RADC-TR-77-168
Final Technical Report
May 1977

VIKING SOFTWARE DATA

Martin Marietta Corporation

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

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

This report has been reviewed and is approved for publication.

APPROVED: Roger W. Weber

ROGER W. WEBER
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM
Assistant Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-168	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): VIKING SOFTWARE DATA	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report 24 May 1976 - 23 May 1977
6. PERFORMING ORG. REPORT NUMBER: MCR-77-114	
7. AUTHOR(s): Nelson H. Prentiss, Jr	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0269
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Martin Marietta Corporation, P. O. Box 179, Denver CO 80201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55810277
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: May 1977
	13. NUMBER OF PAGES: 293
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: Roger W. Weber (ISIM)

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

Software Development, High Order Language, Emulation, Executive, Software Data Base, Software Standards, Software Changes, Software Management, Software Audits, Software Validation, File Management, Software Control, Computer Loading Analysis.

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

This report describes the approach used by Martin Marietta Corporation to develop the Flight, Test and Mission Operations Software Systems that were used to support scientific investigations on the surface of Mars by two Viking Landers in 1976. Overviews of each of the three software systems are included, together with descriptions of a number of techniques used to develop them. The report stresses problems encountered and their solutions and indicates the major lessons learned.

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1 13 225

EVALUATION

This report describes the software development technology and management practices employed on a large and complex system development by the Martin Marietta Corporation.

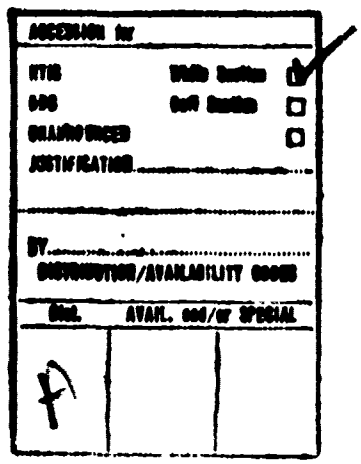
The intent of the RADC program to which this document relates, TPO V/3.4, is to describe and assess software production and management tools and methods which significantly impact the timely delivery of reliable software.

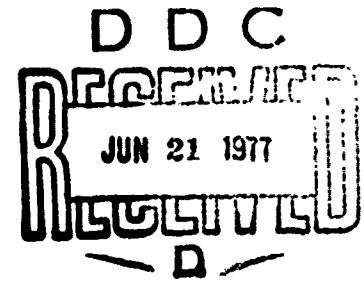
The study contract is one of a series of six, with different firms, having the similar purpose of describing a broad range of techniques which have been found beneficial.

RADC is engaged in promoting utilization of Modern Programming Technology, also called Software Engineering, especially in large complex Command and Control software development efforts.

Roger W. Weber

ROGER W. WEBER
Project Engineer





PREFACE

This document is organized to permit the reader to extract any of the three overviews or 30 techniques describing Viking software development as stand-alone papers. In addition, the first page for each technique provides the reader with a summary, applications consideration and recommendation for that technique. Thereafter, the history, description, qualitative results and quantitative impact of using the technique are described.

The preparation of this report could not have been accomplished without considerable assistance from fellow Martin Marietta employees who were or are members of the Viking Flight Team. Grateful acknowledgement is expressed to the following individuals for their support.

J. R. Anderson	J. D. Goodlette	C. A. Ourada
W. B. Anthony	K. W. Graham	C. W. Ratliff
J. A. Beacon	J. R. Herrington	D. G. Roos
R. Carney	G. E. Heyliger	J. E. Rowe
B. A. Claussen	J. R. Hill	A. R. Schallenmuller
D. L. Davidson	J. K. Kerekes	E. A. Scown
P. A. DeMartine	W. S. Lakins	P. S. Stafford
N. G. Freeman	W. J. Luckow	K. F. Thompson

TABLE OF CONTENTS

	Page
Mission Operations Software Techniques	
Mission Operations Software Development Overview	4
High Order Language Utilization	25
Different Development/Integration Sites	30
Computer Loading Prediction Analyses	37
Lander Command Simulation for On-Board Device Control	43
Program & Data Base Interface Management	48
On-Line Data File Management System	55
Integrated Software Functional Design	69
Mission Build Process	74
Cognizant Engineer/Cognizant Programmer	84
Software Data Base Document	89
Flight Operations Software Subgroup	95
Flight Software Techniques	
Flight Software Development Overview	102
Emulated On-Board Computer	116
Viking Lander Computer Executive Program	129
Hardware/Software Integration Laboratory	141
Independent Verification of On-Board Programs	148
On-Board Computer Timing and Memory Size Monitoring	150
Requirements Generator for Flight Hardware & Software	156
System Test Equipment Software Techniques	
System Test Equipment Software Development Overview	160
Test Data Base Structure	174
Viking Test Language	185

TABLE OF CONTENTS - (Continued)

	Page
Test System Simulator	198
Flight Systems Test and Checkout Program	209
Science Instrument Performance Verification	214
Viking Test Sequence Generation	219
Management and Development Support Tool Techniques	
Software Change Request/Impact Summary	222
Viking Software Standards	229
Flight Operations Software Plan	237
Software Development Management Visibility	244
Comprehensive End-to-End System Level Testing	250
Technical and Management Audits	261
Ground Data System Testing	267
Additional Topics	
Technique Relationships to Structured Programming Series	281
Recommendations for Air Force Application	285
Recommendations for Further Study and Analysis	288
Acronyms and Non-Standard Abbreviations	290

Mission Operations Software Development Overview

1.0 Introduction

- 1.1 The Operational Software System
- 1.2 Multi-Agency Responsibilities
- 1.3 Quantitative Software Description

2.0 The Requirements and Design Phase

- 2.1 Organizing for the Task
- 2.2 Defining the Software System
- 2.3 Three Mission Operations Software Systems
- 2.4 Different Development Philosophies
- 2.5 Milestones and Schedules
- 2.6 The Development Cycle
- 2.7 Additional Comments

3.0 The Test and Integration Phase

- 3.1 User Acceptance Testing
- 3.2 Unit Verification Testing
- 3.3 Lander/Orbiter Software Test Integration
- 3.4 The Mission Builds
- 3.5 Single Thread Tests
- 3.6 Sequence Tests
- 3.7 Operational Considerations

4.0 Lessons Learned

Mission Operations Software Development Overview

1.0 Introduction

The Viking Mission Operations Software System (MOSS) was developed over a three year time period. Phased deliveries of integrated software systems were needed to support test, training, launch, cruise and planetary operations. Capability was added and improved with each new system. This overview presents a brief history of the development of these systems, stressing the problems encountered and their resolutions. Each system was delivered on schedule. The overall approach taken by Viking management was an excellent one that led to the availability of a very efficient software system during planetary operations. Because of the soundness of this approach, the problems encountered during development were all minor in nature.

1.1 The Operational Software System

The Viking Mission Operations Software System (MOSS) consists of six interrelated software subsystems. They were designed to support Mission Planning, Tracking and Flight Path Analysis, Orbiter Uplink, Orbiter Downlink, Lander Uplink, and Lander Downlink activities. The system was installed in the Viking Mission Control and Computing Center (VMC^3) at the Jet Propulsion Laboratory (JPL), Pasadena, California. The VMC^3 consists of three facilities; a Mission Test Computer Facility (MTCF); a Mission Control Computer Facility (MCCF); and a General Purpose Computer Facility (GPCF).

Viking Orbiter (VO) real time telemetry software and near real time first order image processing software resided in a dedicated 1230/1219/1616 computer system in the MTCF.

Viking Lander (VL) real time telemetry software, VO and VL real time command software, and VL near real time first order image processing software were processed by a multi-mission (MVM, HELIOS, PIONEER, VIKING) real time 360/75 computer system in the MCCF. A second 360/75 computer set was supplied by the MCCF to support batch operations. The software processing functions assigned to this computer were VL command generation, power and thermal performance, system data record decommutation and decalibration, experiment data record generation, VO command generation, and Viking ground

support scheduling. The batch operation was under control of a real time operating system and lacked many features common to general purpose computers.

The GPCF provided two 1108 general purpose computers to the project. One was used for Mission Planning, science analysis and data record generation; the second was used for flight path analysis and sequence generation processing.

Second order image enhancement software was developed by the Image Processing Laboratory (IPL), a separate division of JPL. This software was not considered a part of the MOSS and was not subject to Viking MOSS Configuration Control.

The Deep Space Network (DSN) supported the command and telemetry link between the spacecraft and JPL. High speed and wide band data lines connected the lab with Deep Space Stations (DSS), where command stack and telemetry receipt software interfaced the MOSS with the ground radar portion of the communication link. This DSN software was considered an integral part of the MOSS and was subject to Viking Configuration Control.

1.2 Multi-Agency Responsibilities

The Langley Research Center (LRC) was directed by NASA to manage the Viking Project. Contracts were awarded to the Denver Division of the Martin Marietta Corporation (MMC) and Divisions 220, 430 and 910 of the Jet Propulsion Laboratory to develop the operational software system.

MMC was responsible for VL batch and Mission Planning Software, specifying requirements for VL real time software, and the integration of the six software subsystems.

Division 220 of JPL was responsible for VO batch and Tracking and Flight Path Analysis software, specifying requirements for VO real time software, and support the integration of the VO Flight Path Analysis software subsystems.

Division 910 of JPL was responsible for VMC[3] institutional software, implementing the real time VL and VO software requirements, and maintaining the integrity of the operational software system thru a process called the Mission Build.

Division 430 of JPL was responsible for the development and maintenance of the software installed at the Deep Space Stations.

1.3 Quantitative Software Description

A total of 278575 source cards were delivered to MCCC batch operations for the 22 Viking Lander software functions developed by MMC in Denver. Approximately 24000 pages of documentation was written to support these deliveries. The cost to accomplish this task was 1783 man months.

These figures account for all activities conducted by the Cognizant Engineers and Cognizant Programmers to develop the twenty-two programs from mid 1972, when the effort to write the Software Requirements Documents began, until early 1976 when the final versions of the programs used to support planetary operations were delivered.

The documentation figure includes all Functional Requirements, Software Requirements, General Design, Program Design, Users Acceptance Test Plan, and Users Guide documents developed by the CEs and CPs for the 22 programs.

The estimated effort expended by development phase is as follows:

Requirements	20%
Design	10%
Code and Debug	15%
Test and Integration	25%
Change Traffic	30%

The requirements, design and code phases cover initial program development. The test and integration phase covers certification tests at MMC, program conversion at JPL, acceptance testing, and redeliveries caused by errors detected during initial integration plus any new requirements incorporated prior to January, 1975. At that time all planetary programs had been delivered to the integration build and all launch and cruise programs were incorporated on the initial launch and cruise operational software system (MOSS 2.1). The change traffic phase represents the level of effort required to redeliver programs following MOSS 2.1 for reasons of new requirements, program errors, and performance improvements.

2.0 The Requirements and Design Phase

2.1 Organizing for the Task

A Flight Operations Working Group (FOWG) was formed and made responsible for the development of the Mission Operations Software System. Its membership was made up of the managers responsible for the development of VL, VO, VMC^3 and DSN software, and it was chaired by the LRC Project Software Manager.

The FOWG created a Software Subworking Group to manage the details of the MOSS development. Its first assignment was to document a Flight Operations Software Development Plan. The subworking group consisted of a Project Software Systems Engineer from LRC, an Integration Contractor Software Systems Engineer (ICSSE) from MMC, a Viking Orbiter Software Systems Engineer (VOSSE) from Division 220, and a Data Systems Project Engineer (DSPE) from the VMC^3. As chairman of the subworking group, the ICSSE was responsible for the coordination of inter-agency agreements and the software development plan.

The Flight Operations Software Development Plan became the controlling document for the development of the operational software system. It defined the change control procedures to be followed within and among the software developing agencies, specified program documentation requirements on a paragraph by paragraph basis, defined development, test, integration and delivery milestones that would permit the FOWG to monitor development progress, identified roles and responsibilities, and specified configuration management control procedures. It was concurred upon by each member of the FOWG and approved by the Viking Project Manager.

2.2 Defining the Software System

Software Functional Descriptions (SFD's) were written to document the purpose, description, input/output requirements, and estimates of frequency of use and computer CPU, core and mass storage resources required for each operational software system candidate program. They were used to develop an Integrated Software Functional Design (ISFD) which showed the top down design of the data flow for the six software subsystems. This task required a considerable amount of iterative effort in obtaining inter-agency coordination and agreement. Functions were combined, separated, created and discarded.

The SFD/ISFD concept proved to be an extremely useful software management tool. It provided a basis for developing schedules, planning resources, and making personnel assignments on a program by program basis. The ISFD was the baseline for program definition and interface requirements. It provided the means by which both Preliminary and Critical Design Reviews on system functions and data flow were held. The SFD's became the basis for elimination of duplicate functions and for computer loading studies, which were used to make program/computer assignments.

2.3 Three Mission Operations Software Systems

During the development of the Integrated Software Functional Design the Flight Operations Working Group formulated plans for its implementation. Resources available to support software integration, spacecraft compatibility testing, and personnel training dictated that three Mission Operational Software Systems (MOSS) would be required.

MOSS 1 would contain only those software functions required to support Data System Compatibility tests. MOSS 2 would incorporate the additional software functions required to support Data System Pathfinder Compatibility testing, Ground Data System Launch and Cruise Configuration testing, Flight Operations Launch and Cruise Configuration Personnel Test and Training, Flight Article Compatibility testing, and Flight Operations for Launch and Cruise. MOSS 3 would contain all software functions identified in the ISFD and would be used to support Ground Data System Planetary Operations Configuration Testing, Flight Operations Planetary Operations Configuration Personnel Test and Training, and Flight Operations for Planetary Operations.

2.4 Different Development Philosophies

The Jet Propulsion Lab had more than a decade of experience in developing software to support space exploration missions. Most of the software functions needed to support the Viking Orbiters were therefore obtained by modifying existing programs already operational in the GPCF. This led to a bottom up program development approach which required that the Viking Orbiter software subsystems adapt to the established conventions and procedures for using the individual programs. Subsequent computer loading studies, geared

from a cost effectiveness point of view, took into account where software already existed. As such, only one VO program, the Command simulator, was moved to the MCCF to balance the computational load of the MOSS.

The challenge to MMC to develop the Viking Lander software subsystems was significantly different. Some descent analysis, power and thermal programs had been developed on MMC computers that could be modified to support lander Flight Path Analysis and spacecraft performance functions. But the mission planning, ground resource, sequence generation, command generation, flight computer simulation, data decommutation and decalibration, and science analysis functions had to be built from scratch. The process was further complicated when the computer loading studies indicated that these software subsystems would have to be split between the MCCF and GPCF. This added the requirement that a file management program be designed to control inter-computer data transfers to prevent the overloading of available tape drive resources.

A top down approach to VL software system development was adopted. It included parameter passing and common data base file management control functions, common time utilities used by all VL programs, and required the use of unique file header records that were compatible with both the MCCF and GPCF.

Commitments by JPL to other projects limited computer resources available to the Viking Project. For this reason the MMC software was developed in Denver on non-target computers, and the VMC[3] issued a Guidelines and Constraint document that specified module size, number of tape drives and mass storage requirements for the off-site developed software.

Programs destined for the GPCF were developed on CDC 6500 series computers, and those for the MCCF on IBM 370 computers. Minimal HOL coding standards were adopted to simplify the process of converting to the target computers. Pathfinder studies were made that indicated the conversion process would not pose any serious problems.

Considerable effort was expended in an attempt to standardize interface naming conventions and header record requirements for the VO and VL software subsystems. However, because of the differences in development

philosophies, a common approach agreeable to both parties could not be found. Eventually, the VOSSE was made responsible to integrate the VO subsystems under supervision of the ICSSE, who remained responsible for the integration process. Interfaces between the two subsystems were kept at a minimum. They were individually negotiated, often with considerable compromise. However, because the VO/VL interfaces were non-standard relative to each system, they received greater attention then they might have otherwise.

