational site.

6.3.1.2.2 Discussion

There were 173 problems and 59 man months of effort reported over a 37 month period, which appears to be low, compared to Build F. However, the low number of problem reports is attributed, on the most part, to only hardware integration versus the combination of software and hardware integration. The logic and data handling errors were found only in a few programs which had not been completely tested on the hardware prior to being put into the Build.

The peak months of problems reported in Build F occurred in the field when intensive testing and fine tuning of the system was being performed. In some instances, data formats and interface bit configurations were changed to make the system more efficient. There were also changes made to software to bypass hardware fixes which were more costly.

Figure 6-3 shows the errors that were reported each month and the problem categories they represented. The Build was so dependent on hardware scheduling that it is impossible to generate curves representing software reliability. The software was tested in spurts over the 37 month period. The other variable in the software testing was that all hardware was not available for testing until late in the 25th month of the Build.

While analyzing the types of problem reports, there was a definite resemblance to all other builds with respect to percentage of problems by problem category. Table 6-9 reflects the types of problems and their percentage of the total number or problems.

Approximately 50% of the problems were devoted to user requested changes or product improvements. The data handling errors reflected 22% of the problems and logic errors 14%. All remaining problems only accumulated to 14% of the total problems.

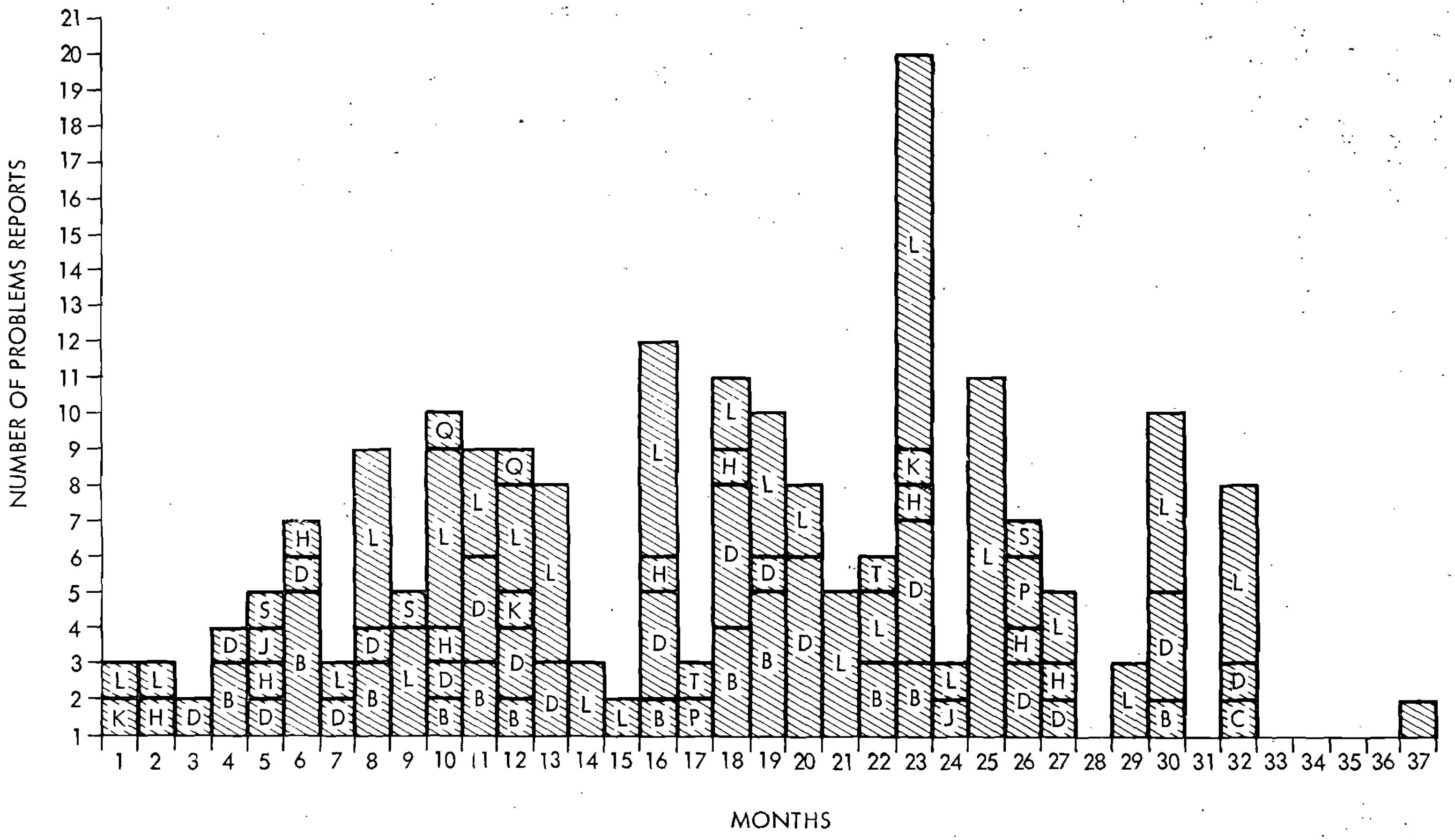

Figure 6-3 - Build "G" Problem Reports by Month and Error Category

TABLE 6-9

BUILD "G" PROBLEM CATEGORY DATA

Problem Category	Number of Problems	Percentage of Total Problems
AA	0	
BB	25	14.45
CC	1	0.58
DD	38	21.96
EE	0	
FF	0	
GG	0	
HH	9	5.20
I I	0	
JJ	2	1.15
KK	3	1.73
LL	86	49.71
MM	0	
NN	0	
PP	3	1.73
QQ	2	1.15
RR	0	
SS	2	1.15
TT	2	1.15
UU	0	
VV	0	
Total	173	

6.3.2 Acceptance Test Data

Acceptance test data is sparse and unreliable. Most often the authors of SPRs did not indicate on the SPR that the problem occurred during an acceptance test. Only 19 SPRs were so marked. This made it impossible to gather significant information about the impact of software problems on the acceptance test process including the impact on other testing. There were a total of 19 Acceptance Test SPRs or 0.9% of the total. Of the 19, 17 were critical, one was an Improvement, and one was Low Seriousness. The 17 critical SPRs were corrected in an average of 4.3 days, with a standard deviation of 4.3 days. The distribution of errors by category group is shown in Table 6-10.

TABLE 6-10
ACCEPTANCE TEST ERRORS BY CATEGORY

Category Group		Number of SPR's
AA	Computational	2
BB	Logic	3
CC	I/O	1
KK	Data Base Interface	1
LL	User Requested Changes	11
SS	Unidentified	1

6.3.3 Operational Data

Operational demonstrations took place at a remote site. Again data is sparse with respect to the impact of software errors on the entire test effort. Of the 162 operational SPRs, 31 were designated as critical. The 31 critical SPRs were corrected in an average of 11.6 days, with a standard deviation of 11.3 days. The distribution of errors by category group is shown in Table 6-11.

