AD-A042 186 CHARLES STARK DRAPER LAB INC CAMBRIDGE MASS F/G 9/2
SOFTWARE SYSTEMS DEVELOPMENT: A CSDL PROJECT HISTORY.(U)
JUN 77 P RYE, F BAMBERGER, W OSTANEK F30602-76-C-0151
UNCLASSIFIED R-1073 RADC-TR-77-213 NL

1 OF 2
ADA042186

(12)

RADC-TR-77-213
Final Technical Report
June 1977

SOFTWARE SYSTEMS DEVELOPMENT: A CSDL PROJECT HISTORY

The Charles Stark Draper Laboratory, Inc.

Approved for public release; distribution unlimited.

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

This report contains some pages which are not of the highest printing quality, but because of economical consideration, it was determined to be in the best interest of the government that they be used in this publication.

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

Do not return this copy. Retain or destroy.

UNCALSSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-213	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SOFTWARE SYSTEMS DEVELOPMENT: A CSDL PROJECT HISTORY	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, 1967 - 1971
6. PERFORMING ORG. REPORT NUMBER: R-1073	
7. AUTHOR(s): P. Rye, F. Bamberger, W. Ostanek, N. Brodeur, J. Goode	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0151
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F 55811408
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS) Griffiss AFB NY 13441	12. REPORT DATE: June 1977
	13. NUMBER OF PAGES: 91
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 Development
Software Modification
Software Reliability
APOLLO Flight Software

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

This report provides a description of the data delivered to RADC for inclusion in a Software Data Repository. The data consists of a complete history of software modifications to the APOLLO on-board flight software for the period 1967 through 1971. Background material on the project that was the source of the data is provided, as well as tabular and graphic summaries of the data. Some recommendations for future work are made.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 386

CONTENTS

CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF TABLES

EVALUATION

The cost of producing software has increased at a rapid pace, and, as a result, much has been said about the unreliability problems associated with producing this software. The need by the software industry in general, and the military establishment in particular, to produce reliable, maintainable and quality software, is ever increasing. Investigations into the types of software errors and the reasons for their occurrence are taking place, but are somewhat deterred by an insufficient quantity of error data. The error data base can be utilized for in-depth analysis as well as testing software error prediction models.

In an attempt to overcome this insufficient data base and to respond to these needs, this effort was initiated. The effort is in accord with the goals of RADC TPO No. 5, Software Cost Reduction (formerly RADC TPO No. 11, Software Sciences Technology); particularly in the area of Software Quality (Software Data). The report provides a description of the delivered data. The data consists of a history of software modifications to an on-board flight software project for a specific period. The value of acquiring this data is that it will be analyzed for the purpose of developing software measurements and will also be used to support current software model development projects. In addition, this data will be used concurrently with other procured software error data, to aid in establishing a baseline for on-board flight software projects in quantitative terms. This class of information will, in the future, influence better methods of developing on-board flight software projects.

James V Cellini, Jr

JAMES V. CELLINI, Jr.
Project Engineer

SECTION 1

INTRODUCTION

This study is based on a unique experience: Project Apollo, the manned lunar landing, an effort combining many technological disciplines. The Instrumentation Laboratory of The Massachusetts Institute of Technology (MIT/IL), since incorporated as The Charles Stark Draper Laboratory, Inc. (CSDL), was given responsibility by NASA for the guidance, navigation and control (GNC) functions of the Apollo space vehicles. The data investigated by this study are derived from the software developed for the computers of the onboard Primary GNC Systems (PGNCS) of both the Command Module (CM) and Lunar Module (LM).

This study compiles and categorizes modifications that were made to the flight software programs during the development of the PGNCS of both CM and LM. This material was recorded on magnetic tape; the data contents are described in Section 4 of this report. The purpose of this effort is to contribute to the establishment of a software error data base to support research in software reliability.

A great deal of support software was produced at the laboratory for the Apollo project in addition to the flight software: simulators, assemblers, data management systems, post processors, and engineering simulators. Figures for computer usage and staffing include this total software effort; the modification data collected and described in this report, however, are only from the flight programs. These data are compiled from changes made to the flight software for Apollo flights 7 through 17. Apollo flights 16 and 17 used the same software as Apollo 15. The data were collected from flight software developed over the years 1967 to 1971.

This report provides a description of the project used as the source for these data, and describes the data base itself. Section 2 of this report describes the flight computer architecture, the functional nature of the software, its production, testing and management. Section 3 discusses the life cycle of the software. Section 4 describes in detail the collected data and its format. Section 5 summarizes the data in graphic and tabular form. Section 6 presents some conclusions and recommendations drawn from this activity.

SECTION 2

BACKGROUND

2.1 FLIGHT COMPUTER ARCHITECTURE

The Apollo Guidance Computer designed and developed by the MIT/IL was advanced for its day, being small, light-weight and highly reliable. During all the hours of testing and actual flight, not one failure was ever recorded. The AGC was used throughout all the Apollo, Skylab, Apollo-Soyuz and F-8 Phase I programs.

2.1.1 MEMORY ORGANIZATION

The computer consisted of a fixed wired program memory (called a "core rope*") of 36,864 words in 36 banks and a read-write ("erasable") memory of 2048 words in 8 banks. Words were 15 bits plus parity; the memory cycle time was 12 micro-seconds. Fixed memory could not be changed after manufacture.

There were 34 possible machine instructions. Since 15 bits were not sufficient to specify an op-code and all 38,912 memory addresses, computer core was divided into banks and programmable bank selection registers were provided in the CPU. Any instruction could specify any address within its own bank and could also address the first 3 banks of erasable as well as the first 2 banks of fixed memory. Access to any other bank was accomplished by using the bank registers. The limited size of erasable memory forced the time-sharing of these locations. Many software modifications resulted from this memory organization and appear in such error categories as V050, program memory optimization, N010, items in wrong location, F040, organization problem.

Throughout all Apollo programs, there was a "memory crunch". Extreme efforts were expended by programmers to be "clever" in order to save even a few memory locations. Not surprisingly, this added to the difficulty of the debugging process and made ongoing program modification a tricky business. In some instances modifications were not made even though potential problems were known to exist because it was felt that known problems were less hazardous than new problems that could be introduced by a complex correction. Care was taken that the problems were well understood and work-around procedures were developed, when required, to avoid them.

---

* The term "rope" came to be used by Apollo engineers to denote each program intended for release, even while it was under development. "Rope" is also used elsewhere in this report in that sense.

The fact that the program memory was hard-wired led to an effort (not covered by this study) which extended through the late Apollo flights, Skylab, and the Apollo-Soyuz programs. It consisted of the development of Erasable Memory Programs (EMPs) which were implemented in order to provide modifications to a flight program while avoiding remanufacture of the "rope". The EMPs resided in erasable memory, time-sharing even further that scarce resource. Needless to say, extreme caution was used in specifying, designing, implementing and testing each EMP. An experienced team of experts participated in each phase of EMP development, and the astronauts and ground crews were thoroughly briefed on the use and limitations of each EMP.

2.1.2 INTERRUPT SYSTEM

Ten interrupt levels were provided. They are shown in Table 2-1. Of these ten, four were programmable counters of varying granularity, the finest being 10 milliseconds. These counters were utilized to provide cyclic servicing of the vehicle control functions, hardware calibration and other servicing such as display refresh and telemetry.

One of the counters was used to service a time queue by which the operating system dispatched asynchronously timed tasks to be processed in the interrupt mode. Processing in the interrupt mode was constrained to a time limit of 14 milliseconds, since other interrupts were masked while the current interrupt was being processed. Computer hardware failure monitors were provided and are described in Table 2-2. Any of these failures caused a GOPROG interrupt and a subsequent software restart.

2.1.3 HARDWARE INTERFACES

The computers were interfaced to the several hardware components shown in Figures 2-1 and 2-2.

All input/output (I/O) took place through counters and channels. Counters were used for the transmission and reception of numeric data; channels were used for the communication of discrete data. Counters were accessed by direct memory access on a cycle-steal basis.

2.2 FLIGHT SOFTWARE FUNCTIONS

2.2.1 MISSION PROGRAMS

The purpose of the AGC was to compute guidance, targeting, navigation, and control functions for the Apollo space vehicles for all mission phases. Many guidance, targeting and navigation functions were computed by the ground control system and

Table 2-1 INTERRUPT STRUCTURE

INTERRUPT	PURPOSE	HIERARCHY
GOPROG	RESTARTS	1
T6RUPT	JET TURN ON/OFF	2
T5RUPT	DAP	3
T3RUPT	WAITLIST	4
T4RUPT	IMU/OPTICS MONITORING	5
KEYRUPT(2)	KEYSTROKE PROCESSING	6,7
UPRUPT	UPTELEMETRY PROCESSING	8
DOWNRUPT	DOWNLINK PROCESSING	9
RADRUPT	RADAR RETURN PROCESSING	10

Table 2-2 COMPUTER HARDWARE INTERRUPTS

OSCILLATOR FAIL Occurs if loss of oscillator 1.02 MHz square wave happens. In addition a logic circuit insures a RESTART condition for a 250 millisecond interval upon transferring from STANDBY to OPERATE.
TRANSFER CONTROL (TC) TRAP Occurs if too many or too few TC instructions are requested. The period for "too many" or "too few" is from 5 to 15 milliseconds in duration.
PARITY ALARM Occurs if any accessed word in fixed or erasable memory whose address is 10 octal or greater contains an even parity of "ones." All locations of 10 octal or greater are stored in fixed or erasable memory with odd parity.
NIGHTWATCHMAN FAIL Occurs if the computer should fail to access address 67 within a period whose duration varies from .64 to 1.92 seconds. This assures that the computer is still operating during an extended idle period and is tied up in an interrupt loop.
INTERRUPT (RUPT) LOCK Occurs if an interrupt is either "too long" or "too infrequent" .
VOLTAGE FAIL Occurs if the AGC voltages are out of limits for 157 to 470 microseconds.

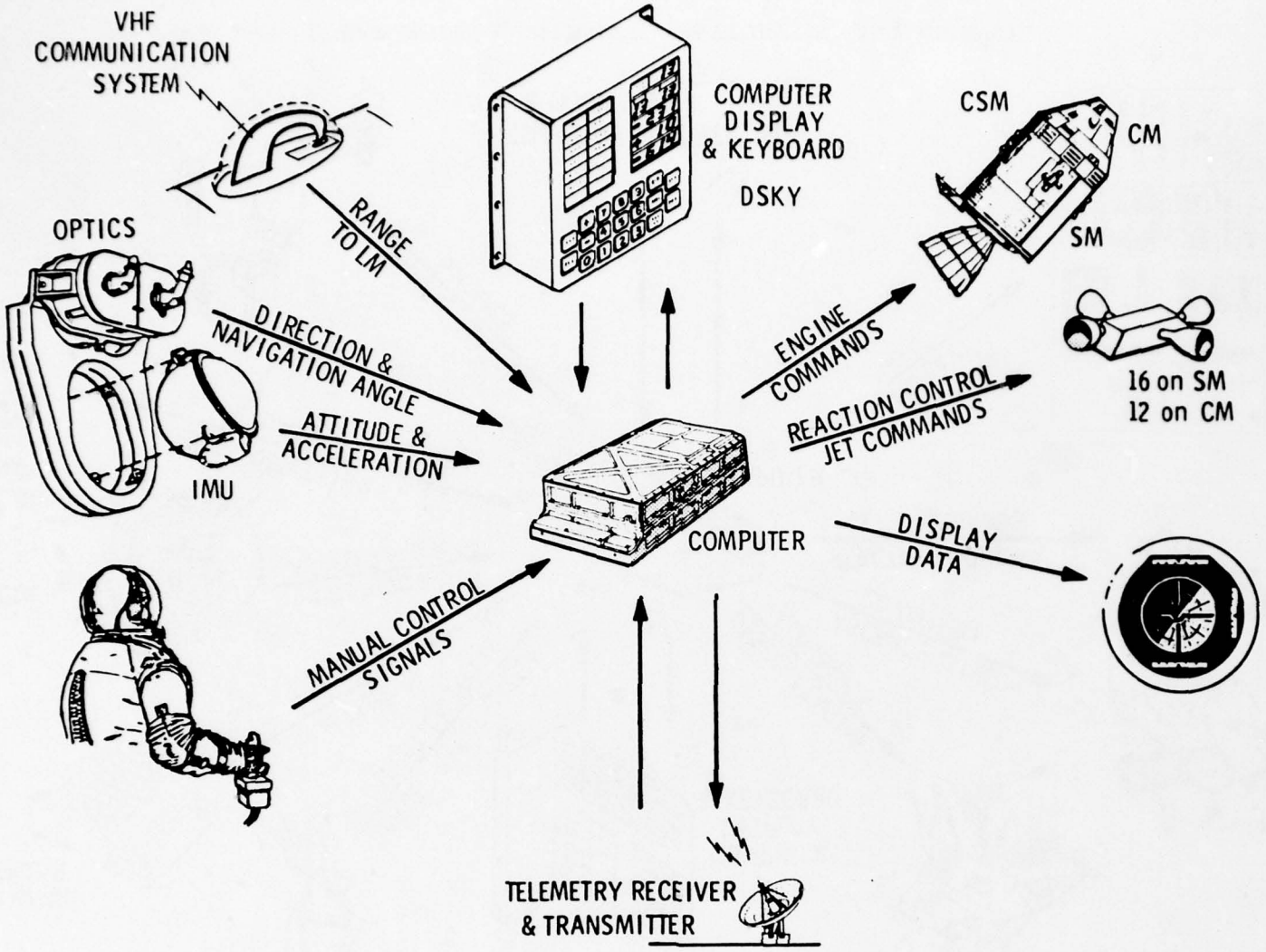


Figure 2-1 COMMAND MODULE GUIDANCE, NAVIGATION AND CONTROL

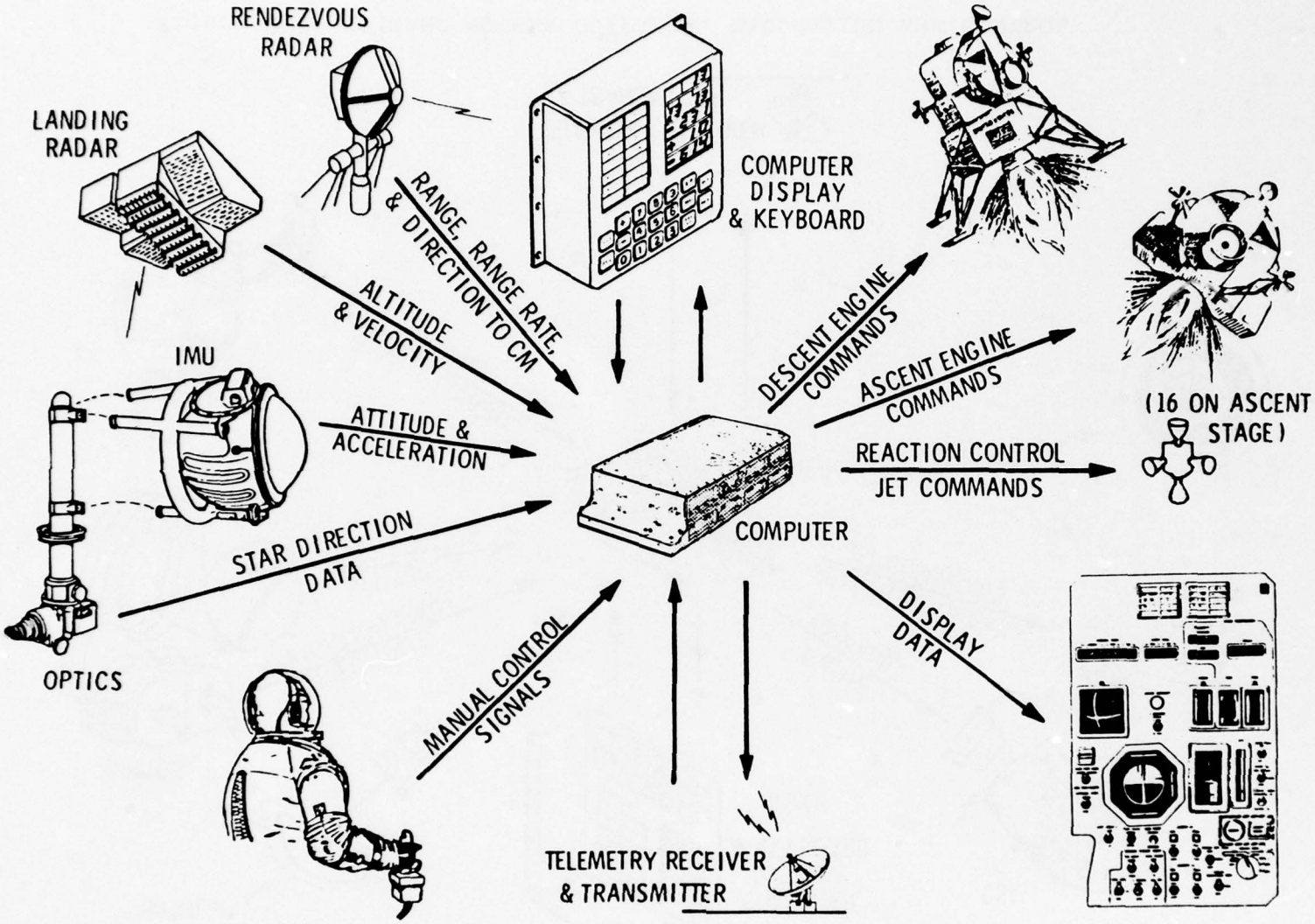


Figure 2-2 LUNAR MODULE GUIDANCE, NAVIGATION AND CONTROL

duplicated on-board, but control functions were, of course, an on-board responsibility; so too were computations of range, range-rate, velocity requirements, and other parameters for lunar landing and rendezvous. These computations were based on radar and optics sensor inputs and monitored by the astronauts, who had final authority over all activity. The following definitions of these functions appear in reference 1.

> Navigation - the measurement and computation necessary to determine the present spacecraft position and velocity.
>
> Targeting - the computation of the maneuver required to continue to the next step in the mission.
>
> Guidance - the continuous measurement and computation during accelerated flight to generate steering signals necessary to assure that the position and velocity changes of the maneuver will be those required by navigation measurements and targeting computations.
>
> Control - the management of spacecraft attitude motion; the rotation to and maintenance of the desired spacecraft attitude during free-fall coasting flight and powered accelerated flight.

Table 2-3 lists the major mission programs that were available for astronaut selection.

2.2.2 CREW INTERFACE

Because of the importance of the man-machine interface, a significant portion of the software effort was devoted to displays and handling of crew keyboard inputs. At significant points in the processing, a display was presented to the crew; the program did not advance further until directed to do so by the crew (depression of a PROCEED key was the standard method of crew approval).

These paragraphs are adapted from material presented in reference 1.

> The basic man/computer interface device is the display keyboard (DSKY) (shown in Figure 2-3). Through the DSKY the astronaut could initiate, monitor, or change programs being processed by the computer. He could request the display of specific data or enter new data. Communication with the DSKY was two-way: the astronaut could exercise command via the DSKY and the computer could request the astronaut to monitor, approve, or enter data when necessary. There were two

Table 2-3 MISSION PROGRAMS

COMMAND MODULE	LUNAR MODULE
00 CMC Idling	00 LGC Idling
01 Prelaunch Initialization	06 GNCS Power Down
02 Gyro Compassing	ASCENT
03 Verify Gyro compassing	12 Ascent Guidance
06 CMC Power Down	NAVIGATION
07 IMU Ground Test	20 Rendezvous Navigation
EARTH ORBIT INSERTION	21 Ground Track
11 Earth Orbit Insertion	22 Surface Navigation
NAVIGATION	25 Preferred Tracking Attitude
20 Rendezvous Navigation	27 LGC Update
21 Ground Track Determination	RENDEZVOUS
22 Orbital Navigation	30 External Delta V
23 Cislunar Navigation	31 Lambert Aimpoint Maneuver
27 CMC Update	32 Coelliptic Sequence
RENDEZVOUS TARGETING	33 Constant Delta Height
30 External Delta V	34 Transfer Phase Initiation
31 Lambert Aimpoint	35 Transfer Phase Midcourse
32 Coelliptic Sequence	POWERED FLIGHT
33 Constant Delta Height	40 DPS Maneuver
34 Transfer Phase Init	41 RCS Maneuver
35 Transfer Phase Midcourse	42 APS Maneuver
37 Return to Earth	47 Thrust Monitor
POWERED FLIGHT	IMU ALIGNMENT
40 SPS Maneuver	51 IMU Orientation
41 RCS Maneuver	52 IMU Realign
IMU ALIGNMENT	57 Lunar Surface Align
51 IMU Orientation	LANDING
52 IMU Realign	63 Braking Phase
53 Backup IMU Orientation	64 Approach Phase
54 Backup IMU Realign	65 Landing Phase (auto)
ENTRY	66 Landing Phase (ROD)
61 Maneuver and Separation	67 Landing Phase (manual)
62 Separation and Reentry	68 Landing Confirmation
63 Entry-Initiation	BACKUP
64 Entry-Post .05 g	70 DPS Abort
65 Entry-Up Control	71 APS Abort
66 Entry-Ballistic	72 CSM CSI Targeting
67 Entry-Final Phase	73 CSM CDH Targeting
LM RENDEZVOUS TARGETING	74 CSM TPI Targeting
72 LM Coelliptic Sequence	75 CSM TPM Targeting
73 LM Constant Delta Height	76 Target Delta V
74 LM TPI	
75 LM TPM	
76 Target Delta V	

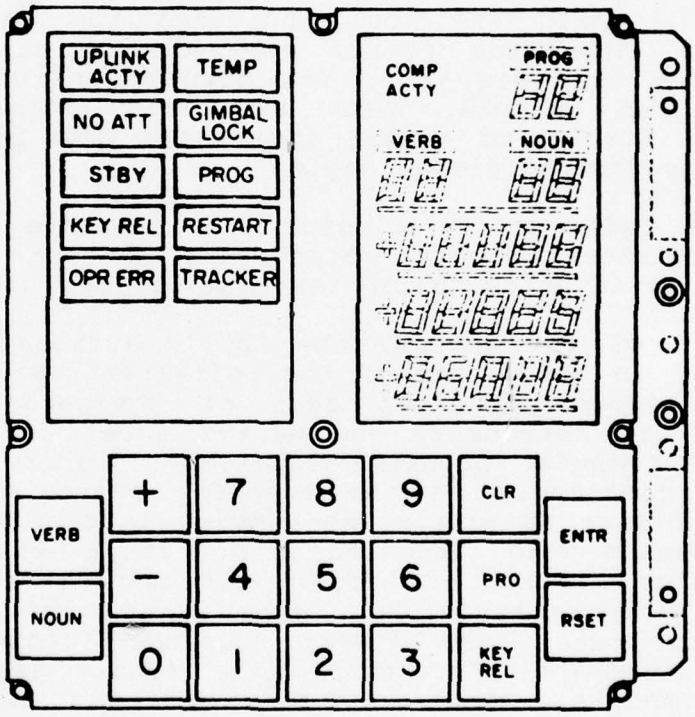


Figure 2-3 DISPLAY AND KEYBOARD

DSKYs available in the CM and one in the LM. Each DSKY had a keyboard, several electro- luminescent displays, and activity and alarm lights. The activity lights were for the computer and the telemetry uplink, and the alarm lights were for the computer and inertial subsystems. These aided the astronaut in monitoring the status of the G&N system. The alarm lights indicated equipment failure and program alarms.