2.5 Milestones and Schedules

A hierarchy of schedules were developed to provide an orderly delivery of software to the VMC[3] that would not compromise available computer resources. High level schedules provided significant milestones for upper management visibility. Lower level schedules were very important for monitoring programmers progress on coding and testing. They were very detailed.

2.6 The Development Cycle

A cognizant engineer and a cognizant programmer were assigned to each program. The cognizant engineer formalized the program requirements in a Software Requirements Document (SRD). Approval of the SRD authorized the cognizant programmer to design the basic flow for the program and write a General Design Document (GDD). After the GDD was approved, the cognizant programmer began coding and the cognizant engineer wrote a Users Acceptance Test Plan (UAT). The rationale was that the programmer would test his code during the debug stage and implement the design, whereas the engineer would assure that the program formally met the requirements specified for it in the SRD. Some software developers do not like this approach, claiming the programmer, rather than the engineer, knows best how to test the program. Nevertheless, the process adopted by Viking proved very effective. Its weaknesses were that many vague SRD's were approved because users did not understand all that was needed, which led to confusion, replanning, reprogramming, retesting and redelivery, and that some engineers failed to write UATs that fully tested the requirements. Its strengths were that it uncovered numerous misunderstandings of requirements by programmers and disclosed cases of poor program design. Observe that the weaknesses can be controlled by management, whereas the strengths are difficult to realize if the programmer testing approach is adopted.

Concurrent with the requirements/design/code phase were the development of two extremely important and useful documents. They were the Software Data Base Document (SDBD) and the Lander Orbiter Software Test Plan (L/OST Plan).

The SDBD described in exacting detail each file and parameter that would reside in mass storage accessible by Viking program software. The cognizant engineers and programmers were required to sign an agreement for all files produced or processed by their programs. This agreement indicated that they understood the file structure and data contents, and that the file was compatible with their program. The interface agreements were then approved by the ICSSE, concurred upon by the VOSSE, VLSSE and/or DSPE as appropriate, and included as part of the file descriptions. The SDBD was invaluable in locating errors and resolving interface problems; when an interface test failed the document invariably could be used to point directly at the cause. The SDBD also contained descriptions of utility programs and the Common Data Base.

The L/OST Plan specified the requirements for individually testing each interface shown in the ISFD. It included test descriptions, resources required, success criteria, and procedures. This permitted management to foresee, early in the development cycle, the facilities, personnel and data that would be required. The plan also described single thread tests for the major software subsystems that would demonstrate the data flow and indicate what procedures would be required to use the software as a system.

During the software coding time period plans as to how the software system would be integrated and implemented were finalized. MMC software would be required to pass a formal certification test in Denver prior to being taken to JPL. The test would be similar to the UAT. A Viking Lander Software Systems Engineer (VLSSE) was appointed as a member of the Software Subworking Group to monitor the certification process. The UAT's would be run from private software sources at JPL. The ICSSE was responsible for VL programs, the VOSSE for VO programs, and the DSPE for VMC^3 programs. The ICSSE would chair all Viking software post UAT reviews. Following this the programs would be subject to change control, placed on an Integration Build and unit verified by Data Systems Integration (DSI). L/OST integration

would then be performed by the ICSSE and VOSSE. DSI next would test the software to assure it's compatible with the multimission environment. At specified points in time, copies of the Integration Build would be made that became the current version of the Mission Build. Spacecraft compatibility and Ground Data System test and training could then be conducted using the Mission Build. Finally, after test and training were completed, the Mission Build would become the Mission Operations Software System.

Post UAT changes to programs caused by new requirements or software failures would be controlled by an Integration Change Control Board (ICCB). Requirements changes prior to the UAT complete milestone already were under control of the ICCB. This occurred when the SRD was approved.

2.7 Additional Comments

During the Requirements and Design phase, progress towards generating the three Mission Operations Software Systems (MOSS) proceeded relatively smoothly. Milestones were added and changed under Flight Operations working configuration control as the process unfolded. Schedules were modified to accommodate new requirements, and plans for future testing evolved as management gained insight into the system description and integration approach. A clear cut workable approach to the development cycle was formalized.

Management had some concern about the constant reworking of schedules caused by changing requirements. The software was being developed in parallel with Flight hardware and software. Changes in those areas created the need for changes to the operational software under development. The result was that the time period allotted for integration functions had to be reduced, since the delivery date for the on-line MOSS's could not be changed.

The schedule for MOSS 1 proved to be overly optimistic in that it moved scheduled software delivery dates forward by several months. This was to impact the development of MOSS 2 significantly, because it prevented programmers from developing MOSS 2 software during the MOSS 1 UAT and integration time period. The impact should have been foreseen, but it was not.

File management software should be developed before any program that will be dependent upon it is developed. This was not accomplished on Viking

because of a combination of events. When MOSS 1 was defined, the file management software delivery schedule was moved forward two months so that it would be delivered first. The Software Integration group, responsible for its development, was understaffed at the time, so that only one programmer was available to write the software. This was further complicated by the fact that the JPL computer systems were not well documented at that time. The result was that the MOSS 1 file management software was poorly designed relative to MOSS 2 software requirements, especially in the area of VO/VL interfaces. It became mandatory that a redesign effort be taken in parallel with MOSS 2 software deliveries, resulting in frequent failures during MOSS 2 interface testing. Had it not been for the fact that it was under development by an exceptionally competent and dedicated individual, serious schedule slippages would have occurred.

An item that was not worked properly during this time period was the negotiation of the structure, contents, and naming conventions of VL/VO interface files. The VOSSE and ICSSE agreed to negotiate file naming conventions and header record structures, and continued exchanging information as the individual systems developed. As matters turned out, the ICSSE did not have sufficient Project support to force compliance with agreements made with the VOSSE, and because of the divisional organization at JPL the VOSSE did not have full control of the VO software. This proved to be a mistake, since final negotiations impacted developed software in both systems.

Software requirements should have been given far more attention by middle management than they received. Items that should have been stressed more include program run time, printed output formats and quantities, and plot requirements. In addition, had this attention been extended to include critical reviews of the initial program designs, some of the design problems uncovered after program delivery may have been avoided.

A rather interesting technique was used by NASA to validate the management approach and system design. NASA gathered a committee of software experts from around the country to review and critique Viking software during this phase. The committee agreed with the overall approach, and contributed many constructive suggestions.

3.0 The Test and Integration Phase

3.1 User Acceptance Testing

Conversion of MMC developed software and VO and VL User Acceptance Testing proceeded nominally on the 1108 computers in the GPCF, but were difficult to accomplish on the 360 computers in the MCCF. The 1108 was a general purpose computer with considerable mass storage capability, which made it easy to use. In addition, turn-around time on the 1108 was reasonable.

The 360 was controlled by a real time operating system designed to be efficient for command and telemetry functions. Batch operations were restricted to 400 - 500 Kbytes core. No roll out/roll in features were available. Programs were scatter loaded. Direct access storage space was limited. The user was required to request a specific disk pack computer configuration be mounted to permit Viking software to operate.

The first programmers to bring their software to the MCCF for conversion and User Acceptance Testing began slipping their schedules almost immediately because of poor turnaround time. Two to three day delays were not uncommon. It became apparent to the ICSSE and the DSPE that Viking software, which was the first major batch software system supported by the MCCF, would require special treatment.

The resolution of the 360 computer turnaround problem was to block computers for Viking users. The ICSSE submitted 360 computer usage forecasts to the DSPE on a weekly basis. Schedules were then issued which allowed Viking users to know when computers would be available during the following week. Typically, four to six hour blocks of time were scheduled on second and third weekday shifts, and during daytime hours on weekends. When a computer was blocked for Viking, programmers could access it from peripheral equipment located in an adjacent user area. They could monitor the computer run and load status, and receive their output promptly.

The 360 blocked computer environment caused programmers to develop bad habits. They would come prepared to make as many job submittals as possible during block time, often making conflicting ones that would hang the computer.

They would overload the computer, causing the system to crash or their programs to abend. They only glanced at their output during block time, and submitted many sloppy and unnecessary runs. They overworked themselves and became inefficient. However, generous amounts of block time were made available to them, and ample time had been scheduled for conversions and UAT. As such, they were able to meet their delivery date commitments.

Reviews were held at the completion of User Acceptance Testing. The cognizant engineers were required to demonstrate that their programs had met all success criteria specified in the User Acceptance Test Plans. The UAT Review was a profit incentive milestone. Management kept a close eye on conversion and UAT progress to assure that the milestone would be met on or ahead of schedule.

Occasionally waivers had to be issued for specific subfunctions that were not included, or because the UAT demonstrated that a program violated a computer set constraint of size, run time or peripheral equipment usage. Programs delivered with waivers often were required to be scheduled for corrective redelivery at a later date.

Waivers were also required for functions that had to be tested artificially because the true environment was not available. This occurred for some early deliveries because the file management functions that accessed the common data base, the common data base itself, the time utilities, or interfacing programs were not ready.

3.2 Unit Verification Testing

As soon as the UAT review was completed, the program was delivered to DSI to be incorporated on the Viking Integration Build, where it was no longer accessible to the programmer for modification. During peak delivery periods, the Integration Build was updated weekly. The DSI was required to unit verify the program to assure that it had been correctly incorporated on the build. The unit verification test (UVT) was a subset of the UAT; the data and procedures to run it were included with program delivery.

The UVT sometimes failed because the program had not been incorporated on the integration build properly. This happened mostly with 360 operations

because of the complexity of the build decks, or that a required module was not on the build, or that a programmer had not turned all of the required build slip forms into operations. When this occurred the program, or components thereof, had to be redelivered via the same responsive but rigid change control procedure used for the initial delivery. The fix would then be incorporated with the next integration build update, and DSI would rerun the UVT. Since a month was allotted between the UAT complete milestone and the UVT complete milestone, these failures rarely jeopardized schedules.

3.3 Lander/Orbiter Software Test Integration

The initial L/OST integration was conducted following unit verification testing of the MOSS 1 programs on the integration build. It demonstrated that what had been delivered was a collection of programs that individually worked fine to perform their required functions, but could not communicate with one another to form a workable software system. This finding was to prove true for MOSS 2 programs as well.

It should be emphasized that the purpose of L/OST integration was to assure that every interface would work. Since they were being tested for the first time, it was anticipated that a large number of errors would be uncovered.

The high rate of failure detected by L/OST integration established the extreme value of the SDBD. The reasons for failures could be detected very quickly. Invariably, only minor changes to code were required to correct the situation. Had the SDBD exercise not been done as detailed and complete as it was, it is reasonable to conclude that the L/OST integration failure impact would have been major, and the Mission Build would have been compromised. In addition, the SDBD permitted management the visibility to know that when a change to a program did not affect the SDBD in any way, the change would not affect any other program in the system. Thus, the software system itself was structured by the SDBD as well as the ISFD.

The reasons for failures detected by L/OST integration were numerous: there had been misunderstandings in the file header structure, precision requirements, file access methods, fixed vs floating point data, file structures, number and types of records generated, data units, and operating

system differences between originating and receiving computer sets. Overall, eighty percent of the interface tests failed the first time they were attempted, and fifty percent of retesting uncovered new errors.

The fact that the UAT was made an incentive milestone was a contributing factor to this finding. Rarely was a schedule missed. But emphasis had been placed on unit testing. Interfaces rarely had been tested because it was not required. Scaffolds had been used to demonstrate that interfaces would work, but they had been built based on individual programmers interpretations.

The significant failure rate uncovered by L/OST integration, combined with a much greater change traffic caused by new requirements than had been anticipated, was not compatible with meeting Mission Build schedules using the UAT/Integration Build/UVT/L/OST integration development cycle. Therefore, after failure reports were issued and program corrections made, L/OST integration was performed prior to redelivering the software to DSI. This permitted corrections to be made for newly discovered interface failures prior to placing the redelivered software under rigid change control. This modification of the development cycle proved to be workable, permitting programs to be placed on the integration build that formed a usable software system.

3.4 The Mission Builds

The concept of the JPL Mission Build is probably of major importance whenever one conceives of real software control. Despite all the problems and headaches encountered during the development of Viking operational software, which extended well into the compatibility test phase, the build process permitted management to control at all times what was operational, what was to become operational, and how and when it would become operational. It allowed management to know what the operational capabilities would be before a system was delivered. This was because Mission Builds were copies of the current known status of an integration build. It guaranteed the structured integrity of the system. Finally, it provided a means for the software system to evolve and mature over the eighteen month period prior to planetary operations.

As previously mentioned, Mission Builds were copies of the Integration Build taken at scheduled points in time. Each was given a unique MOSS designation, and changes to them were not permitted. Controlled overrides to portions of programs on the build were possible, but highly visible. Such patches were permitted to be incorporated to correct local problems only when authorized by the Mission Director.

MOSS 1 consisted of a small subset of Viking software. It was not adaptive, and virtually worked with "canned" runs. MOSS 1.1 had to be scheduled to incorporate changes to the real time portion of the system. It was adequate to support the spacecraft hardware communication compatibility test for which it was designed.

MOSS 2 incorporated all of the basic Viking launch and cruise operational software functions. Management realized that the heavy change traffic caused by new requirements, plus the fact that single thread sequence and compatibility testing could uncover errors, required that several more mission builds be scheduled. Furthermore, MOSS 2 demonstrated to them that some of the programs were long running resource hogs that would require performance improvement to support the mission.

For these reasons, five mission builds were scheduled as updates to MOSS 2. Each would be designed to incorporate changes required to support specific spacecraft compatibility tests conducted in preparation for launch. The final update, MOSS 2.5, would support launch and cruise operations.

Following the first mid-course maneuver, MOSS 3 would be placed on-line to incorporate planetary software functions, to support lander spacecraft compatibility testing, Flight Team test and training, and cruise operations. Four updates to MOSS 3 were scheduled to incorporate potential, but unknown, software changes authorized by the Mission Director.

UAT, UVT and L/OST integration milestones and schedules were revised for the MOSS 2.1 - 2.5 and MOSS 3.0 builds. The development cycle now overlapped compatibility testing for launch and cruise operations.

3.5 Single Thread Tests

Originally, sequence tests had been incorporated into the L/OST Plan as the final step of the integration phase. They were intended to demonstrate that the software system could meet the mission requirements. They were extremely involved, requiring resources, personnel and data that could not be supported by the project during the period scheduled for L/OST integration. For this reason single thread subsystem tests that were designed to establish that data could be passed between and among programs in a coherent fashion were scheduled. The single thread tests generally worked the first time they were tried, and demonstrated the operability of the system using the integration build. They were conducted for uplink and FPA software scheduled for the MOSS 2 build series. Single thread tests for descent and downlink science software systems were not performed because the analytical nature of the programs required realistic data, which was not available.

3.6 Sequence Tests

The fact that the final planetary software system did not have to be placed on line until after launch permitted sequence integration tests to be conducted from the integration build, rather than from a MOSS. This proved to be a very fortunate turn of events because the tests ran into problems caused by the lack of good test data during User Acceptance Testing. Sequence tests were conducted independently for the descent and landed science software systems. Single thread uplink capabilities had been established, and all program interfaces functioned properly prior to the start of these tests.

The integration build software system was used to generate uplink commands, which were written to tape and sent to Denver to be processed by the Viking Proof Test Capsule and associated test simulators. There, realistic data was generated to simulate descent and landed science. This data was written to tape and sent to JPL to drive the downlink portions of these software systems.