TABLE 6-11
OPERATIONAL ERRORS BY CATEGORY

Category Group		Number of SPR's
AA	Computational	4
BB	Logic	24
CC	I/O	3
DD	Data Handling	35
GG	Interface - Routine/Routine	1
HH	Interface - Routine/System	5
JJ	Interface - User	1
KK	Interface - Data Base	3
LL	User Requested	45
MM	Preset Data Base	5
PP	Recurrent	6
RR	Requirements Compliance	2
SS	Unidentified	24
TT	Operator	2
UU	Questions	1
VV	Requirements Specification	1

Again the high level of user requested changes reflects the evolutionary nature of the development.

Table 6-12 indicates the load placed on the software in the operational environment. This may be useful in the analysis of operational errors.

TABLE 6-12

EXECUTION LOADING BY MODULE TYPE

Module Type	Light Load	Heavy Load
Control	10%	10%
Mathematical	0	44%
Logic	11%	16%
Data Manipulation	13%	26%
I/O	3%	3%
	37% Loaded	99% Loaded

7. RECOMMENDATIONS

As mentioned in the introduction the intended use of this data base is to support the development of software reliability models. During the process of building the data base, the primary purpose of this project, some thought was given to the significance of the data and the uses to which data of this type might be put. This section identifies some of the characteristics of the subject project and data which may influence the accuracy of the models. Recommendations are also made with respect to the collection of such data in future projects and the potential uses of the data while it is still "fresh."

7.1 Subject Project Characteristics That May Affect Modeling

Several characteristics of the subject project may be of some interest to those constructing software models. While quantitative data was not gathered for this project, these characteristics might serve to assist in the selection of an applicable model as well as indicating possible areas for future extension of models. For the subject project these characteristics were:

1) evolutionary development of software requirements
2) evolutionary development of the system
3) parallel hardware development
4) multiple system configurations
5) build process
6) uneven application of resources
7) previously existing software
8) lack of development phase data

As mentioned earlier in this report, the software requirements for the subject project were issued in several releases over a two year period. Due to schedule pressure, informal or preliminary releases were also made. This characteristic probably contributed heavily to the large percentage of "User Requested" changes to the software. Many large DOD system developments have this characteristic. It is really related to the evolutionary approach to system development which seeks to minimize risk by testing concepts and evolving the system in a step-by-step orderly fashion. This approach is common when a system is being developed which does not use off-the-shelf components and proven technology.

Another characteristic of this project was parallel hardware development. Early users of the new hardware suffered from the "serial-number 1" syndrome and the high incidence of hardware failures had a pronounced effect on the software development. However, since most of the early failures were immediately recognized as being hardware problems, no software problem reports were filed. The data was not captured.

Software developed for the subject project was executed on three similar computer configurations, each "slightly" different in its usage of input/output channels and its suite of peripherals. These "slight" differences contributed to the high incidence of Input/Output errors. Software checked out at the integration facility would require minor modifications in input/output areas when executed at the acceptance test facility and later at the operational site. Each of these modifications was recorded via a SMN to maintain configuration control, and so entered the error data base. This type of "error" should be filtered out before using the data in a reliability model as these modifications are really adaptations.

Another possible problem for the modeling effort is the build process. In such a process, each successive build jeopardizes the reliability function (R(t)) of the previous build. Therefore, R(t) should increase as build testing progresses. Then, at the next build, it would probably decrease. The new functions that are added to each build differ in size and complexity. As one would expect, the simple functions were integrated before the more complex functions. Therefore, succeeding builds became more difficult to test because of the larger number of interconnections and interactions between the various modules. Therefore, the total errors (E_t) increase with each succeeding build.

A careful look at Figure 6-1 reveals several sharp dips in the number of SPRs opened. Several of these occur at the end of the calendar year, the end of the fiscal year, and at the time of summer vacation. Most likely, the intense activity just preceding the dip occurred at a build release or a major system milestone which are likely to fall just prior to these above-mentioned times and are followed by a lull in activity. These indicate uneven application of resources, primarily manpower, and supplementary data on applied manpower is needed to normalize the data and accurately relate error discovery to applied effort.

Another area which affects software reliability is the extent to which previously developed software is used. Previously developed software may occur as library routines, entire programs, or as published algorithms. It is known that a small percentage of the software (probably $<10\%$) of the subject project was developed previously, but the actual data is lost in the past.

Software error data from the development phase is not available. Many of the error categories (e.g., compiler errors, job control errors, etc.) would show up predominantly in this early phase. It is a reasonable suspicion that a program with poor reliability during the development phase is likely to have poor reliability in later phases, but it would be helpful to have hard facts in this area. On the other hand a program may have high reliability during the development phase and poor reliability during integration. This would indicate problems in development testing, or interface design.

7.2 Data Collection

Reference 1 emphasizes the need to provide accurate error categorizing at the time the error is identified. To do this at a later date requires some degree of interpretation from historical documentation which can introduce further error and distort the reliability information. We recommend that the programmer who creates the fix for the problem also does the error category assignment. The assigned category should be independently verified, possibly by a software quality assurance engineer. Since the error category assignment does involve an element of interpretation, this concurrence would enhance the reliability of the assignment.

One problem with the fault taxonomy used for this data base development was the large number of categories, some of which were overly specific (e.g., time conversion error). This overspecifying of error categories led to incompleteness and it seemed to us that a level of generality was needed (e.g., conversion error). The major complaint by the category assigners was that the number of categories was too large and the amount of subjectivity involved in assignment led to an uncomfortable feeling that some assignments were ambiguous. Subsequent to our categorization of errors, the final report was issued (reference 2) and the number of categories were reduced to 79, less than half the original list. (See Table 3-2, of reference 2). We believe that this taxonomy is a significant improvement.

7.3 Use of Fresh Data

We recommend that data also be collected during the development phase. This could be done in larger systems by automatic collection of data during compilation and testing and would allow important feedback to the developers that would allow improvements to be made early enough to have an effect on the software reliability. This feedback of "fresh" data could be used to provide improvements in the areas of training and development tools. For example, a high incidence of improperly formatted data errors might indicate that further training in the data definition capability of the HOL in use is necessary. In the subject project, Input/Output software had a high incidence of software errors (36 SPRs/1000 Source Lines). This can partially be attributed to the fact that different configurations of hardware required different I/O coding. It is also probable that the fact that the Digital Simulator had no I/O simulation capability, caused software to be released to integration testing without actually exercising the I/O code. This feedback early in the project could have resulted in I/O simulation being added to the Digital Simulator.

This potential feedback benefit would also justify the collection during the development process rather than "after-the-fact," and therefore increase its own reliability.

APPENDIX A
DATA BASE DESCRIPTION FILE FORMATS

File #1 Software Module Descriptions

The Software Module Description file contains software descriptive data and consists of one record per module. It is used to validate file #2 data and provide statistics.