The basic language used for communication between the operator and the computer was a pair of two character numbers that represented a verb/noun combination. The verb code indicated the operation to be performed, while the noun indicated the operand to which the operation (verb) applied. Typical of the verb codes used are those for displaying and loading data. For example, VERB 16 NOUN 20 provided a monitored display of gimbal angles and VERB 25 NOUN 18 loaded the desired gimbal angles.

Noun codes called up groups of erasable registers within computer memory. Processing of nouns provided information in units scaled for ease of crew use.

The users of the system, the Apollo astronauts, were very much involved in the design of the software/crew interface. In the early flights, a great deal of keyboard activity was required of the astronauts who wanted total control over the selection of computer program sequences. Later in the Apollo program, as confidence in the computer system increased, the crews were willing to relinquish some of that authority and the program sequences were automated to a greater extent. One significant effort in that direction was the development of the MINKEY (for minimum keystroke) sequence in the command module rendezvous program. The MINKEY sequence automated what was formerly a burdensome period for the single astronaut aboard the command module. Another effort that was undertaken after the first manned flights was the incorporation into the software of the error checking of all crew inputs. Experience had shown that illegal or incorrect inputs caused aberrant behavior and could have seriously impacted mission performance. Legality checks in the software, although consuming scarce memory resources, were deemed necessary to guard against a potentially dangerous source of error.

2.2.3 RESTARTABILITY

A unique aspect of the Apollo software was its built-in, real-time error recovery and restart capability. This capability was implemented by storing restart pointers at strategic points in each process. The pointers were stored in "restart tables",

which accommodated up to six separate processes at any time. Determination of restart points was a complex process based on the repeatability of coding sequences. Updating of variables was protected by storing a copy of the updated value, inserting a restart point, then replacing the old values with the newly computed ones. Other coding sequences posed more difficult logic problems for the programmers, sometimes necessitating multiple restart points within a few lines of code. The multi-programmed nature of the software contributed to the complexity of the restart design. The restart tables were time-shared, so that the assignment of table locations to various processes required expert knowledge of the total program activity throughout the mission. Also, insertion of restart points in the code had to take into consideration the possibility that processes could be preempted by a higher priority process at almost any time. Although this capability was expensive in terms of memory usage, as well as being a significant source of errors during development, this feature was directly responsible for the success of the Apollo 11 and 12 missions, which may well have failed without it.*

Modifications to the software for the implementation of this capability are recorded under categories H040, inadequate restart, or H050, errors in restart logic.

2.2.4 MULTIPROGRAMMING AND ASYNCHRONISM

The AGC operating system was designed and implemented very early in the program, before the time period covered by this study. It provided great flexibility in that it allowed for both synchronous, precisely timed cyclic processing and asynchronous tasks which could themselves be either cyclic, timed processes or priority-driven jobs. Timed processing was interrupt-driven and was itself non-interruptable. For this reason timed processing was subject to a time limit of 14 milliseconds. Typical synchronous timed processes were the digital autopilot (DAP) and the hardware monitoring and service routines. Dedicated interrupts were assigned to these processes.

*Lightning struck the Apollo 12 vehicle during the ascent phase and caused a series of power transients in the computer system. The software recovery system successfully restarted the program allowing the mission to continue. The recovery system was similarly involved in the lunar landing phase of Apollo 11 when erroneous radar inputs caused a computational overload. The recovery system deleted low priority functions, performing only essential computational sequences, and thus relieving the computer overload and allowing successful completion of the landing.

Asynchronously timed processes were dispatched by the operating system from a time queue (called WAITLIST) to which another interrupt was dedicated. These tasks were scheduled in response to real-time program requirements.

Priority-driven jobs were also scheduled as required by current demands on the system, often in direct response to crew inputs. These jobs were preemptable; that is, they could be interrupted by timed tasks (unless interrupts were specifically masked) or by jobs of higher priority.

Interaction between processes in real time, the sharing of resources necessitated by limited memory, and the fact that processes could be initiated asynchronously and unpredictably, created a complexity in program behavior that required extensive testing and often heroic debugging efforts. The Assembly Control Supervisor (para 2.4.1) and a relatively small group of software experts were often called upon to dig into a problem to help explain the seemingly inexplicable.

2.3 FLIGHT SOFTWARE PRODUCTION AND TESTING

The laboratory began its Apollo work in 1961, and the early years were primarily devoted to hardware development, including computer design and prototyping. Early software work was performed by a small group of engineers who were familiar with the computer design work and who had been closely involved with the working groups that developed the mission requirements. All of the programs for the manned Apollo flights were based on this initial existing "core" of system (e.g., executive, display and hardware interface) software. Thus each new program was essentially a modification, although usually a large-scale modification, of a previous release. With the approach of manned missions, software requirements grew and the programming and verification group was expanded. Figure 2-4 shows how this effort was staffed over the years.

2.3.1 PRODUCTION

The group responsible for the flight software included guidance, navigation, and control engineers, programmers, and test engineers, led by a small team who had been associated with the early work at the laboratory and who had themselves developed the first test and flight programs. Experts in the various disciplines worked closely together and in many cases a single person participated in all phases of development of a particular software module, i.e., initial engineering studies, programming and testing.

The coding was done both in the assembly language of the AGC and in the interpretive language (INTERPRETER) developed for the

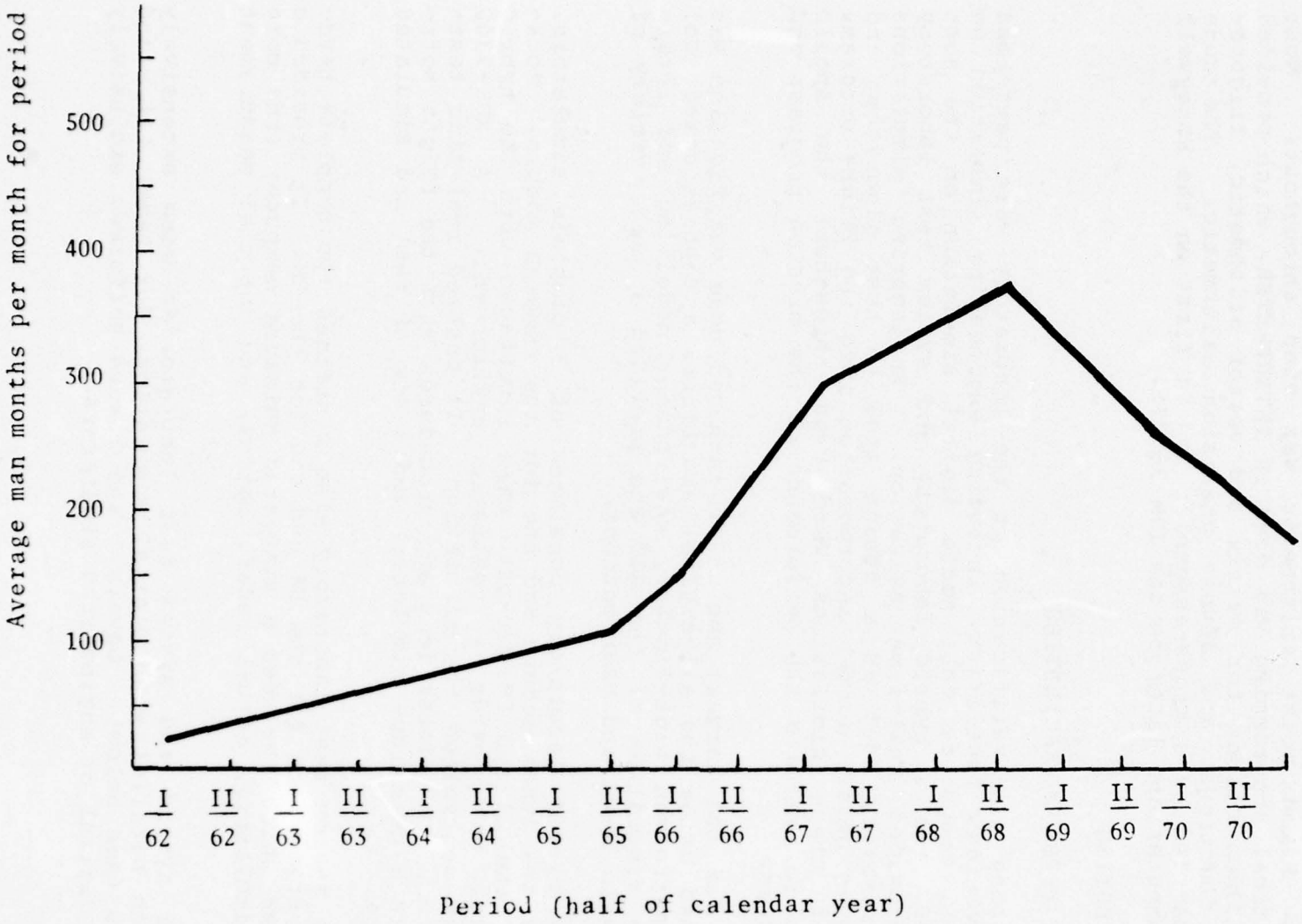


Figure 2-4 MANPOWER USAGE BY MIT/IL SOFTWARE EFFORT

project. Fixed point arithmetic was used throughout. Most mathematical programming was done in INTERPRETER, which provided pseudo instructions for matrix and vector arithmetic, trigonometric functions, and double precision arithmetic. The code was assembled by a cross-assembler hosted first on the Honeywell 1800 computer and later on the IBM 360/75.

2.3.2 TESTING

2.3.2.1 TYPES OF FACILITIES

Testing and verification at the laboratory were performed using various facilities, including engineering simulation on the host computer, full scale digital simulation on the host computer, and a hybrid laboratory and system test laboratory that provided real-time execution. Engineering simulations were primarily used as a design tool to test algorithms and techniques before actual incorporation into the flight program. Some of these simulations were used throughout the Apollo program to evaluate the performance of the mission programs and procedures.

By far the largest amount of in-house code verification was performed using the all-digital simulator, a sophisticated tool which performed high-fidelity environment modelling and interpretive simulation of the AGC and provided a large variety of diagnostic tools and user options.

The hybrid laboratory consisted of 2 complete simulators, one for the lunar module and one for the command module. Mock-ups of the CM and LM cockpits were interfaced with the hybrid computers to provide a realistic environment. A XDS-9300 computer controlled the simulation. It enabled real-time testing of the crew interfaces and procedures with the flight software (in a core rope simulator) and a mix of real and simulated hardware.

The system test laboratory also contained two complete hardware systems, one for the LM and one for the CM. It provided a test bed which included a simulated guidance computer (the core rope simulator), actual radar, optics, and inertial measurement units.

The hybrid and system test laboratories were extensively used, in parallel with digital simulation, for level 3,4,5 and 6 tests (see below). Levels 1 and 2 were performed exclusively on the digital or engineering simulators.

2.3.2.2 LEVELS OF TESTING

Several levels of testing were performed.

Level 1 tests were high order language (HOL) programs run on the host computer to test algorithms. The MAC (MIT Algebraic Compiler), developed at MIT/IL, was used for this effort.

Level 2 was the AGC counterpart of these programs. The results of the two were compared to establish the accuracy of the AGC equivalent. The errors found at this level were primarily computational.

Level 3 was intended to verify the operation of a complete program or routine including crew interface and realistic physical environment models. The errors discovered at this level were primarily logic and display interface problems. This level was performed only when a routine was incorporated into the flight program.

Level 4 testing was intended to verify mission phases, e.g., ascent, rendezvous. The multi-programmed environment was exercised extensively and therefore uncovered priority, timing, and erasable-sharing problems.

Level 5 repeated the level 4 tests on the final rope which was released for manufacture. This was required because even though the level 4 tests had been successfully completed, they may not have run on the version of the program that was released.

Level 6 took place after the ropes were released for manufacture and were intended to verify the program using actual mission data and the flight time-line. These runs were run with 1 sigma and 3 sigma errors in the simulated instruments.

2.4 IN-HOUSE CONFIGURATION CONTROL

The very early programs, produced by a small cadre of dedicated engineers, were not subject to formal configuration control procedures. By the time of the period covered by this study, however, the magnitude of the task and the large number of contributors necessitated control procedures to insure the continuing integrity of the software.

2.4.1 ASSEMBLY CONTROL SUPERVISOR

The assembly control function was established to localize responsibility for the quality of each program as it was being developed. The Assembly Control Supervisor (ACS) was a person

who was knowledgeable about all aspects of the program, and especially expert in the system software, subprogram interfaces, initialization and crew interfaces. The ACS could call on a group of "experts" in the various disciplines to act as consultants to resolve problems. All code was submitted to the ACS for approval before it was incorporated into the official assembly. The code was documented using coding forms* which provided information about the nature of the code, the reason it was being incorporated, the official change notice or change request reference (if any), and the signatures of the programmer and his supervisor, which guaranteed that the code had been checked out independently before it was submitted.

It was only after carefully scanning the code and analyzing it with respect to its interfaces with existing software that the ACS approved each submittal for incorporation into the official assembly. The ACS issued memos documenting in detail each change that had been included into each new assembly revision.

2.4.2 ASSEMBLY CONTROL BOARD

The assembly control activity described above was under the overall management of a system integration group. A board consisting of an integration supervisor, the ACS for each separate program** and a group of experts was established to provide policy guidance and resolve technical issues not able to be decided by the ACS alone.

For testing and checkout before submittal to the official assembly, engineers could snapshot a version of the assembly, incorporate their new code, run simulations, and modify and remodify their programs.

2.4.3 ERASABLE COMMITTEE

This system, with its built-in checks at several levels (engineer, supervisor, ACS, experts), worked well to provide visibility and control during development phases of the project. Because only 2048 words of read-write (erasable) memory were available, it was necessary to time-share data memory among many software modules. Assuring the integrity of erasable data required an overview of the program behavior and a knowledge of

---

*These coding forms served as the source of the data prepared for this study.

** There were always at least two programs under development, the LM and CM programs and at times programs for different flights.

the timing characteristics and interactions of the various program modules. To this end a committee was established to oversee the management of erasable memory; the ACS was an important member of this committee. Requests for data storage from engineers working on the software were reviewed by the committee which assigned data locations on the basis of global needs and time-sharing considerations.

Some consideration was given to automating the process of erasable assignments. Studies that were conducted at the time, however, led to the conclusion that the effort and accuracy required to provide correct input describing data requirements and time-sharing characteristics would be as difficult for the automated process as it was for the manual activity.

Figure 2-5 is a typical erasable memory map showing the data overlays in one bank of erasable memory.

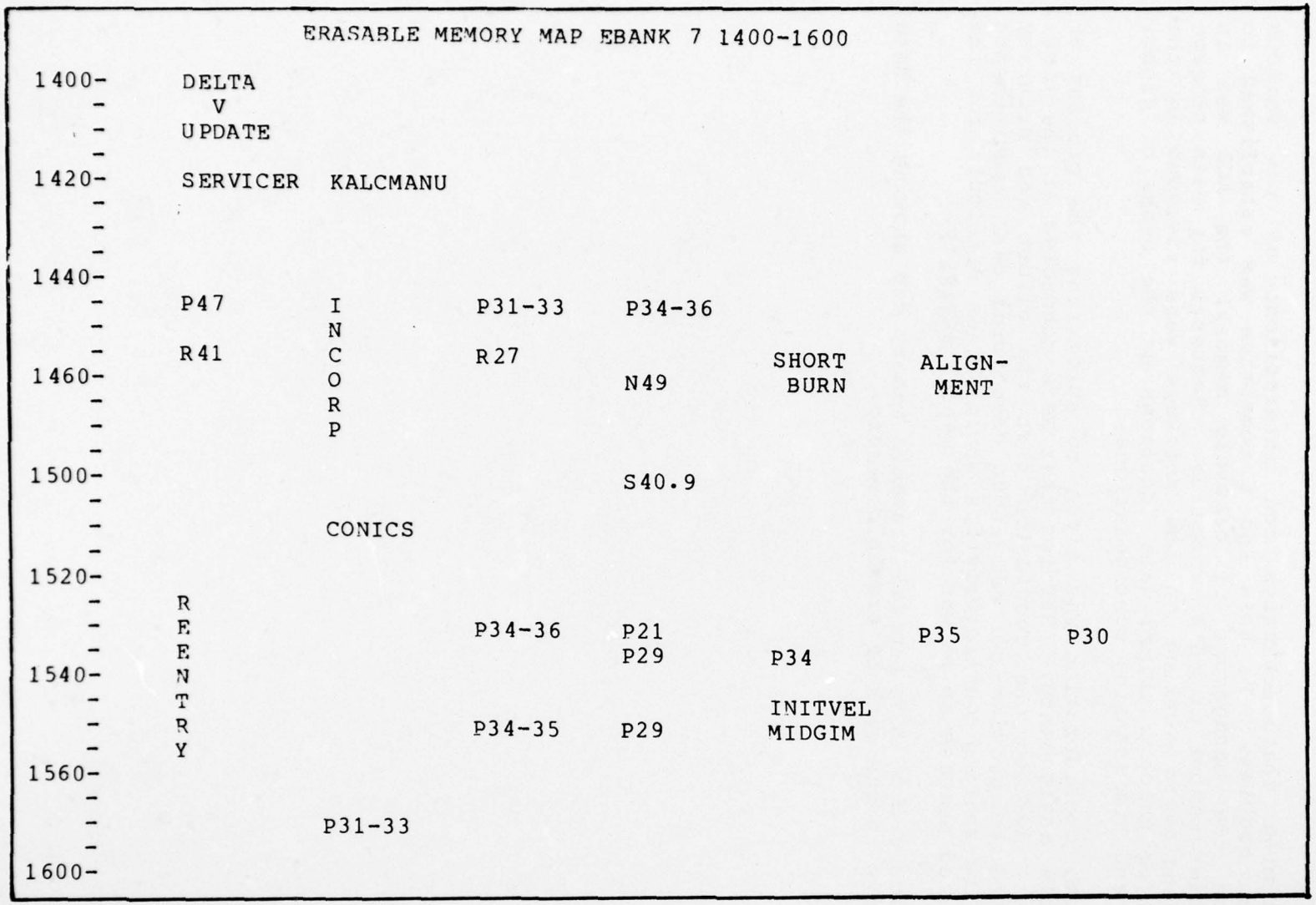


Figure 2-5 ERASABLE MEMORY MAP

ERASABLE MEMORY MAP EBANK 7 (1600-1777)

```
1600-                          P34-35
     -    R
     -    E
     -    E          P30-40     P36
1620-     N
     -    T                     P31-33    P34
     -    R          P30-40
     -    Y          PL REFS-
1640-                    MAT
     -                P40
     -
     -
1660-                P47
     -                          R61       N88
     -
     -                 S40.1
1700-
     -                          MARKBUF
     -
     -                P20       MINKEY                          EMP
1720-
     -    P11                                                   SL-52
     -    ENTRY                 R27
     -
1740-
     -
     -
     -                          S40.1
1760-
     -
     -
     -
2000-
```

Figure 2-5 ERASABLE MEMORY MAP (continued)

SECTION 3

SOFTWARE LIFE CYCLE

3.1 SYSTEM SOFTWARE

Much Apollo flight software was developed before the controls discussed in Section 2 were established. This included both system software and mission software. Although the mission-related software was eventually replaced by software capable of performing the lunar landing mission, the system software was basically unchanged. This software included the executive, display interface, interpreter, much of the hardware interface logic, interrupt handling and computer self-test. In general, the system design was not formally documented and was produced by a relatively small group of engineers who were thoroughly knowledgeable of the performance requirements of the hardware, the sensor instruments and the computer. As the programs continued to develop and as more and more testing was done, some changes were made to these programs for improvement and error corrections, but these corrections were relatively few.

3.2 MISSION SOFTWARE REQUIREMENTS

The requirements for the Apollo mission were determined jointly by NASA and the laboratory. Once the lunar orbit rendezvous technique was established as the method of accomplishing the ultimate goal of Apollo, some obvious and desirable requirements became apparent, e.g., navigation, guidance, etc. The software was designed to perform the mission functions on-board even though many functions were duplicated on the ground.

3.3 MISSION PROGRAM DEVELOPMENT

For the period of time covered by this study, software development consisted of creating a program for a particular mission, testing that program, and releasing it for manufacture. This cycle was repeated for each release. The major utilization of host computer resources during a program's life cycle was in testing. Testing continued after every release to evaluate any deviation from the planned mission.

In the development of software capabilities, use was made of engineering simulations of varying complexity to validate the algorithms and techniques before actual implementation into the flight software.

During the development phase a data base management system, List Processing Service, LIPSVC, was used. The LIPSVC system provided reliablity and visibility in that earlier program re-

visions could be recreated at any time; it provided control of the frequent assemblies and permitted easy creation of "off-line" versions while recording all modifications to these versions. This allowed new programs to be debugged without interfering with the main line production. The interim modifications to the off-line versions were not centrally maintained and therefore do not appear in the data collection.

There often was parallel development of software for the same vehicle. For example, Colossus 3, the software program which flew on Apollo flights 15, 16 and 17, was begun by snapping a version of the program named Comanche under the new name, Artemis. This new version was begun in parallel with the Apollo 12 program. Since extensive changes were planned for Apollo 15,16 and 17, a major effort was undertaken to recode vast portions of the existing program solely for the purpose of gaining space for the new features. Rather than waiting for the release of the Comanche program for these changes, the new version was developed.

3.4 SPECIFICATIONS AND DOCUMENTATION

The controlling document in the life cycle of the Apollo software was the Guidance System Operation Plan (GSOP), a document which served as the specification for the software efforts. Development and control of the GSOP were important activities in planning, controlling, and documenting a flight program. For early flights the GSOP was a single volume document. With the advent of manned flights, it was expanded to six separate volumes. Each volume of the GSOP was dedicated to controlling a different aspect of the AGC software. Section 1 controlled the AGC prelaunch activities. Section 2 dealt with data links, uplink, downlink, and telemetry. Section 3 dealt exclusively with the digital autopilot. Section 4 governed operational modes including PGNCS interfaces with the flight crew and mission control. Section 5 contained the guidance and navigation equations. Section 6 specified the data used in the digital and hybrid simulators in support of the verification of the AGC programs.

In addition to the function of the GSOP as a NASA control document, it served as an internal working document, as a testing guide, and as a crew training aid.

Changes to the software or the GSOP were controlled by the documents described below.

- Program Change Request (PCR)

  A PCR was a request for a change in the specification (GSOP) for a flight program. It was given a preliminary review by a laboratory engineer and by the NASA Flight Software Branch and then held for Software Control Board action. The SCB was composed of representatives of various branches of NASA. SCB could disapprove the change, require more detailed evaluation by MIT/IL or require implementation of the change. This decision involved overall mission considerations and scheduling, as well as the particular software considerations.

- Program Change Notice (PCN)

  A PCN originated at MIT/IL and was a notification that a change was being made rather than a request for a change. The PCN was used for clerical corrections to the flight program specifications or for changes that were required to the program for development to continue. PCNs required approval by the SCB, but approval was usually automatic. If the SCB disapproved, a change to undo the PCN was generated.

- Anomaly Report

  Anomaly Reports reported a discrepancy between a program's specification and its operation. Anomaly Reports required official disposition ,but did not always result in program modification. Work-around procedures or EMPs were sometimes developed to correct the anomaly. Table 3-1 shows the number of known anomalies that existed in each flight.