The science data went thru the downlink system relatively smoothly. A new user group was conducting the test, and they experienced some delays in learning how to use the system. All programs ran as advertised, the

data flowed to the science analysis programs, which processed the interface files without error and produced page after page of wrong answers. This finding was directly attributable to vagueness in specifying software requirements and in the lack of adequate test data during User Acceptance Testing. Once again the SDBD proved its value. The science cognizant engineers turned to it to learn what their data should look like at each stage as it passed thru the software system. They were able to state precisely what errors had occurred within each program in the loop, including their own. All the errors were easy to uncover, and were minor in nature. Failure reports were written, programs were corrected and the downlink portion of the landed science sequence test was rerun prior to program redelivery. This time the answers were all correct.

The downlink portion of the descent sequence test uncovered a different problem. The lander decommutation and decalibration function could not process the data using the available computer set resources. The function had passed UAT using scaffold test data that had been good enough to demonstrate all program functions worked correctly, but not good enough to demonstrate the program would be unable to handle the quantity of data that would be required to support descent operations. It took almost a month to finally piecemeal process this descent data. The descent analysis programs were then run, and a few minor errors were uncovered. A Tiger Team was formed to resolve the decommutation and decalibration problem. The ultimate solution was to design a software procedure for allocating computer set resources that would not overload the system. Part of the redesign required a constraint waiver that permitted use of additional tape drives when data for the two Viking descents were to be processed. The fix was tested and incorporated six months before it was needed to support planetary operations.

The sequence tests were the final step in the Mission Operations Software Development process.

3.7 Operational Considerations

Users tended to get their own job done by any means available to them, and without regard to other users. They hogged resources, which often caused other programs to fail. Because of the limited resources in both the

MCCF and GPCF, and the large amounts of data expected to flow thru the system during operations, the software integration group came up with the means by which management could solve the problem. An on-line data file management system automated file control in the MCCF that controlled the use of the limited direct access storage space available in that system. Conflicting programs were not allowed to be placed in a computer at the same time. Users were required to release tape drives and direct access space as soon as possible. Files were removed from the system with regularity by the Data Processing Team. Users were disciplined to get their jobs done at prescribed times.

During cruise emphasis was placed on permitting limited software changes as approved by the Project Manager. Generally, modifications were permitted that improved program efficiency. The user engineers responded favorably and creatively in this atmosphere.

Shortly before planetary operations began, a crackdown on users was made by Management. Visibility into the controlled MOSS made it easy to detect that some unauthorized software was being run on the system. Management rationalized that engineers would very naturally view the computer as a tool by which they could make their own job easier. It was apparent that several small programs had been developed for this purpose. Each of these programs was required to be identified, and the individual users were required to explain their purpose and functions to the Mission Directors. All of the functions were worthwhile and innovated. None of them affected the integrity of the software system. Some functions were deemed necessary to support Viking operations. They were required to be delivered to DSI, unit tested, and placed on the MOSS. The remaining functions were declared to be non-essential software. Since they were useful, their users were permitted to run them during low activity, non-critical periods, or on off-line computers.

4.0 Lessons Learned

Management's requirement that the software system be placed on line ten months before launch so that it would be available for test and training was paramount in the success of planetary operations. It allowed time to uncover software errors, time to determine additional needs, time to redesign for performance improvement, time to let the system mature, and time to develop user discipline and procedures.

Programs should be developed on target computers whenever possible. When this is not possible, sound management planning for the conversion process is mandatory if costs and schedules are to be met.

The Integrated Software Functional Design is an extremely effective means to allow management to monitor and approve the overall design of a software system.

The Software Data Base Document is an invaluable aid in solving interface and integration problems.

Development of program dependent system software should precede development of program software.

Person to person communications, requirements specifications, and test data generation are the hardest things to do well. Strong emphasis should be placed in these areas early in the development cycle.

Software development can meet reasonable schedules if it is monitored and managed down to its root level.

The Software test and integration effort will be at least as costly as the software requirements, design and coding effort.

Software should be brought under configuration control at an early date and tested as a system before it is to be delivered as a system.

Management should have the ability to adapt to new situations as they arise, and be prepared to replan schedules and resources.

Rushing software development will not produce useful software.

Software programs should be tested to meet their requirements, not their code. Software systems should be tested to meet their mission objectives, not their design.

The development of any major software system requires that competent software systems engineers, who understand both the needs of the system and the individual programs that make up the system, be given firm control over the development process.

Meaningful software milestones should be defined to provide the means by which progress can be monitored.

TECHNIQUE

NAME: HIGH ORDER LANGUAGE UTILIZATION

SUMMARY: During the initial phase of the development of Viking Mission Operational software the requirement that FORTRAN be used for code was made mandatory. Waivers were granted that permitted the use of assembly language for specific functions that could not be implemented by a FORTRAN compiler. Waivers were eventually granted for some functions that permitted replacement of FORTRAN code with assembly language.

APPLICATION CONSIDERATIONS: The decision to use FORTRAN was based on several reasons. Orbiter software could be generated by modifying existing software already coded in FORTRAN. Lander software was required to be developed on non-target computers. Each computer system had assembly languages which had different instruction sets, but all computers had FORTRAN compilers. Many of the software functions were analytical in nature, making FORTRAN appear to be an ideal HOL. The level of programmer expertise required to program in FORTRAN is not too great, reducing the potential impact caused by personnel turnover. Finally, high order languages ease the task of locating errors in logic when anomolous conditions are detected.

RECOMMENDATION: The concept of using HOL for all functions that can be accomplished by it and using an appropriate assembly language for the remaining functions will produce a software system with a good basic design. In the event that the system must operate within limited computer resources, timelines, and budget, as was the case with Viking, some functions will be inefficient. These should be replaced with assembly language to get improvements as required for specific applications.

HISTORY: The basic functions required to be developed for the Viking operational software system were examined during the very early stage of Viking. None of them seemed to be complicated, and it appeared at the time that they could all be implemented reasonably well using FORTRAN. The use of that high order language was looked upon favorably by management because the program to computer assignments were unknown at the time. In addition they felt it would simplify the conversion process as well as make the software logic readable to a far wider range of individuals, thereby making management less dependent on the individual talents and personalities of programmers.

This desire to make the software more visible and less computer dependent led management to mandate that all Viking software functions would be developed in FORTRAN. The problem of permitting the use of a low order language would not be resolved until an actual situation occurred that required resolution.

The first indication that some assembly language software would be required came during the requirements definition phase for the Lander Command Simulation (LCOMSM) program. An Interpretive Computer Simulation (ICS) program was available in FORTRAN and could be modified for Viking. However, its size and run time violated Viking computer resource constraints and mission timeline requirements. Considerable effort was made to resolve this problem, but no solution could be found. As a result, an innovative scheme which required low order language development became the only viable alternative to the Viking managers. A project wide waiver system was adopted to permit functions to be developed in assembly language in the event that FORTRAN could not be used to meet Viking needs.

Eventually, the waiver was used to permit assembly language subfunctions to be developed to improve program efficiencies.

DESCRIPTION: With the exception of the Lander Command Simulation program, all software functions planned for the Viking operational system were developed in FORTRAN. After the original implementation of the software, assembly language subfunctions were required to be developed for the reasons outlined below.

The file management program that transferred interfacing data files between the UNIVAC 1108 computer and the IBM 360/75 computer required an interactive capability with the two operating systems in order to become sufficiently adaptive to changing requirements so that its code would not be impacted.

The IBM FORTRAN compiler did not provide some functions that were available on the UNIVAC compiler. This resulted in poor core utilization and unacceptably large CPU requirements. Assembly language subfunctions, similar to those available in the UNIVAC compiler, were developed for the IBM programs to resolve this problem.

A number of programs increased in size, because of new requirements, to where they violated the 65 K word maximum core restraint imposed on the UNIVAC 1108. A number of subfunctions used in common by these programs were rewritten in assembly language to conserve core, which solved the problem.

FORTRAN DATA statements, used by the Lander Sequence of Events program, were replaced with assembly language functions that took advantage of 1108 operating systems capabilities to provide output capabilities more readable to the users.

Conversion of floating point numbers transferred between the 1108 and 360/75 computer systems was accomplished using assembly language to satisfy the guidelines and constraints of core utilization.

The requirement for on-line data management functions materialized for IBM 360/75 operations. This required interactions with the operating system that could only be accomplished using assembly language code.

Finally, assembly language functions were developed for both the 1108 and 360 computer systems to improve computer run times and reduce computer resource loads. These were in the areas of dynamic core allocations, compressing the use of disk space, freeing unused core during program execution, and supporting bit manipulation.

The final software system consisted of approximately 90 percent FORTRAN and 10 percent assembly language.

QUALITATIVE RESULTS: A significant use of low order language was required to be developed and included in the Viking Mission Operations Software System. The overriding reason for this was that available computer resources could not support mission load and timeline requirements because of inefficiencies in CPU, core and mass storage caused by FORTRAN compilers. There were only a few instances in which assembly languages were used to perform functions that could not be done in FORTRAN. Given sufficient computer resources, even these functions could have been circumvented.

The quality of the software itself was more a function of programmer experience rather than whether FORTRAN or assembly language was used; some programmers were limited to using FORTRAN.

QUANTITATIVE IMPACT: The estimate of the cost impact to the Viking Project caused by replacing unacceptable FORTRAN code with assembly language is subjective and difficult to assess. The personnel who accomplished the task did so in parallel with extensive program modifications caused by program failures and new software requirements. Many of the assembly language functions needed to be developed were available in the JPL library or had been developed by the Lander Command Simulation program.

A large amount of this task was accomplished by the 360 and 1108 computer consultants, who, as members of the Software Integration Group, were funded to support just such activities. They would have been available whether or not assembly language modifications had been required. For these reasons, the cost to replace FORTRAN with assembly language probably did not exceed two man years, and may have been much less.

The cost that the project would have borne had it not permitted any assembly language to be developed is also subjective, but easy to assess. Assuming that mission timelines could have been met with the exclusive use of FORTRAN, one additional 360 and one additional 1108 would have been required to support Flight Team training and planetary operations. This is a net cost increase of fourteen computer months, which represents a 50 percent increase in computer resources over a seven month period.

TECHNIQUE

NAME: DIFFERENT DEVELOPMENT/INTEGRATION SITES

SUMMARY: The Viking Lander software subsystem of the Mission Operations Software System was developed at MMC on non-target computers. Programs were unit tested prior to being delivered to JPL to be integrated into the MOSS.

APPLICATION CONSIDERATIONS: The forecasts for the loads on the computer sets at JPL indicated that the MMC portions of the Viking operational software system could not be developed on them. The MMC facility contained CDC 6400, CDC 6500 and IBM 360/75 computer sets, whereas the JPL facility contained UNIVAC 1108 and IBM 360/75 computer sets. The operating systems of the IBM computers at the two facilities were different. No equipment similar to the JPL UNIVAC 1108 system was available in the Denver area. Pathfinder studies indicated software could be developed on non-target computers without creating any serious conversion problems.

RECOMMENDATION: Manpower, computer time, and schedule slippages can be reduced by planning, organizing and controlling software development activities in a manner which will allow for an easier conversion process. Functions developed in a minimal HOL can be converted with relative ease. Functions required to interplay with the operating system should be developed on target computers. Programmer education should be stressed. The same compilers should be used on both the development and integration computers.

HISTORY: One of the problems which faced the management of the Viking mission operations lander software development was that the development computers differed from the operational computers. The Martin Marietta facility consisted of CDC 6400 and CDC 6500 and IBM 360/75 whereas the JPL computer facility consisted of IBM 360/75s and UNIVAC 1108s. Thus the software conversion task became a management concern very early in the software development activities.

Great amounts of manpower and machine-time can be spent converting software. This coupled with the extra costs of not meeting scheduled delivery dates can cause software conversion to be an expensive task. Many times much of this manpower, machine-time, and schedule slippages could be reduced by planning, organizing and controlling the software development activities relative to the conversion process. The principle methods used in the management of the conversion process by Martin Marietta on the Viking project were: 1) predominate use of a minimal high order language; 2) education of programmers and engineers on system differences and similarities; 3) emphasis on the fact that the software would eventually reside in another computer system; 4) pathfinder operations to seek out problems which might occur prior to the actual conversion process; 5) establishment of MMC computer consultants at JPL to gain familiarity with both the various computer systems and the operational procedures; 6) the requirement that the software be demonstrated to unit function properly prior to the conversion process.

DESCRIPTION: Prior to initiating software development the MMC management examined the need for standards required to develop software on non-target computers. One output of this analysis was to require the use of FORTRAN except when individual software requirements represented an actual need for assembly programming. This decision was based on several reasons. FORTRAN compilers were available on the various computer systems at both JPL and MMC. Some of the prototype software for mission operations were already written in FORTRAN. The programming skills of the MMC engineers and programmers was limited in many cases to FORTRAN.

Once the decision to use FORTRAN was made, a study was undertaken to determine what differences existed between the various FORTRAN compilers on the UNIVAC 1108, IBM 360, and CDC 6000 series computers. This study showed that there were variations in permissible number sizes, that some features of FORTRAN were not available on all the FORTRAN compilers, and that some implementations went beyond the standard FORTRAN. A document entitled "Characteristics of FORTRAN, CDC 6000 Series, IBM System/360, UNIVAC 1108" was written which showed the minimal language that would be used to reduce conversion costs between the development computer and the integration computer. A second reason set forth for the usage of the minimal language was that for some programs the development machine at MMC had been determined but the final machine at JPL had not been decided upon. The target computer would be determined at a later date when computer loading studies could be made and analyzed.

The education of the engineers and programmers in the conversion process was next undertaken. Lectures were given which involved discussions regarding the minimal language document and its contents. The minimal language document was incorporated along with JPL documentation on the IBM 360s and the UNIVAC 1108s into a document entitled "Viking Flight Operations Programmer Guide". This was distributed to all programmers and engineers. The primary point stressed during this education process was that by following the minimum language requirements during software development the conversion efforts of the programmer would be lessened. Other points stressed during this education were that by using the guidelines set forth, along with thorough documentation, change over between programmers and engineers caused by changes in manpower would be less costly.

A pathfinder study was made that converted representative software which had been developed on the CDC 6500 using the minimal language to both the IBM 360 and UNIVAC 1108 at JPL. The time required for conversion of code was recorded and the software was then run on the UNIVAC 1108 and IBM 360 until the test cases matched the CDC 6500 runs.

The computer runs were logged and all problems were noted. The results of the pathfinder study showed that by using the minimal language documents the conversion process would be simplified. The pathfinder study also demonstrated that it was easier to convert from CDC 6500 to UNIVAC 1108 then from CDC 6500 to IBM 360. Thus all software that would eventually reside on the UNIVAC 1108s at JPL would be developed on the CDC 6500 at MMC.

The next step taken to ease the conversion process was to establish MMC computer consultant personnel on site at JPL to learn about the specific operational differences between the MMC computer systems and the JPL computer systems. This initially consisted of one person who was the focal point for all questions by MMC personnel on the JPL computer systems. The procedure for gathering and disseminating information was to funnel all questions through one individual at MMC, who then contacted the MMC computer consultant at JPL. The consultant would then be responsible for contacting various JPL individuals to gain the answers to the questions. These answers were then distributed via inter-office memos to all the MMC development programmers of mission operations software.

The computer consultant proved to be of such value that a second consultant was appointed. One now had responsibility for inquiries into the differences between the MMC IBM facility and JPL IBM facility, and the other had responsibility for inquiries between the UNIVAC facility at JPL and the CDC facility at MMC. The consultants were organizationally members of the lander software integration group.

After a mission operations software program completed certification testing at the MMC facility it was brought to JPL for conversion and user acceptance testing. The MMC consultants at JPL established the methods by which the programs were brought to JPL. This was done by generating special purpose software or by defining the actual utility programs which were to be used in generation of the tapes required for program conversion and data tapes for program testing.

The MMC consultants took an active role in the software testing at JPL, and were responsible for insuring proper delivery of the software to Data System Integration.