Record Format:

Columns	Field	Code
1	File Identification	"1"
2-6	Project Identification	Alphanumeric
7-8	Project Code	Alphanumeric
9-15	Module Identification (left justified)	Alphanumeric
16-17	Version Identification	Alphanumeric
18	Module Function X = Control P = Input/Output L = Logical D = Data Manipulation M = Mathematical T = Test Driver C = Confidence Test B = Data Base O = COMPOOL R = Microcode	Alphanumeric
19	Module Complexity S = Simple M = Medium C = Complex	Alphabetic
20	Source Language A = Assembler J = JOVIAL F = Fortran D = Direct Code	Alphabetic
21-25	# Source Statements Right justified	Numeric

Columns	Field	Code
26-30	Object Size Including literals and local data. Not including buffers. Must be in decimal. Right justified.	Numeric
31	Mode of Construction 0 = Unstructured 1 = Modular 2 = Top Down 3 = Modular Top Down 4 = Structured 5 = Modular Structured 6 = Top Down Structured 7 = Modular Top Down Structured	Numeric

File #2 Software Problem Reports

This file consists of data from Software Problem Reports and Software Modification Notices and consists of one record per module.

Record Format:

Columns	Field	Code
1	File Identification	"2"
2-6	Project Identification	Alphanumeric
7-8	Project Code	Alphanumeric
9-12	SPR Number Right justified Blank if no SPR#	Numeric
13-19	Module Affected Identification Left justified	Alphanumeric
20-21	Version Identification	Alphanumeric

Record Format:

Columns	Field	Code
22-29	Date SPR Opened (MM/DD/YY) Blank if no SPR	Alphanumeric
30	Termination Code Blank = Terminated Normally S = Software Aborted H = Hardware Aborted	Alphabetic
31	Seriousness of Problem 1 = Critical 2 = Medium priority 3 = Low priority 4 = Suggested important	Numeric
32	Test Period D = Development - Unit Test V = Validation - Unit Acceptance I = Integration A = Acceptance of Build O = Operational Demonstration	Alphabetic
33-37	Error Category	Alphanumeric
38-41	Applicable SMN Number	Numeric
42-46	Type of Correction New Module Update X in Col 42 Document Update X in Col 43	Alphabetic

Record Format:

Columns	Field	Code
42-46	COMPOOL Change X in Col 44 Data Base Change X in Col 45 Explanation X in Col 46 Leave column blank if not applicable. Use more than one type if several apply.	
47-54	Date SPR Closed (MM/DD/YY) The SPR is closed by an SMN, therefore, this data is taken from the SMN.	Alphanumeric
55-57	Days Open Total of working days between the open and closed date. If only an SMN appears it reflects 1 day open. Right justified.	Numeric

File #3 Error Categories

This file contains the error categories and descriptions. It is used to validate file #2 data and is listed for reference. It consists of one record per error category.

Record Format:

Columns	Field	Code
1	File Identification	"3"
2-6	Error Category	Alphanumeric
7-80	Error Description	Alphanumeric

APPENDIX B

SOFTWARE MODULE DESCRIPTIONS
FILE NO. 1 LISTING

MODULE ID	VERSION	MODULE FUNCTION	COMPLEXITY	SOURCE LANGUAGE	SOURCE SIZE	OBJECT SIZE	MODE OF CONSTRUCTION	N-SPRS
PROG001	1E	LOGICAL	MEDIUM	JOVIAL	428	861	UNSTRUCTURED	19
PROG002	0K	CONTROL	MEDIUM	JOVIAL	206	339	UNSTRUCTURED	11
PROG005	0C	CONTROL	MEDIUM	JOVIAL	645	2251	UNSTRUCTURED	3
PROG006	0B	MATHEMATICAL	COMPLEX	JOVIAL	247	448	MODULAR	6
PROG007	2K	DATA BASE	MEDIUM	JOVIAL	850	998	MODULAR	6
PROG008	0B	DATA BASE	MEDIUM	JOVIAL	328	850	MODULAR	12
PROG009	2J	LOGICAL	MEDIUM	JOVIAL	349	848	UNSTRUCTURED	44
PROG011	4D	I/O	MEDIUM	JOVIAL	583	1279	UNSTRUCTURED	29
PROG012	2G	DATA MANIPULATION	SIMPLE	JOVIAL	279	725	UNSTRUCTURED	34
PROG013	2D	LOGICAL	MEDIUM	JOVIAL	416	822	MODULAR	7
PROG014	0C	LOGICAL	MEDIUM	DIRECT	120	139	MODULAR	4
PROG015	1B	MATHEMATICAL	MEDIUM	JOVIAL	103	165	UNSTRUCTURED	6
PROG016	1C	LOGICAL	SIMPLE	JOVIAL	233	471	UNSTRUCTURED	4
PROG017	3C	TEST DRIVER	MEDIUM	JOVIAL	1714	3769	MODULAR	45
PROG018	0A	TEST DRIVER	MEDIUM	JOVIAL	41	82	UNSTRUCTURED	23
PROG019	0A	DATA BASE	SIMPLE	DIRECT	256	256	UNSTRUCTURED	1
PROG020	2B	DATA MANIPULATION	MEDIUM	JOVIAL	178	524	MODULAR	5
PROG021	2A	LOGICAL	MEDIUM	JOVIAL	282	491	UNSTRUCTURED	29
PROG022	0F	DATA MANIPULATION	MEDIUM	JOVIAL	1654	3424	UNSTRUCTURED	30
PROG024	1B	MATHEMATICAL	COMPLEX	JOVIAL	183	434	UNSTRUCTURED	54
PROG025	4D	DATA MANIPULATION	MEDIUM	JOVIAL	1923	4566	UNSTRUCTURED	43
PROG026	2C	DATA MANIPULATION	MEDIUM	ASSEMBLER	492	461	MODULAR	6
PROG027	2G	MATHEMATICAL	MEDIUM	DIRECT	342	347	UNSTRUCTURED	14
PROG028	3F	MATHEMATICAL	MEDIUM	JOVIAL	169	283	UNSTRUCTURED	37
PROG029	2A	DATA MANIPULATION	MEDIUM	JOVIAL	588	925	UNSTRUCTURED	54
PROG030	2A	DATA MANIPULATION	MEDIUM	JOVIAL	890	1443	UNSTRUCTURED	16
PROG031	2B	TEST DRIVER	MEDIUM	JOVIAL	815	3413	UNSTRUCTURED	1
PROG032	1F	CONTROL	MEDIUM	JOVIAL	636	600	UNSTRUCTURED	20
PROG033	1C	I/O	MEDIUM	DIRECT	74	69	UNSTRUCTURED	8
PROG034	2C	I/O	MEDIUM	ASSEMBLER	492	488	UNSTRUCTURED	13
PROG035	2C	I/O	MEDIUM	ASSEMBLER	1537	4774	UNSTRUCTURED	25
PROG036	4J	CONTROL	COMPLEX	ASSEMBLER	5911	4798	MODULAR	128
PROG038	0A	I/O	MEDIUM	JOVIAL	65	939	UNSTRUCTURED	6
PROG039	0F	MATHEMATICAL	COMPLEX	JOVIAL	370	529	UNSTRUCTURED	10
PROG040	1C	TEST DRIVER	MEDIUM	JOVIAL	1157	2642	UNSTRUCTURED	113
PROG041	0A	LOGICAL	MEDIUM	DIRECT	65	99	UNSTRUCTURED	0
PROG042	1C	DATA BASE	MEDIUM	JOVIAL	2047	0	MODULAR	36
PROG043	0C	LOGICAL	COMPLEX	JOVIAL	283	416	UNSTRUCTURED	5
PROG045	0C	LOGICAL	MEDIUM	JOVIAL	98	139	UNSTRUCTURED	3
PROG046	3D	DATA BASE	SIMPLE	JOVIAL	1148	1804	MODULAR	285
PROG047	2B	COMPOOL	SIMPLE	JOVIAL	5142	0	MODULAR	50
PROG049	0A	TEST DRIVER	SIMPLE	JOVIAL	97	334	MODULAR	1
PROG050	0B	DATA MANIPULATION	MEDIUM	JOVIAL	121	231	UNSTRUCTURED	2
PROG051	2C	MATHEMATICAL	MEDIUM	ASSEMBLER	856	767	UNSTRUCTURED	4
PROG052	0F	MATHEMATICAL	COMPLEX	JOVIAL	561	1054	MODULAR	23
PROG053	0F	MATHEMATICAL	MEDIUM	JOVIAL	713	1227	MODULAR	16
PROG057	7B	CONFIDENCE TEST	SIMPLE	ASSEMBLER	3405	10056	UNSTRUCTURED	4
PROG058	2B	MATHEMATICAL	COMPLEX	JOVIAL	621	1564	MODULAR	13
PROG059	1A	LOGICAL	SIMPLE	JOVIAL	65	254	UNSTRUCTURED	8
PROG060	2A	MATHEMATICAL	COMPLEX	JOVIAL	435	566	UNSTRUCTURED	10
PROG061	1A	MATHEMATICAL	MEDIUM	JOVIAL	400	1000	UNSTRUCTURED	32