- Assembly Control Board Request

  An Assembly Control Board Request was prepared by MIL/IL's Assembly Control Board and initiated a program change. This change did not change the program's specification. Instead it was in the nature of an in-house improvement.

Table 3-1 ANOMALIES IN FLIGHT PROGRAMS

FLIGHT	COMMAND MODULE	LUNAR MODULE
7	32	-
8	60	-
9	28	10
10	10	49
11	9	28
12	9	13
13	8	9
14	13	12
15	12	11

3.5 SOFTWARE RELEASES

When the laboratory was directed to release the mission program, a magnetic tape was produced for delivery to the manufacturer. The manufacturing process, which required at least 45 days, started by processing the magnetic tape to produce 2 mylar tapes. One tape, the core rope weaver, actuated the weaving machines that produced the module memory. The other tape was used to verify the memory fabrication process.

Figure 3-1 shows the development of the flight ropes for the Command and Lunar Modules.

3.6 MISSION SUPPORT

All missions were supported around the clock by laboratory personnel both in Cambridge and at NASA facilities. Although no software problems occured in flight, work-around procedures invoked through the software were occasionally required to solve GNCS problems*. These procedures had to be developed very quickly to allow as much testing as possible before they were transmitted to the crew.

*Apollo 14 required a procedure to bypass checking the failed LM abort switch by the software.

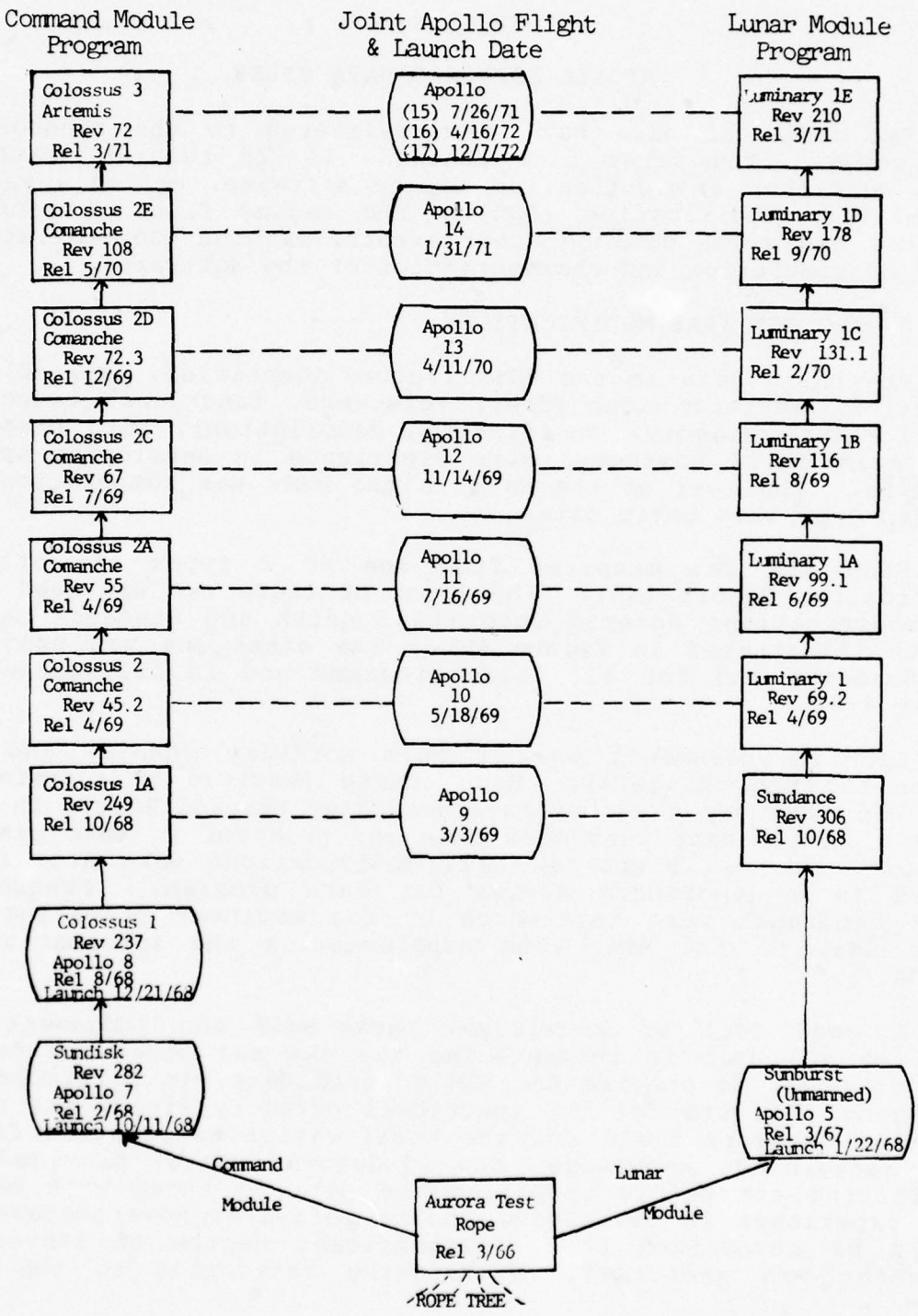


Figure 3-1 SOFTWARE RELEASES

SECTION 4

APOLLO SOFTWARE DATA FILES

Two files of data have been delivered to the sponsor of this study. The first file contains 11,729 records, each of which describes a modification to the software, called a Record of Software Modification (RSM). The second file contains 83 records, and gives data on other errors, machine use statistics, speed of simulation and charateristics of the software.

4.1 APOLLO SOFTWARE MODIFICATIONS

The basic data in the RSMs (record identifier, date of modification, revision identifier, reference, functional category, modification category, modification description), were prepared by a team of 14 engineers with experience in developing Apollo software. The rest of the data in the RSMs was computationally derived from this basic data.

Each RSM was prepared from one of 2 types of software Modification Report (MR). The first of these two was used only for modifications entered into the Sundisk and Sundance series and is illustrated in Figure 4-1. The other one was used for the Sundance and for all later programs and is illustrated in Figure 4-2.

Each MR documents one or more software changes and the reason for the change(s). Each change resulted in a different RSM. In this way a MR may have generated several RSMs. In some cases a change went into more than one program; in that case an RSM for each was prepared. All modifications were also documented in a memorandum series for each program. Frequently these memoranda were referenced by the engineer preparing the basic data of the RSM, thus supplementing the information in the MR.

A good deal of knowledge, background and judgement was required and used in interpreting the MRs and other documentation in order to prepare the RSM's basic data, in particular in supplying the data for the functional category field, the modification category field and the modification description field. The background knowledge and judgement were provided by restricting the people preparing the data to those with extensive experience in developing Apollo software; nevertheless, it should be recognized that a significant degree of subjective judgement was exercised in assigning categories to the data items.

Each RSM contains 9 fields, which are described below. In all numeric fields leading zeroes are replaced by blanks.

P982-3

AS207-SUNDISK AS208-SUNDANCE SUBROUTINE_____

MODIFICATION REPORT

SUNDISK REVISION #: 99

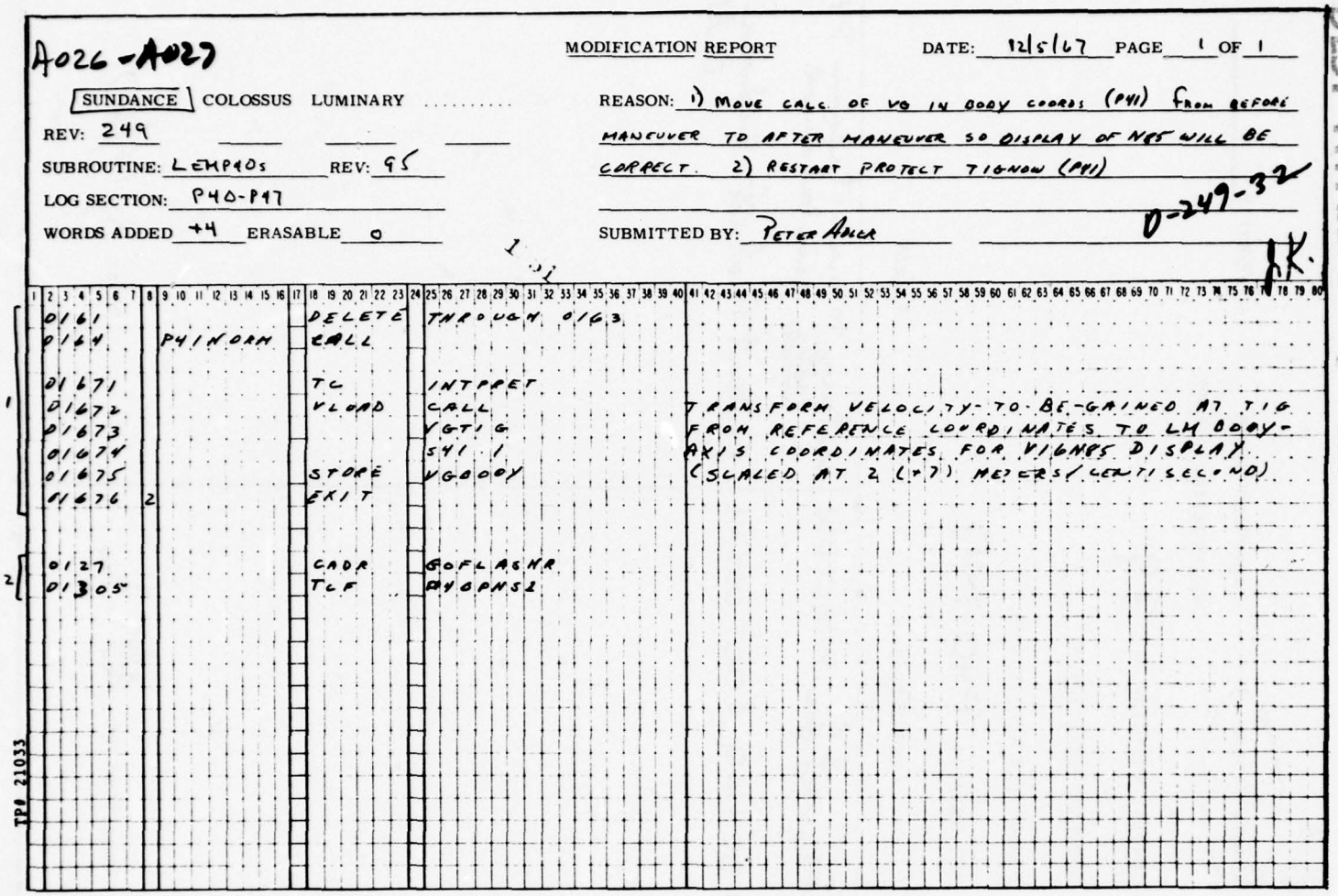
SUNDANCE REVISION #:

SUBROUTINE REVISION #:

LOG SECTION: P40-P47

DESCRIPTION	REASON
Job Request for P40CTDN	return to mainstream P40 after gimbal drive
2PH SCHNG 8 OCT 00174	Restart Pretest Findvec call
Submitted By: AGEJ	Date: 3/14/67

Figure 4-1 MODIFICATION REPORT FOR SUNDANCE AND SUNDISK



A026-A027

MODIFICATION REPORT DATE: 12/5/67 PAGE 1 OF 1

SUNDANCE COLOSSUS LUMINARY .....

REV: 249

SUBROUTINE: LEMP40s REV: 95

LOG SECTION: P40-P47

WORDS ADDED +4 ERASABLE 0

REASON: 1) MOVE CALC OF VG IN BODY COORDS (P41) FROM BEFORE MANEUVER TO AFTER MANEUVER SO DISPLAY OF NBS WILL BE CORRECT. 2) RESTART PROTECT TIGNOW (P41)

0-249-32

SUBMITTED BY: PETER ADLER

B.K.

	01611		DELETE	THROUGH 01613	
	01614	P41NORM	CALL		
	01671		TC	INTPRET	
1	01672		VLOAD	CALL	TRANSFORM VELOCITY-TO-BE-GAINED AT TIG
	01673			VGTIG	FROM REFERENCE COORDINATES TO LM BODY-
	01674			S41.1	AXIS COORDINATES FOR VIGNBS DISPLAY
	01675		STORE	VGBODY	(SCALED AT 2(+7) METERS/CENTISECOND)
	01676	2	EXIT		
2	01127		CADR	GOFLASHR	
	01305		TCF	P41OPNS2	

TP# 21033

BEST AVAILABLE COPY

Figure 4-2 MODIFICATION REPORT FOR LATER PROGRAMS

4.1.1 RECORD IDENTIFIER (Columns 1 - 4)

Each record identifier is unique in the file of RSMs. A record identifier consists of a letter followed by a three decimal digit number. These unique record identifiers were generated at the time of the preparation of the record. They provided traceability during the data preparation process, in that the letter identifies the engineer who prepared the record. At the time of the preparation of the RSM, its record identifier was written on the MR from which the RSM was prepared, thus providing references from the file of MRs to the file of RSMs.

4.1.2 DATE OF MODIFICATION (Columns 6 - 13)

An entry in this field is in the form YY-MM-DD, for year-month-day. MM and DD may contain a leading blank, but no leading zero. All entries in this field were checked to make sure that they were valid dates.

This date is the date the modification was submitted by the programmer to the Assembly Control Supervisor for approval and inclusion in the program. The date that the modification was actually incorporated into the program is not known, but was usually within a day or two, or at most a week, of its submittal.

In most cases this date is the date shown on the MR. In those few cases where the date field on the MR was left blank, the revision identifiers (see paragraph 4.1.3, below) were used as the basis for estimating this date. Estimating this date sometimes required a linear interpolation, interpolating between revision numbers with approximately known dates.

4.1.3 REVISION IDENTIFIER (Columns 15 - 18)

This field was taken directly from the MR and denotes the program revision into which the modification was incorporated. The programs to which the data applies consist of six distinct program series: Sundisk, Colossus, Comanche, Artemis (all Command Module programs), Sundance and Luminary (Lunar Module programs). As shown in Table 4-1, sixteen separate flight programs were manufactured from these series.

The revision identifier consists of a letter followed by a three decimal digit number. The letter identifies the program series; the number is the revision number within that series.

For some revisions there are no RSMs. This could happen for one of two different reasons. (1) For Sundisk revisions 1 through 88, and for Sundance revisions 1 through 82 there were no reports on modifications available. (2) In some cases a

Table 4-1 REVISION IDENTIFIERS, FLIGHT PROGRAMS, SIZES, FLIGHTS

Prog. Code	Revi-sions	Program Name	Flight Program	Total Words	New Words	Space Vehicle	Apollo Flight
D	1-282	Sundisk	Sundisk	36480	23100	C	7
C	1-237	Colossus	Colossus 1	37757	11770	C	8
C	238-249	Colossus	Colossus 1A	37854	110	C	9
M	1- 45	Comanche	Colossus 2	38575	2035	C	10
M	46- 55	Comanche	Colossus 2A	38610	110	C	11
M	56- 67	Comanche	Colossus 2C	38702	215 !	C	12
M	68- 72	Comanche	Colossus 2D	38702	34 !	C	13
M	73-108	Comanche	Colossus 2E	38402	1692 !	C	14
A	1- 72	Artemis	Colossus 3	38485 *	770 #	C	15
S	1-306	Sundance	Sundance	36424	28600	L	9
L	1- 69	Luminary	Luminary 1	37904	10560	L	10
L	70- 99	Luminary	Luminary 1A	38646	2310	L	11
L	100-116	Luminary	Luminary 1B	38502	700 !	L	12
L	117-131	Luminary	Luminary 1C	38502	150 !	L	13
L	132-178	Luminary	Luminary 1D	38202	940 !	L	14
L	179-210	Luminary	Luminary 1E	38452 +	770 #	L	15

Legend for column headed "Space Vehicle":

"C" means Command Module
"L" means Lunar Module

Legend for the columns headed "Total Words" and "New Words":

* Number taken from the program listing for Artemis 72
\+ Number estimated by experienced assembly contol supervisor
! Number taken from Rankin's thesis
\# Number estimated by averaging the numbers for the previous 4 flights

revision was created without a MR being prepared. This happened only in the case of an abort during the assembly process, where due to a clerical error or a machine failure, a spurious revision was created. In those cases, the assembly supervisor simply resubmitted the original modifications.

The following revision numbers do not appear on the RSMs:

Comanche 45.1, 45.2
Comanche 72.1, 72.2, 72.3
Luminary 69.1, 69.2
Luminary 99.1
Luminary 131.1

The revision number on the RSM contains only the digits to the left of the decimal point; thus a modification to revision Luminary 69.1, for example, is recorded in the RSM as occurring in L 69. Fewer than 15 RSMs contain these truncated revision numbers. These revisions are the result of re-releases that were made after further development had continued on the program. To illustrate: Luminary 69 was released for manufacture; meanwhile Luminary 70, 71, 72, etc. were being created (as updates of Luminary 69). Testing of the released program conducted during this period revealed a problem serious enough to warrant re-release. Apollo support software allowed the retrieval of any previous revision; revision 69 was therefore retrieved, copied, and corrected without disturbing the later revisions. The offshoot revision was entitled Luminary 69.1, which later spawned Luminary 69.2 when still another serious problem surfaced.

4.1.4 REFERENCE (Columns 20 - 25)

For about 13% of the RSMs, another document is referenced. The document is identified in the reference field in the manner shown in Table 4-2. The referenced document is either a Program Change Request, a Program Change Notice, an Anomaly Report or an Assembly Control Board Request (see paragraph 3.4). The document established the basis for the change. If it was written before the MR, then it served to initiate the change. If after the MR, then it served to rationalize the change.

When such a document was cited in an MR, it was recorded in the reference field of the corresponding RSM.

For the remaining RSMs the reference field is blank.

Table 4-2 REFERENCE NUMBERS

Columns 20-25	Meaning	Count
PCRddd	Program Change Request ddd	949
PReddd	Program Change Request eddd	"
PCNddd	Program Change Notice ddd	78
PNeddd	Program Change Notice eddd	"
COLddd	Colossus Anomaly ddd	40
ACBxxx	Assembly Control Board Request xxx	357
COMddd	Comanche Anomaly ddd	43
LNYddd	Luminary Anomaly ddd	40

symbol used above	explanation
d	a decimal digit
e	a decimal digit other than zero
x	a decimal digit or a capital letter

.1.5 FUNCTIONAL CATEGORY (Column 27)

This field identifies the function within the flight prog-am which is being modified. 22 functional categories were dentified and assigned code letters "A" through "V". The func-ional categories and their codes are shown in Table 4-3. The ield, then, contains a letter denoting the functional category f the code being modified.

This field was coded on the basis of the information in he MR, supplemented by other documentation and the background nowledge of the individual preparing the data.

.1.6 MODIFICATION CATEGORY (Columns 29 - 32)

The entry in this field denotes that category out of the 08 preselected categories (shown in Table 4-4) which best escribes the modification or the reason for the modification. he entry itself is a code denoting the modification category nd consists of a letter followed by a three decimal digit umber.

184 categories were defined by the sponsor in the Statement f Work, each with its unique code. First these codes, which ere 5 characters long, with the first and second character lways identical, were changed by deleting the second redundant haracter of the code to shorten it, and by replacing leading eroes in the numerical portion of the code by blanks.

Secondly, it became apparent in the course of this study hat a large proportion of Apollo programming errors do not fit he 184 categories defined by the sponsor. This mismatch is ue primarily to the fact that the Apollo code was written for real-time multiprogrammed application, whereas the error data nalyzed previously under the sponsor's auspices was derived rom non real-time applications. Real-time errors manifest hemselves as routine-to-routine interface conflicts as well as riority or time conflicts. The sequence of operation of outines in real time is frequently dependent upon the proper etting of flags and other interface data by other routines. rogram sequencing and restart protection is considerably com-licated by the actions of an on-line astronaut. Finally, the xistence of an interactive user creates a series of possible an/machine interface problems.

In order to avoid forcing the Apollo real-time errors into nnatural categories, 24 new categories were defined, each with new code. 5 other categories had their descriptions modified ut not their code; one category had its description modified nd was given a new code; and 1 category was given a new code ithout its description being modified. We thus arrive at the

Table 4-3 FUNCTIONAL CATEGORIES, THEIR CODES AND SIZES

Code	Functional Category	Colossus 237	Luminary 98
A	Digital Autopilot	5815	3464
B	Error Detection/Recovery	332	280
C	Crew Interface	1771	2220
D	Telemetry	404	242
E	I/O	NA	NA
F	Executive	773	1173
G	Sequence Initialization/Reinitialization	1009	953
H	Display	3603	3107
I	Navigation	4043	4655
J	Coordinate Transforms	616	474
K	Vehicle Attitute Computations/Maneuvers	939	998
L	Tracking	2929	3560
M	Targeting	3728	2684
N	Powered Flight Maneuvers	2479	3103
O	Guidance Computations	NA	1570
P	Alignments	1588	2101
Q	Interpreter	2145	2150
R	Math Subroutines	71	126
S	Software Utility Routines	196	162
T	Hardware Failure Monitor	1291	1780
U	Hardware Service Routines	1982	967
V	Manual Operations (non-software)	NA	NA

"NA" means that the size of the code for the functional category is not available.