QUALITATIVE RESULTS: The software conversion task was done on schedule, although not without some problems. Many of the problems that did occur had to do with the altering of the engineers and programmers normal work habits. This occurred in the IBM 360 conversion effort. Due to the operational procedures at JPL, in order to receive the necessary turnaround time to do program conversion and testing, blocks of computer time had to be scheduled. These blocks of time were usually 4 to 6 hours in duration during the week starting at 9:00 PM to 4:00 AM, and up to 48 hours duration on the weekends. The pressures of trying to make as many runs as possible and meet delivery schedules forced many extra errors into the software. Instead of doing a detailed analysis of the code and the dump of the program the programmer or engineer would shot-gun many runs to try to fix errors. During longer block-time stretches many engineers and programmers would work until they introduced new errors due to physical and mental exhaustion. This altering of normal work habits did not occur on the 1108 conversion effort. The turnaround was excellent and the machine was available for use during normal working hours. Blocks of time were only required in very special cases and were planned by the engineer and programmer so that the use of the time was optimized. In short, the conversion process requires good computer turnaround during normal working hours to be efficient.

The MMC IBM 370 to JPL IBM 360 conversion effort had some problems than were not foreseen. The JPL version of OS was a real-time system with standard OS features but was a different release than the MMCs IBM OS. This caused problems in the conversion effort. In one instance the difference in FORTRAN compilers between MMC IBM and JPL IBM caused schedule slippages due to errors in the target compiler that were not discovered until the conversion process began. This problem happened early in the conversion process and two steps were

taken to help solve the problems: 1) the release of the compiler JPL had was installed at MMC for use by the software still in development and 2) the release of the MMC compiler was installed at JPL for use when needed.

There were technical problems as well. An example on the 1108 was that the lander trajectory simulation program (LATS) had a subroutine which contained a namelist for input which was very large. This large namelist coupled with a large amount of equivalent statements caused an overflow in the internal tables the compiler used. Thus the routine was not compiled correctly. In order to solve this problem, a special version of the compiler with larger tables was generated strictly for use by this subroutine. Eventually the namelist was split into many namelists which then allowed the standard JPL compiler to process the subroutine.

There was one other problem which occurred that should be reported. This was an internal conversion effort done at MMC on the CDC. The operating system was changed from MACE to SCOPE during the period in which the software was being developed. This change caused many schedule impacts in the Viking software development. Much time was spent by the development programmers changing their job control cards, their file naming conventions, their file structures, and in learning how the system worked. This coupled with extra down time and running of two operating systems caused much confusion. The conclusion is that any operating system change should not be allowed concurrent with a major software development activity.

The software should have been maintained at JPL after its initial delivery to JPL. Deliveries of the software were scheduled to include various programs and sub-sets of the programs required for test and training operations which were carried on throughout the spacecraft compatibility test phase. When a program was delivered to support such a test the programmers and engineers would take the final converted

program back to MMC, convert back to the MMC computer, and then do the development required for the next delivery. This was an extra task that should never be required.

It should be stressed that by following the concepts discussed herein, the conversion of the MMC developed Viking mission operations software was done on schedule within budget, and successfully supported two Viking missions.

QUANTITATIVE IMPACT: Total manpower for cognizant engineers and programmer activities involved in developing, testing and documenting Viking lander operational software produced an average of 7 lines of code per day. Individual programs varied from a low of 3.3 lines of code per day to a high of 12.4 lines of code per day. These figures are reasonably within the industry for software developed on target computers, indicating the impact of conversion was relatively low. One program was delivered two months late because of differences in compilers, which was a conversion problem. All other programs were converted on schedule. The conversion process represented approximately five percent of the development effort for 1108 programs, and approximately ten percent of the development effort for 360 programs.

TECHNIQUE

NAME: COMPUTER LOADING PREDICTION ANALYSES

SUMMARY: Two types of computer loading prediction analyses were conducted for the Viking Mission. They included hand analyses of predicted CPU requirements and use of the General Purpose Simulation System (GPSS) program. Both indicated that the planned Mission timelines could not be met. The Viking Flight Team (VFT) conducted a full scale test that verified the findings of the loading analyses. The key decisions resulting from these efforts included changing the Mission Design from two parallel to two serial landed missions, acquiring an additional 1108 computer set to support operations, and decreasing the planned frequency for commanding the vehicles.

APPLICATION CONSIDERATIONS: Operations analyses of timelines and computer loading analyses are applicable to software systems supporting missions subject to environmental surprises, such as Viking. On the other hand if the software system is being developed for use in a steady-state environment a computer loading analyses made for any point in time should be sufficient. Two techniques for accomplishing the loading analyses are available. One approach is to use a complex computer program such as GPSS (available from either UNIVAC or IBM) to model the throughput of each computer used by the system. The second approach is to perform hand analyses of requirements versus capabilities for basic parameters such as CPU time.

RECOMMENDATION: Computer loading analyses are a proven useful tool to software managers. Two areas should be concentrated upon independent of the technique selected; 1) the quality of the inputs will determine the accuracy of the output, and 2) careful interpretation of the output is required if correct conclusions are to be drawn.

HISTORY: All computer loading analyses performed for Viking were accomplished in conjunction with operations analyses of mission timelines. Three basic stages to the process evolved that were not apriori planned. These were operations analysis, critical period analysis, and VFT Test and Training.

During 1973 management was developing the Viking Mission design. The key factors of concern were the Mars encounter dates, the landing dates, and the frequency and duration of the biological investigations. At that time the planned nominal sequence was as follows:

1. Viking I encounters Mars
2. VL-1 lands 12 days later
3. VL-1 initiates biological investigations 4 days later
4. Viking II encounters Mars 8 days later
5. VL-2 lands 12 days later
6. VL-2 initiates biological investigations 4 days later
7. VL-1 terminates biological investigations 60 days later
8. VL-2 terminates biological investigations 24 days later

The prime Viking landed missions were each planned to have 90 day durations, with about 65 days of overlap. Shortly after the mission was established the Integrating Contractor Software System Engineer made a hand computer loading analysis based on Mariner 9 experience and proposed seven day look-ahead for lander planning. The analysis indicated the computer complex at JPL would not be adequate for Viking.

Management therefore directed that an analysis of a two day mission period should be conducted. The fifth and sixth days after the Viking II landing were selected. Representatives from each Viking Flight Team defined the products they would produce, the inputs they would require and the computer runs they would need. After this data was gathered a cursory time line/computer loading analysis was made that indicated that neither the VFT nor the computers could support the events planned for these days; people were both waiting to use computers and not ready to use them when they were available. As a result of this analysis the Mission Design was changed to the following sequence:

1. Viking I encounters Mars
2. VL-1 lands 19 days later
3. VL-1 initiates biological investigation 7 days later
4. Viking II encounters Mars 21 days later
5. VL-1 terminates biological investigation 27 days later
6. VL-2 lands 3 days later
7. VL-2 initiates biological investigation 7 days later

It is emphasized that the decision to slow down the pace and go to a serial mission could have been made from either an operations analysis of people overload or a computer loading analysis that demonstrated computer overload. Furthermore, the decision to go to a serial mission was made when the software system was less than half finished, so that only rough estimates of the characteristics of the programs were known.

At this time the Viking Project issued a separate contract to develop a General Purpose Simulation System Model (GPSSM) for the Mission Control and Computing Center (MCCC) system. This was a joint effort by MMC and JPL personnel, using the IBM GPSS as the basis for the model. A Critical Period Analysis covering a 12 day period of the serial mission was conducted in parallel with the GPSSM development. The purpose of the analysis was to study timelines and computer loading in more detail and to prepare for VFT test and training activities. By this time more software was developed allowing better inputs for the computer loading analysis.

Because the GPSSM had not yet been verified, a hand analysis was again made. The results of the analysis indicated that more UNIVAC 1108 capability would be needed to avoid another mission simplification. Since lead time for purchasing an 1108 prior to planetary operations was becoming tight, the time saved by hand analysis was quite beneficial.

At this time the Viking Project and JPL faced the choice of purchasing or leasing a third 1108 or of adding a second Central Processing Unit to each of the two existing 1108s. In order to make this decision, a full scale test involving the VFT was scheduled to be

conducted during the three day Memorial weekend in May of 1975. Prior to the test both the GPSSM and hand analyses were used to predict the outcome of the test. The test confirmed both analyses, and was the tool that verified the accuracy of the GPSS model. The results of the test led to the decision to purchase a third 1108.

It was not that Viking needed three 1108s but rather that Viking plus the non-Viking JPL users would require that capacity. The decision to add a third 1108 rather than increase the CPU capability was based on the fact that it would result in less sharing between Viking and non-Viking users.

Early in 1976 VFT test and training exercises were conducted in a series of tests using the simplified Mission design, the extra 1108, and modified operational strategies based on the results of computer loading analyses. The tests demonstrated that the system could operate to support the mission timelines.

DESCRIPTION: The processes used to conduct the computer loading analyses were as follows.

Programmers estimated program run characteristics, such as CPU time, I/O time, core usage, disk space, and tape drives. Each program could be used for different tasks having different characteristics. Data was gathered for every case.

Engineers then estimated when, how and why they would use the programs in each mode. Each team would indicate what time or times during the day that each program mode would need to be run in order to satisfy mission objectives. Estimates of how long it would take to analyze the results of a computer run before the next run could be made were also included.

The hand computer analyses were then based on adding up the CPU time required for each computer during each team's shift and matching it with the inputs supplied by the programmers and engineers. A subjective judgement factor then had to be applied. If during any eight hour period, CPU requirements in excess of 4 hours for the 360/75 or 5 hours for the 1108 indicated insufficient computer capacity was available. The early analyses indicated 8 hours CPU time on the 1108 would be required, stressing the need for an additional computer.

The GPSS modeled all of the inputs obtained from the engineers and programmers. The program modeled both 1108 and 360/75 operations. The characteristics included core capacity, CPU rate, I/O rate, tape drives, and delays caused by humans examining computer runs. The model would not allow runs to begin when core or tape drives were unavailable, nor would it allow a program to begin if it required input from another program that had not completed. As such, the model did not overload the computers. Rather, it would indicate that one day's work required two days to complete if insufficient computer power was available.

The GPSS modeled the scatter load feature of the IBM 360/75, but did not model the core swapping feature of the UNIVAC 1108. It accounted for print and plot times, but not for system outages. The model is now a standard tool used by JPL as a scheduling tool and for the purpose of estimating potential system performance improvement.

QUALITATIVE RESULTS: Sufficient evidence for the potential usefulness of computer loading analyses to managers of software development has been presented. A few important aspects are worth noting.

The primary error source of the analysis is the quality of the input. Since the quality of the output is limited by the quality of the input, the cheaper and faster hand analysis offers advantages over a complex simulation model, which would have to be validated before it could be relied upon. In cases where most runs can be executed overnight, one day granularity is sufficient to achieve an understanding of the problem. In a mission operations environment where a sequence of runs needs to be accomplished during a working day shift, a granularity of 4 or 8 hours should be used.

Most members of the VFT felt that the hand analyses were adequate and the GPSS was an expensive luxury, although a significant number of team members did not hold this point of view. The use of the GPSS did have two generally agreed upon side benefits. People generating inputs for analysis tended to be more careful and thorough when they knew their data would be used in GPSS rather than for a hand analysis.

Also, management was more impressed with GPSS results and paid more attention to the implications, regardless of whether or not they understood how the GPSS worked.

The Memorial Day test by the VFT to verify the GPSS provided additional guidance concerning operational procedures and file management problems. These areas were ignored by the hand analyses and treated inadequately by GPSS.

One final point should be made relative to the computer loading analyses. They only reflected how the VFT thought the mission would be run. They did not take into account the affect that the resolution of the 360/75 file management problem had on changing the way operations there were conducted (refer to the on-line data file management system technique). They did not account for the 16 day delay in finding a satisfactory landing site for Viking I, nor for the additional Mission Planning activities for both Vikings I and II (neither Viking was set down at its originally planned landing site). They did not account for maintaining a reduced Viking I mission during the primary Viking II mission. Finally, they did not account for the data rate changes resulting from the scientists analyzing data and then trying experiments that were not originally planned. Their primary value was to determine the level of activity that could be supported, which allowed management to realistically assess how much could be accomplished.

QUANTITATIVE IMPACT: Since the computer loading analyses were conducted in conjunction with operations timeline study analyses, it is difficult ot determine their specific costs. Probably one man year was spent by the VFT to generate inputs. Less than one man year was spent on hand analyses. About three man years were required to develop the GPSS model. An additional one to two man years were required to maintain the model.

Approximately 200 members of the VFT supported the three day Memorial weekend tests used to validate the GPSS. Had it not been for the computer loading studies, that test probably would not have been scheduled.

TECHNIQUE

NAME: LANDER COMMAND SIMULATION (LCOMSM) FOR ON-BOARD DEVICE CONTROL

SUMMARY: Simulation of Lander on-board commanding and computation was required at the bit-for-bit level. An innovative technique permitted such fidelity without the ponderously slow Interpretive Computer Simulation (ICS) technique ordinarily employed in such situations.

APPLICATION CONSIDERATIONS: Several proven approaches exist for on-board flight computer simulations. Emulation, at real time speeds, usually requires specialized hardware and may be embedded in a "hot-bench" testing facility. The ICS approach is proven and popular, but significant resources are expended in their use. Reasons for poor performance are several, with typical expansion ratios of 20 to 100 times real time. LCOMSM achieved significant improvement through two mechanisms: 1) taking advantage of pseudo real time, i.e. segments where there is no activity are skipped over during the simulation, and 2) the high overhead due to execution time interpretation is avoided by performing a translation time interpretation of static source code, and substituting an equivalent sequence of simulation computer instructions for each target computer operation. The latter sequence is carefully tailored to represent the bit for bit result of on-board computer operations. The improvement in simulation run times are significant.

RECOMMENDATION: An ICS has become a tool traditionally supplied by the computer vendor. Typically written in FORTRAN to meet portability requirements, an inherently slow process becomes more cumbersome. The LCOMSM approach is attractive where many hours of simulated time is anticipated. This class of application warrants the special tailoring required to substitute simulation computer code sequences for each object computer Op-Code. The same level of execution visibility can be attained with any approach; trace, conditional snapshots, conditional halts, and the like. A single host monitor could provide the necessary common services for a variety of distinct target computers.

HISTORY: Initially, a FORTRAN Interpretive Computer Simulation (ICS) was considered to operationally simulate the Viking Lander Flight Program. An ICS was available to the Viking Project that could be used for this purpose. Analysis indicated that thirty hours of computer time would be required to simulate one landed day's activities. Modifications to the ICS scheme could be made to reduce the computer run time to an estimated three to five hours. Mission operation timelines required that the simulation not exceed thirty minutes, with a goal of less than twenty minutes. Therefore, the Viking Project's digital simulation program for the Mars lander computer became an attempt to solve two resource problems inherent with interpretive computer simulations. These problems were the resources required to develop a new simulation and the computer resources used by the simulation operationally.

The problem of computer resource consumption was addressed by first designing the LCOMSM program to minimize the work required to simulate a Viking lander computer. The approach taken was to place the object code of the Flight Program into the IBM 360/75 computer. Two methods to accomplish this were considered, a source code instruction translation process and an interpretive process.

The source code instruction translation process could be automated by subprograms called Instruction Translation Macros, one for each of the Flight Computer instruction set. The Flight program would be translated into source code which, when assembled into object code and executed on the IBM machine, exactly would simulate the Guidance, Control and Sequencing Computer (GCSC) processing of the original Flight program.

As a consequence of the source code expansion effect of the translation process, the simulation object code core requirements are large. However, having done the translation and assembly, the actual execution of the object code would be very fast. Since the code resident in core is a translated version of the Flight Program, a simulated GCSC memory map would not be immediately available. Rather, the contents of IBM core would have to be mapped, by address, to the contents of the simulated GCSC core. This approach imposed a requirement on the original source code. Namely, the source code had to define a monotonic mapping, by address, of object code into core. Unfortunately, the

existence of unusable GCSC memory locations as well as programming techniques utilized in the Flight Program made it impossible to meet this requirement. Consequently, the translative simulation approach had to be discarded even though it offered the fastest operational simulation.