MODULE ID	VERSION	MODULE FUNCTION	COMPLEXITY	SOURCE LANGUAGE	SOURCE SIZE	OBJECT SIZE	MODE OF CONSTRUCTION	N.SPRS
PROG062	1E	DATA MANIPULATION	MEDIUM	JOVIAL	386	1064	UNSTRUCTURED	27
PROG063	0B	MATHEMATICAL	COMPLEX	DIRECT	242	248	UNSTRUCTURED	2
PROG064	1D	MATHEMATICAL	MEDIUM	JOVIAL	1070	2378	UNSTRUCTURED	11
PROG065	3D	MATHEMATICAL	COMPLEX	JOVIAL	129	269	UNSTRUCTURED	8
PROG066	5C	DATA MANIPULATION	MEDIUM	JOVIAL	1911	2959	MODULAR	157
PROG067	3C	MATHEMATICAL	MEDIUM	JOVIAL	906	1673	MODULAR	44
PROG068	0C	TEST DRIVER	MEDIUM	JOVIAL	1651	4707	MODULAR	40
PROG070	6C	MICROCODE	MEDIUM	ASSEMBLER	498	3466	MODULAR	29
PROG071	3J	CONFIDENCE TEST	MEDIUM	ASSEMBLER	5620	6144	UNSTRUCTURED	8
PROG072	2D	LOGICAL	COMPLEX	JOVIAL	546	992	UNSTRUCTURED	14
PROG073	0B	DATA BASE	SIMPLE	JOVIAL	256	528	UNSTRUCTURED	5
PROG075	0F	MATHEMATICAL	MEDIUM	JOVIAL	686	1453	MODULAR	16
PROG076	0D	LOGICAL	SIMPLE	JOVIAL	50	88	UNSTRUCTURED	10
PROG077	2A	DATA BASE	SIMPLE	JOVIAL	597	1866	UNSTRUCTURED	4
PROG078	7D	LOGICAL	SIMPLE	JOVIAL	79	150	UNSTRUCTURED	14
PROG079	2G	CONFIDENCE TEST	MEDIUM	ASSEMBLER	1547	4909	UNSTRUCTURED	12
PROG080	2A	CONTROL	MEDIUM	ASSEMBLER	1396	4064	UNSTRUCTURED	5
PROG081	3E	MATHEMATICAL	COMPLEX	JOVIAL	1142	2083	UNSTRUCTURED	41
PROG082	0H	CONTROL	SIMPLE	JOVIAL	470	821	UNSTRUCTURED	31
PROG083	0B	CONFIDENCE TEST	SIMPLE	JOVIAL	1677	3271	UNSTRUCTURED	2
PROG084	1C	CONTROL	SIMPLE	JOVIAL	246	529	UNSTRUCTURED	3
PROG085	4D	DATA MANIPULATION	MEDIUM	JOVIAL	5122	13207	UNSTRUCTURED	23
PROG086	0C	MATHEMATICAL	MEDIUM	JOVIAL	123	196	UNSTRUCTURED	4
PROG087	0B	DATA MANIPULATION	SIMPLE	JOVIAL	113	245	UNSTRUCTURED	1
PROG088	0C	LOGICAL	MEDIUM	JOVIAL	1049	1413	UNSTRUCTURED	7
PROG089	0D	CONFIDENCE TEST	MEDIUM	JOVIAL	10058	19821	UNSTRUCTURED	16
PROG090	0A	CONTROL	SIMPLE	JOVIAL	102	214	UNSTRUCTURED	3
PROG091	0B	CONFIDENCE TEST	MEDIUM	JOVIAL	2139	3961	UNSTRUCTURED	6
PROG092	0C	LOGICAL	SIMPLE	JOVIAL	54	156	UNSTRUCTURED	4
PROG093	0C	CONFIDENCE TEST	MEDIUM	JOVIAL	2176	4727	MODULAR	1
PROG094	0H	CONFIDENCE TEST	MEDIUM	JOVIAL	6352	18136	MODULAR	21
PROG095	1E	LOGICAL	MEDIUM	JOVIAL	320	301	MODULAR	13
PROG096	0A	CONFIDENCE TEST	MEDIUM	JOVIAL	1789	6190	MODULAR	7
PROG097	1J	CONFIDENCE TEST	MEDIUM	JOVIAL	2522	4652	MODULAR	17
PROG098	1C	CONFIDENCE TEST	SIMPLE	DIRECT	1285	1029	UNSTRUCTURED	7
PROG099	0D	CONFIDENCE TEST	MEDIUM	ASSEMBLER	8931	8379	UNSTRUCTURED	7
PROG100	1C	CONFIDENCE TEST	MEDIUM	ASSEMBLER	8888	8098	UNSTRUCTURED	5
PROG101	1C	CONFIDENCE TEST	MEDIUM	JOVIAL	2078	5200	MODULAR	1
PROG102	1C	CONFIDENCE TEST	MEDIUM	JOVIAL	1926	4655	UNSTRUCTURED	6
PROG103	0B	CONFIDENCE TEST	MEDIUM	JOVIAL	2001	6646	MODULAR	1
PROG104	1B	CONFIDENCE TEST	SIMPLE	JOVIAL	1194	3492	UNSTRUCTURED	2
PROG105	0B	CONFIDENCE TEST	MEDIUM	JOVIAL	345	714	UNSTRUCTURED	8
PROG106	1C	DATA BASE	SIMPLE	JOVIAL	64	184	MODULAR	5
PROG107	0B	CONFIDENCE TEST	MEDIUM	JOVIAL	447	958	UNSTRUCTURED	1
PROG108	1H	MATHEMATICAL	MEDIUM	JOVIAL	572	1317	UNSTRUCTURED	44
PROG109	1G	CONFIDENCE TEST	MEDIUM	ASSEMBLER	7778	8722	MODULAR	15
PROG110	0H	MATHEMATICAL	MEDIUM	JOVIAL	157	305	UNSTRUCTURED	11
PROG111	0F	LOGICAL	MEDIUM	JOVIAL	319	585	UNSTRUCTURED	23
PROG112	0H	I/O	MEDIUM	JOVIAL	218	559	UNSTRUCTURED	27
PROG113	0B	LOGICAL	MEDIUM	JOVIAL	333	621	UNSTRUCTURED	5
PROG114	0B	LOGICAL	MEDIUM	JOVIAL	260	397	UNSTRUCTURED	1