Table 4-4 MODIFICATION CATEGORIES

A 0	COMPUTATIONAL ERRORS
A 10	Total number of entries computed incorrectly
A 20	Physical or logical entry number computed incorrectly
A 30	Index computation error
A 40	Wrong equation or convention used
A 41	Mathematical modeling problem
A 42	Sampled data problem
A 50	Results of arithmetic calculation inaccurate/not as expected
A 60	Mixed mode arithmetic error
A 70	Time calculation error
A 71	Time conversion error
A 72	Time truncation/rounding error
A 80	Sign convention error
A 90	Units conversion error
A100	Vector calculation error
A110	Calculation fails to converge
A120	Quantization/truncation error (precision)
A130	Fixed point scaling error
B 0	LOGIC ERRORS (within an individual module)
B 10	Limit determination error
B 20	Wrong logic branch taken
B 30	Loop exited on wrong cycle
B 40	*Incomplete processing*
B 50	Endless loop during routine operation
B 60	Missing logic or condition test
B 61	Index not checked
B 62	Flag or specific data value not tested
B 63	Open branch
B 70	Incorrect logic design
B 71	Incorrect implementation
B 80	Sequence of activities wrong
B 90	*Filtering error*
B100	Status check/propagation error
B110	Iteration step size incorrectly determined
B120	Logical code produced wrong results
B130	Logic on wrong routine
B140	Physical characteristics of problem to be solved overlooked or misunderstood
B150	Logic needlessly complex
B160	Inefficient logic
B170	Excessive logic
B180	Storage reference error (software problem)
C 0	I/O ERRORS (DISPLAY, TELEMETRY, ETC.)
C 10	Missing output
C 20	Output missing data entries
C 30	Error message not output
C 40	Error message garbled
C 50	Output or error message not compatible with design documentation (including garbled output)
C 60	*Misleading or inaccurate error message text*
C 70	Output format error (including wrong location)
C 80	Duplicate or excessive output
C 90	Output field size inadequate
C100	Debug output problem (relative to design documentation)
C101	Lack of debug output
C102	Too much debug output
C110	Header output problem
C120	Output tape format error
C130	Output card format error
C140	Error in printer control
C150	Line count/page eject error
C160	Needed output not provided in design
C161	Insufficient output options
D 0	DATA HANDLING ERRORS
D 10	Valid input data improperly set/used
D 20	Data written in or read from wrong memory location
D 30	Data lost/not stored
D 40	Data index or flag not set or set/initialized incorrectly
D041	Number of entries set incorrectly
D 50	Data, index, or flag modified or updated incorrectly
D 51	Number of entries updated incorrectly
D 60	*Extraneous entries generated (table, array, etc)*
D 70	Bit manipulation error
D 71	Error using bit modifier
D 80	Floating point/integer conversion error
D 90	Internal variable error (definition or set/use)
D100	Data packing/unpacking error
D110	Routine looking for data in non-existent record
D120	Bounds violation
D130	Data chaining error
D140	*Data overflow or overflow processing error*
D150	Read error
D151	All available data not read
D160	Long literal processing error
D170	Sort error
D180	Overlay error
D190	Subscripting convention error
D200	Double buffering error
E 0	OPERATING SYSTEM/SYSTEM SUPPORT SOFTWARE ERRORS
E 10	Language produces erroneous machine code
E 20	OS missing needed capability
F 0	CONFIGURATION ERRORS
F 10	Assembly errors
F 11	Program segmentation (layout of subroutines, etc.)
F 12	Real time organization problem (imcompatible modes)
F 20	Illegal instruction
F 30	Unexplained program halt
F 40	Organization problem (location of code, etc.)
G 0	ROUTINE/ROUTINE INTERFACE ERRORS
G 10	Routine passing incorrect amount of data (insufficient or too much)
G 20	*Routine passing wrong parameters or units*
G 30	Routine expecting wrong parameters
G 40	Routine fails to use available data
G 50	Routine sensitive to input data order

Table 4-4 MODIFICATION CATEGORIES (continued)

G 60	Calling sequence or routine/routine initialization error
G 61	Real time routine/routine initialization error
G 62	Priority conflict
G 53	Time conflict
G 70	Routines communicating through wrond data block
G 80	Routine used outside design limitation
G 90	Routine won't load (routine imcompatibility)
G100	Routine overflows core when loaded
H 0	ROUTINE/SYSTEM SOFTWARE INTERFACE ERRORS (AGC Executive)
H 10	OS interface error (calling sequence or initialization)
H 20	Routine uses existing system support software incorrectly
H 30	Routine uses sense/jump switch improperly
H 40	Inadequate restart (was JJ100)
H 50	Errors in restart logic
I 0	TAPE PROCESSING INTERFACE ERRORS
I 10	Tape unit equipment check not made
I 20	Routine fails to read continuation tape
I 30	Routine fails to unload tape ofter completion
I 40	Erroneous input tape format
J 0	USER INTERFACE ERRORS
J 10	Incompatibility with external requests
J 20	Multiple physical card/logical card processing error
J 30	Input data interpreted incorrectly by routine
J 40	Valid input data rejected or not used by routine
J 50	Input data rejected but used
J 60	Input data read but not used
J 70	Illegal input data accepted and processed
J 80	Legal input data processed incorrectly
J 90	Poor design in man/machine interface
K 0	DATA BASE INTERFACE ERRORS
K 10	Routine/data base incompatibility
K 20	Uncoordinated use of data elements by more than one user
L 0	USER REQUESTED CHANGES
L 10	Simplified interface and/or convenience
L 20	New and/or enhanced functions
L 21	CPU
L 22	Disk
L 23	Tape
L 24	I/O
L 25	Core
L 26	Man/machine real time spec change
L 27	Error recovery spec change
L 30	Security
L 40	New hardware/OS capability
L 50	Instrumentation
L 60	Capacity
L 70	Data base management and integrity
L 80	External program interface
L 90	Implementation of original spec.
M 0	PRESET DATA BASE ERRORS
M 10	Data or operations request card descriptions
M 20	Error message text
M 30	Nominal, default, legal, max/min values
M 40	Physical constants and modelling parameters
M 41	Ephemeris constants
M 50	Dictionary (bit string) parameters
M 60	Missing data base settings
N 0	GLOBAL VARIABLE/COMPOOL DEFINITION ERRORS
N 10	Items in wrong location (wrong data block)
N 11	Definition sequence error
N 20	Data definition error
N 21	Table definition incorrect
N 30	Length of definition incorrect
N 40	Comments error
N 50	Delete unneeded diefintions
N 60	Make room in E-Bank (erasable memory)
N 70	Reorganize data
N 80	Missing data definition
P 0	RECURRENT ERRORS (ERRORS MADE IN FIXING PREVIOUS ERRORS)
P 10	Problem report reopened
P 20	Problem report a duplicate of previous report
Q 0	DOCUMENTATION ERRORS
Q 10	Routine limitation
Q 20	Operation procedures
Q 30	Difference between flow chart and code
Q 40	Tape format
Q 50	Data card/operation request card format
Q 60	Error message
Q 70	Routine's functional description
Q 80	Output format
Q 90	Documentation not clear/not complete
Q100	Test case documentation
Q110	Operation system documentation
Q120	Type/editorial error/cosmetic change
R 0	COMPLIANCE ERRORS
R 10	Excessive run time
R 20	Required capability overlooked or not delivered at time of report
S 0	REQUIREMENTS ERRORS (SPECIFICATIONS)
S 10	Requirements error (insufficient, inadequate)
S 20	Requirements enhancement
T 0	OPERATIOR ERROR
T 10	Test execution error
T 20	Routine compiled against wrong Compool/Master Common
T 30	Wrong data base used
T 40	Wrong master configuration used
T 50	Wrong tape(s) used
U 0	QUESTIONS
U 10	Data Base
U 20	Master configuration
U 30	Routine
VV 0	IN-HOUSE PROGRAM IMPROVEMENTS
V 10	Simplified interface and/or convenience
V 20	New and/or enhanced functions
V 30	Data base management and integrity
V 40	CPU time optimization
V 50	Program memory optimization

final count of 208 categories. Care was taken to avoid proliferation of categories. In most cases, the new real-time categories are simply subsets of existing, closely related non real-time categories. Descriptions of categories were modified to clarify their meaning within the Apollo context. Categories were recoded to provide a clearer hierarchy of categories. All these changes are shown in Table 4-5.

The value of the modification category field was entered by the RSM's preparer on the basis of 1) all the information in the MR, 2) other documentation related to the modification, 3) using his or her knowledge of the context in which the MR had been written and 4) using his or her knowledge of the categorization scheme that was being used.

The MRs document each change to an assembly and the reason for the change. Ideally it would be possible for experienced Apollo programmers to go through the MRs categorizing each modification according to the categorization scheme being employed. However, when the hand-written MRs were originally produced, they were not produced with anything like the present purposes (e.g, carrying out error analysis) in mind. As it is, the MRs frequently do not contain sufficient information to categorize a modification and thus other documentation is required. The appropriate Apollo memo series was used for this purpose, since that series contained memos provided by the Assembly Control Supervisor which gave for each revision a detailed description of all modifications incorporated in the revision. Also, if the MR references a PCR, PCN, Anomaly Report or an Assembly Control Board Request (see paragraph 4.1.4), supporting data identifying the source of the change was obtained, if needed, from the referenced document. Even with all this documentation, often not enough information was available about the nature of the original change and about the context in which the change was made: in many cases, judgement based on experience was relied on.

4.1.7 MODIFICATION DESCRIPTION (Columns 34 - 83)

This field contains a brief textual description of the program change or the reason for the change. Generally this is just a fuller description of the modification or the reason for it then what is denoted in the modification category field (paragraph 4.1.6). (Note that "reason for the modification" and "description of the modification" are largely synonymous.)

The description in this field, like the code in the modification category field, was entered by the RSM's preparer on the basis of all the information in the MR, other available documentation, as well as his or her knowledge of the context in which the MR had been written.

Table 4-5 CHANGES TO THE ERROR MODIFICATION CATEGORIES

Code	Status	Category Description. (Explanation of the Change)
A 42	A	Sampled data problem; characteristic of real-time data was not as expected.
A130	A	Fixed point scaling error.
B 63	A	Open branch; possibility of logic branch to undefined location.
B 70	C	Incorrect logic design. ("design" added to differentiate category from the one below coded "B 71")
B 71	A	Incorrect implementation; logic designed correctly but incorrectly implemented.
D 20	C	Data written in or read from wrong memory location. ("disk" changed to "memory")
F 11	C	Program segmentation (layout of subroutines, etc.). (Original category was specialized to apply only to changes to the organization of subroutines).
F 12	A	Real time organization problem (incompatible modes); incompatible modes of operation of routines in real time (includes restart problems).
F 40	A	Organization errors (location of code, etc.); errors in location of code, deletion of unused code, etc.
G 61	A	Real time routine/routine initialization error; initialization error resulting in real-time sequencing problem.
G 62	A	Priority conflict; error in establishment of priority of real-time routine.
G 63	A	Time conflict; real-time routine does not meet time constraints.
H 40	A	Inadequate restart capability (was J100); addition or modification of restart tables, etc.

Table 4-5 CHANGES TO THE ERROR MODIFICATION CATEGORIES (cont.)

Code	Status	Category Description. (Explanation for the Change)
H 50	A	Errors in restart logic; modification in restart procedures.
J 10	C	Incompatibility with external requests; changes made to conform to external requests.
J 90	C	Poor design in man/machine interface. ("operator" changed to "man/machine")
J100	D	Inadequate interrupt and restart capability. (Code for the category changed to "H 40")
K 11	D	Uncoordinated use of data elements by more than one user. (Code for the category changed to "K 20")
K 20	A	Uncoordinated use of data elements by more than one user. (Code for this category was "K 11" originally)
L 26	A	Man/machine real-time spec change; user requested change in man/machine interface.
L 27	A	Error recovery spec change; user requested change in restart mechanism.
L 90	A	Implementation of original spec.; new code required to conform to original specification.
N 60	A	Make room in E-Bank (erasable memory).
N 70	A	Reorganize data.
N 80	A	Missing data definition; primarily includes those modifications to the Erasable Assignments that parallel coding changes that fall into such categories as "incorrect implementation of logic design", "missing logic", and others of a similar nature.
S 10	A	Requirements error (insufficient, inadequate); coding changes due to an insufficient or inadequate specification.
S 20	A	Requirements enhancement; coding changes due to changes in requirements.

Table 4-5 CHANGES TO THE ERROR MODIFICATION CATEGORIES (cont.)

Code	Status	Category Description; Explanation for the Change
V 0	A	IN-HOUSE PROGRAM IMPROVEMENTS.
V 10	A	Simplified interface and/or convenience. (Description same as for the category coded "L 10")
V 20	A	New and/or enhanced functions. (Description same as for the category coded "L 20")
V 30	A	Data base management and integrity. (Description same as for the category coded "L 70")
V 40	A	CPU time optimization.
V 50	A	Program memory optimization.

Legend for column headed "Status":

"A" means that the code was added.
"C" means that the description was changed, but the code was not.
"D" means that the code was deleted.

4.1.8 FLIGHT PROGRAM (Columns 85 - 87)

A modification is intended for a particular flight program then under development. This study covers the development of 16 different flight programs.

The field has one of 16 values, and indicates that the modification is intended for the flight program denoted by the value and under development at the time of the modification. Table 4-1, page 30, shows which value denotes which flight program.

This field also indicates that the modification was intended for a particular space vehicle (paragraph 4.1.8.1) and a particular Apollo flight (paragraph 4.1.8.2). However, there are no modifications that were intended for the Lunar Module of Apollo flight 7 or 8, because there was no Lunar Module on Apollo flights 7 and 8. Thus, instead of 18 (i.e., 2*9) possible combinations of vehicle and flight number, there were only 16.

The entry for this field is computed from the revision identifier (see paragraph 4.1.3), using the information in Table 4-1.

4.1.8.1 SPACE VEHICLE (Column 85)

This field has one of two values. "C" denotes that the modification was made to a Command Module program, "L", to a Lunar Module program.

The entry for the field was derived from the leading character of the revision identifier (see paragraph 4.1.3), using the information in Table 4-1.

4.1.8.2 FLIGHT NUMBER (Columns 86 - 87)

A flight program (rope) is designed to play its part in one or more Apollo flights. The Apollo flights are numbered. The flight programs covered in this study flew on Apollo flights 7 through 17. Note that the two flight programs intended for Apollo flight 15 are also for Apollo flights 16 and 17, since these three flights employed the same programs.

This field identifies for which one of these Apollo flights the modification was first implemented, by containing in the field a two decimal digit number identifying the Apollo flight.

The entry for this field was derived from the revision identifier (see paragraph 4.1.3), using the information in Table 4-1.

4.1.9 SOFTWARE DEVELOPMENT PHASE (Column 89)

This field indicates whether the modification was made during the development or during the verification phase of the flight program's software development cycle. The letter "D" entered in the field indicates that the change was made during the development phase, the letter "V", during the verification phase.

Note that, if the modification is to correct an error, then this field does not generally indicate (neither does any other field) in which phase of the software development cycle the error was found (recognized). Such data is not generally available. However, it is safe to assume that in most cases the error was discovered in the same phase that it was corrected. This is, of course, necessarily the case if the error was corrected during the development phase.

Note further that, if the modification is to correct an error, then this field does not generally indicate in which phase of the software development cycle the error was introduced into the program. However, the modification category field generally indicates whether an error was introduced during the specification phase or later.

The value of this field was derived from two other values contained in the RSM, (1) the date the modification was completed (paragraph 4.1.2), and (2) the flight program (paragraph 4.1.8), as well as from (3) the date the verification phase began for that flight program. The value of the field is "D" if the date the modification was submitted was earlier than the date of the beginning of the verification phase; otherwise the value of the field is "V".

The date the verification phase began is the date that is underlined in Table 4-6 for that flight program. This date is either the date that configuration control by NASA began, if that date is available, or else the date that Level 4 testing started. The NASA configuration control date was preferred, if available, over the Level 4 testing date, because it is considered a more accurate and reliable estimator of the beginning of the verification phase.

Other and more refined breakdowns of that portion of the software development cycle during which program modifications were made were considered, but were rejected, either because the resulting phases would not be sufficiently meaningful or because the available data, whether on the basis of partitioning the series of revisions into phases or on the basis of establishing dates that separate phases, was not sufficiently reliable.

Table 4-6 FLIGHT PROGRAM RELEASE DATES, etc.

Flight Program	Space Vehicle	Apollo Flight	NASA Control	Release Date	- - Level 4 - - Start	Complete
Sundisk	Command Module	7	67-10- 4	68- 2	NC	NC
Colossus 1	Command Module	8	68- 7-26	68- 8	NC	NC
Colossus 1A	Command Module	9	68- 8-23	67-10-28	NC	NC
Colossus 2	Command Module	10	69- 3- 7	69- 4- 2	NC	NC
Colossus 2A	Command Module	11	NA	69- 4-18	69- 3-14	69- 4- 4
Colossus 2C	Command Module	12	NA	69- 7-18	69- 7- 7	69- 7-17
Colossus 2D	Command Module	13	NA	69-12-12	69-10-10	69-10-24
Colossus 2E	Command Module	14	NA	70- 5-29	70- 5- 8	70- 5-26
Colossus 3	Command Module	15	NA	71- 3- 1	70- 7-20	71- 1-28
Sundance	Lunar Module	9	68- 4-12	68-10	NC	NC
Luminary 1	Lunar Module	10	NA	69- 4- 2	68-11-22	68-11-22
Luminary 1A	Lunar Module	11	NA	69- 6-17	69- 2-28	69- 4-14
Luminary 1B	Lunar Module	12	NA	69- 8-12	69- 7-14	69- 8-12
Luminary 1C	Lunar Module	13	NA	70- 2- 5	69-10-20	69-11- 5
Luminary 1D	Lunar Module	14	NA	70- 4-18	70- 4-13	70- 5-24
Luminary 1E	Lunar Module	15	NA	71- 3-20	70-12- 7	71- 1-18

Legend for date fields:

NA Date not available
NC Date not compiled, but probably available

4.2 SUPPLEMENTAL DATA

The data described in this section was derived from CSDL Apollo project documentation. It is contained in the second file delivered to the sponsor. This file contains 83 records of 80 characters each. The order of the items in each record is listed below:

Field 1	Columns 1 - 5	paragraph of the sponsor's Statement of Work
Field 2	Columns 7 - 16	alphanumeric tag
Field 3	Columns 18 - 29	numeric value
or		
Field 3	Columns 18 - 19	NA (for not available)
or		
Field 3	Columns 18 - 76	multiple subfields (computer hours table)
Field 4	Columns 31 - 80	comment (for records other than those containing computer hours table)

The information contained in this file is described below.

4.2.1 ERRORS

No errors were due to failures in the computer hardware only.

The number of software errors that resulted in abnormal processor termination is not available, since such data would have to be based on a detailed history of digital simulation runs.

The number of software errors that resulted in normal processor termination is not available, since such data would have to be based on a detailed history of digital simulation runs.

There were no software errors for which the exact cause of the error was unknown when the corresponding software problem report was closed, since such reports were not closed until the problem was understood.

4.2.2 MACHINE USE STATISTICS

The total amount of CPU time used for the Apollo project per month is shown in Figures 4-3 and 4-4 and Table 4-7. This CPU time is provided only for Draper Laboratory's Honeywell 1800 and its IBM 360/75, and not for the Draper Laboratory's hybrid computer, its System Test Laboratory, or testing facilities used outside the Laboratory. The CPU time given, for the Honeywell 1800 and the IBM 360/75, is in IBM 360/75 equivalent

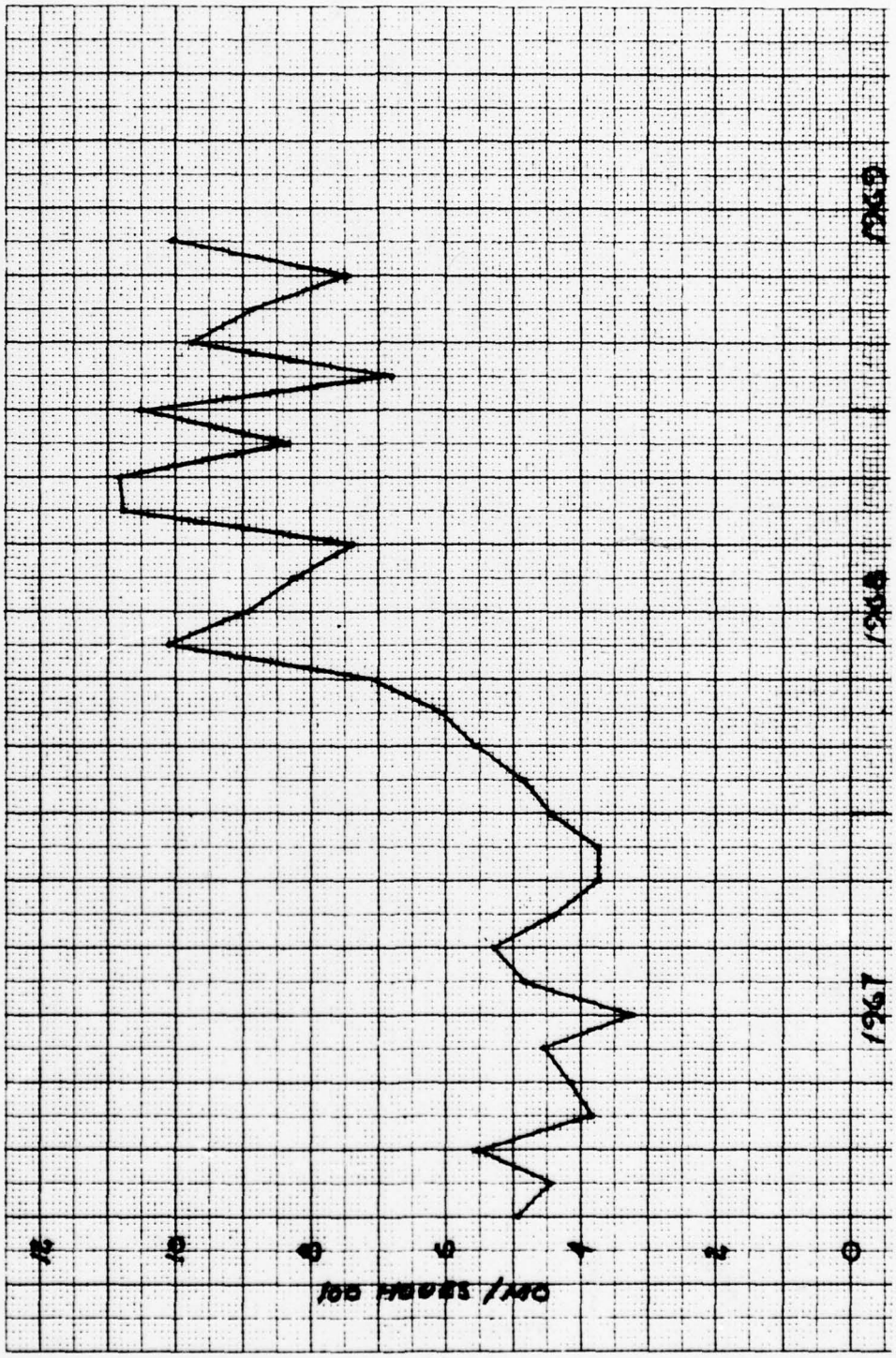


Figure 4-3 COMPUTER USAGE THROUGH APOLLO 5-11

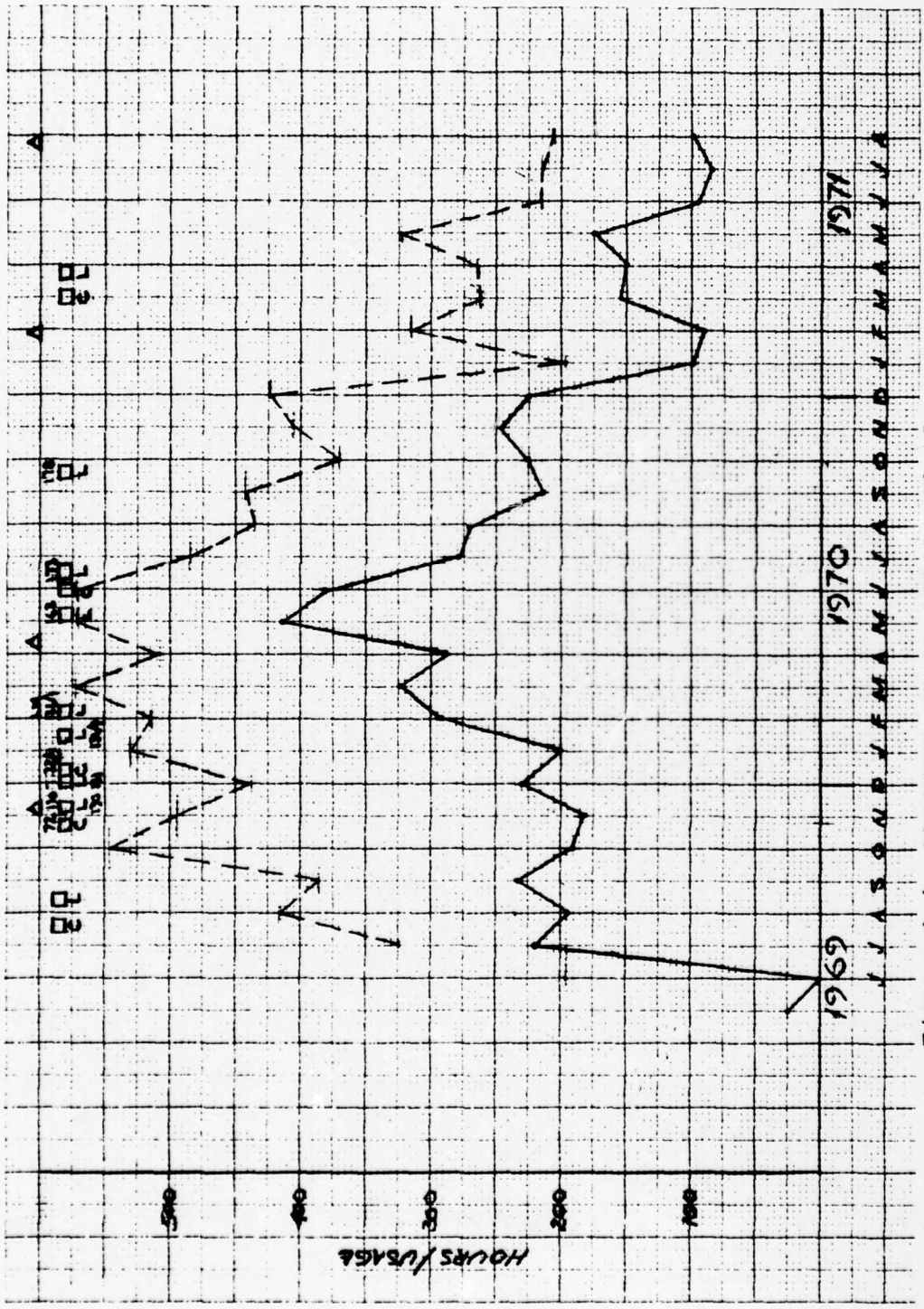


Figure 4-4 COMPUTER USAGE APOLLO 12-15

Table 4-7 COMPUTER USAGE (in hours)

Year	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sep	Oct	Nov	Dec
1967	491	444	553	385	414	453	325	485	529	439	377	378
1968	446	484	555	606	703	1010	932	915	1097	1372	1267	1042
1969	1008	569	678	802	546	623	325	416	386	545	496	438
1970	529	513	573	508	571	571	484	435	441	370	406	423
1971	197	315	262	264	323	216	214	206				

hours. (4 hours of Honeywell 1800 time are considered equivalent to 1 hour of IBM 360/75 time.) The total amount of CPU time used for the Apollo project is provided on a monthly basis, not on a daily basis.