DESCRIPTION: An interpretive simulation of the GCSC was used by LCOMSM. The design approach was to assemble the Flight Program source code directly into object code, load it into core on the IBM machine just as on the GCSC (allowing for differences in word lengths), and simulate the GCSC processing of the identical object code in the course of execution on the IBM. The GCSC was simulated by interpreting each object code instruction in terms of the GCSC's response to it, as would have been done had the translative approach been taken. The major difference was that, rather than solving the problem once and for all at the source code level before execution begins, the interpretive process must be performed for each instruction as it is encountered in the course of execution. Therefore, the interpretive approach required more CPU time for a given simulation. However, since the source code was not translated prior to assembly, the core requirements were smaller in the interpretive approach. Most significantly, since the object code was an exact representation of the actual GCSC load, a memory map of the GCSC being simulated was immediately available at any desired simulation time with a simple readout of the IBM core.

The development of the interpretive simulation for a real time process was a difficult task, since it was hard to realize if a real time process was being simulated correctly and even harder to know why it was not. For this reason a major emphasis was placed on providing as much visibility as possible of what the simulation was doing. The program was designed to produce simulated program execution traces for both high and low level processing. As soon as the program was ready for testing it incorporated a full instruction and processor state trace. Because of the volume of data this trace produced, a post processor was developed to allow scans of the data in several different modes. In addition, the capability was introduced to trace the functions

of the simulated lander executive for both Input/Output and scheduling. This allowed a higher level of program execution visibility, which meant that much less output was produced during a given run. The program was designed to allow for tracing of both selected data cell usage or instruction execution to facilitate finding incorrect simulation or invalid simulated program loads. To verify simulator accuracy, the program was designed to compare the simulated processor commands issued with the set defined by a functional simulation of the same period.

In order to minimize the work required, the LCOMSM program design included the mapping of several of the hardware capabilities of the GCSC into the hardware capabilities of the IBM 360. This consisted, in part, of the mapping of several GCSC registers to IBM 360 general purpose registers. In addition, a unique timekeeping system was devised which eliminated most of the usual processor time associated with the task.

The program design was enhanced to include the concept of Dynamic Algorithm Replacement (DART). As incorporated in LCOMSM, the simulator could identify, during a run, that the simulated program was doing some algorithm for which a replacement existed. The replacement would do precisely the same thing as the simulated algorithm with the exception that the instructions did not have to be individually interpreted. Instead, a host computer code body was executed with the subsequent savings of processor time.

QUALITATIVE RESULTS: The LCOMSM program was able to model elapsed times for 90 day simulation runs without error. The program was developed in less than 30% of the original development estimate. The simulation provided a time compression slightly greater than 200 to 1, which should be compared with the 8 to 1 time compression estimated to be the best that could be obtained by modifying an available FORTRAN ICS program.

QUANTITATIVE IMPACT: The operational impact of the LCOMSM design can best be judged by comparing its computer resource requirements with the Orbiter Computer Simulation Program (OCOMSM), which used a FORTRAN Interpretive Computer Simulation. The Lander Flight Pregram was four

times larger than the Orbiter Flight Program. LCOMSM required 120 Kbytes of Main Core and 138 Kbytes of Large Capacity Storage (LCS), whereas OCOMSM required 183 Kbytes of Main Core and 287 Kbytes of LCS. Any Viking 360 batch program, except OCOMSM, could be run in the computer while LCOMSM was executing. No other Viking batch program could be run in the computer while OCOMSM was executing. This was important due to the throughput characteristics of the two programs. LCOMSM would require approximately 5-1/2 minutes of CPU time to simulate one landed days operations and throughput the job in 6-1/2 minutes. OCOMSM would require 8-1/2 minutes of CPU to simulate one synchronous orbits operation, but the throughput time for the job took approximately 70 minutes. The systems impact of this was that about half of the Viking batch IBM 360/75 computer time had to be dedicated to OCOMSM exclusively. This proved to be a continuing cause for delays in obtaining output for any other batch job. In addition, because LCOMSM throughput time was relatively good, the program allowed for job resubmittals without causing a major impact to normal operations.

The final version of LCOMSM used for planetary operations contained 11535 source cards. The cost to develop the program from requirements through delivery of the final version used for planetary operations was 87 man months, or approximately 70 minutes per source card.

TECHNIQUE

NAME: PROGRAM & DATA BASE INTERFACE MANAGEMENT

SUMMARY: A common data base was used by all Viking Lander operational programs to access critical tables and constants, such as flight computer turn on times, lander coordinates, and length of a Martian day. Inter computer file transfer software permitted user files to be readily available on any computer system, transparent to the users.

APPLICATION CONSIDERATIONS: Viking Lander operational software was required to be developed from scratch rather than by modifying an existing system. Mission planning, sequence generation, flight path analysis, spacecraft health and science analysis programs used significant amounts of common tables and constants. These programs operated on two different computer sets and required large amounts of interface data to function correctly. Using tape drives to transfer data between programs during operations would have compromised mission timelines. Coordination of large amounts of data separately used by programs is subject to considerable human error. For these reasons inter-computer transfer and common data base file management software was developed and used by these programs.

RECOMMENDATION: A single source for accessing critical data subject to a low rate of change can be a useful tool in reducing chances of human error. Care should be taken to coordinate data values among all users. A single source for transferring data between computers is attractive when large amounts of data are to be transferred and computer tape drive resources are limited.

HISTORY: At least as early as March 1972, it was realized that the Viking Project would have some unique data base problems. One of the data bases identified as being desirable was the Flight Operations Data Base (the data used and generated by the Flight Operations software). One segment of the FOS data base was named the Common Data Base. This data base would contain data items used by more than one program (but in some cases, data used by only one program). The types of data eligible for admission to this data base were tables and constants whose values were not expected to change more than a few times during the mission. This data base was meant to replace data normally compiled within the program as DATA statements or as data input into the program without change each time the program was executed. Having a system of programs operating from a common data base offered many attractive features. It forced consistency of data among the various application programs. That is, all programs used identical values for tables and constants, such as epochs and clock drift tables. A common data base allowed updating of constants to take place simultaneously among all using programs. Data base management procedures allowed control and documentation of values used, and change history information. Operational responsibility for data availability resided with a central data base manager rather than being divided among the operational groups.

Another segment of the Viking data base was the management of inter-program data files. Viking file management was defined to be an automated system which would checkpoint files and transfer files between machines. The checkpointing activity would provide file security and load/off-load on-line mass storage space. File transferral was complicated by the usage of IBM 360 and UNIVAC 1108 computers for the Viking Flight Operation Software System. Programs on one system required files created on the other.

Early attempts at defining methods and operation of a general file checkpoint system were frustrated by lack of agreement among the affected user groups and lack of definition of system usage. As a result, several different methods of checkpointing were eventually used for Viking. The file transfer scheme used 7-track magnetic tape as a universal transfer medium. An electrical interface was established between one computer pair at a time (IBM-UNIVAC).

The file transfer software concept fluctuated wildly during the early systems integration time period. The detailed techniques for translation and the extensive Input/Output requirements necessitate thorough familiarization with the machines and operating systems. This was initially a problem since the early software development was done in Denver without easy access to the machines that were to use the system. Since file manipulation is dependent on file structure, either great flexibility is required or total definition of file structure must be available. Total definition is impossible until the file generating software is totally defined. For Viking the file transfer algorithms were designed to maximize flexibility and generalize. The design goal was to have greater capacity than required at the time, but excess capacity was continually used up as the systems integration activities progressed. The decision to perform data translation by the file transfer software was contested early in the preliminary design. The alternatives are to design the data files in such a way that translation requirements are reduced (i.e. transfer files in external BCD); to have the programs which use transferred files embed translation within their own structure; or to have specific file translators attached as pre-processors to the using programs.

DESCRIPTION. The Common Data Base on the IBM 360 system used for Viking was based on ISAM (Indexed Sequential Access Method). Subroutines were written which allowed read only access of data base records by name. The data base could be read directly or sequentially: the subroutines were linked into the application programs. The using programs and the data base were totally insulated from each other. Several programs were written to permit the basic necessary management functions for data base operations. These functions were: load, modify, list, and reorganization. The actual data for the data base was collected, punched into cards and loaded. An access method, similar to ISAM, was developed for the UNIVAC 1108.

Inter-machine file transfer was enacted by pairs of programs. A program on the source machine copies a file to tape for the tape transfer, or enques files to a real time communication program for electrical

interface file transfer. A program on the destination computer read the tape and/or electrical interface communication line and placed the files on regular on-line mass storage. Files were translated to the word and record format of the receiving computer when necessary. Tables describing the structure of files were placed onto the common data base to supply specific translation details. Two types of tables were used, transfer control blocks and translation control blocks. Transfer control blocks, TRCBs, contain basic input file information such as file format, output file information such as record size, file space allocation parameters and flags indicating cataloging/allocation techniques, and for non-character data, a string of symbolic addresses of translation control blocks, TRBs. TRBs are a linked set of tables describing specific translation details and translation criteria. Translation details are ordered pairs of numbers of words and translation technique to be used. Translation techniques are integer to integer, character to character, single precision floating point to single or double precision, straight binary transfer, et al. Translation criterion is a data condition and one or two addresses of TRBs that are branched to if the condition is met. Unformatted files, when transferred between unlike computers, must be accompanied with detailed information about record structure. These details are contained in TRBs and reside in the Common Data Base. The symbolic addresses of the TRBs are contained in the TRCB for the file being received. Each TRB contains triggering information and translation information. Triggering is the branching process that allows different TRBs to be enacted. At the beginning of file reception, TRB number 1 is given control. Upon the detection of a specified condition, another TRB will be given control. The condition can be either the length of the record being translated or the contents of any single data word contained within the record. TRB branching can be on the basis of the condition being true or untrue. That is, triggering to a new TRB can be based on a record length being equal to some quantity or not equal to that quantity. On the IBM receiving program, only the first 32 bits of a data item participate in the compare. Should the trigger condition be met, the reference number of the TRB

to be branched to is located in the old TRB. A new TRB number outside the range of one to fifteen will terminate the translation and set the error flag within the receiving program. This can be used to detect anomalous conditions. The time of enactment of new TRBs is variable. If the enactment time field contains the character string 'ELSE' or 'PRES', triggering will take place after the current record is translated by the present TRB pattern. For 'PRES' and 'PØST', triggering takes place only when the trigger condition is met. For 'ELSE' triggering always takes place. When the trigger condition is met, one TRB is activated; when the trigger condition is not met, a different TRB is activated. The purpose of the 'ELSE' trigger is to provide a double branch capability. The purpose of the 'PRES' and 'POST' triggers is to use a single TRB translation pattern to process a string of identically structured records and then branch to a new TRB when the string is broken by a differently structured record, or a record containing different data.

Transfer control blocks are keyed to the receiving program by information contained in the first data record. As each file was generated, a two record header was created. The first record contained (at least) a five character generic file name. This generic file name is the symbolic key for access to the common data base. In this way, file transfer is dependent only upon data content. Two other methods of synchronizing received files with transfer control blocks are: providing manual inputs to the receive program or using a stringent file naming convention. Manual inputs are prone to error; file naming conventions are difficult to adhere to rigidly, although file name method of keying TRCBs eludes the manual input problems and places no restriction on data file format. It also allows transfer of files which were not originally expected to be transferred. However, the independence of file content and file name was judged to be the overriding concern, and seems to be more in keeping with the concept of utility routines. Since file name syntax differs between systems, the file names were changed by the receiving program to the syntax of the receiving operating system.

QUALITATIVE RESULTS: There exist lessons to be learned from development and usage of these data base and file transfer schemes. Since the data base is essential to proper development of application programs, data base software and data base techniques should be thoroughly debugged and tested before the application program development is started. This was done in parallel with application program development for Viking and resulted in an initial lack of confidence by the application programmers. The confidence gap manifested itself in somewhat poor usage of data base concepts and in invariably assuming it to be responsible for application software anomalies. Gathering the data for insertion into the data base is a monstrous task and requires great cooperation from the suppliers. Data quality and responsibility should be assigned to cognizant groups of people for ongoing maintenance and control. However, all responsibility for manipulation of the physical data base should reside within a single authority.

Changes to data during operations on a common data base must be performed with great care so that all using groups know ahead of time the details of the change. In general, agreement of all using groups should be obtained before changes are done.

During the Viking mission, the data base software and technique worked well in practice and conceptually, and no modifications of scheme or concept seem to be errant.

QUANTITATIVE IMPACT: The Viking data base access method software and data base utility software required about four manmonths of programming and checkout effort for both the IBM and UNIVAC systems. The file transfer software development cost was about one man year. The total effort for the software function of which these were the major components produced 7923 operational source cards for a cost of 57 man months. These figures include the development of an experiment data record generation capability and reflect all costs from requirements development through final delivery of the program version used for planetary operations.

The use of a Common Data Base had a negative impact during development, test and training. Approximately one man month was lost by users believing software errors were the cause of test repeatability failures, whereas the actual causes were ultimately traced to changes to the common data base. No impact of this type was observed during operations.

Use of the file transfer software reduced the tape drive allocation requirements during operations by approximately 30 percent for the entire Viking software system.

TECHNIQUE

NAME: ON-LINE DATA FILE MANAGEMENT SYSTEM

SUMMARY: The on-line data file management system (OLDFMS) was designed to assist in the management of data sets suited for residence on direct access devices in the IBM 360/75 system configuration for Viking batch operations. The system was developed after determining that the proposed manual procedure for handling files on the 360/75 batch computer could not adequately accommodate the anticipated 200-400 daily data management activities.

APPLICATION CONSIDERATIONS: The volume of data forecast to be processed daily on the 360/75 computer necessitated the development of a process to perform data management functions with minimum human intervention. Direct access space (DAS) was limited to approximately 12000 tracks for non-temporary storage of lander files, which were expected to require a minimum of 9000 new tracks daily. Thus data set creation could not exceed file removal, which in turn could not be performed until adequate checkpoint was completed. This translated into 200-400 actions required daily to create, checkpoint, remove, and restore mission files. Maintenance of good records indicating which data sets were ready for removal and checkpointing was mandatory to allow DAS space to be made available. Tracking a data set's location also required accurate records.

RECOMMENDATION: The on-line data file management system is perhaps unique to the Viking Lander 360/75 batch operations. It was developed to resolve a very difficult data management problem that did not exist for Viking 1108 operations, since adequate data management tools were available there. It proved to be superior to the 1108 file management system. It greatly simplified user requirements, significantly reduced the existing job failure rate and did not lose data sets. The lesson learned is that designing a data management tool based on user needs, computer set idiosyncracies and software system design produces a more efficient system than using existing albeit adequate management tools.

HISTORY: The data management procedures used on the IBM 360/75 were found to be unacceptable to meet Viking Mission time lines. Those on the 1108 were deemed adequate.

The Univac 1108 operating environment was such that it promoted standard file management practices to be performed by each user of the system. Maintaining permanent data sets on mass storage is generally impractical because:

1. special coordination is required with operations personnel to identify a data set having permanent status, primarily intended for program libraries, data bases, etc.;
2. accounts are charged for data sets remaining on direct access storage at the end of each day (non-permanent data sets are then purged after accounting);
3. if execution is desired on another available machine, unless a data set is permanent on the object configuration, the user must move data sets.

Item 1 above virtually eliminated Viking utilization of permanent data sets because the naming conventions required that each data set be given a unique name to help identify the file's content. Thus, if a data set was required on subsequent days, it was the user's responsibility to ensure that his files were properly placed on tape prior to terminating a session. In addition, the charge on abandoned data sets was incentive enough to remove files when no longer required.