MODULE ID	VERSION	MODULE FUNCTION	COMPLEXITY	SOURCE LANGUAGE	SOURCE SIZE	OBJECT SIZE	MODE OF CONSTRUCTION	N-SPRS
PROG115	3A	CONFIDENCE TEST	MEDIUM	JOVIAL	1413	5898	UNSTRUCTURED	9
PROG116	1D	CONFIDENCE TEST	MEDIUM	JOVIAL	2251	5563	UNSTRUCTURED	4
PROG117	5G	DATA BASE	MEDIUM	JOVIAL	346	514	MODULAR	25
PROG118	0A	LOGICAL	MEDIUM	JOVIAL	97	81	UNSTRUCTURED	0
PROG119	0A	LOGICAL	MEDIUM	DIRECT	173	128	UNSTRUCTURED	0
PROG120	0A	CONTROL	SIMPLE	JOVIAL	1493	2135	UNSTRUCTURED	0
PROG121	0A	LOGICAL	MEDIUM	JOVIAL	483	746	UNSTRUCTURED	0

APPENDIX C

SOFTWARE PROBLEM REPORTS
SAMPLE OF FILE NO. 2 LISTING

SPR NUMBER	MODULE AFFECTED	VERSION	DATE OPENED	TERMINATION	SERIOUSNESS	TEST PERIOD	ERROR CATEGORY	SMN NO.	CORRECTION TYPE CD MDOAE DOMTX DCPAP		DATE CLOSED	DAYS OPEN
0892	PROG066	5C	12/01/72	NORMAL	MEDIUM	INTEGRATION	LL040	0892	X		01/16/73	28
0893	PROG025	4D	12/01/72	NORMAL	IMPROVEMENT	INTEGRATION	LL026	0893	X		01/15/73	30
0894	PROG067	3C	12/01/72	NORMAL	MEDIUM	INTEGRATION	BB010	0894	X		01/15/73	28
0895	PROG009	2J	12/01/72	NORMAL	LOW	INTEGRATION	BB170	0895	X		02/26/73	60
0898	PROG051	2C	12/08/73	NORMAL	MEDIUM	INTEGRATION	LL010	0898	X		01/31/74	35
0899	PROG051	2C	12/08/72	NORMAL	MEDIUM	INTEGRATION	LL010	0899	X		02/05/73	38
0901	PROG026	2C	12/20/72	SOFTWARE	CRITICAL	INTEGRATION	KK010	0901		X	04/24/73	88
0902	PROG078	7D	12/21/72	NORMAL	CRITICAL	INTEGRATION	DD050	0902			01/02/73	5
0903	PROG078	7D	12/21/72	NORMAL	CRITICAL	INTEGRATION	DD050	0903	X		01/02/73	5
0904	PROG078	7D	12/28/72	NORMAL	LOW	INTEGRATION	DD090	0904	X		01/02/73	2
0905	PROG036	4J	12/18/72	NORMAL	LOW	INTEGRATION	DD040	0905	X		01/03/73	9
0906	PROG036	4J	12/18/72	NORMAL	MEDIUM	INTEGRATION	DD100	0906	X		01/03/73	9
0907	PROG036	4J	12/18/72	SOFTWARE	CRITICAL	INTEGRATION	DD100	0907	X		01/03/73	9
0908	PROG036	4J	12/20/72	SOFTWARE	CRITICAL	INTEGRATION	DD180	0908	X		01/03/73	7
0909	PROG066	5C	12/07/72	NORMAL	CRITICAL	INTEGRATION	DD071	0909	X		12/29/72	14
0910	PROG066	5C	12/07/72	NORMAL	CRITICAL	INTEGRATION	BB070	0910	X		12/29/72	14
0911	PROG066	5C	12/07/72	NORMAL	MEDIUM	INTEGRATION	DD050	0911	X		12/29/72	14
0912	PROG066	5C	12/07/72	NORMAL	MEDIUM	INTEGRATION	BB070	0912	X		12/29/72	14
0913	PROG066	5C	12/07/72	NORMAL	MEDIUM	INTEGRATION	DD041	0913	X		01/02/73	15
0914	PROG066	5C	12/20/72	NORMAL	MEDIUM	INTEGRATION	KK010	0914	X	X	01/02/73	6
0915	PROG066	5C	12/20/72	NORMAL	MEDIUM	INTEGRATION	DD050	0915	X		01/02/73	6
0916	PROG066	5C	12/20/72	NORMAL	MEDIUM	INTEGRATION	DD050	0916	X		01/02/73	6
0917	PROG066	5C	12/28/72	NORMAL	CRITICAL	INTEGRATION	BB020	0917	X		12/29/72	1
0918	PROG068	0C	12/21/72	NORMAL	MEDIUM	INTEGRATION	TT060	0918	X		01/02/73	5
0919	PROG068	0C	12/21/72	SOFTWARE	CRITICAL	INTEGRATION	BB070	0919	X		01/02/73	5
0920	PROG068	0C	12/21/72	NORMAL	MEDIUM	INTEGRATION	HH020	0920	X		01/02/73	6
0921	PROG068	0C	12/28/72	NORMAL	MEDIUM	INTEGRATION	KK010	0921	X		01/02/73	2
0922	PROG068	0C	12/28/72	NORMAL	MEDIUM	INTEGRATION	DD050	0922	X		01/02/73	2
0924	PROG066	5C	01/02/73	NORMAL	MEDIUM	INTEGRATION	BB140	0924	X		01/02/73	1
0925	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	BB060	0925	X		01/04/73	2
0926	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	DD050	0926	X		01/04/73	2
0927	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	BB070	0927	X		01/04/73	2
0928	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	AA130	0928	X		01/04/73	2
0929	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	DD050	0929	X		01/04/73	2
0930	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	DD050	0930	X		01/04/73	2
0931	PROG068	0C	01/02/73	NORMAL	MEDIUM	INTEGRATION	DD050	0931	X		01/04/73	2
0932	PROG104	1B	01/02/73*	NORMAL	MEDIUM	INTEGRATION	DD050	0932	X		01/02/73	1
0933	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	BB070	0933	X		01/24/73	14
0934	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	BB070	0934	X		01/24/73	14
0935	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	DD050	0935	X		01/24/73	14
0936	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	DD050	0936	X		01/24/73	14
0937	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	DD050	0937	X		01/24/73	14
0938	PROG115	3A	01/04/73	NORMAL	MEDIUM	INTEGRATION	DD050	0938	X		01/24/73	14
0941	PROG066	5C	01/10/73	NORMAL	MEDIUM	INTEGRATION	BB020	0941	X		03/02/73	37
0942	PROG066	5C	01/10/73*	NORMAL	MEDIUM	INTEGRATION	BB020	0942	X		01/10/73	1