The monthly CPU time is based on Apollo project documentation.

Note that the CPU time used for the Apollo project went mainly into (1) testing (i.e., finding errors), (2) assembling programs to correct errors, (3) assembling programs to modify the programs to meet new requirements and (4) developing the equations. Thus, except for the time that went into developing the equations--a rather small proportion of the total time used, the preponderance of the CPU time used for the Apollo project went into finding and correcting errors or effecting requirements changes.

4.2.3 NUMBER OF SIMULATION RUNS

Data on the number of simulation runs for each period in the software development cycle is not available; the record of this item, therefore, contains the letters "NA".

Even if data regarding the number of simulation runs were available, it is not clear how meaningful this data would be, since the simulation testing done at Draper on Draper's IBM 360/75 and on Draper's Honeywell 1800 represented only a small portion of all the simulation testing that was done. A great deal of simulation testing was also done at Cape Kennedy, NASA/JSC, Grumman, Rockwell, Delco and on Draper's hybrid computer and in Draper's System Test Laboratory. (These remarks also apply to the data discussed under paragraph 4.2.2, above.)

4.2.4 SPEED OF SIMULATION

The simulator to real time processing ratio varied considerably, depending on the rate of activity of the AGC being simulated. At best, when the AGC (the on-board computer) was performing the coasting flight functions (i.e., when only a small percentage of its processing capacity was being used), about 1 unit of IBM 360/75 time was required to simulate 8 units of AGC time. At worst, when the AGC was very heavily loaded (i.e., all of its processing capability was being used), about 4 units of IBM 360/75 time was required to simulate 1 unit of AGC time. With an average load on the AGC, about 3 units of IBM 360/75 time were required to simulate 2 units of AGC time.

These figures remained fairly constant over the period covered by this study. The Honeywell 1800 program that simulated the AGC was, when correcting for the 4 to 1 ratio in

processing power between the IBM 360/75 and the Honeywell 1800, approximately as efficient in simulating the AGC as the IBM 360/75 simulator program.

The user of the simulator program was able to vary the fidelity (high and low) with which the environment was simulated. Users specified high fidelity only when high accuracy was required, thus effecting some savings in run time.

4.2.5 SOFTWARE CHARACTERISTICS

4.2.5.1 SIZE

The total number of computer words in all the 16 releases covered by this study is approximately 610,000.

A second estimate of the total size of the Apollo flight software covered by this study is provided by the sum for all ropes of the number of words added or changed since the last rope, excluding, however, all words changed to correct programming errors. This number is 83,866.

The second of these two numbers is a more meaningful estimate of the total size of the Apollo project, if we consider primarily the extent of the development effort. However, the first is the more meaningful estimate, if we consider the size of the testing effort, since every rope had to be tested anew, and there was relatively little carry over to the testing of one rope from the testing of its parent. In particular, approximately the same amount of Level 4 through Level 6 testing was performed for each rope.

The first of these two numbers can be fairly well approximated by multiplying the size of the computer's memory (38,912 words) by the number of ropes (16, i.e., one rope for each of the Apollo flights 7 and 8 and two ropes for each of the flights 9 through 15). This provides an estimate, albeit high, of 622,592 words. However, a more accurate estimate, 610,000 words, was obtained by using the data of reference 11. The sizes of Colossus 3 and Luminary 1E are not available in reference 11. The total number of words for flight program Colossus 3 was taken from the program listing for Artemis revision 72. The total for Luminary 1E was estimated by an experienced Assembly Control Supervisor. Table 4-1, page 30, lists the sizes for each rope.

The best estimate of the second of these numbers is the sum of the entries of the sixth column (headed "New Words") of Table 4-1. This column lists, for each of the 16 flight programs, an estimate of the number of words added or changed since the last flight program (rope) for the same space vehicle,

excluding, however, all words changed to correct programming errors. 8 of the 16 numbers of this column are taken from reference 12. 6 of the 16 are taken from reference 7. The other 2 were estimated by averaging the numbers in the column for the previous 4 flights.

For an explanation of why size is estimated in terms of numbers of computer words, see the discussion in paragraph 4.2.5.1.1.

4.2.5.1.1 SIZE OF FUNCTIONAL CATEGORIES

Table 4-3, page 34, provides an estimate of the amount of code (in numbers of words) used for each functional category for two representative revisions, Colossus revision 237 (the final revision for the Command Module for Apollo flight 8), and Luminary revision 98 (one of the later revisions for the Lunar Module for Apollo flight 11). Where, for a size, "NA" is entered instead of a number it means that the size of the code for the functional category is not available, since the code is embedded among the code that was counted for other functional categories.

These numbers have not been computed for the other ropes, because the program listings on the basis of which these numbers are computed are not readily available. The NASA archives in Houston may contain a complete or near complete set, but this has not been investigated.

The size of each component module (i.e., the amount of code used for each functional category) is stated in terms of number of machine words (each word consisting of 15 bits of information, with a sixteenth bit being used as a parity check), and includes constants (data) and interpretive code, as well as instructions. Although a listing of a program indicates for each word whether the word is an instruction word (the word is unmarked), a constant (the word is marked with a "C"), or a word of interpretive code (the word is marked with an "I"), it would still be an arduous task to count these separately (for each rope over 30,000 words would have to be categorized), and no clear benefit would be obtained from this. Typically and very roughly, about 53% of the words of a program represent basic machine instructions, 34% represent interpretive code and 13% constants.

Measuring size in terms of machine words conforms to the sponsor's requirement that programs be measured in terms of number of machine instructions for that portion of a program written in assembly language. The AGC instructions can be meaningfully viewed as being of fixed (i.e., constant) length, each the size of a word. Thus it makes sense to measure the number of machine instructions in terms of the number of words

the machine instructions occupy. (This makes sense even in terms of those machine instructions that employ more than one word. For example, a conditional branch instruction requires 4 words, yet those 4 words are generally occupied by 4 instructions.)

Measuring size in terms of machine words conforms to the sponsor's requirement that programs be measured in terms of number of lines of source code for that portion of a program written in a higher order language, since each line of the listing representing interpretive code can be considered to be a line of source code and corresponds to a machine word.

Further, since interpretive code, machine instructions and constants are apt to be interspersed throughout a log section, subroutine or functional category, it makes sense to use the same yardstick for interpretive code as for machine instructions. Note that this situation is different from the typical one, where different modules are written in different languages, in that one segment of code is often written partly in a higher order language (interpretive code) and partly in a (more normal) assembly language.

Since modules are viewed as functional categories, the size of each module given here is based on an approximation obtained by considering each "log section"* as containing one function. To the extent that this is true, the sizes are correct. It should be recognized, however, that, although log sections in general were assigned on a functional basis, certain code embedded in any given log section properly belonged to a function other than that represented by the log section. As an example, a log section that was, quite reasonably, judged to belong to the functional category of "navigation" (since a very large proportion of its code is dedicated to that function), contains within it some code that is "display", some that is "I/O", and perhaps some other categories as well. Since the functional category assigned to the code modification (column 27 in the first data file) was assigned on the basis of actual function, there is an inconsistency between the sizes given here and the modification functional categories.

The first step in computing the size of the functional categories for the two programs was to assign each log section of each of the two programs to one (or, in some cases, to none or to two) functional categories, depending on which category (or categories) best described the log section's principal function. The result of this step is shown in the Table 4-8

*An explanation of "log section" and "subroutine"--these fields appear on the Modification Report (MR)--is given in Appendix B.

headed "Functional Categories Applied to Colossus 237" and Table 4-9 headed "Functional Categories Applied to Luminary 98". The second step was to take the listing for each of the two programs in turn, look up on the listing the size of each log section and add this number to the size for the appropriate functional category.

One minor problem connected with this procedure arises from the fact that, for each of the programs, just 2 (out of some 80) log sections are assigned to more than one functional category. In particular, in both Colossus revision 237 and in Luminary revision 98 the log section T4RUPT PROGRAM is assigned to functional category "E" (I/O) as well as "T" (Hardware Failure Monitor), and IMU COMPENSATION PACKAGE assigned to "E" and "U" (Hardware Service Routine). Clearly we do not want to add the size of a log section to the size of each of two functional categories. There are at least the following 3 ways of resolving this problem: (1) We arbitrarily split the size of the log section in two, and add half to one functional category and half to the other. (2) We could specify, for each log section which is assigned to more than one functional category, what proportion it is most appropriate to assign to one and what to the other. (3) We assign the size of the log sections to the more appropriate of the two categories. We chose the third of these 3 ways. The reasoning was as follows: All the log sections in question involved I/O. Input/Output is a more subsidiary function than is Hardware Services and Hardware Failure Monitor, and is thus best subsumed under (embedded in) the more primary function. Input/Output may not be a meaningful functional category in the first place.

Tables 4-8 and 4-9 were prepared by two engineers with extensive experience with Apollo software. For each log section the functional category (see paragraph 4.1.5) was selected which best described the function to which the log section contributed. Some log sections consisted only of constants, which participated in many, if not all functions. The functional category column in the table for these log sections is marked "all". The log section ENTRY LEXICON of Colossus 237 consisted of a comment and did not produce executable code, hence could not be assigned to a functional category, and is marked "comment".

4.2.5.2 MODE OF CONSTRUCTION

All the 16 flight programs were developed using conventional programming techniques. Structured programming was not in general use at the time of the Apollo development.

Table 4-8 FUNCTIONAL CATEGORIES APPLIED TO COLOSSUS 237

Sub-routine	Log Section	Functional Category (coded)
KILERASE	ERASABLE ASSIGNMENTS	all
	INTERRUPT LEAD INS	F
	T4RUPT PROGRAM	E,T
	DOWNLINK LISTS	D
	FRESH START AND RESTART	G
	RESTART TABLES	B
	SXTMARK	I
	EXTENDED VERBS	C
	PINBALL NOUN TABLES	H
KOOLADE	CSM GEOMETRY	J
	IMU COMPENSATION PACKAGE	E,U
	PINBALL GAME BUTTONS AND LIGHTS	H
	P60,P62	K
	ANGLFIND	K
	GIMBAL LOCK AVOIDANCE	K
	KALCMANU STEERING	K
	SYSTEM TEST STANDARD LEAD INS	T
	IMU CALIBRATION AND ALIGNMENT	U
	GROUND TRACKING DETERMINATION PROGRAM - P21-P29	L
	P34-P35, P74-P75	M
SMOOCH	P31	C
	P76	M
	P80	C
	STABLE ORBIT - P38-P39	M
	P11	N
	TPI SEARCH	M
	P20-P25	L
	P30,P37	M
	P40-P47	N
	P51-P53	P
	LUNAR AND SOLAR EPHEMERIDES SUBROUTINES	R
PANDORA	P61-P67	A
	SERVICER207	I
	ENTRY LEXICON	comment
	REENTRY CONTROL	A
	CM BODY ATTITUDE	K
	P37,P70	M
	S-BAND ANTENA FOR CM	C
	LUNAR LANDMARK SELECTION FOR CM	I

Table 4-8 FUNCTIONAL CATEGORIES APPLIED TO COLOSSUS 237 (cont.)

Sub-routine	Log Section	Functional Category (coded)
DAPCSM	TVCINITIALIZE	A
	P15	G
	TVCEXECUTIVE	A
	TVCMASSPROP	A
	TVCRESTARTS	B
	TVCDAPS	A
	TVCSTROKETEST	T
	TVCRCIIDAP	A
	TVCGENBFILTERS	A
	MYSUBS	R
	RCS-CSM DIGITAL AUTOPILOT	A
	AUTOMATIC MANEUVERS	K
	RCS-CSM DAP EXECUTIVE PROGRAMS	A
	IFT SELECTION LOGIC	A
	CM ENTRY DIGITAL AUTOPILOT	A
SATRAP	DOWN-TELEMETRY PROGRAM	D
	INTER-BANK COMMUNICATION	F
	INTERPRETER	Q
	FIXED-FIXED CONSTANT POOL	all
	INTERPRETIVE CONSTANTS	all
	SINGLE PRECISION SUBROUTINES	R
	EXECUTIVE	F
	WAITLIST	F
	LATITUDE LONGITUDE SUBROUTINES	J
	PLANETARY INERTIAL ORIENTATION	J
	MEASUREMENT INCORPORATION	I
	CONIC SUBROUTINES	I
	INTEGRATION INITIALIZATION	I
	ORBITAL INTEGRATION	I
	INFLIGHT ALIGNMENT ROUTINES	P
	POWERED FLIGHT SUBROUTINES	N
	TIME OF FREE FALL	M
	STAR TABLES	I
	AGC BLOCK TWO SELF-CHECK	T
	PHASE TABLE MAINTENANCE	B
	RESTARTS ROUTINE	B
	IMU MODE SWITCHING ROUTINES	U
	KEYRUPT, UPRUPT	D
	DISPLAY INTERFACE ROUTINES	H
	SERVICE ROUTINES	S
	ALARM AND ABORT	B
	UPDATE PROGRAM	D
	RTB OP CODES	S

Table 4-9 FUNCTIONAL CATEGORIES APPLIED TO LUMINARY 98

Sub-routine	Log Section	Functional Category (coded)
LUMERASE	ERASABLE ASSIGNMENTS	all
LEMONAID	INTERRUPT LEAD INS	F
	T4RUPT PROGRAM	E,T
	RCS FAILURE MONITOR	T
	DOWNLINK LISTS	D
	AGS INITIALIZATION	B
	FRESH START AND RESTART	G
	RESTART TABLES	B
	AOTMARK	I
	EXTENDED VERBS	C
	PINBALL NOUN TABLES	H
	LEM GEOMETRY	J
	IMU COMPENSATION PACKAGE	E,U
	R63	K
	ATTITUDE MANEUVER ROUTINE	K
	GIMBAL LOCK AVOIDANCE	K
	KALCMANU STEERING	K
	SYSTEM TEST STANDARD LEAD INS	T
	IMU PERFORMANCE TESTS 2	T
	IMU PERFORMANCE TESTS 4	T
	PINBALL GAMES BUTTONS AND LIGHTS	H
	R60,R62	K
	S-BAND ANTENNA FOR LM	C
LEMP20S	RADAR LEADIN ROUTINES	U
	P20-P25	L
LEMP30S	P30,P37	M
	P32-P35, P72-P75	M
	GENERAL LAMBERT AIMPOINT GUIDANCE	M
KISSING	GROUND TRACKING DETERMINATION PROGRAM - P21	L
	P34-P35, P74-P75	M
	R31	C
	P76	M
	R30	C
	STABLE ORBIT - P38-P39	M
FLY	BURN, BABY, BURN -- MASTER IGNITION ROUTINE	N
	P40-P47	N
	THE LUNAR LANDING	N
	THROTTLE CONTROL ROUTINES	O
	LUNAR LANDING GUIDANCE EQUATIONS	O

Table 4-9 FUNCTIONAL CATEGORIES APPLIED TO LUMINARY 98 (cont.)

Sub-routine	Log Section	Functional Category
	P70-P71	N
	P12	N
FLY	ASCENT GUIDANCE	O
	SERVICER	I
	LANDING ANALOG DISPLAYS	C
	FINDCDUP - GUIDAP INTERFACE	O
LEMP50S	P51-P53	P
	LUNAR AND SOLAR EPHEMERIDES SUBROUTINES	R
	DOWN-TELEMETRY PROGRAM	D
	INTER-BANK COMMUNICATION	F
	INTERPRETER	Q
	FIXED-FIXED CONSTANT POOL	all
	INTERPRETIVE CONSTANTS	all
	SINGLE PRECISION SUBROUTINES	R
	EXECUTIVE	F
	WAITLIST	F
	LATITUDE LONGITUDE SUBROUTINES	J
	PLANETARY INERTIAL ORIENTATION	J
	MEASUREMENT INCORPORATION	I
	CONIC SUBROUTINES	I
	INTEGRATION INITIALIZATION	I
SKIPPER	ORBITAL INTEGRATION	I
	INFLIGHT ALIGNMENT ROUTINES	P
	POWERED FLIGHT SUBROUTINES	N
	TIME OF FREE FALL	M
	AGC BLOCK TWO SELF-CHECK	T
	PHASE TABLE MAINTENANCE	B
	RESTARTS ROUTINE	B
	IMU MODE SWITCHING ROUTINES	U
	KEYRUPT, UPRUPT	D
	DISPLAY INTERFACE ROUTINES	H
	SERVICE ROUTINES	S
	ALARM AND ABORT	B
	UPDATE PROGRAM	D
	RTB OP CODES	S
	T6-RUPT PROGRAMS	A
	DAP INTERFACE SUBROUTINES	A
	DAPIDLER PROGRAM	A
	P-AXIS RCS AUTOPILOT	A
LMDAP	Q-R AXIS	A
	KALMAN FILTER	R
	TRIM GIMBAL CONTROL SYSTEM	A
	AOSTASK AND AOSJOB	A
	SPS BACK-UP RCS CONTROL	A

4.2.5.3 LANGUAGES USED

The language used on this project was assembly language, with interpretive code interspersed throughout an assembly language program.

Typically interpretive code and machine instructions, as well as data, are interspersed throughout a log section, subroutine or functional category. Thus there would not likely be a log section consisting only of interpretive code.

The interpretive language being used is primarily oriented toward doing three kinds of arithmetic, 1) one that operates on 28 bits plus sign fixed point scalar numbers, 2) one that operates on 42 bits plus sign fixed point scalar numbers, and 3) the third that operates on three element vectors, each of whose elements is a 28 bit plus sign fixed point scalar number. The vector arithmetic includes provision for multiplying three element vectors and 3 by 3 matrices. Even though this language is thus considerably more powerful than a typical assembly language, its form (syntax) is that of an assembly language rather than of a higher order language. For a fuller description of the language see Appendix A. Use of the interpretive language instead of assembly language (or machine code) generally saves storage space in the AGC memory at the expense of speed of execution.

SECTION 5

DATA SUMMARIES

The data described in Section 4 have been summarized and presented in this section in tabular and graphic form. The following observations are made from examining these summaries with a knowledge of the software development.

5.1 PATTERNS OF FLIGHT SOFTWARE DEVELOPMENT

The general pattern over time of the number of software modifications accurately reflects the history of program development. Figure 5-1 shows that the periods of greatest activity would be expected before the Apollo 9 flight and before the Apollo 15 flight, since a long lead development time was scheduled for those flights and they were developed in parallel with others. The data bear this out (Figures 5-2, 5-3, 5-4; Tables 5-1, 5-2, 5-3, 5-4). In mid-1970 a new flurry of activity followed the release of Apollo 14, reflecting the fact that a large number of new capabilities were specified for the Apollo 15 flight. The increased modification activity shown for the Apollo 15 flight reflects the space saving activity that took place to enable the implementation of those major improvements.

At least two ropes were under development at all times. Throughout 1967 and for a period in 1969 and 1970, three or more were being developed simultaneously. Many of the modifications tabulated for these periods, therefore, are multiple implementations of the same change.

Apollo 9 was the first joint flight with the Lunar Module; not surprisingly, Table 5-4 shows that the vast majority of modifications for Apollo 9 were those made to the Lunar Module program, Sundance.

5.2 FUNCTIONAL CATEGORIES

Figure 5-5 and Tables 5-5, 5-6, and 5-7 show that the greatest modification activity was in mission-oriented functions; powered flight, navigation, tracking, targeting and digital autopilot (DAP). These functions were all specifically related to the lunar landing and the rendezvous programs that were being newly developed during the period covered by this study.

5.3 MODIFICATION ACTIVITY RELATED TO MEMORY SIZE

It was expected that a correlation would be observed between the size of the functional categories and the number of modifications to them. The data do not exhibit this, however,

as shown in Figure 5-6 and Table 5-8.

It is probably true that modification activity is more directly correlated to the specific development taking place, as discussed in Section 5-2, than to memory size.

A factor that should be considered is the method used in this study to determine the sizes of the functions. The sizes were taken directly from two intermediate programs that were thought to be representative. Furthermore, the determination of size and the assignment of functional categories to the modifications were done on different bases: the sizes were based on log sections, while the assignment of a category to a modification was based on the actual content of the modification itself. Thus the usefulness of the figures on sizes of functions is questionable.

Off-line development and checkout were more common for some functions than for others. Data on this off-line activity are not available, but it may well be a factor in these statistics. It is known, for example, that the digital autopilot (DAP) programming group did a large amount of off-line work, while the powered flight programming group did not. The data show that the percentage of modifications vs. size for these two functions is in direct opposition, 7.4% of the modifications vs. 13% of the memory size for the DAP, 13.1% of the modifications vs. 7.8% of memory size for powered flight.

5.4 MAJOR MODIFICATION ACTIVITY

It was expected even prior to the examination of any of the data summaries that large numbers of modifications would fall into the categories of logic, in-house improvements, compool definition, interfaces, configuration control and user requests. The nature of the Apollo project and its computer architecture led to this expectation, which is borne out by the data (see Figure 5-7, Tables 5-9, 5-10 and 5-11).

The use of assembly language undoubtedly contributed to the preponderance of logic errors; had a higher order language been used, the percentage of these errors would no doubt have been smaller.

The number of in-house improvements was expected to be, and was, large. This category included memory optimizations as an ongoing activity.