These procedures with which 1108 users are burdened were necessarily followed throughout software deliveries to JPL and user acceptance testing. Thus learning to use the system effectively was virtually mandatory. It is interesting to note that most 1108 programs were originally developed on the CDC system, in Denver, where the same concept of no permanent direct access storage prevailed.

By contrast, the 360/75 system accommodated permanent direct access storage via user packs during software deliveries and user acceptance testing. This was different from the 370 development environment in Denver in that available space was reduced by roughly a factor of six.

Though this posed few problems initially, because data sets were resident until manual action was taken to remove the entry, a build-up of leftovers became a growing problem. Since Disk pack contents were considered permanent, when personnel departed after completing user acceptance testing there was no incentive to remove data sets no longer required, and there was no obvious need to move data sets to tapes which might be necessary for the next delivery phase. What happened was that other teams would arrive to deliver and test software only to discover that there was insufficient direct access space for them to operate. So they would resort to deleting data sets left by other users, many of which were important.

In short, there was little visibility or control of direct access storage usage. No incentives existed to use the resource in a rational manner. Because no self-training could develop as occurred on the 1108, it became obvious that self-policing would not accomplish the goals of maintaining direct access storage (DAS).

Attempts were made to guarantee that data sets would be checkpointed to tape and subsequently could be retrieved on direct access storage space. This task was cumbersome and somewhat ineffective because there was no way to assure that the data set contents was the latest. Regardless of the shortcomings of the checkpointing procedures, no method could be devised to remove unneeded data sets, except for polling and policing users. Frequently, even the user could not recall the criticality of a file's contents.

While it was possible to clear direct access storage daily on the 360/75, it was impractical for several reasons:

1. No reliable software existed that could readily checkpoint a collection of data sets to tape that could be retrieved individually (an entire pack could be moved to tape, but nothing short of the entire pack could be restored).
2. There was no firm central control over pack contents because: it would have been a full-time job for which no funds were authorized; using the available software tools would require an immense amount of computer time to be consumed; to ensure

data set contents, the pack control time would have to be separate from pack user time; and finally, coordination (i.e. file names to be checkpointed) would be difficult, if not impossible, due to varied block time/shift assignments.

Users might have been required to checkpoint files, but the available software utilities were extremely burdensome, error-prone, and would have consumed an estimated 20-30% of available CPU time. In addition, most 360 users were accustomed to having their data sets permanently resident on direct access storage (DAS) in the Denver installation. Even though direct access storage was more abundant, moving files to tape was not an alien concept. However, because of the general pain associated with the effort, and the tremendous amount of storage available, the action was seldom taken.

The computer environment at JPL was significantly different from that in Denver during development. In Denver, the computer was generally accessible twenty-four hours a day, whereas computer availability at JPL was by assigning the computer for exclusive use by Viking lander program developers for a specified period known as block-time. This approach resulted in prolific activity within a typical 4-6 hour block, generating an atmosphere of frenzy that sometimes bordered on panic, and left little room for thought about data management. Block-time participants worked almost always in a sloppy, barely-look-at-results mode, trying to use as much of the precious available computer time as possible before the blocktime ended. It was very difficult to perform data management activities during non-block time periods because a modification to the non-block time computer configuration was invariably required.

The Viking lander file naming convention allowed for one to identify: a data set as being mission or test; the type of data it contained; whether the contents were applicable to mission A or B, or both; and a final unique qualifier identifying the source link of data. In particular, the first level qualifier "VS" denoted a test data set and "VM", a mission data set. A second level qualifier identified the type of data set using an agreed to five character generic which was documented in the Software Data Base Document. The third and final qualifier

indicated lander association by the characters: XA, XB, or XX, for Lander A, Lander B, or both, respectively. This was followed by five digits, the meaning of which was well defined for the file. For example, Lander downlink data files used the day of year the data was recorded on earth and the version for that day. In particular, the Lander power program, LPWR, creates three data sets to generate the electrical load profile. They are: the PROFL file, which contains a complete load profile; the COMPR file, which contains a detailed power profile for each individual component; and the PWRIC data file, which will be used to provide initial power profile conditions to process data on the next downlink. A forecast run of Day 250 for mission B landed, nominal temperature, would have a qualifier of XB250L3. Should another run be required for the same time frame, and the original files were to be retained, the qualifier would appear as XB250L4.

The full mission data set names of these files are VM.PROFL.XB250L3, VM.PWRIC.XB250L3, and VM.COMPR.XB250L3.

An obvious advantage to using a well defined naming convention is to reduce, and make more meaningful user input. As for the above example, the PROC parameters might be the following:

//MISSION=B,DAY=250,TYPE=3

File headers were conceived in anticipation of the naming conventions to allow receiving programs the opportunity to verify that the data set they were using was in fact the one requested. In addition, a creation date was included to pinpoint the time the data set was written, thereby guaranteeing the uniqueness of each data set.

Whereas naming conventions could have been implemented earlier, they could not have preempted file headers, since the header was an encoded part of data set creation and could not be inadvertently modified. The file headers provide an additional check on the data set contents. However, it should be noted that no requirements were imposed to guarantee that the file version field would be properly maintained.

The data management problems encountered during user acceptance testing were amplified when the data systems integration (DSI) group began verifying software deliveries. Test checks were submitted to

DSI for program delivery verification, but test data to be processed by these runs often pointed to a direct access pack that was not part of the DSI computer configuration. Thus, more often than not, a required data set was not accessible.

It should be noted that it was required to have data sets available, but frequently they were on the wrong packs (in practice often data flight packs and development packs had identical identifications which enhanced the confusion), or sometimes they ended up on temporary copies of packs. Temporary copies of packs were required to be mounted when, as often occurred, two computers whose configuration requirements overlapped were scheduled concurrently. Rather than being concerned with individual data sets, emphasis, by default, was placed on maintaining direct access volumes with virtually no regard for content integrity. This approach was not because of any lack of concern, but was rather due to the lack of proper, usable tools to cater to individual data sets which, no doubt, fostered undefined procedures--other than identifying the location of data sets--for delivering test data.

The GDS testing conducted during the spring of 1975 brought the worsening data management problem into the limelight when one test was a total failure because required data sets were missing. As a result of this, procedures were defined to assure that required data sets would be placed on the appropriate packs. A tape copy was required to be available in case the direct access copy was, for any reason, not online at the initiation of the test. Finally, the proper personnel were required to be present for the test to work out any anomalies which might arise, such as fetching a mission data set from tape.

Despite these procedures, requirements and precautions, some test sequences had to be cancelled because the required data sets were not available. This was not the result of negligence; numerous data sets had been copied to the appropriate packs, but subsequently some were removed because of a breakdown in communications about the validity of the presence of the data sets on the direct access storage. Furthermore, the required backup tape copy did not exist because of a lack of computer availability combined with an additional breakdown in communications that led to the presumption that the action had been performed.

The inception of Viking flight-batch operations established a DAS configuration which was permanently on-line (for all practical purposes), thus automatically eliminating problems directly attributable to block-time processing. The initial DAS configuration provided two 2314 packs (4000 tracks each) with plans to incorporate a third for lander activities when mission/testing activities increased.

Throughout most of 1974 and the first half of 1975, manual procedures were defined to handle the management of Viking data files. This included the establishment of logs to be maintained by each user group to track their files and a central point of control to actually manage the DAS and better utilize tape facilities. This responsibility was assumed by a data processing team (DPT) figure known as fileman, whose original charter was to expedite file transfers between the 1108 and 360 computer systems. It became fileman's function to respond to flight team needs for file checkpointing, deleting, and restoring. In addition, fileman, being a focal point for DAS management, was responsible for coordinating with users to remove no longer required data sets when DAS space was close to extinction. This involved personally polling each user with a volume table of contents (VTOC) of each of the packs to determine which data sets could be removed.

As might be expected, users were most reluctant to remove data sets from DAS unless there was virtually no space available, at which point fileman prompted a campaign to personally contact each user and confront them with the hard facts. Although there was some improvement, removal was never adequately performed voluntarily, employed only during a crisis.

User reluctance to remove no longer required data sets from the DAS was based on a fear of not being able to retrieve the file if it was needed later and a somewhat innate lack of confidence in the many manual interactions necessary to perform the desired function--particularly in a timely manner. Looking more closely at the steps required, it is apparent that a number of weak links undermined the confidence level of the user.

The user was more than familiar with the problems of locating DAS space for a data set, even when file sizes were unknown. This required checking the VTOC listing of each pack to determine which might be likely to have adequate available space at the time allocation would be attempted (when the job was run). If the job was rerun, old copies required purging. Unless the user acquired a VTOC to note the size of a data set prior to checkpointing and removal from DAS, there was no record maintained of space used. Space allocation was the principle problem fileman had to face to restore a data set, after checking that the data set name was not already in existence. The possibility of the job running half-way and then resubmitted (or inadvertently rerun) grossly complicated the allocation task. It should be noted that the utility could allocate space, but its default values were generally much larger than required and that amount of space was frequently not available.

Then of course, there was the conveying of the tape and file to fileman, which had to be transformed into input to the IEHMOVE utility. Two more pitfalls quickly arose: possibility of transposing information; and generating a syntax error on the utility control card. Once the restore was performed, the user had to be informed of the success of the generation so they could submit their run.

Although central point was now coordinating the checkpointing, removing and restoring functions, there was no central source for data set tracking. Thus the thermal personnel had to query the power personnel for the location of a needed file. This required that either the on-duty power people were cognizant of the information, or that someone had remembered to make note of it before going home for the week-end. The stumbling blocks are simple to perceive, but their impacts were overwhelming. Though numerous procedures were written, and elegant logging forms were designed, the implementation of the original file management scheme was unsatisfactory, relying too heavily on people to record log entries immediately upon receipt of the information, without error, and to interact with other individuals to convey mundane - albeit important - information concerning data set location. Even the most conscientious personnel failed because of a timing problem, or a misplaced message. The coordination effort, already intensified to cope

with activities associated with mission activities, could not realistically be broadened to encompass file management problems. In a critical situation, more time was devoted to maintaining the basic vital functions than to coping with the actual crisis.

DESCRIPTION: Just prior to the launch of Viking I, OLDFMS development was initiated. The initial concern was to provide immediate relief to fileman by greatly simplifying the interface with standard OS utilities. This would simultaneously reduce the potential for error and increase file management capacity by decreasing the amount of information required to perform each function. For example, to checkpoint to tape and remove three data sets from DAS, the fileman was required to keypunch the following information onto cards:

```
//   EXEC CHKPOINT,TAPE=1783
                                                    if not cataloged
     COPY DSNAME=VS.OPDFE.XA00000,TO=2400=(1783,1) 1, [FROM=2314=VIK001]
     COPY DSNAME=VS.TEMPF.XA00201,TO=2400=(1783,2)  , [FROM=2314=VIK002]
     COPY DSNAME=VS.LPWRF.XA00604,TO=2400=(1783,3)  , [FROM=2314=VIK002]
//   EXEC REMOVE
     SCRATCH DSNAME=VS.OPDFE.XA00000,VOL=2314=VIK001
     SCRATCH DSNAME=VS.TEMPF.XA00201,VOL=2314=VIK002
     SCRATCH DSNAME=VS.LPWRF.XA00604,VOL=2314=VIK002
  if
cataloged UNCATLG DSNAME=VS.OPDFE.XA00000
          UNCATLG DSNAME=VS.TEMPF.XA00201
          UNCATLG DSNAME=VS.LPWRF.XA00604
```

If the data set being copied to the first file could not be done for some reason (data set already scratched, keypunch error, etc.), then subsequent requests could not be honored and the program would abnormally terminate. Before the data sets could be removed, the output of the checkpoint run had to be scanned visually to ensure everything was copied successfully. Upon delivery of the Phase I File Management program, the same process could be done by merely keypunching:

```
//    EXEC CHKPOINT, TAPE=1783,FILE=1,NAME=FILEMAN,SCRATCH=YES
      D=VS.OPDFE.XA00000
      D=VS.TEMPF.XA00201
      D=VS.LPWRF.XA00604
```

If the data set was not cataloged, the program would search the DAS packs to locate the file. The scratch would not be performed unless the data set was successfully copied.

In addition, software was provided to punch system catalog entries allowing fileman to extract pre-punched cards to respond to user requests (via a file request form), thus eliminating keypunch errors.

The Phase I File Management also provided for a simplification of restoring and allocating data sets. For example, pre-Phase I restoration of a pre-allocated data set requiring 150 tracks would appear as:

```
//    EXEC PGM=IEFBR14
//A   DD DSN=VM.TEMPF.XA00010,DISP=(,CATLG),UNIT=SYSDA,
//    VOL=SER=VIK002,SPACE=(TRK,(150,10))
//    EXEC LRESTORE,TAPE=9768
      COPY DSNAME=VM.TEMPF.XA00010,FROM=2400=(9768,13),
      TO=2314=VIK0002
```

Notice the requirement to explicitly define the object DAS volume, necessitating an educated guess as to which volume would contain the data set. Phase I could accomplish the task as follows:

```
//    EXEC LRESTORE,NAME=FILEMAN,TAPE=9768,FILE=13
      D=VM.TEMPF.XA00010,TRKS=150
```

Available DAS volumes would be searched until the required space could be allocated, if it was available.

Data sets could also be pre-allocated for programs to alleviate the necessity to choose a volume prior to job submittal.

```
//    EXEC RESERVE, NAME=USER
      D=VM.LPWRF.XA07633,TRKS=700,SEC=100
      D=VM.CMPAR.XA07633,TRKS=100
```

However, this method of input was in addition to specifying the data set names in the program process, thus complicating run submittals for the user. Phase II would allow for data set names and space requirements to be extracted from standard proc entries, making the allocation phase totally transparent to the user.

Phase II incorporated the logging of data set transactions into a permanently available data set which recorded such things as when a data set was allocated, checkpointed, removed, or restored. In addition, the number of tracks required for the data set and its DCB (data control block) attributes were recorded.

Thus a central point of information was established, readily accessible to any user at any time. Fileman generally acquired a log listing several times a day for reference purposes, removing the burden from flight team users of acquiring a log report.

The log report was presented by generic and mission/test qualification. That is, data sets, under VM.IANCO would appear together, separate from VS.LANCO data sets. For each entry space was provided to display the date of creation, a date for terminating on-line residence, two checkpoint tapes/files, tracks consumed, data set organization, record format, block-size, record length, the current status of the data set (original, inactive, restored), and last time of log entry modification. If a comment were entered for a given entry, it would optionally be displayed on the following line.

Log entries could be made only for generic-mission/test qualifiers that followed naming conventions and were identified to the log by fileman via log configuration software. Additions were made to the allocation, checkpointing, removing, and restoring modules to update by entries (if a log were present--complete downwards compatibility was maintained throughout), and new software was provided for manual updating.

Log initialization (/reconfiguration) allowed for each type entry (e.g., VM.OPDFE, VM.TEMPF, VM.GCSCI) to be assigned an owner. The report would contain the owner specification to further identify the data sets. However, the owner identification was more intended for use in Phase III software to assist in performing automatic checkpointing/removing of data sets. In addition, the log contained (for each type entry) a nominal space requirement and retention period (i.e., number of hours data sets of the given type would be required on DAS), intended strictly for Phase III operations, but part of Phase II to let the concept gradually develop. Thus, when automatic capabilities became available, most data set types had been identified.

Phase III software incorporated automatic checkpointing/removing, and a new function known as PROVIDE. All three packages used information from the OLDFMS log to perform their tasks. A Viking mobile tape rack that could hold over 100 tapes was provided to support the delivery, since the requirement for humans to know what tapes were needed was taken over by the software.

Automatic checkpointing would scan the log by owner, type entry, or in entirety, selecting data sets which were original, had been written (as opposed to being just allocated), and had not yet been checkpointed adequately (some data sets required double checkpointing). The data sets entries which met criteria were then passed to the standard checkpointing software.