APPENDIX D

ERROR CATEGORIES (FAULT TAXONOMY) FILE NO. 3 LISTING

ERROR CATEGORY	DESCRIPTION	N-SPRS
AA000	*** COMPUTATIONAL ERRORS ***	0
AA010	TOTAL NUMBER OF ENTRIES COMPUTED INCORRECTLY	1
AA030	INDEX COMPUTATION ERROR	1
AA040	WRONG EQUATION OR CONVENTION USED	32
AA041	MATHEMATICAL MODELING PROBLEM	3
AA050	RESULTS OF ARITHMETIC CALCULATION INACCURATE/NOT AS EXPECTED	9
AA060	MIXED MODE ARITHMETIC ERROR	0
AA070	TIME CALCULATION ERROR	8
AA071	TIME CONVERSION ERROR	1
AA080	SIGN CONVENTION ERROR	6
AA090	UNITS CONVERSION ERROR	2
AA100	VECTOR CALCULATION ERROR	1
AA110	CALCULATION FAILS TO CONVERGE	0
AA120	QUANTIZATION/TRUNCATION ERROR	20
AA130	SCALING ERROR	31
BB000	*** LOGIC ERRORS ***	1
BB010	LIMIT DETERMINATION ERROR	16
BB020	WRONG LOGIC BRANCH TAKEN	46
BB030	LOOP EXITED ON WRONG CYCLE	2
BB040	INCOMPLETE PROCESSING	13
BB050	ENDLESS LOOP DURING ROUTINE OPERATION	6
BB060	MISSING LOGIC OR CONDITION TEST	69
BB061	INDEX NOT CHECKED	2
BB062	FLAG OR SPECIFIED DATA VALUE NOT TESTED	8
BB070	INCORRECT LOGIC	118
BB080	SEQUENCE OF ACTIVITIES WRONG	28
BB090	FILTERING ERROR	3
BB100	STATUS CHECK/PROPOGATION ERROR	2
BB110	ITERATION STEP SIZE INCORRECTLY DETERMINED	0
BB120	LOGICAL CODE PRODUCED WRONG RESULTS	0
BB130	LOGIC ON WRONG ROUTINE	1
BB140	PHYSICAL CHARACTERISTICS OF PROBLEM, OVERLOOKED OR MISUNDERSTOOD	47
BB150	LOGIC NEEDLESSLY COMPLEX	0
BB160	INEFFICIENT LOGIC	2
BB170	EXCESSIVE LOGIC	16
BB180	STORAGE REFERENCE ERROR (SOFTWARE PROBLEM)	2
CC000	*** I/O ERRORS ***	0
CC010	MISSING OUTPUT	9
CC020	OUTPUT MISSING DATA ENTRIES	0
CC030	ERROR MESSAGE NOT OUTPUT	0
CC040	ERROR MESSAGE GARBLED	1
CC050	OUTPUT OR ERROR MESSAGE NOT COMPATIBLE WITH DESIGN DOCUMENTATION	3
CC060	MISLEADING OR INACCURATE ERROR MESSAGE TEXT	1
CC070	OUTPUT FORMAT ERROR (INCLUDING WRONG LOCATION)	1
CC080	DUPLICATE OR EXCESSIVE OUTPUT	0
CC090	OUTPUT FIELD SIZE INADEQUATE	1
CC100	DEBUG OUTPUT PROBLEM (RELATIVE TO DESIGN DOCUMENTATION)	2
CC101	LACK OF DEBUG OUTPUT	1

ERROR CATEGORY	DESCRIPTION	N.SPRS
CC102	TOO MUCH DEBUG	1
CC110	HEADER OUTPUT PROBLEM	0
CC120	OUTPUT TAPE FORMAT ERROR	1
CC130	OUTPUT CARD FORMAT ERROR	0
CC140	ERROR IN PRINTER CONTROL	0
CC150	LINE COUNT/PAGE EJECT ERROR	0
CC160	NEEDED OUTPUT NOT PROVIDED IN DESIGN	0
CC161	INSUFFICIENT OUTPUT OPTIONS	0
DD000	*** DATA HANDLING ERRORS ***	0
DD010	VALID INPUT DATA IMPROPERLY SET/USED	27
DD020	DATA WRITTEN ON OR READ FROM WRONG TAPE OR DISK LOCATION	2
DD030	DATA LOST/NOT STORED	12
DD040	DATA, INDEX, OR FLAG NOT SET OR SET/INITIALIZED INCORRECTLY	65
DD041	NUMBER OF ENTRIES SET INCORRECTLY	11
DD050	DATA, INDEX, OR FLAG MODIFIED OR UPDATED INCORRECTLY	196
DD051	NUMBER OF ENTRIES UPDATED INCORRECTLY	5
DD060	EXTRANEOUS ENTRIES GENERATED (TABLE ARRAY, ETC)	0
DD070	BIT MANIPULATION ERROR	7
DD071	ERROR USING BIT MODIFIER	2
DD080	FLOATING POINT/INTEGER CONVERSION ERROR	0
DD090	INTERNAL VARIABLE ERROR (DEFINITION OR SET/USE)	30
DD100	DATA PACKING/UNPACKING ERROR	15
DD110	ROUTINE LOOKING FOR DATA IN NON-EXISTENT RECORD	0
DD120	BOUNDS VIOLATION	2
DD130	DATA CHAINING ERROR	1
DD140	DATA OVERFLOW OR OVERFLOW PROCESSING ERROR	33
DD150	READ ERROR	0
DD151	ALL AVAILABLE DATA NOT READ	0
DD160	LONG LITERAL PROCESSING ERROR	0
DD170	SORT ERROR	0
DD180	OVERLAY ERROR	1
DD190	SUBSCRIPTING CONVENTION ERROR	0
DD200	DOUBLE BUFFERING ERROR	0
EE000	*** OPERATING SYSTEM/SYSTEM SUPPORT SOFTWARE ERRORS ***	0
EE010	JOVIAL PRODUCES ERRONEOUS MACHINE CODE	4
EE020	OS MISSING NEEDED CAPABILITY	0
FF000	*** CONFIGURATION ERRORS ***	1
FF010	COMPILATION ERROR	16
FF011	SEGMENTATION PROBLEM	1
FF020	ILLEGAL INSTRUCTION	0
FF030	UNEXPLAINABLE PROGRAM HALT	0
GG000	*** ROUTINE/ROUTINE INTERFACE ERRORS ***	0
GG010	ROUTINE PASSING INCORRECT AMOUNT OF DATA INSUFFICIENT OR TOO MUCH	2
GG020	ROUTINE PASSING WRONG PARAMETERS OR UNITS	3
GG030	ROUTINE EXPECTING WRONG PARAMETERS	1
GG040	ROUTINE FAILS TO USE AVAILABLE DATA	1
GG050	ROUTINE SENSITIVE TO INPUT DATA ORDER	0