Modifications to compool variables were expected to be large in number because of the time-sharing of erasable memory. This limitation imposed continuing requirements for modifica-

tions throughout the development.

Interfaces are a traditional source of error; in the Apollo project, the complexities of real-time interactions and the use of assembly language coupled with interpretive language exacerbated the problems in this area.

The fixed bank architecture of the computer required specific code to address remote banks; in addition, the memory organization was changed frequently to accommodate program changes, both actual and anticipated.

The "user requested" category is actually a more significant factor than is shown by the number of modifications in that category, since more often than not, a single change involved one or more large blocks of code. Data on the size of the individual changes are not available.

It was expected that the number of computational errors would be significant, due to the large percentage of "number-crunching" code and especially because of the fixed point scaling that was used. The data show, however, that this category was relatively small in comparison to the others discussed above.

5.5 DEVELOPMENT PHASE VS. VERIFICATION PHASE

Although Figure 5-8 and Table 5-12 indicate that most of the modifications to each rope were made during the development phase as expected, the information presented is probably not entirely accurate. The configuration control dates for Apollo flights 7,8, and 9 are available and, therefore, the phase indicators for these flights are reliable. Phase indicators for later flights were based on the completion of level 4 testing, which approximates the configuration control dates.

It should be pointed out that some errors were found after configuration control but were not corrected for that flight and, therefore, will not appear in the verification phase data; instead; corrections were implemented in the development phase of the next flight. Figure 3-1, page 23, shows that a significant number of known anomalies were allowed to remain in the flight programs.

5.6 REFERENCED DOCUMENTS

Tables 5-13 and 5-14 illustrate the references in the data to supporting documents- PCRs, PCNs, ACBs and Anomaly Reports.