Auto-checkpointing was always performed selectively rather than querying the entire log. The reasons for this are as follows:

1. Data sets were categorized by retention requirements; those required for only 30 days; those required for 60 days; those required to be retained until end of mission. Thus it was desirable to not intermix groups on the same tape.
2. The OS utility IEHMOVE (utilized for checkpointing) could not support multi-volume checkpoints; therefore, it would generally be impossible for all data sets requiring checkpointing to be processed in a given run. However, multi-volume checkpoints could have caused more harm than benefit when considering the possibility of restoring a data set spanning tape volumes.
3. Checkpoints could be performed in parallel (several jobs), reducing the elapsed time to process.

Automatic removing would scan the log similar to auto-checkpointing, with identical selective options. The auto-remove software would basically remove data sets whose active periods had elapsed. It was possible to increment or decrement the current time to allow for additional flexibility in defining data sets eleigible for removal. Original, restored, or all expired data sets could be optioned as candidates for processing.

Original data sets could not be removed unless adequate checkpoints had been created. Thus if an original data set was scheduled for removal, but was not adequately checkpointed, a message would be displayed to reflect the situation.

Phase III also enhanced the checkpointing function by using the log to ensure tape files could not inadvertently be over-written. Thus, for multiple checkpoint runs using the same tape, fileman did not have to maintain and enter the "next" available file number and, indeed, could not enter one which could destroy checkpointed data sets without substantial effort to override several safeguards and suffer verbal abuse issued by the software before succumbing to the hazard.

QUALITATIVE RESULTS: The PROVIDE function was probably the most effective piece of software in the entire OLDFMS package, the culmination of efforts to virtually eliminate manual user intervention in the file management scheme. Though extremely simple in concept, the impact of the user being able to say to the system "I need data set X", and it is made available, is unsurpassed. PROVIDE instilled user confidence in the on-line data file management system.

The ability to obtain an input data set for processing regardless of its status (active or inactive) prompted users to define extremely short active periods for many data sets (some as short as a few hours). As a result, most data sets were retained on DAS only while required for processing. PROVIDE promoted extensive flexibility in managing the DAS resources by allowing fileman to remove data sets which were perhaps still required, in an effort to free-up space for other users.

The incidence of job failures due to "data set not found" went to zero when PROVIDE was incorporated. If the data set was in the system, it was available, whether active on DAS, or residing on a checkpoint tape to be restored. A user's job requirements were virtually self-contained.

OLDFMS was able to greatly enhance mission software operations because:

1. it reduced substantially job reruns due to lack of DAS space or missing data sets;

2. it eiliminated the necessity for manual interfaces concerning file management details;
3. it allowed fileman to respond to critical file action requests in a timely manner;
4. it established a central point of information for data set tracking;
5. no data sets were lost or misplaced.

Had OLDFMS been available for development, UAT's and system testing, substantial time savings could have been realized with reruns because of missing files or inadequate DAS space reduced and additional processing eliminated to reproduce data files inadvertently destroyed.

Software to maintain and use OLDFMS logging was somewhat underscoped, partially because its user acceptance exceeded all expectations. More effort could have been spent on software to remove no longer maintained data sets and associated tape recycling though existing tools could perform the task, requiring substantial manual processing.

QUANTITATIVE IMPACT: Prior to development of OLDFMS, operational timelines could not be met. The job failure rate due to inadequate direct access space and files being removed from the system ranged from 5 percent to 30 percent, depending on the level of activity. After implementing OLDFMS, no job failed for these reasons, and mission timelines could be met. Delays inherent in preparing to submit a typical run were reduced by an average of thirty minutes per run. Manpower costs for the design and implementation of OLDFMS were approximately 5 man months, distributed over a nine-month time period.

This relatively low cost of development was possible because many file transfer software functions, which were used by OLDFMS, had already been developed. As such, the cost to develop OLDFMS from scratch would have been more like one man year.

TECHNIQUE

NAME: INTEGRATED SOFTWARE FUNCTIONAL DESIGN

SUMMARY: Software Functional Descriptions (SFD) were written for each candidate Mission Operations software function. Concurrence by Flight Team members established the requirement for a program. The SFD's were combined to form an Integrated Software Functional Design (ISFD) of the entire software system. The ISFD was subjected to preliminary and critical design reviews by the Flight Team Directors and the Mission Director. Upon acceptance by the Mission Director, the ISFD was placed under change control to establish the baseline design for the Mission Operations Software System.

APPLICATION CONSIDERATIONS: Each Flight Team group was in a position to define the functions they would need to support and control the Viking spacecraft during Mission Operations. A requirement that these functions be documented offered management a tool by which they could combine functions used by more than one group, determine which should be performed through procedures and which through software, and establish program need date as a function of mission phase. By integrating and combining the individual functions, the data flow for the entire software system could be established.

RECOMMENDATION: A software integrated functional design provides an excellent means for management to understand and structure a software system. It can be used to determine the amount and type of software necessary to be developed, thereby laying the foundation for manpower and computer resource requirements. By establishing the data flow of the system, it makes visible the integration requirements for the system.

HISTORY: The Flight Operations Software Plan specified the need for software functional descriptions and an integrated software functional design of the Mission Operations Software System. The lander, orbiter and institutional software systems engineers were required to gather software functional descriptions for each candidate element of their software systems. They delivered them to the Integrating Contractor Software System Engineer (ICSSE), who was responsible for publishing them and generating an ISFD from them.

This proved to be a lengthy iterative process, wherein numerous meetings were held among the software systems engineers and the various Flight team groups to understand the need for and interplay between the various functions. An initial ISFD was generated and subjected to considerable review by each of the Flight team groups, primarily to determine interface rquirements and uncover system deficiencies.

Eventually, six software systems were defined to support mission planning, lander and orbiter uplinks and downlinks, and tracking and flight path analysis. Diagrams for each of these systems were used by the ICSSE to conduct a preliminary design review of the system, which was held before the mission directors and representatives of the various Flight team groups.

Following the PDR the iteration process continued and brief text descriptions were developed for each of the software systems.

The critical design review was held in a high school auditorium before an audience of several hundred people; included were Flight team members, directors, and outside software experts brought in by NASA to critique the Viking software development approach. To accommodate such a large audience, the ICSSE used very large diagrams for each subsystem, the largest of which was ten feet high and forty feet wide.

Following the CDR, the Mission Director approved the ISFD. The SFD's and ISFD were then incorporated as an appendix to the Flight Operations System Design document and placed under Viking Integration Change Control. The software system was now structured such that any change to the ISFD would affect system data flow and impact more than one program.

DESCRIPTION: The software design and development process began with the identification of software which was the basis of the ISFD. Identification of software was accomplished by the appropriate Flight Operation subgroup by preparation of a SFD for each required function. The SFD was written in accordance with the following format and requirements.

1. Program Title
2. Functional Description - Give a brief description of the general functions to be performed by the program. While the functions are of main interest, some information on capabilities and mathematical method is also desirable.
3. Utilization - Describe the intended use for Viking operations in general terms with reference to mission phase, frequency of use, use in program run streams, etc. For new programs, a step-by-step functional description of the program operation is recommended to facilitate the integration process.
4. Input/Output - Describe all data and program interfaces, both internal and external to the particular operations software element. This section shall be broken up into subsections entitled "Input" and "Output".
5. General - State whether the program is essentially a new program, a current existing program, or one that will be derived from an existing program. If the latter, name the baseline program, computer developed on and the magnitude of modification. If possible, give some indication of expected program size and running time. Specify any anticipated or known program constraints.
6. Bibliography - Give references to pertinent documents which would provide more information about the planned program or which describes the existing program.

The ISFD was developed in a series of increasingly complex stages. A target program, such as the Lander Sequence Generation Program (LSEQ), was shown as a box. The input section of the SFD for this target program was then used to determine what information was required to be made available to the program. A logical subset of the remaining SFDs were examined to see if they could or did generate output for the

target program. When a source for input to the target program was found, the source program was added to the diagram and an arrow was drawn to indicate the data flow. This sometimes required modifying the output section of the SFD of the source program. If no input source for the target program could be found, the input was shown to be manual. The output section in the SFD of the target program was treated in a similar fashion, showing each output item either going to another program, to a printer, a plotter, or to archives. In this fashion, a simplistic overview of the entire software system was generated which accounted for each SFD.

The basic system flow diagram was next iterated upon by the various Flight team groups to determine if manual inputs would require new software functions to be defined, to ascertain which prints and plots would generate information required to produce manual input to other programs, and to assess whether some functions should be moved from one program to another.

Following this iteration, more detailed diagrams were drawn that indicated the means by which interfaces would be accomplished. Symbols were used to show mass storage files, tape files, card files, and manual interfaces. The latter illustrated that printed output from one program would become punched card input to another program. File management functions were now determined and added to the diagrams.

Computer loading studies were conducted to balance the computational loads on the available computer systems. The ISFDs then were expanded to show the computer systems involved. A rough estimate of throughput time for data to be passed through the system could now be made.

The final step in the generation of the ISFD was to write a short narrative describing the software programs used by each subsystem, how they would be operationally used, and how information would flow through the system.

QUALITATIVE RESULTS: The ISFD was extremely valuable to the success of developing a software system that was both reliable and minimized the amount of software necessary to be developed to support all mission

objectives. It permitted the project to assess system capabilities during the design stage. It provided a structure for the overall system that was visible and easily controllable by management. It lay the foundation for the system integration requirements and the development of the Software Data Base Document.

The document was maintained through integration of the software system, after which the Software Data Base Document was used to control changes to the system structure and integrity.

QUANTITATIVE IMPACT: The cost to develop the SFDs and ISFD was approximately five man years. Changes to and maintenance of the descriptions cost an estimated two additional man years. A total of 130 SFDs were written to describe 61 lander, 56 orbiter and 13 institutional software functions. From these, six ISFDs were developed showing 8 Mission Planning, 24 VO Uplink, 18 VL Uplink, 40 VO Downlink, 53 VL Downlink and 13 FPA functions. The total number of functions shown in the ISFDs adds up to 156, which illustrates that 26 adaptations were made to allow functions to support more than one subsystem. Thus, had the ISFD approach not been taken, it is reasonable to conclude that some redundant software functions would have been developed at additional cost in manpower and computer resources.

TECHNIQUE

NAME: MISSION BUILD PROCESS

SUMMARY: Viking operational software code, including object, source and program listing, was placed under strict control at the beginning of the system integration phase through a process known as the Mission Build. Software could only be added to the Mission Build by Viking Mission Control and Computing Center (VMCCC) personnel, who were responsible for maintaining the Build. During integration and operations, only object code on the Build was available to users in a read only mode. The Mission Build process is a software control process that assures deliverables will function as built within a computer system.

APPLICATION CONSIDERATIONS: The Mission Build process was developed at the Jet Propulsion Laboratory to control the development, integration and use of operational software systems in a multi-mission environment. The process permitted only authorized software to be made available to users, and prevented the software of one project from conflicting with the software of any other project. Use of this process was mandatory for real time operational software. The Viking Project elected to invoke the process for all batch operations as well, since it afforded them a practical and established means to control their software system.

RECOMMENDATION: The Mission Build process is conceptually an excellent means by which management can control a software system. The process should include a capability to provide temporary overrides that are transparent to users. The process of generating, updating and maintaining a build will be costly in manpower and computer resources. The resultant configuration control is well worth the additional cost. The override feature of the build offers the advantage of being able to correct errors or add new functions without inadvertently introducing errors to delivered software. This feature should not be used for real time systems except in extreme emergencies, but should be incorporated as the standard procedure for modifying batch systems.

HISTORY: Both Integrating Contractor Lander/Orbiter software integration and VMCCC integration testing were conducted using software programs resident on the same mass storage media, called the Integration Mission Build. The method of adding programs to or updating programs on the Integration Mission Build varied by facility, as described below. The Integrating Contractor integration was limited to running Viking Lander and Orbiter software under control of the VMCCC operating system, whereas the VMCCC integration also included running Institutional MCCC software and non-Viking Project software.

Software System One (MOSS 1) was obtained by copying and saving the Integration Mission Build when all the integration for that system had been completed. The integration process continued, adding and modifying programs on the Integration Mission Build and performing the integration tasks for each of the remaining software systems. Copies of the Integration Mission Build were made as the integration for each software system was completed.

The Mission Control Computing Facility (MCCF) supported the Viking Project with two IBM 360/75 computer systems. One was used to support multi-mission, multi-project, real time program operations. The second computer system was used to support both Viking and MCCF batch program operations. The initial concept was to convert Lander programs developed on Martin Marietta computers by submitting program decks as over the counter batch jobs to MCCF operations personnel. During preparation for Users Acceptance Testing, Project software was then to be incorporated on a Development Mission Build, which was available on a daily basis but did not usually have the same operating system as the Integration Mission Build. This sounded fine in principal but did not work in practice. The software developers not only wanted to use the Integration Mission Build operating system, but they also wanted to link edit their programs to controlled delivered software library elements rather than linking to uncontrolled software library elements whose status could change at any time without visibility to the user. For this reason they placed their programs on a private disk pack, known as DSN005, which was assigned to and controlled by the Integration Contractor

Software System Engineer (ICSSE). During conversion and Users Acceptance Testing, the software developers requested the Integration Mission Build packs, or copies thereof, plus DSN006, to be mounted on the system. This precluded being able to run during daytime operations when the Development Mission Build packs were mounted. Blocks of time were therefore made available to these users during second and third shifts and on weekends to allow them to use controlled software. Following Users Acceptance Testing, Project software was placed on the Integration Mission Build and unit verified by the Data Systems Integration Group (DSI) of the VMCCC. The ICSSE then performed the required Lander/Orbiter integration tests.

Frequently program malfunctions were detected during ICSSE integration. When this occurred, modifications to the program were made and written to disk pack DSN006 and the integration test was completed using that pack to override the appropriate portions of the Integration Mission Build. Upon completion of this testing, failure reports were written against those portions of the program that malfunctioned on the Integration Mission Build, but worked with the DSN006 override. This failure report procedure permitted the ICSSE to authorize redelivery of the corrected and tested portions of the program to the VMCCC to be incorporated as permanent updates to the Integration Mission Build. Each redelivery required the DSI to unit verify those portions of the program that had malfunctioned, after which the ICSSE repeated the integration test without DSN006 being mounted.

The General Purpose Computing Facility (GPCF) supported the project by making either of two general purpose UNIVAC 1108 computer systems available on a daily basis. The initial conversion of Lander programs from MMC computers to the GPCF computers was by submitting programs decks as over the counter batch jobs to GPCF operations personnel. The same EXEC-8 operating system was available for both general use and the Integration Mission Build and programmers could map their software to controlled delivered software library elements during standard daytime operations, using the qualifier VIKING. During preparation for User Acceptance Testing, individual program Bench Mark tapes were

written and submitted to be run as batch jobs. The Users Acceptance Tests were conducted in the same environment. Following this, GPCF program integration tapes containing source, object and listing of the program, were submitted to DSI. These were combined onto a series of GPCF integration tapes under control of the DSI. The software was made available to users by the DSI who had the contents of the tapes read into the computer under the qualifier VIKING.

Unfortunately, qualifier VIKING was frequently not on the system during second and third shifts and on weekends during development and ICSSE integration. This caused problems in submitting overnight or weekend runs, which were desirable since computer rates were considerably cheaper during these periods. For this reason the ICSSE, who controlled the delivery of the software, created an identical Integration Mission Build under his control that contained the originals of the controlled software elements. This ICSSE Build was made available at all times to users under the qualifier VIKINT. Integration testing was conducted under this qualifier, and program malfunctions were treated in a fashion similar to that described above for the MCCF. Effectively what had happened was that the Integration Mission Build was for all practical purposes merely a copy of the ICSSE Build; actual control of the software system during this period had by default passed from the DSI to the ICSSE.