ERROR CATEGORY	DESCRIPTION	N-SPRS
GG060	CALLING SEQUENCE OR ROUTINE/ROUTINE INITIALIZATION ERROR	6
GG070	ROUTINES COMMUNICATING THROUGH WRONG DATA BLOCK	1
GG080	ROUTINE USED OUTSIDE DESIGN LIMITATION	0
GG090	ROUTINE WON'T LOAD (ROUTINE INCOMPATIBILITY)	1
GG100	ROUTINE OVERFLOWS CORE WHEN LOADED	1
HH000	*** ROUTINE/SYSTEM SOFTWARE INTERFACE ERRORS ***	0
HH010	OS INTERFACE ERROR (CALLING SEQUENCE OR INITIALIZATION)	12
HH020	ROUTINE USES EXISTING SYSTEM SUPPORT SOFTWARE INCORRECTLY	4
HH030	ROUTINE USES SENSE/JUMP SWITCH IMPROPERLY	1
II000	*** TAPE PROCESSING INTERFACE ERROR ***	0
II010	TAPE UNIT EQUIPMENT CHECK NOT MADE	0
II020	ROUTINE FAILS TO READ CONTINUATION TAPE	0
II030	ROUTINE FAILS TO UNLOAD TAPE AFTER COMPLETION	0
II040	ERRONEOUS INPUT TAPE FORMAT	0
JJ000	*** USER INTERFACE ERRORS ***	0
JJ010	OPERATIONS REQUEST OR DATA CARD/ROUTINE INCOMPATIBILITY	0
JJ020	MULTIPLE PHYSICAL CARD/LOGICAL CARD PROCESSING ERROR	0
JJ030	INPUT DATA INTERPRETED INCORRECTLY BY ROUTINE	2
JJ040	VALID INPUT DATA REJECTED OR NOT USED BY ROUTINE	1
JJ050	INPUT DATA REJECTED BUT USED	0
JJ060	INPUT DATA READ BUT NOT USED	1
JJ070	ILLEGAL INPUT DATA ACCEPTED AND PROCESSED	3
JJ080	LEGAL INPUT DATA PROCESSED INCORRECTLY	1
JJ090	POOR DESIGN IN OPERATOR INTERFACE	1
JJ100	INADEQUATE INTERRUPT AND RESTART CAPABILITY	1
KK000	*** DATA BASE INTERFACE ERRORS ***	0
KK010	ROUTINE/DATA BASE INCOMPATIBILITY	27
KK011	UNCOORDINATED USE OF DATA ELEMENTS BY MORE THAN ONE USER	5
LL000	*** USER REQUESTED CHANGES PRODUCT IMPROVEMENTS NOT ERRORS ***	1
LL010	SIMPLIFIED INTERFACE AND/OR CONVENIENCE	116
LL020	NEW AND/OR ENHANCED FUNCTIONS	212
LL021	CPU	0
LL022	DISK	0
LL023	TAPE	9
LL024	I/O	40
LL025	CORE	7
LL026	DISPLAY *	9
LL030	SECURITY	0
LL040	NEW HARDWARE/OS CAPABILITY	40
LL050	SOFTWARE INSTRUMENTATION	89
LL060	CAPACITY	3
LL070	DATA BASE MANAGEMENT AND INTEGRITY	71
LL080	EXTERNAL PROGRAM INTERFACE	30
LL090	MODIFICATION FOR SPECIAL TEST PURPOSES *	137
MM000	*** PRESET DATA BASE ERRORS ***	0

ERROR CATEGORY	DESCRIPTION		N=SPRS
MM010	DATA OR OPERATIONS REQUEST CARD DESCRIPTIONS		3
MM020	ERROR MESSAGE TEXT		2
MM030	NOMINAL, DEFAULT, LEGAL, MAX/MIN VALUES		56
MM040	PHYSICAL CONSTANTS AND MODELING PARAMETERS		69
MM041	EPHEMERIS PARAMETERS		0
MM050	DICTIONARY (BIT STRING) PARAMETERS		1
MM060	MISSING DATA BASE SETTINGS		31
NN000	*** GLOBAL VARIABLE/COMPOOL DEFINITION ERRORS ***		0
NN010	ITEMS IN WRONG LOCATION (WRONG DATA BLOCK)		8
NN011	DEFINITION SEQUENCE ERROR		5
NN020	DATA DEFINITION ERROR		9
NN021	TABLE DEFINITION INCORRECT		10
NN030	LENGTH OF DEFINITION INCORRECT		12
NN040	COMMENTS ERROR		0
NN050	DELETE UNNEEDED DEFINITIONS		1
PP000	*** RECURRENT REPORTS		0
PP010	PROBLEM REPORT REOPENED		3
PP020	PROBLEM REPORT A DUPLICATE OF PREVIOUS REPORT		36
QQ000	*** PROGRAM COMMENTS		0
QQ010	ROUTINE LIMITATION		0
QQ020	OPERATING PROCEDURES		0
QQ030	DIFFERENCE BETWEEN FLOW CHART AND CODE		0
QQ040	TAPE FORMAT		0
QQ050	DATA CARD/OPERATION REQUEST CARD FORMAT		0
QQ060	ERROR MESSAGE		0
QQ070	ROUTINE=S FUNCTIONAL DESCRIPTION		0
QQ080	OUTPUT FORMAT		0
QQ090	DOCUMENTATION NOT CLEAR/NOT COMPLETE		2
QQ100	TEST CASE DOCUMENTATION		0
QQ110	OPERATING SYSTEM DOCUMENTATION		1
QQ120	TYPO/EDITORIAL ERROR/COSMETIC CHANGE		12
RR000	*** REQUIREMENTS COMPLIANCE ERRORS ***		0
RR010	EXCESSIVE RUN TIMES		2
RR020	REQUIRED CAPABILITY OVERLOOKED OR NOT DELIVERED AT TIME OF REPORT		8
SS000	*** UNIDENTIFIED ERRORS ***		0
SS010	HARDWARE ERROR	*	13
SS020	NON RECURRING PROBLEM	*	14
SS030	NO ERROR	*	40
SS040	INSUFFICIENT INFORMATION FOR ERROR ANALYSIS	*	10
TT000	*** OPERATOR ERROR NOT SYSTEM ERRORS ***		0
TT010	TEST EXECUTION ERROR		3
TT020	ROUTINE COMPILED AGAINST WRONG COMPOOL/MASTER COMMON		2
TT030	WRONG DATA BASE USED		1
TT040	WRONG MASTER CONFIGURATION USED		1
TT050	WRONG TAPE(S) USED		3