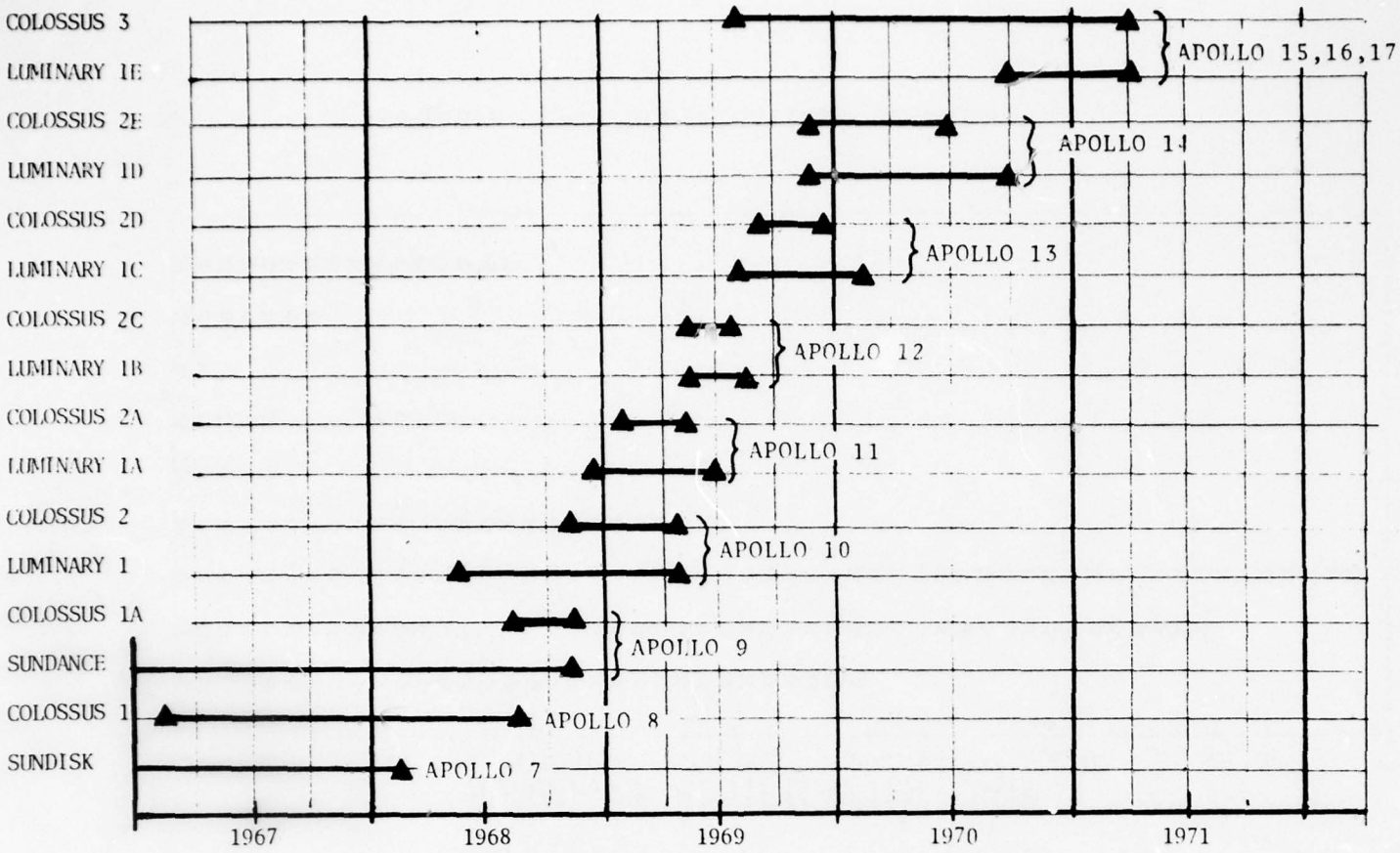


Figure 5-1 FLIGHT SOFTWARE DEVELOPMENT

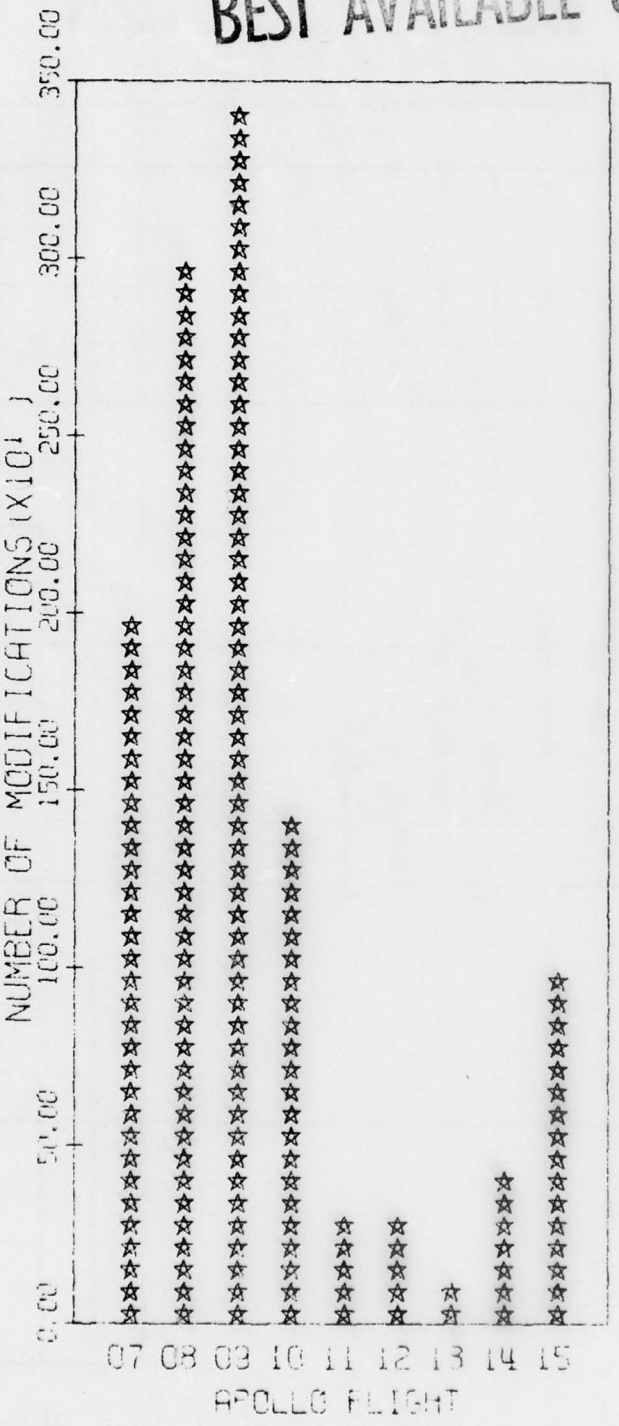


Figure 5-2 MODIFICATIONS BY FLIGHT

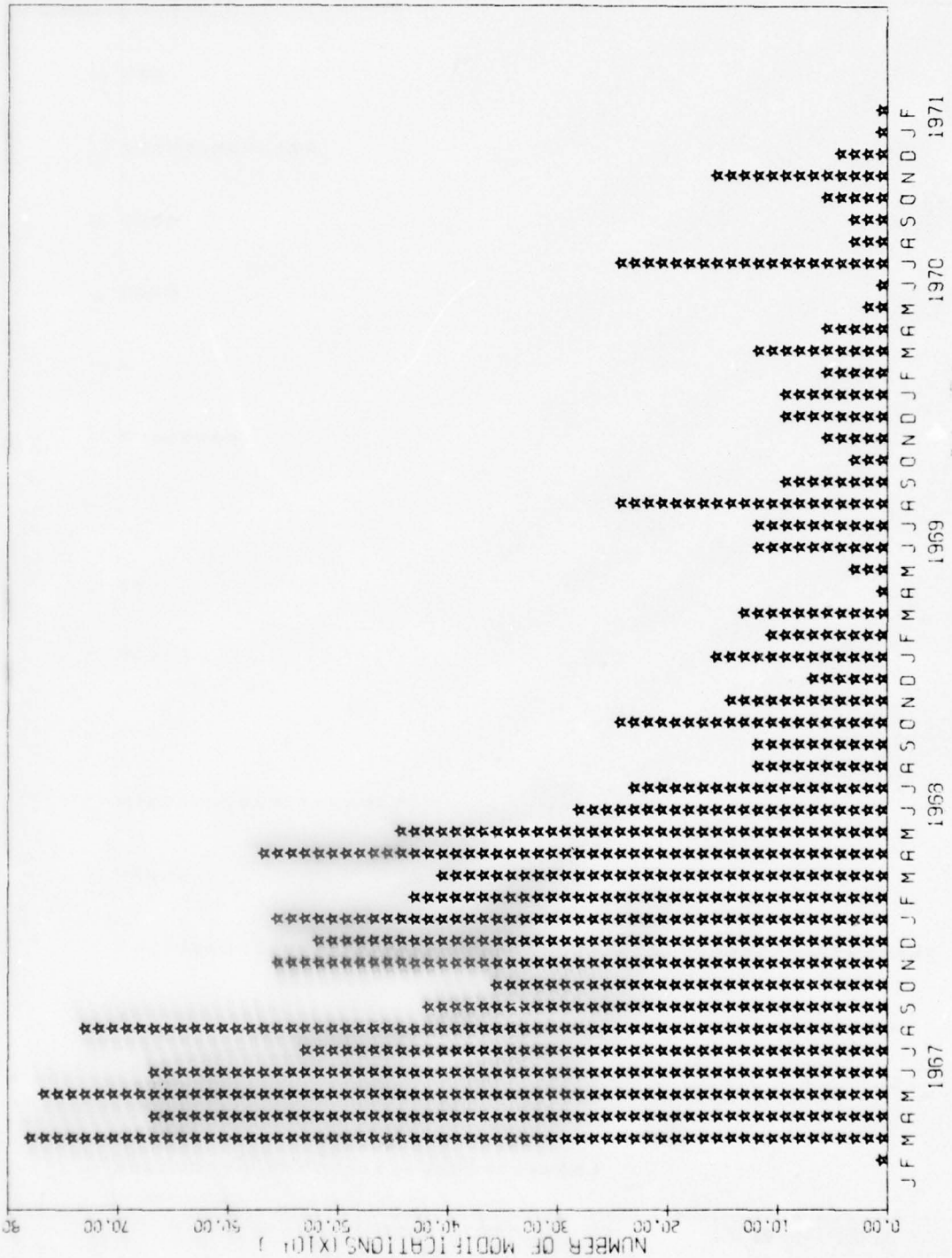


Figure 5-3 MODIFICATIONS BY MONTH

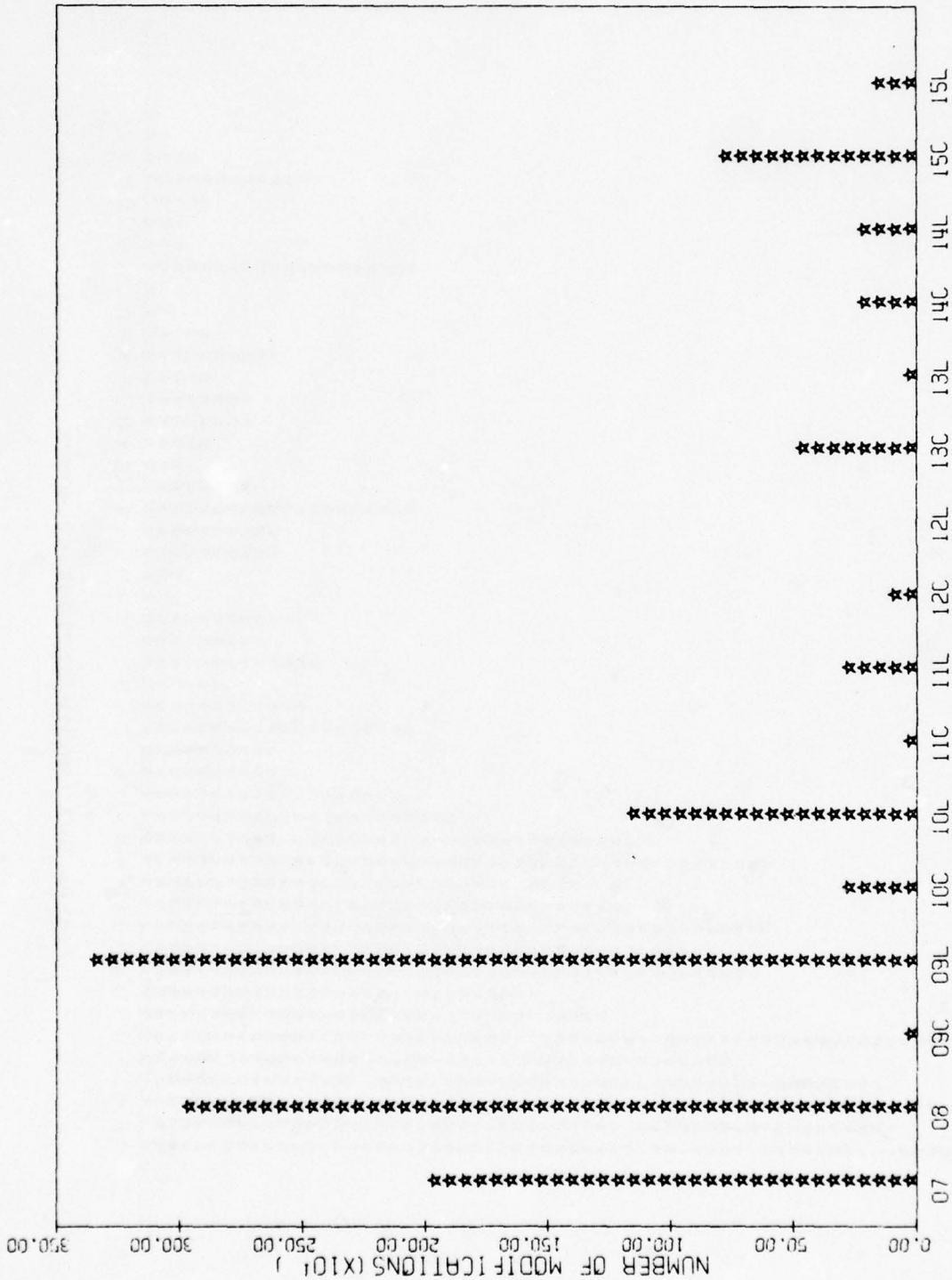


Figure 5-4 MODIFICATIONS BY ROPE

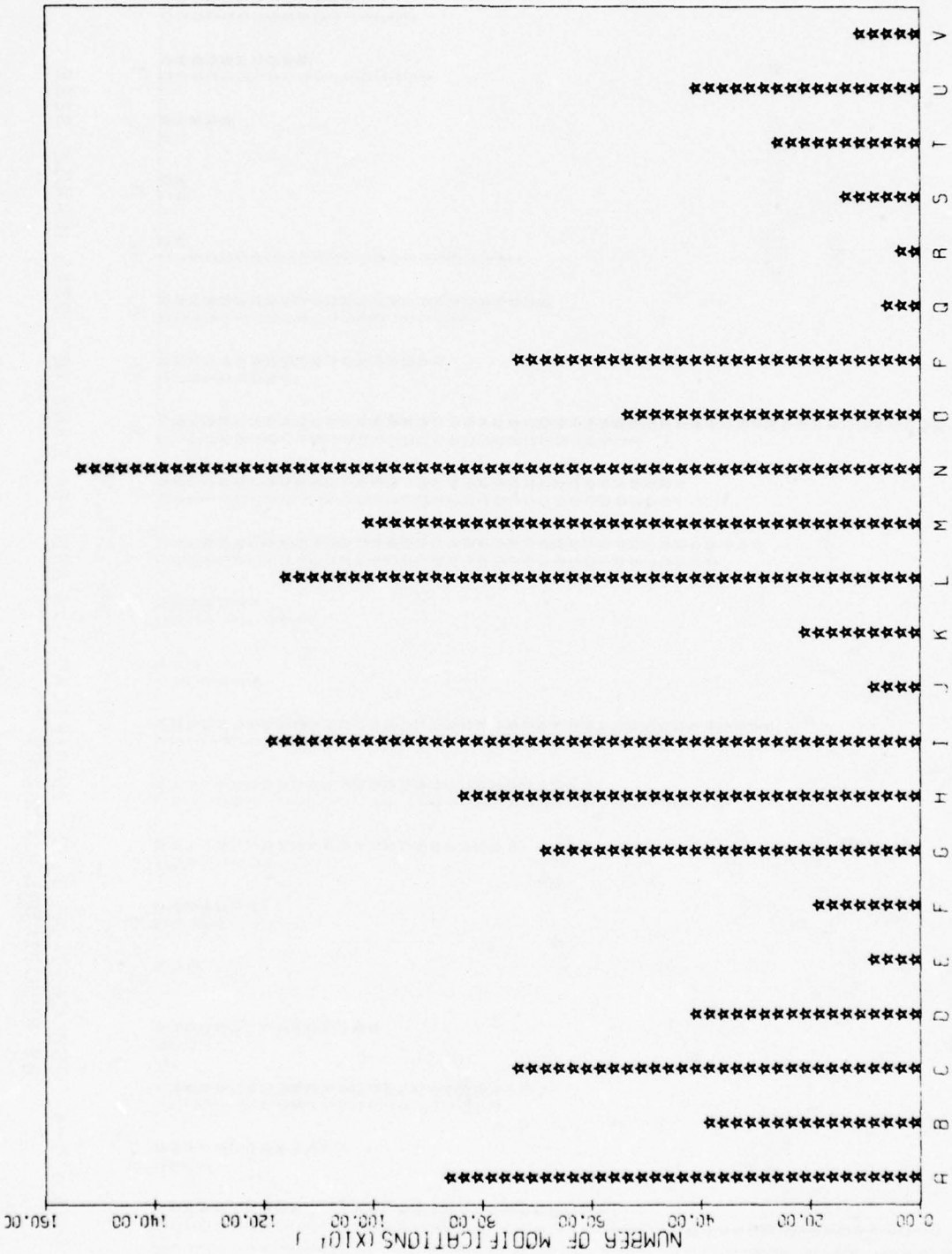


Figure 5-5 MODIFICATIONS BY FUNCTIONAL CATEGORY

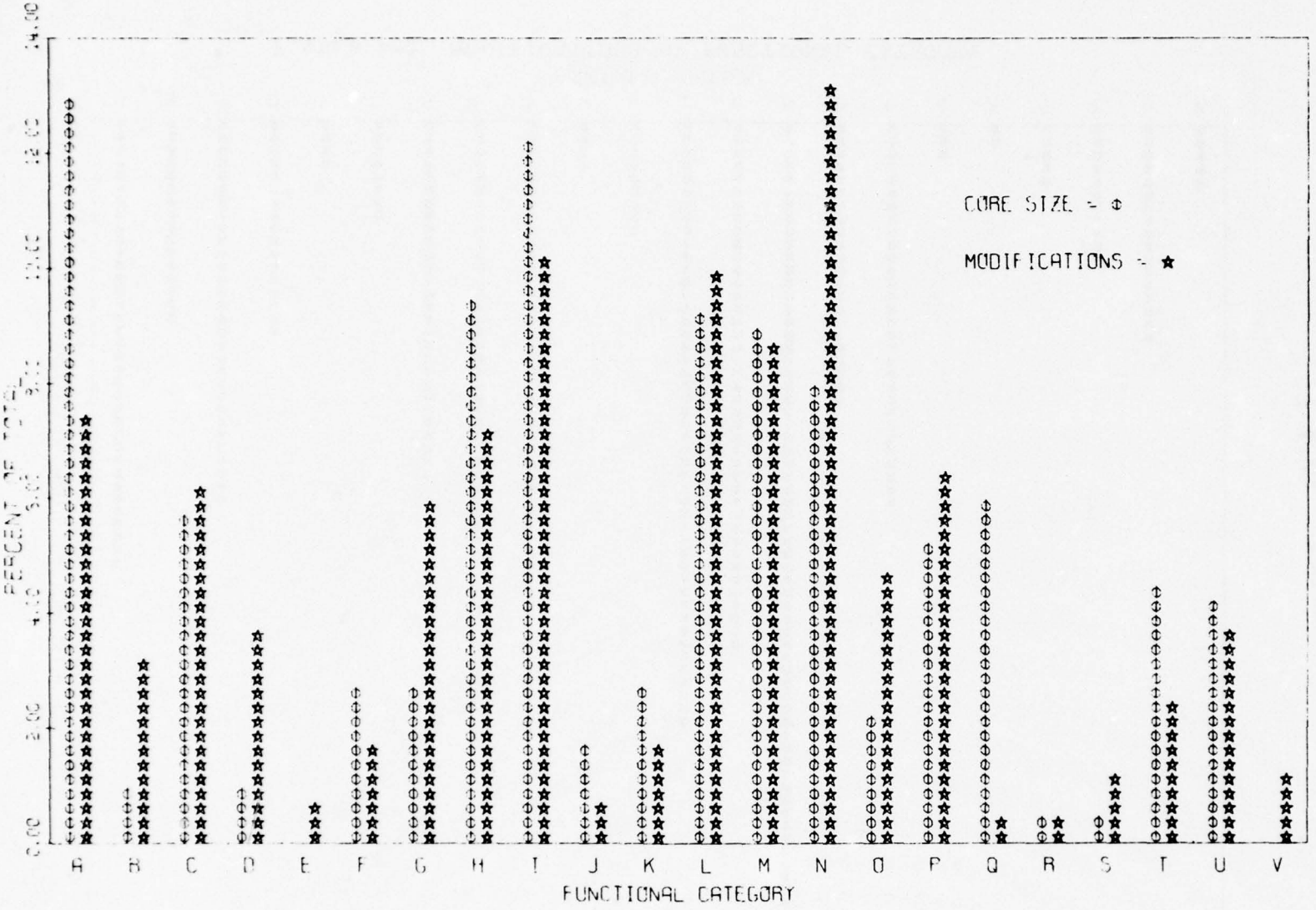


Figure 5-6 PER CENT OF TOTAL MODIFICATIONS AND PER CENT OF TOTAL MODULE SIZE BY FUNCTIONAL CATEGORY

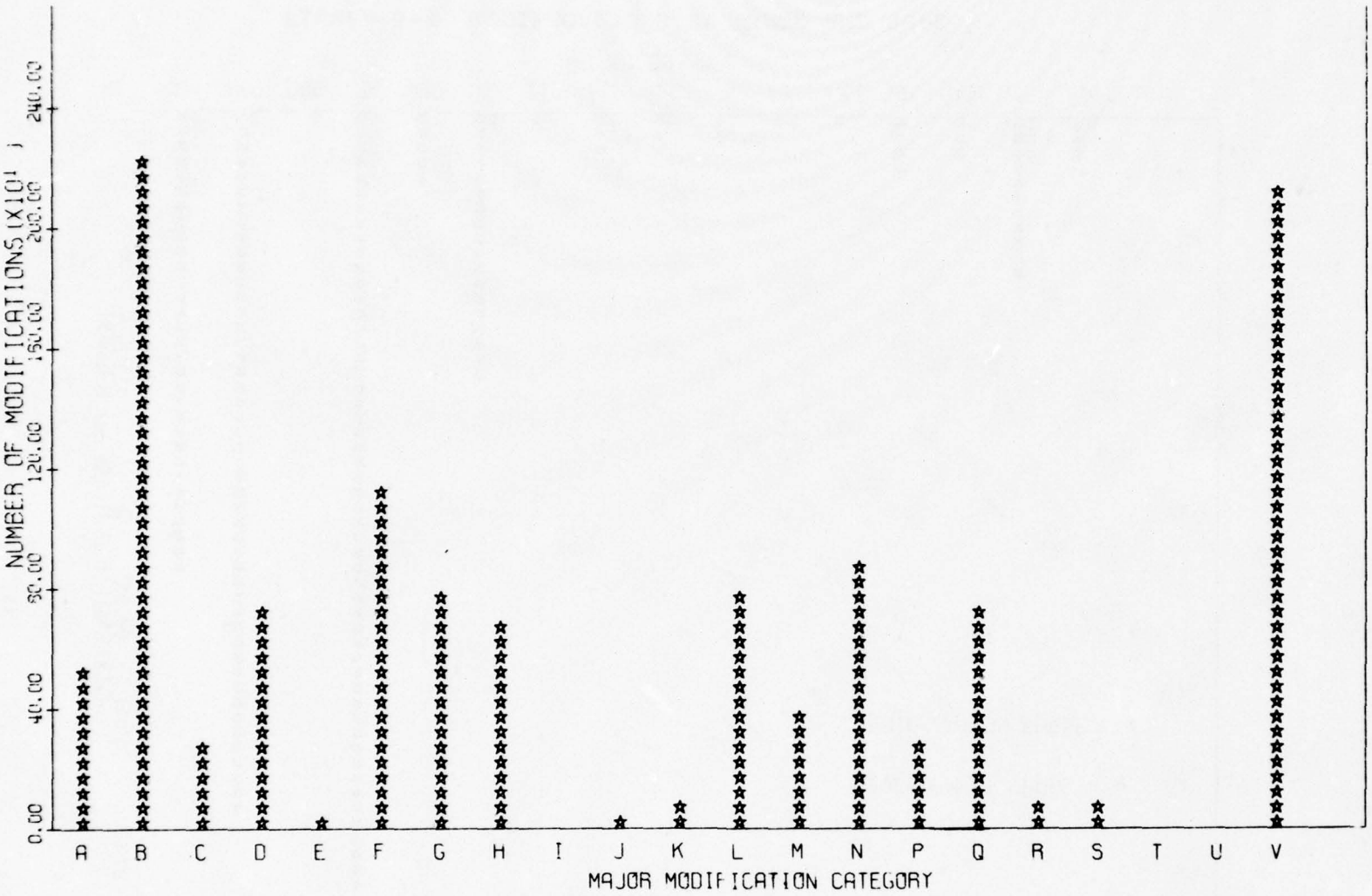


Figure 5-7 MODIFICATIONS BY MAJOR MODIFICATION CATEGORY

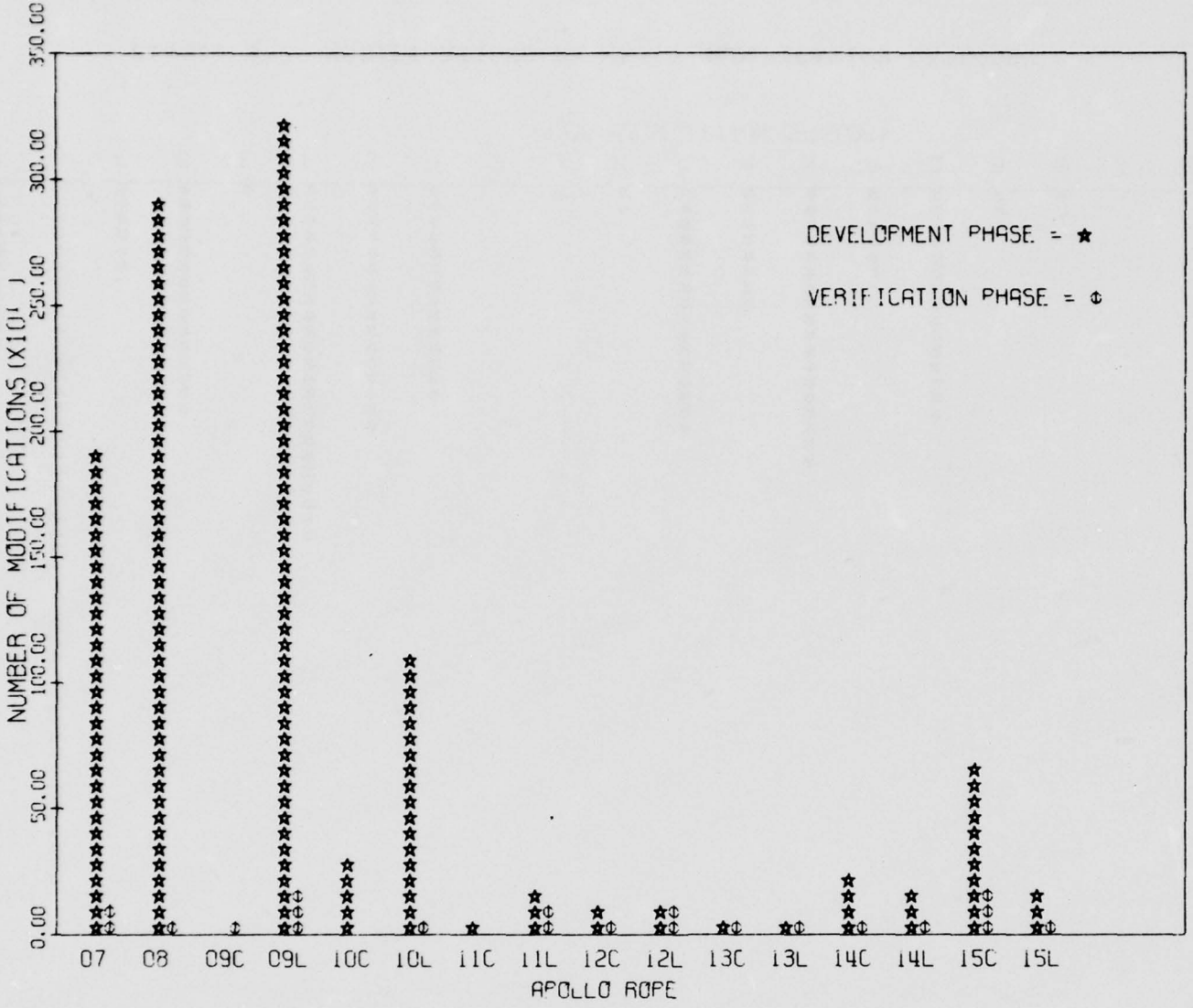


Figure 5-8 MODIFICATIONS BY PHASE AND ROPE

TABLE 5-1 MODIFICATIONS BY ROPE AND MONTH

YEAR	MONTH	APOLLO FLIGHT PROGRAM															
		07	08	09C	09L	10C	10L	11C	11L	12C	12L	13C	13L	14C	14L	15C	15L
1967	JANUARY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FEBRUARY	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	MARCH	426	0	0	353	0	0	0	0	0	0	0	0	0	0	0	0
	APRIL	334	56	0	283	0	0	0	0	0	0	0	0	0	0	0	0
	MAY	311	291	0	170	0	0	0	0	0	0	0	0	0	0	0	0
	JUNE	258	258	0	155	0	0	0	0	0	0	0	0	0	0	0	0
	JULY	158	178	0	193	0	0	0	0	0	0	0	0	0	0	0	0
	AUGUST	248	240	0	245	0	0	0	0	0	0	0	0	0	0	0	0
	SEPTEMBER	134	132	0	158	0	0	0	0	0	0	0	0	0	0	0	0
	OCTOBER	39	83	0	230	0	0	0	0	0	0	0	0	0	0	0	0
	NOVEMBER	23	142	0	391	0	0	0	0	0	0	0	0	0	0	0	0
	DECEMBER	9	172	0	336	1	0	0	0	0	0	0	0	0	0	0	0
1968	JANUARY	24	341	0	190	0	1	0	0	0	0	0	0	0	0	0	0
	FEBRUARY	0	204	0	223	0	0	0	0	0	0	0	0	0	0	0	0
	MARCH	0	165	0	225	0	11	0	0	0	0	0	0	0	0	0	0
	APRIL	0	196	0	72	0	306	0	0	0	0	0	0	0	0	0	0
	MAY	0	198	0	25	0	225	0	0	0	0	0	0	0	0	0	0
	JUNE	0	125	0	48	0	108	0	0	0	0	0	0	0	0	0	0
	JULY	0	120	0	5	0	111	0	0	0	0	0	0	0	0	0	0
	AUGUST	0	46	0	0	2	66	0	0	0	0	0	0	0	0	0	0
	SEPTEMBER	0	0	14	0	4	97	0	0	0	0	0	0	0	0	0	0
	OCTOBER	LAUNCH	0	37	21	69	111	0	0	0	0	0	0	0	0	0	0
	NOVEMBER	0	0	0	0	38	104	0	1	0	0	0	0	0	0	0	0
	DECEMBER	0	LAUNCH	1	0	65	0	0	1	0	0	0	0	0	0	0	0
1969	JANUARY	0	0	0	1	98	0	1	58	0	0	0	0	2	0	0	0
	FEBRUARY	0	0	0	0	14	0	3	88	0	0	0	0	0	0	0	0
	MARCH	0	0	LAUNCH		0	0	28	106	0	0	0	0	0	0	0	0
	APRIL	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0
	MAY	0	0	0	0	LAUNCH		0	4	6	21	0	0	0	0	0	0
	JUNE	0	0	0	0	0	0	0	0	67	56	0	0	0	0	0	0
	JULY	0	0	0	0	0	0	LAUNCH		21	74	0	0	0	0	19	0
	AUGUST	0	0	0	0	0	0	0	0	0	21	0	9	0	3	207	0
	SEPTEMBER	0	0	0	0	0	0	0	0	0	0	15	5	0	0	73	0
	OCTOBER	0	0	0	0	0	0	0	0	0	0	4	32	0	0	1	0
	NOVEMBER	0	0	0	0	0	0	0	0	LAUNCH		3	1	44	3	10	0
	DECEMBER	0	0	0	0	0	0	0	0	0	0	0	2	45	20	21	0
1970	JANUARY	0	0	0	0	0	0	0	0	0	0	0	0	47	42	0	0
	FEBRUARY	0	0	0	0	0	0	0	0	0	0	0	0	22	14	24	0
	MARCH	0	0	0	0	0	0	0	0	0	0	0	0	27	66	27	0
	APRIL	0	0	0	0	0	0	0	0	0	0	LAUNCH		15	19	27	0
	MAY	0	0	0	0	0	0	0	0	0	0	0	0	9	9	6	0
	JUNE	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0
	JULY	0	0	0	0	0	0	0	0	0	0	0	0	0	22	215	2
	AUGUST	0	0	0	0	0	0	0	0	0	0	0	0	0	6	23	0
	SEPTEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	23
	OCTOBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	38
	NOVEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	92	68
	DECEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	35
1971	JANUARY	0	0	0	0	0	0	0	0	0	0	0	0	LAUNCH		1	10
	FEBRUARY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

TABLE 5-2 MODIFICATIONS BY FLIGHT

APOLLO FLIGHT	NUMBER OF MODIFICATIONS
7	1964
8	2947
9	3377
10	1431
11	296
12	266
13	71
14	421
15	956

TABLE 5-3 MODIFICATIONS BY MONTH

YEAR	MONTH	NUMBER OF MODIFICATIONS
1967	JANUARY	0
	FEBRUARY	1
	MARCH	779
	APRIL	673
	MAY	772
	JUNE	671
	JULY	529
	AUGUST	733
	SEPTEMBER	424
	OCTOBER	352
	NOVEMBER	556
	DECEMBER	518
1968	JANUARY	556
	FEBRUARY	427
	MARCH	401
	APRIL	574
	MAY	448
	JUNE	281
	JULY	236
	AUGUST	114
	SEPTEMBER	115
	OCTOBER	238
	NOVEMBER	143
	DECEMBER	67
1969	JANUARY	160
	FEBRUARY	105
	MARCH	134
	APRIL	6
	MAY	31
	JUNE	123
	JULY	114
	AUGUST	240
	SEPTEMBER	93
	OCTOBER	37
	NOVEMBER	61
	DECEMBER	88
1970	JANUARY	89
	FEBRUARY	60
	MARCH	120
	APRIL	61
	MAY	24
	JUNE	6
	JULY	239
	AUGUST	29
	SEPTEMBER	30
	OCTOBER	60
	NOVEMBER	160
	DECEMBER	39
1971	JANUARY	11
	FEBRUARY	1

TABLE 5-4 MODIFICATIONS BY ROPE

APOLLO ROPE	NUMBER OF MODIFICATIONS
07	1964
08	2947
09C	52
09L	3325
10C	291
10L	1140
11C	32
11L	264
12C	94
12L	172
13C	22
13L	49
14C	211
14L	219
15C	780
15L	176

TABLE 5-5 MODIFICATIONS BY FUNCTIONAL CATEGORY

FUNCTIONAL CATEGORY	NUMBER OF MODIFICATIONS	PERCENT OF TOTAL
A DIGITAL AUTOPILOT	873	7.4
B ERROR DETECTION/RECOVERY	380	3.2
C CREW INTERFACE	728	6.2
D TELEMETRY	422	3.6
E I/O	85	0.7
F EXECUTIVE	198	1.7
G SEQUENCE INITIALIZATION/REINITIALIZATION	700	6.0
H DISPLAY	826	7.0
I NAVIGATION	1195	10.2
J COORDINATE TRANSFERS	79	0.7
K VEHICLE ATTITUDE COMPUTATIONS/MANEUVERS	206	1.8
L TRACKING	1165	9.9
M TARGETING	1025	8.7
N POWERED FLIGHT MANEUVERS	1533	13.1
O GUIDANCE COMPUTATIONS	534	4.6
P ALIGNMENTS	740	6.3
Q INTERPRETER	53	0.5
R MATH SUBROUTINES	35	0.3
S SOFTWARE UTILITY ROUTINES	133	1.1
T HARDWARE FAILURE MONITOR	272	2.3
U HARDWARE SERVICE ROUTINES	425	3.6
V MANUAL OPERATIONS (NON-SOFTWARE)	122	1.0

TABLE 5-6 MODIFICATIONS BY ROPE AND FUNCTIONAL CATEGORY

FUNCTIONAL CATEGORY	NUMBER OF MODIFICATIONS PER ROPE															
	07	08	09C	09L	10C	10L	11C	11L	12C	12L	13C	13L	14C	14L	15C	15L
A DIGITAL AUTOPILOT	188	287	13	190	16	64	4	8	6	11	2	3	15	2	52	12
B ERROR DETECT/RECOV	74	86	2	113	13	24	1	5	10	2	2	0	8	6	19	15
C CREW INTERFACE	109	195	6	224	40	39	5	0	20	16	5	0	8	1	57	3
D TELEMETRY	102	121	0	106	6	16	0	2	3	8	0	4	14	11	25	4
E I/O	11	21	0	30	0	7	0	0	0	2	0	0	6	0	6	2
F EXECUTIVE	21	24	0	64	3	21	0	0	3	1	0	0	5	9	34	13
G SEQUENCE INIT/REINIT	116	171	5	172	24	94	0	7	0	10	0	1	23	7	65	5
H DISPLAY	142	199	2	247	12	96	6	11	3	13	0	2	11	11	54	17
I NAVIGATION	109	360	3	400	12	158	0	30	6	9	3	1	11	52	31	10
J COORDINATE TRANSFERS	12	27	0	17	1	3	0	4	1	1	0	0	5	0	8	0
K VEH ATTITUDE COMP	20	33	0	85	4	15	1	0	0	1	0	1	9	0	35	2
L TRACKING	148	342	8	240	48	141	5	37	4	24	2	8	37	24	80	17
M TARGETING	151	282	4	295	68	78	4	12	4	9	3	5	19	2	82	7
N POWERED FLT MANEUVER	410	395	4	398	23	132	3	29	4	22	1	7	9	18	67	11
O GUIDANCE COMPUTATION	1	4	0	230	3	127	1	76	0	25	0	10	0	31	7	19
P ALIGNMENTS	213	133	1	194	9	63	2	26	3	6	0	1	10	4	51	24
Q INTERPRETER	11	10	1	17	0	4	0	0	1	0	0	0	0	0	9	0
R MATH SUBROUTINES	1	16	0	4	2	0	0	0	0	1	0	0	1	1	9	0
S S/W UTILITY ROUTINES	18	38	0	47	0	6	0	3	1	8	0	0	1	0	10	1
T H/W FAILURE MONITOR	50	65	0	152	0	0	0	0	0	0	0	0	3	0	2	0
U H/W SERVICE ROUTINES	52	131	3	93	7	33	0	5	3	3	0	2	14	4	71	4
V MANUAL OPS (NON-S/W)	5	7	0	7	0	19	0	9	22	0	4	4	2	27	6	10

TABLE 5-7 MODIFICATIONS BY ROPE AND FUNCTIONAL CATEGORY (PERCENT OF ROPE TOTAL)

FUNCTIONAL CATEGORY	PERCENT OF MODIFICATIONS PER ROPE															
	07	08	09C	09L	10C	10L	11C	11L	12C	12L	13C	13L	14C	14L	15C	15L
A DIGITAL AUTOPILOT	9.6	9.7	25.0	5.7	5.5	5.6	12.5	3.0	6.4	6.4	9.1	6.1	7.1	1.0	6.7	6.8
B ERROR DETECT/RECOV	3.8	2.9	3.8	3.4	4.5	2.1	3.1	1.9	10.6	1.2	9.1	0.0	3.8	2.9	2.4	8.5
C CREW INTERFACE	5.5	6.6	11.5	6.7	13.7	3.4	15.6	0.0	21.3	9.3	22.7	0.0	3.8	0.5	7.3	1.7
D TELEMETRY	5.2	4.1	0.0	3.2	2.1	1.4	0.0	0.8	3.2	4.7	0.0	8.2	6.6	5.2	3.2	2.3
E I/O	0.6	0.7	0.0	0.9	0.0	0.6	0.0	0.0	0.0	1.2	0.0	0.0	2.8	0.0	0.8	1.1
F EXECUTIVE	1.1	0.8	0.0	1.9	1.0	1.8	0.0	0.0	3.2	0.6	0.0	0.0	2.4	4.3	4.4	7.4
G SEQUENCE INIT/REINIT	5.9	5.8	9.6	5.2	8.2	8.2	0.0	2.7	0.0	5.8	0.0	2.0	10.9	3.3	8.3	2.8
H DISPLAY	7.2	6.8	3.8	7.4	4.1	8.4	18.8	4.2	3.2	7.6	0.0	4.1	5.2	5.2	6.9	9.7
I NAVIGATION	5.5	12.2	5.8	12.0	4.1	13.9	0.0	11.4	6.4	5.2	13.6	2.0	5.2	24.8	4.0	5.7
J COORDINATE TRANSFERS	0.6	0.9	0.0	0.5	0.3	0.3	0.0	1.5	1.1	0.6	0.0	0.0	2.4	0.0	1.0	0.0
K VEH ATTITUDE COMP	1.0	1.1	0.0	2.6	1.4	1.3	3.1	0.0	0.0	0.6	0.0	2.0	4.3	0.0	4.5	1.1
L TRACKING	7.5	11.6	15.4	7.2	16.5	12.4	15.6	14.0	4.3	14.0	9.1	16.3	17.5	11.4	10.3	9.7
M TARGETING	7.7	9.6	7.7	8.9	23.4	6.8	12.5	4.5	4.3	5.2	13.6	10.2	9.0	1.0	10.5	4.0
N POWERED FLT MANEUVER	20.9	13.4	7.7	12.0	7.9	11.6	9.4	11.0	4.3	12.8	4.5	14.3	4.3	8.6	8.6	6.3
O GUIDANCE COMPUTATION	0.1	0.1	0.0	6.9	1.0	11.1	3.1	28.8	0.0	14.5	0.0	20.4	0.0	14.8	0.9	10.8
P ALIGNMENTS	10.8	4.5	1.9	5.8	3.1	5.5	6.3	9.8	3.2	3.5	0.0	2.0	4.7	1.9	6.5	13.6
Q INTERPRETER	0.6	0.3	1.9	0.5	0.0	0.4	0.0	0.0	1.1	0.0	0.0	0.0	0.0	0.0	1.2	0.0
R MATH SUBROUTINES	0.1	0.5	0.0	0.1	0.7	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.5	0.5	1.2	0.0
S S/W UTILITY ROUTINES	0.9	1.3	0.0	1.4	0.0	0.5	0.0	1.1	1.1	4.7	0.0	0.0	0.5	0.0	1.3	0.6
T H/W FAILURE MONITOR	2.5	2.2	0.0	4.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.4	0.0	0.3	0.0
U H/W SERVICE ROUTINES	2.6	4.4	5.8	2.8	2.4	2.9	0.0	1.9	3.2	1.7	0.0	4.1	6.6	1.9	9.1	2.3
V MANUAL OPS (NON-S/W)	0.3	0.2	0.0	0.2	0.0	1.7	0.0	3.4	23.4	0.0	18.2	8.2	0.9	12.9	0.8	5.7