ERROR CATEGORY	DESCRIPTION		N-SPRS
TT060	MISSING CARDS IN COMPILED PROGRAM	*	5
UU000	*** QUESTIONS ***		0
UU010	DATA BASE		1
UU020	MASTER CONFIGURATION		1
UU030	ROUTINE		1
VV000	*** REQUIREMENTS SPECIFICATION	*	0
VV010	INADEQUATE/INEFFICIENT REQUIREMENTS	*	7
VV020	ENHANCEMENT REQUIREMENTS	*	4

APPENDIX E
STATIC STATISTICS FOR JOVIAL SOURCE MODULES

Nine modules were examined by the U1108 JOVIAL program (STATGT) to see how frequently certain statements are used in practice. Tables E-1 and E-2 show the distribution of statement types. Also, calculations are provided for executable statement types. Certain changes were made to the data to eliminate discontinuities*. The most frequently used language construct is the = sign. This is because of its use in the assignment statement (23 percent). The next most used construct is subscription (14 percent), followed by GOTO (8 percent) and IF (8 percent). Nothing can be said about the procedure call mechanism because the same construct is used for other features. The BEGIN-END delimiters are used about 6 percent of the time. This implies some blocking in the decision making logic. The EQ relational operator was most highly used (5 percent). The most used executable statements were assignment (54 percent), IF (19.7 percent), and GOTO (19.6 percent).

A typical program consisted of assignment statements and blocked condition checking statements. Programming with the use of tables appears to be prevalent. Some explicit loops are seen. Bit and byte manipulation do not appear to be frequently used.

---

*See Note 3 of Table E-1.

TABLE E-1

DISTRIBUTION AND MODULE USAGE OF STATEMENT TYPES
(9 OPERATIONAL MODULES)

No.	Constructs	Number	Percent All
1	( )[1]	454	6.76
2	IF	512	7.62
3	GOTO	534	7.95
4	FOR	82	1.22
5	TEST	19	0.28
6	CLOSE	15	0.22
7	RETURN	33	0.49
8	STOP	2	0.03
9	=[2]	1543	23.0
10	AND	24	0.36
11	OR	64	0.95
12	EQ	307	4.57
13	GR	89	1.32
14	GQ	23	0.34
15	LQ	45	0.67
16	LS	67	1.0
17	NQ	67	1.0
18	+	241	3.6
19	-	246	3.66
20	*	138	2.0
21	/	28	0.42
22	**	4	0.06
23	ABS ( )	13	0.19
24	(/ /)	12	0.18
25	NENT	21	0.31
26	NWDSEN	13	0.19
27	ALL	5	0.07
28	ENTRY	3	0.04

TABLE E-1 (Cont.)

No.	Constructs	Number	Percent All
29	'LOC	13	0.19
30	ASSIGN	25	0.37
31	BIT	57	0.85
32	BYTE	97	1.4
33	,[3]	438	- -
34	.[3]	330	- -
35	$[3]	3251	- -
36	BEGIN-END	401	5.98
37	START-TERM	9	0.13
38	DIRECT-JOVIAL	71	1.06
39	($-$)	929	13.8
40	ITEM	438	6.5
41	TABLE	26	0.38
42	ARRAY	4	0.06
43	PROC	20	0.3
44	SWITCH	14	0.2
45	OVERLAY	6	0.09
46	'PROGRAM	0	0
47	BLOCK	0	0
	Subtotal	10733	
	less	4019	
	Total	6714	100

Note:
1) expression grouping, procedure, function call
2) assignment, FOR, procedure call parameter delimiting
3) deleted from total for reasons of ambiguity

TABLE E-2

DISTRIBUTION AND MODULE USAGE OF EXECUTABLE STATEMENTS

No.	Constructs	Percent All
1	IF	19.70
2	GOTO	19.60
3	FOR	3.18
4	TEST	0.73
5	CLOSE	0.58
6	RETURN	1.27
7	STOP	0.07
8	=(assignment)	54.00

APPENDIX F

CONSTITUENT PROGRAM MODULES OF BUILDS "F" AND "G"

Refer to Appendix B (Software Module Descriptions) for further information about each of these modules listed.

Build F - Operation Build (55 modules)

PROG001, 6, 8, 9, 11, 12, 13, 14, 15, 16, 20, 21, 24, 25, 27, 28, 29, 36, 39, 41, 43, 45, 46, 50, 52, 53, 58, 59, 60, 62, 64, 65, 66, 67, 72, 75, 76, 81, 82, 84, 86, 87, 88, 92, 95, 106, 108, 110, 111, 112, 113, 114, 117, 118, 119.

Build G - Initialization Build (25 modules)

PROG002, 57, 70, 71, 77, 79, 85, 89, 91, 93, 94, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 107, 109, 116, 120.

REFERENCES

1) Thayer, T. A., et al, "Software Reliability Study," TRW Defense and Space Systems Group, Interim Technical Report, RADC-TR-74-250, October 1974. AD-787-784.

2) Thayer, T. A. et al, "Software Reliability Study," TRW Defense and Space Systems Group, Final Technical Report (16 Oct 73 - 27 Feb 76), RADC-TR-76-238, August 1976. AD-A030-798.

3) Air Force Manual AFM 100-24, "Standard Computer Programming Language For Air Force Command and Control Systems," CEC-2400, 21 April 1972.

4) Sukert, A. N., "A Software Reliability Modeling Study," RADC-TR-76-247, August 1976.

BIBLIOGRAPHY

1) Tucker, A.E., "The Correlation of Computer Programming Quality with Testing Effort," SDC, TM-2219/000/00, 26 January 1965.

2) Barney, D. R., Giloth, P. K., and Kiengle, H. G., "System Testing and Early Field Operation Experience," Bell System Technical Journal, December 1970, pp. 2975-3004.

3) Knuth, D. E., "An Empirical Study of FORTRAN Programs," Software Practice and Experience, 1, 1971, pp. 105-133.

4) Shooman, M. L., "Operational Testing and Software Reliability Estimation During Program Development," 1973 IEEE Symposium on Computer Software Reliability, 30 April - 2 May 1973, pp. 51-57.

5) Boehm, B. W., "Software and Its Input: A Quantitive Assessment," DATAMATION, May 1973.

6) Wagoner, W. L., "The Final Report on a Software Reliability Measurement Study," ASCO, TOR-0047(4112)-1, 15 August 1973.

7) Brooks, F. D., Jr., "The Mythical Man-Month-Essays on Software Engineering," Addison-Wesley, 1975.

8) Thayer, T. A., "Understanding Software through Empirical Reliability Analysis," Proceedings of the National Computer Conference, 1975.

9) Elshoff, J. L., "An Analysis of Some Commercial PL/1 Programs," IEEE Transactions on Software Engineering, Vol. SE-2, No. 2, June 1976, pp. 113-120.

10) Wichmann, B. A., "A Comparison of Algol 60 Execution Speeds," CCU Report No. 3, NPL, Teddington, Middlesex.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

Quantity	Unit	SI Symbol	Formula
plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Quantity	Unit	SI Symbol	Formula
Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. 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.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976