TABLE 5-8 PERCENT OF TOTAL MODIFICATIONS AND PERCENT OF TOTAL MODULE SIZE BY FUNCTIONAL CATEGORY

FUNCTIONAL CATEGORY	PERCENT OF MODIFICATIONS	PERCENT OF SIZE
A DIGITAL AUTOPILOT	7.4	13.0
B ERROR DETECTION/RECOVERY	3.2	0.8
C CREW INTERFACE	6.2	5.6
D TELEMETRY	3.6	0.9
E I/O	0.7	0.0
F EXECUTIVE	1.7	2.7
G SEQUENCE INITIALIZATION/REINITIALIZATION	6.0	2.7
H DISPLAY	7.0	9.4
I NAVIGATION	10.2	12.2
J COORDINATE TRANSFERS	0.7	1.5
K VEHICLE ATTITUDE COMPUTATIONS/MANEUVERS	1.8	2.7
L TRACKING	9.9	9.1
M TARGETING	8.7	9.0
N POWERED FLIGHT MANEUVERS	13.1	7.8
O GUIDANCE COMPUTATIONS	4.5	2.2
P ALIGNMENTS	6.3	5.2
Q INTERPRETER	0.4	6.0
R MATH SUBROUTINES	0.3	0.3
S SOFTWARE UTILITY ROUTINES	1.1	0.5
T HARDWARE FAILURE MONITOR	2.3	4.3
U HARDWARE SERVICE ROUTINES	3.6	4.1
V MANUAL OPERATIONS (NON-SOFTWARE)	1.0	0.0

TABLE 5-9 MODIFICATIONS BY MAJOR MODIFICATION CATEGORY

MAJOR MODIFICATION CATEGORY	NUMBER OF MODIFICATIONS	PERCENT OF TOTAL
A COMPUTATIONAL ERRORS	541	4.6
B LOGIC ERRORS	2217	18.9
C I/O ERRORS	287	2.4
D DATA HANDLING ERRORS	745	6.4
E OPERATING SYSTEM/SYSTEM SUPPORT SOFTWARE ERRORS	14	0.1
F CONFIGURATION ERRORS	1122	9.6
G ROUTINE/ROUTINE INTERFACE ERRORS	760	6.5
H ROUTINE/SYSTEM SOFTWARE INTERFACE ERRORS	683	5.8
I TAPE PROCESSING INTERFACE ERRORS	0	0.0
J USER INTERFACE ERRORS	42	0.4
K DATA BASE INTERFACE ERRORS	79	0.7
L USER REQUESTED CHANGES	780	6.7
M PRESET DATA BASE ERRORS	355	3.0
N GLOBAL VARIABLE/COMPOOL DEFINITION ERRORS	851	7.3
P RECURRENT ERRORS	280	2.4
Q DOCUMENTATION ERRORS	727	6.2
R COMPLIANCE ERRORS	57	0.5
S REQUIREMENTS ERRORS	66	0.6
T OPERATOR ERROR	0	0.0
U QUESTIONS	0	0.0
V IN-HOUSE PROGRAM IMPROVEMENTS	2123	18.1

TABLE 5-10 MODIFICATIONS BY ROPE AND MAJOR MODIFICATION CATEGORY

MAJOR MODIFICATION CATEGORY	NUMBER OF MODIFICATIONS PER ROPE															
	07	08	09C	09L	10C	10L	11C	11L	12C	12L	13C	13L	14C	14L	15C	15L
A COMPUTATIONAL	83	155	6	175	15	53	4	15	1	9	2	2	2	9	5	5
B LOGIC	406	491	12	727	51	221	5	42	6	50	1	7	35	35	105	23
C I/O	65	102	1	81	2	9	0	1	1	3	0	1	4	0	13	4
D DATA HANDLING	115	170	9	247	46	43	4	6	7	24	2	4	8	12	33	15
E C/S SUPPORT S/W	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F CONFIGURATION	154	331	1	279	36	121	2	30	6	1	3	3	15	41	73	26
G ROUTINE I/F	132	205	3	226	15	95	1	13	3	21	0	2	5	12	23	4
H SYSTEM S/W I/F	149	232	3	151	12	75	1	8	3	1	3	0	15	4	23	3
I TAPE PROCESSING I/F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J USER INTERFACE	6	14	1	10	1	7	0	1	1	0	0	0	0	0	1	0
K DATA BASE INTERFACE	23	28	2	16	2	4	0	2	0	1	0	0	0	0	1	0
L USER REQUESTED CHG	52	134	1	208	10	106	1	49	42	2	4	10	49	41	52	19
M PRESET DATA BASE	49	118	2	77	6	33	1	10	0	6	0	0	12	12	19	10
N COMPOOL DEFINITION	175	192	2	213	34	87	2	25	10	11	3	8	22	13	43	11
P RECURRENT ERRORS	84	61	2	90	13	6	1	3	0	15	0	0	0	0	4	1
Q DOCUMENTATION	162	172	2	183	14	82	5	10	6	0	3	10	19	11	33	15
R COMPLIANCE	26	4	0	26	0	1	0	0	0	0	0	0	0	0	0	0
S REQUIREMENTS	5	32	0	17	1	10	1	0	0	0	0	0	0	0	0	0
T OPERATOR ERROR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U QUESTIONS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V IN-HOUSE IMPROVEMENT	264	506	5	599	33	187	4	49	8	28	1	2	25	20	352	40

TABLE 5-11 MODIFICATIONS BY ROPE AND MAJOR MODIFICATION CATEGORY
(PERCENT OF ROPE TOTAL)

MAJOR MODIFICATION CATEGORY	PERCENT OF MODIFICATIONS PER ROPE															
	07	08	09C	09L	10C	10L	11C	11L	12C	12L	13C	13L	14C	14L	15C	15L
A COMPUTATIONAL	4.2	5.3	11.5	5.3	5.2	4.6	12.5	5.7	1.1	5.2	9.1	4.1	0.9	4.3	0.6	2.8
B LOGIC	20.7	16.7	23.1	21.9	17.5	19.4	15.6	15.9	6.4	29.1	4.5	14.3	16.6	16.7	13.5	13.1
C I/O	3.3	3.5	1.9	2.4	0.7	0.8	0.0	0.4	1.1	1.7	0.0	2.0	1.9	0.0	1.7	2.3
D DATA HANDLING	5.9	5.8	17.3	7.4	15.8	3.8	12.5	2.3	7.4	14.0	9.1	8.2	3.8	5.7	4.2	8.5
E O/S SUPPORT S/W	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
F CONFIGURATION	7.8	11.2	1.9	8.4	12.4	10.6	6.3	11.4	6.4	0.6	13.6	6.1	7.1	19.5	9.4	14.8
G ROUTINE I/F	6.7	7.0	5.8	6.8	5.2	8.3	3.1	4.9	3.2	12.2	0.0	4.1	2.4	5.7	2.9	2.3
H SYSTEM S/W I/F	7.6	7.9	5.8	4.5	4.1	6.6	3.1	3.0	3.2	0.6	13.6	0.0	7.1	1.9	2.9	1.7
I TAPE PROCESSING I/F	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
J USER INTERFACE	0.3	0.5	1.9	0.3	0.3	0.6	0.0	0.4	1.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0
K DATA BASE INTERFACE	1.2	1.0	3.8	0.5	0.7	0.4	0.0	0.8	0.0	0.6	0.0	0.0	0.0	0.0	0.1	0.0
L USER REQUESTED CHG	2.6	4.5	1.9	6.3	3.4	9.3	3.1	18.6	44.7	1.2	18.2	20.4	23.2	19.5	6.7	10.8
M PRESET DATA BASE	2.5	4.0	3.8	2.3	2.1	2.9	3.1	3.8	0.0	3.5	0.0	0.0	5.7	5.7	2.4	5.7
N COMPOOL DEFINITION	8.9	6.5	3.8	6.4	11.7	7.6	6.3	9.5	10.6	6.4	13.6	16.3	10.4	6.2	5.5	6.3
P RECURRENT ERRORS	4.3	2.1	3.8	2.7	4.5	0.5	3.1	1.1	0.0	8.7	0.0	0.0	0.0	0.0	0.5	0.6
Q DOCUMENTATION	8.2	5.8	3.8	5.5	4.8	7.2	15.6	3.8	6.4	0.0	13.6	20.4	9.0	5.2	4.2	8.5
R COMPLIANCE	1.3	0.1	0.0	0.8	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S REQUIREMENTS	0.3	1.1	0.0	0.5	0.3	0.9	3.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
T OPERATOR ERROR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
U QUESTIONS	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
V IN-HOUSE IMPROVEMENT	13.4	17.2	9.6	18.0	11.3	16.4	12.5	18.6	8.5	16.3	4.5	4.1	11.8	9.5	45.1	22.7

TABLE 5-12 MODIFICATIONS BY PHASE AND ROPE

APOLLO ROPE	NUMBER OF MODIFICATIONS	
	DEVELOPMENT PHASE	VERIFICATION PHASE
07	1876	88
08	2890	57
09C	0	52
09L	3194	131
10C	291	0
10L	1124	16
11C	32	0
11L	148	116
12C	82	12
12L	109	63
13C	15	7
13L	23	26
14C	207	4
14L	155	55
15C	626	154
15L	140	36

TABLE 5-13 MODIFICATIONS BY REFERENCE CATEGORY

REFERENCE CATEGORY	NUMBER OF MODIFICATIONS
NO REFERENCE	10222
ASSEMBLY CONTROL BOARD	357
COLOSSUS ANOMALY	40
COMANCHE ANOMALY	43
LUMINARY ANOMALY	40
PROGRAM CHANGE NOTE	78
PROGRAM CHANGE REQUEST	949

TABLE 5-14 MODIFICATION BY ROPE AND REFERENCE CATEGORY

APOLLO ROPE	NUMBER OF MODIFICATIONS BY REFERENCE CATEGORY						
	NOREF	ACE	COL	COM	LMY	PCN	PCR
07	1963	0	0	0	0	0	1
08	2884	1	1	0	0	22	39
09C	28	0	16	0	0	1	7
09L	3286	0	0	0	4	8	27
10C	174	25	23	0	0	25	44
10L	1071	0	0	0	0	5	64
11C	8	2	0	10	0	1	11
11L	162	5	0	0	16	2	79
12C	27	10	0	8	0	1	48
12L	88	2	0	0	19	0	63
13C	4	4	0	8	0	0	6
13L	19	5	0	0	0	0	25
14C	53	26	0	6	0	4	122
14L	59	17	0	0	1	4	129
15C	337	247	0	11	0	1	184
15L	59	13	0	0	0	4	100

SECTION 6

SUMMARY AND RECOMMENDATIONS

6.1 NATURE AND QUALITY OF THE DATA

The set of data provided by this study is derived directly from the program modifications. It is a complete history of the programs over the four-year period. It includes, therefore, all changes made to the programs, including error corrections, enhancements of capability, deletions of obsolete capabilities, changes in mission requirements, optimizations of either memory or execution time, and improvements in elegance of construction.

It is not quite correct to classify these data, then, as an "error history", since a larger portion of the items were in the nature of improvements rather than corrections to outright mistakes. On the other hand, had the system been specified and built perfectly the first time, there would have been no need for any modifications at all, except for those that reflected changing mission requirements. Since the requirements imposed by changes in mission were relatively few in number (although large in number of lines of code), it is fair to say that the data set represents the response to system errors, some of which were software errors (mistakes), some of which represented a failure in specification or design, and others which represent the fact that the job was not done perfectly the first time and, therefore, could be improved.

The value of this data set to prospective analysis should be considered in the light of several factors unique to the Apollo project:

A. The real-time multi-programmed aspects of the programs:

> This led to problems in data integrity and data availability in real-time. In addition, the varying levels of demand on system resources caused problems in timing and assignment of priorities

B. The importance of the man/machine interface:

> The crew, through the DSKY, was an interactive user of the software in that approval was required at each major program step; options presented to the crew enabled selection of alternate program paths. Furthermore, many special functions could be invoked, at any time, by crew request.

C. Ultra-high reliability requirements:

This imposed a heavy programming and testing burden, especially for real-time restart ability and error checking.

D. Severe memory limitations:

This led to extensive modifications for memory optimiztion, as well as the necessity for time-sharing of erasable data locations.

E. The computer architecture:

This led to many modifications to tailor both fixed and erasable code to enable the correct addressing mode.

F. The era in which the development took place:

The strict methodologies in specification techniques, design standards, documentation standards and programming conventions were not in general use at the time.

6.1.1 RELIABILITY OF THE DATA

The initial recording of the data was done for the purposes of management control and visibility at the time; it was not anticipated that the records would be used for later statistical analysis. The format did not, therefore, always provide sufficient information for the categorization process performed in this study. Some items required reference to other material -- memos, management plans, presentation material.

A team of experienced Apollo programmers and engineers was formed to collect and categorize the data; had these specialized personnel not been available, it is doubtful that the job could have been done. Yet it is possible that biases may have been introduced due to their personal involvement with the original project.

The inherent subjectivity of the categorization process should be emphasized. Judgement was exercised by fourteen individuals in assigning categories. The risk was recognized early in the process and consultations produced a general approach to be taken by all; nevertheless, it is impossible to estimate the extent of the divergence in judgement decisions. Also, it must be recognized that inadvertent errors were

undoubtedly made; it was impossible to check for any but mechanical errors (examining the fields of machined data) due to the large volume of data.

Specific fields which may not be completely reliable include:

- reference field: there is no way of assuring that references were always cited on the modification report when they applied; it is reasonably certain, however, that the recorders included all such references when they were cited.

- functional category, modification category, and modification description fields: subjectivity, as discussed above, was a component in determining these fields.

- software development phase field: due to lack of sufficient data, the date chosen for the verification phase was estimated for the later flights (Apollo 10 through Apollo 17).

- size fields: the estimates on which the values of these fields are based are discussed in paragraph 4.2.4.

6.1.2 UNAVAILABILITY OF DATA

Certain items would have been included in the data had they been available. These include:

- the date of initiation of each error correction; the date or program revision of the discovery of each error: no information is available on when errors were found or when correction processes were begun.

- detailed information on the number of simulation runs, the termination conditions of tests, the amount of computer time necessary to isolate each error.

- the size of the modifications.

- traceability from one modification to others: only incomplete information is available to establish the effect, in causation of additional errors, of incorporating a given modification.

6.2 RECOMMENDATIONS

One of the prime goals of software research is to achieve

greater software reliability. One promising approach toward this goal is to find methods of measuring and predicting the reliability of a given software product, much in the way that hardware reliability is measured and predicted.

Another approach is to attack the reliability problem at its source, that is, to ask what kinds of errors happen in software production, to attempt to analyze their causes, and to develop tools designed to deal specifically with those aspects of software production that contribute to the creation of errors.

In order to implement these approaches, large quantities of error data must be analyzed. This means that errors must be tracked from the very beginning of software development, and further, that they must be recorded in a way that will enhance analysis of their types and causes.

A study such as this one reveals the difficulty of compiling data from sources that were originally designed for other purposes. Had more analysis of the data been included at the time it was produced, the job of compiling the data would have been merely a clerical one. As it was, specialized personnel were required to expend considerable effort in the compilation and categorization process. Further, some data that would have been useful for analysis were not available (see paragraph 6.1.2).

The first recommendation, therefore, is that reporting procedures be designed so that the material collected during development will be useful for later analysis. Specific studies should be conducted to determine just what information is relevant to error analysis and an effort to compile for several large current projects should be instituted.

In practice, however, there is extreme difficulty in attempting to maintain error records during project development. The realities of schedules and costs make it almost impossible for project engineers and programmers to devote their time to anything but the process of software production, test, and verification.

Methods must be found, therefore, that can automate the production and collection of error data without significantly impacting the schedule or the cost of the host project.

Studies should be conducted to investigate the feasability of incorporating error analysis and compilation of statistics into existing tools. Modern compilers already contain sophisticated diagnostics; the use of these as a basis for maintaining error statistics may be easily implemented. Similar existing

diagnostics contained in simulators and other testing tools could be enhanced to provide error histories as well.

Much work has already been done to prevent errors at their source. Higher order languages themselves preempt many of the errors common to assembly-coded programs; structured programming is generally recognized as a giant step forward and documentation and control techniques have been greatly improved.

Still, there is a need for a better understanding of the roots of the problem, and work toward this understanding should be founded on a sound statistical base.

The creation of the data base by the sponsor is certainly a step in this direction.

SECTION 7

LIST OF REFERENCES

1. Johnson, M.S. and D.R. Giller, "The Software Effort" MIT'S Role in Project Apollo, Volume V, The Charles Stark Draper Laboratory Report, R-700, July 1971.

2. Hall, E.C. "Computer Subsystem" MIT'S Role in Project Apollo, Volume III, The Charles Stark Draper Laboratory, Report R-700, August 1972.

3. Hand, J.A. "Project Management System Development" MIT's Role in Project Apollo, Volume I, The Charles Stark Draper Laboratory, Report R-700, October 1971.

4. Hamilton, M.H. "Data Links" Guidance System Operations Plan for Manned CM Earth Orbital and Lunar Missions using Program COLOSSUS 3, The Charles Stark Draper Labortory, Report R-700, November 1971.

5. Dunbar, J.C. "Operational Modes" Guidance System Operational Plan for Manual CM Earth Orbital and Lunar Missions using Program COLOSSUS 3, The Charles Stark Draper Laboratory, Report R-577, July 1972.

6. Thayer, T.A. et al SOFTWARE RELIABILITY STUDY, TRW Defense & Space Systems Group, Final Technical Report, AD A030-798. (16 Oct 73 - 27 Feb 76), RADC-TR-76-238, August 1976.

7. Rankin, A.R. A Model of the Cost of Software Development for the Apollo Spacecraft Computer, Master of Science Thesis, Massachussetts Institute of Technology, Cambridge, Mass., June 1972.

8. Hamilton, M.H. "23B TESTING PROCEDURES" The Charles Stark Draper Laboratory, Memorandum, July 21,1970.

9. Hamilton, M.H. "ASSEMBLY CONTROL PROCEDURES" The Charles Stark Draper Laboratory, Memorandum, July 24,1970.

10. Nevins, J. "MISCELLANEOUS PAPERS" the Charles Stark Draper Laboratory, B. DeWolf custodian, Jan 1, 1977

11. Peck, P. "TABLE OF WORDS PER PROGRAM," The Charles Stark Draper Laboratory, Memorandum, December 8, 1970.

12. Kernan, J. "AGC Words through Apollo 11," The Charles Stark Draper Laboratory, Memorandum, February 3, 1971.

13. Copps, S.L. "Apollo Software Management; an Overview,"

The Charles Stark Draper Laboratory, Slide Presentation, May 1972.

14. Hamilton, M.H. "Apollo Software Management," The Charles Stark Draper Laboratory, Slide Presentation, May 1972.

15. Hamilton, M.H. "Management of Apollo Programming" The Charles Stark Draper Laboratory, Mission Program Development Note #18, May 20,1971.

AD-A042 186 CHARLES STARK DRAPER LAB INC CAMBRIDGE MASS F/G 9/2
SOFTWARE SYSTEMS DEVELOPMENT: A CSDL PROJECT HISTORY.(U)
JUN 77 P RYE, F BAMBERGER, W OSTANEK F30602-76-C-0151
UNCLASSIFIED R-1073 RADC-TR-77-213 NL

2 OF 2
ADA042186

END
DATE FILMED
8-77

APPENDIX A

THE INTERPRETIVE LANGUAGE

The interpretive language used as one of the two languages in which to write the flight programs for the Apollo on-board computers can well be viewed as a language in which to write instructions for a virtual machine. This machine, called an interpreter, has its own instruction set, its own multi-purpose arithmetic accumulator, its own arithmetic overflow indicator, its own arithmetic argument stack (push-down list), its own two integer index registers, its two step registers (used by the TIX instruction, which manipulates an index register), its own 60 switches (boolean variables), its own program (control) counter, its own return address register, and its memory, which is essentially the same as the memory of the AGC, which hosts the interpreter. Each of the interpreter's instructions can be viewed as consisting of an operation code followed by 0, 1 or 2 operand designators.

Consider first a class of instructions whose operation code can be followed by either one or no operand designator. The operations invoked by these instructions are binary arithmetic operations, *e.g., add,* multiply, subtract and divide. Each such operation operates on two arguments. The first argument is the content of the interpreter's accumulator. The second is either explicitly designated, in which case the operation code is followed by an operand designator, or not, in which case the operand designator is missing. If the second argument is only implicitly designated, then it is the top of the interpreter's stack, and one of the side effects of the operation is the popping of the stack. If explicitly designated it can be designated statically or dynamically. If designated statically, it is designated by giving a fixed address. If designated dynamically, it is designated by giving a fixed address and by designating one of two index registers, the argument being the one at the address which results from adding to the fixed address the content of the designated index register. The result of the operation is returned to the accumulator, and the interpreter's overflow indicator is set if overflow occurred during the operation.

Consider next another class of instructions whose operation code is never followed by an operand designator. The operations invoked by these instructions are unary arithmetic operations, e.g., square, round, double, sine and complement. The operations operate on one argument, the content of the interpreter's accumulator, with the result of the operation being returned to the accumulator, and the interpreter's overflow indicator being set if overflow occurred during the operation.

Other instructions provide for testing the accumulator or the overflow indicator, transferring data between the interpreter's registers (e.g., accumulator, index registers) and memory, resetting the program counter, resetting the return address register, manipulating and testing index registers, manipulating and testing switches, manipulating the stack, and exiting and conditionally exiting from the interpreter. By an instruction by which testing is carried out is meant an instruction which conditionally resets the program counter (i.e., resets the program counter if the test succeeds).

Each of certain store instructions with N operand designators (N=1 or 2) occupies N words of consecutive storage. Any other instruction with N operand designators (N = 0, 1 or 2) requires N plus a half or N+1 words of (not necessarily consecutive) storage. Thus an instruction requires anywhere from a half to 3 words of storage. The average amount of storage required for an instruction is around one and a half words. (Of course these figures do not include the amount of storage required to house the interpreter, which is itself part of the flight program and takes up around 2150 words of storage.)

Interpretive instructions are interpreted by the interpreter only after a transfer of control to the location INTERPRET is effected by the AGC via a Transfer Control (TC) machine language instruction. The word in the location immediately following the TC instruction is the first to be interpreted after invoking (entering) the interpreter. Transfer of control from the interpreter to a designated real machine instruction is effected by the interpreter via an appropriate exit instruction.

APPENDIX B

LOG SECTIONS AND SUBROUTINES

The fields "Log Section" and "Subroutine" appear on the modification reports (Figures 4-1, page 27, and 4-2, page 28) used as the source of the data of this study. They are explained below.

Log sections, even though recognized by the assembler as program components, should not be identified as modules. (3 examples of log sections are, for subroutine PANDORA of COLOSSUS 237, P11, TPI SEARCH, P20-P25.) Log sections were created during the Apollo development as a bookkeeping convenience, to partition the software into manageable portions. In some cases, but not all, a given log section was assigned to a single group of engineers who had responsibility for maintaining it. Line numbering began anew with each log section; renumbering could be accomplished by an assembler instruction. A log section could not be separately assembled. In most cases (as can be seen from Tables 4-7, page 47 and 4-8, page 53, which are described in paragraph 4.2.5.1.1) a log section was dedicated to a particular functional category, but was not sufficient for carrying out the function. Once more than one log section was usually required to carry out a function.

Subroutines, also recognized by the assembler as program components, again should not be identified as modules. (4 examples of subroutines are, for COLOSSUS 237, KILERASE, KOOLADE, SMOOCH, PANDORA.) Subroutines were also created as a bookkeeping convenience, to partition the software into manageable but more inclusive portions. A subroutine could be separately assembled. However, a subroutine was seldom dedicated to carrying out a particular function. Instead (as can be seen from Tables 4-8, page 53, and 4-9, page 55), different portions (log sections) of a subroutine participated in carrying out different functions. It should be noted that this sense of "subroutine" is not the same as the common concept of a coded module that can be invoked by, and will return to, another module.