DESCRIPTION: The JPL Mission Build process is a mechanism by which the elements of various operational software systems are integrated within the facilities of the Mission Control and Computing Center (MCCC) to support controlled multi-Project, multi-mission operations. The integration portion of the process is relative to the Operating Systems, institutional software and computer complexes; the integration of the operational software systems is the responsibility of the Projects that use the Mission Build process, and is independent of the build process. As such it is a software control process.

There are normally three distinct Mission Build phases:

a) The development Mission Build is used for software development and User Acceptance Testing (UAT). Change control is maintained at the programmer/engineer level. This phase was not used by the Viking Project.

b) The integration Mission Build is created by MCCC Data System Integration (DSI). Only software accepted by DSI is placed on this Mission Build. Change control is maintained at the SSE/DSPE/DSI level.

c) The flight support Mission Operations Software System (MOSS) is the final product of DSI which is delivered to Operations and the Operations Program Data Base (OPDB). Change control is maintained by the multi-project MCCC Change Control Board.

The process used by Viking began with the delivery of post-UAT software elements to DSI. MCCC integration testing then consisted of delivery verification (Unit Verification Tests - UVT), subsystem/program verification, system performance tests, system loading tests, and facility level demonstration tests. How each of these were accomplished will be discussed in the following paragraphs.

After each software element satisfied the UAT requirement, it was delivered to DSI. The items that accompanied each delivery were:

1) The program deliverables as applicable to the facility in which the software end product will be operated;

2) The documentation required to define and use the software element;

3) An Inventory Change Authorization (ICA) form completed and signed by the software element supplier which certifies that the deliverable successfully completed a UAT;

4) A completed Inventory List identifying all items required to be delivered to DSI. Each item included in the delivery package was identified, and a schedule delivery date was provided for each item not included in the package;

5) An Estimated Parameter List that provided adequate information for computer loading analyses and to verify conformance with the MCCC guidelines and constraints;

6) A copy of the authorized change package if the delivery was the result of change action. For changes to the integration Mission Build to correct program errors, the signatures of the appropriate SSEs were required. Changes based on new requirements or to the MOSS also required the signatures of appropriate Mission directors.

The program deliverables to the Mission Control and Computing Facility (MCCF) included a TRIO tape created under Real-Time Program Management (RTPM), Load Module descriptor cards, Mission Build Input cards, and test decks for unit verification and regression tests. The TRIO tape was a scratch tape generated by each final UAT run. It contained program source, object and listing. The contents of the tape were dumped to Bank Disk packs following UAT. Delivery to DSI authorized MCCC personnel to run the Mission Build input cards, which transferred the program elements from the bank disks to the Mission Build packs.

The program deliverables to the General Purpose Computing Facility (GPCF) included a GPCF integration tape and a print of the table of contents of each file on the tape. File 1 contained the runstreams and EXEC-8 run control statements for the UVT test case; file 2 contained the test case data; file 3 contained the absolute program elements; file 4 contained the relocatable elements for each absolute element; file 5 contained the symbolic or source elements of the absolute elements; and file 6 contained any necessary run control cards, data elements, and all symbolic relocatable and absolute elements for each utility program included with the delivery. Delivery to DSI authorized them to concatenate the delivered object code onto Mission Build GPCF tapes, which were the source of the operational software system in this facility.

A UVT was conducted for each program delivery, redelivery or modification to verify that the program was successfully incorporated into the Mission Build. These were user supplied regression tests. Failure reports were prepared when software was not successfully incorporated into the Mission Build which permitted the SSEs to take corrective redelivery actions.

The objective of the subsystem/program test phase was to verify the successful performance of each system or program, and the proper interfacing of each system or program. These tests were essentially combined program acceptance tests which verified each program's proper performance with the integration version of the facility operating system. They were functional performance tests used to examine computer memory and execution time requirements.

The system performance tests were run independently at each facility of the MCCC. The purpose of these tests were to evaluate the overall performance of the elements of the software system when operated concurrently. They were used to examine program to program interference and operating system interference that could cause performance degradation during operations.

The system loading tests were run independently for each facility to identify loading problem areas, recommend alternative solutions, and determine system constraints.

The system demonstration test was conducted with project support to demonstrate to all applicable projects that the software systems could meet all flight support and testing requirements.

Upon the completion of successful demonstrations, the system was made available to Computer Operations.

QUALITATIVE RESULTS: The Mission Build process worked very well for 360/75 MCCF operations, but was only marginally successful for 1108 GPCF operations. The reason for this lies in the differences in the procedures for creating the builds on the two systems.

The TRIO tape concept used by the MCCF had the effect of forcing the programmers to follow a procedure that did not leave them with a working copy of the link-edited object code of their program on a private tape. Also, an easy to control override feature that was transparent to users was available. The override patches could only be generated when the link-edit Mission Build pack, MSC3A9, was mounted on the system. This pack was not a standard mount during operations and could only be placed on the system when authorized by the Mission

Director for a short period of time during which the override patch would be catalogued as a data set on one of the Viking Direct Access storage packs. Finally, if a software element failed, only that element had to be redelivered to the Build, at which time all user programs of that element were re-linked to it.

The GPCF integration tape concept used by the GPCF provided far less software control. The final Bench Mark tapes the programmers were required to generate contained all of the software elemnts needed to run their programs. This made it possible to maintain private sources from which programs could be run, or modified, during operations. In addition, overrides of software elements were not possible. When a software element failed and was corrected, every program that was mapped to that element had to be remapped, using a private source, and then be redelivered.

In both facilities the Mission Build process provided excellent control over all deliverables made to both the integration and operational software systems.

The amount of testing conducted by the DSI during the Mission Build process was greater than needed for the benefits derived. They were essentially stand alone regression tests of proven program runs that allowed DSI to demonstrate to the various projects that the software systems were operable. They did not establish that the software systems could handle mission data flows or timelines, nor did they establish that the individual programs could be used to form an operational software system. The operational software system was established by Project integration, compatibility, and team training testing.

Divorcing the build process from the integration process caused problems and increased expenses. Early in the development phase, the Viking ICSSE failed in an attempt to negotiate combining these efforts. The result was, predictably, that the number of redeliveries caused by detection of malfunctions during integration was more than should have been required. The MCCC objected to this, stating it was taxing their resources to the limit. This prompted the ICSSE to maintain private builds in both facilities, using SECURE in the GPCF and disk

pack DSN005 in the MCCF. In that way, the ICSSE reduced the number of deliveries required. In addition, the ICSSE build in the MCCF was typically more advanced than either the Integration Mission Build or current MOSS. At one point the Mission Director was forced, albeit reluctantly, to use the ICSSE build (renamed the Viking Test Build and given project control through the ICSSE) to support critical compatibility testing in a timely fashion.

The action taken by the ICSSE should have been totally unnecessary had meaningful controls been applied to the Integration Mission Build. The refusal of the MCCC to permit overrides made sense for realtime operations, but was nonsense for batch operations. The build had been conceived to protect real time systems, wherein an error introduced through an override could be fatal to the entire system. Viking was the first batch user of the process. Errors in batch programs only cause the program itself to fail, and not the system. Therefore, it would have increased software control to have permitted override corrections to be made against the batch Integration Build, since the delivered software was unaffected by the overrides (i.e. to use an override one must deliberately point to it, otherwise the override is ignored). The use of DSN006 was a people control process rather than a software control process. Changes, rather than overrides, had to be made. Therefore, unlike the build process, it was possible to introduce errors into previously delivered working software.

QUANTITATIVE IMPACT: The Mission Build Process permitted management to know and control the status of the integration and operational software systems at all times. Without using the build, all batch software functions would have been available under individual rather than management control. Therefore, the cost of the process was an additional price the project was willing to pay to insure system integrity.

Approximately 40 Integration Builds were made by the VMC[3] for Viking 360/75 Batch operations. Each required the exclusive use of a computer for six hours. Unit verification of the programs delivered to this build required an eight man year level of effort and approximately 300 computer hours. In addition, fourteen operating systems were

made by copying the integration build, each of which required eight hours of computer time. Based on this, it is estimated that the additional cost to 360 operations to use the build process was eight man years, 650 hours of computer time and 72 direct access storage disk packs.

The Mission Build process on the 1108 system was considerably different, and its impact will be judged accordingly. The integration build process was accomplished by concatenating new deliveries to the current operational system, rather than maintaining a separate build. Also, unit verification of programs was easier to accomplish on this system because all test data and files were delivered with the program on the GPCF tapes. These were accomplished at a cost of eight man years and approximately 400 computer hours. To store this software an average of six tapes for each of 41 programs (due to redeliveries) and six tapes for each of 12 operational systems, or a total of 318 tapes were required. The cost to the ICSSE to make permanent versions of current software available to users at all times was a 200 dollar per month storage fee for two years. This was actually not an impact, since it was more than off-set by savings realized by using the system when nighttime and weekend rates were effective.

TECHNIQUE

NAME: COGNIZANT ENGINEER/COGNIZANT PROGRAMMER

SUMMARY: Each Mission Operational software program was assigned a Cognizant Engineer (CE) and a Cognizant Programmer (CP). The CEs were responsible for generating program requirements and testing the program to meet those requirements. The CPs were responsible for designing, implementing and defining the procedures for operating the programs.

APPLICATION CONSIDERATIONS: The rationale for adopting this concept was based on the belief that an engineer who understood requirements would not necessarily understand how computer systems could be used to implement them. Once a programmer implemented working software, the engineer would then be in a position to test the software to meet the requirements. The primary reason for assigning a particular programmer to be responsible for the design, development and implementation process, rather than using a software pool, was to have a second individual become thoroughly familiar with all the requirements for the software function. A secondary purpose was to provide an incentive for pride in workmanship.

RECOMMENDATION: Management can increase personnel work incentive by adopting the CE/CP approach to program development. Programmers will generate working software based on their interpretation of requirements, which are not necessarily correct. Requiring the CEs to write the program acceptance test plan provides the balance required to assure the programs will function to meet the engineering requirements.

HISTORY: The Cognizant Programmer/Cognizant Engineer philosophy was first documented in Standards for Viking Software Development, issued by MMC in October 1971. The concept was not adopted for Flight or System Test Equipment software development. In those areas, engineers were assigned to write the Software Requirements Documents, after which they were given new assignments. This in part is responsible for the fact that no formal acceptance test plan or equivalent was ever written for the STE software system, even though it was developed for a general purpose computer. The CE/CP approach did or could not realistically be applied to Flight software, which had to be validated by emulation techniques, by an independent validator, and by tests involving the entire lander hardware/software digital system.

Mission Operations adopted the CE/CP concept and specified it as a requirement in the Flight Operations Software Plan. It was used to develop all Lander and Orbiter operational programs. Two lander programs did not follow this procedure.

The lander power program was originally developed by a single engineer who could code in FORTRAN. The engineer did not understand the scope of the task and thought he could do it by himself. During the coding phase he began slipping his schedule. Management formed a Tiger team to assess the situation, the result of which was to assign a new cognizant engineer plus a cognizant programmer to assure the program would be delivered on schedule.

The file management program developed to support the common data base and inter-computer file management functions was originally assigned a CE and a CP. However, when the program was taken to JPL in December 1973, the CE declined to move to California and dropped off the Viking project. At that time the CP was made both the CE and CP for the function. He was directly supervised by the Integrating Contractor Software Systems Engineer. He was able to handle the task, but was typically delinquent in providing the required support documentation. The result was that members of the Software Integration Group were frequently required to come in at odd hours to show users how to run the program. Eventually, satisfactory documentation was produced.

DESCRIPTION: The roles specified for the cognizant engineers and cognizant programmer in the Flight Operations Software Plan were as follows:

Role of the Cognizant Engineer (MMC/JPL). A Cognizant Engineer (CE) shall be assigned by the appropriate department at the request of the Software System Engineer (SSE). The MMC CE shall serve as the coordinator of, and is responsible for providing the Functional Requirements Document, the Software Requirements Document, and the test documents for the specific program over which he is assigned cognizance. The JPL CE shall have overall responsibility for the development of the software program and is responsible for the FRD, the SRD and test plan for the specific programs over which he is assigned cognizance. The CE, as a member of the software design effort at MMC or JPL, will support the related SSE and the users of the program in the performance of this role. The CE is responsible for the following specific tasks:

a. MMC CE - Develop, with the concurrence of the related SSE, a schedule for the preparation of the SRD and test plans;

b. JPL CE - Develop, with concurrence of the related SSE, a schedule for the development and testing of each program;

c. Establish the detail requirements and prepare the FRD and SRD for which he is cognizant;

d. Review and concur with the General Design Document (GDD) and Schedule and Work Plan generated in response to the SRD;

e. Review all requests for changes to the GDD;

f. Coordinate the inputs, provide the Certification Test/Users Acceptance Test Plan (CT/UATP), the associated test data requirements, and test procedures to be used during the tests;

g. Work directly with the CP and provide resolution of details which have not been clearly defined in the SRD;

h. Perform Certification and/or Users Acceptance Test in accordance with the applicable test plan and write the test report. The necessary assistance in performing this task will be provided by the SSE and CP.

i. Support software configuration management in accordance with the control procedures in Appendix B of this plan;

j. Support the generation of the User's Guide and concur in its readiness for delivery;

k. Support the maintenance of the deliverable items.

Role of the Cognizant Programmer. When requested by the SSE, the Cognizant Programmer (CP) shall be assigned by the appropriate department with the concurrence of the CE. Any reassignment shall be concurred in by the CE and SSE. The CP shall be responsible for the following:

a. The coordination and generation of the General Design Document (GDD), Program Description Document (PDD), User's Guide and inputs to the Software Data Base Document (SDBD);

b. Develop, with the concurrence of the related SSE and CE, a schedule for the program development and documentation;

c. Review all changes requests to the SRD and prepare change request impact summary on software design and development;

d. Support data generation for software certification and user acceptance testing;

e. Generate, with coordination of the CE, test cases for use in compatibility testing and integration;

f. Coordinate with the CE during the final development of the SRD;

g. Support all software testing;

h. Design, code, and test the programs to meet the requirements of the SRD and specifications of the GDD and system constraints identified by the SSE;

i. Support the SSE and CE in the generation of the deliverable items of the program and documentation for TDS/VMCCC integration and operational system release;

j. Support the FOS in training, testing, mission operations, and program maintenance;

k. Implement all changes to the GDD, PDD and SDBD after their release and approval;

l. Support software configuration management in accordance with the control procedures in Appendix B of this plan.

QUALITATIVE RESULTS: The quality of the software produced was a function of the relative abilities of both the cognizant engineer and the cognizant programmer. The talent available to the Viking Project ranged from mediocre to excellent in both categories. When either the programmer or the engineer was excellent, the resultant software end product was very good.

The ability for the engineer to clearly and accurately specify requirements was extremely important. In some cases the programmers learned and understood the requirements as well as the engineers.

Some friction developed on a few of the programming teams. This was only partially due to personalities. Management tended to overemphasize the importance of the engineer to the detriment of the programmer. That is, when everything went well the engineer got the credit, but when problems came up they were too often blamed on the programmer. The roles of the CE and CP should be kept in proper perspective by any project adopting this technique.

The prime disadvantage to selecting the CE/CP technique over using a engineering pool/software programming pool is that each programmer must develop every function required by the program. This makes it more difficult for systems engineering and integration to generate a commonality of utility subroutines used by a multiple of programs. However, the improvement in communications available with the CE/CP more than offsets this disadvantage.

QUANTITATIVE IMPACT: This technique did not entail any kind of a cost impact, since the same number of programmers and engineers would have been needed if an engineering pool/software pool had been used. The development effort for the Viking Lander operational software programs broke down as 45 percent for engineer participation and 55 percent for programmer participation. Half of the engineering hours were spent on requirements generation and documentation; the other half was spent on test plan generation and test support. Two-thirds of the programmer hours were spent on design and code; the remaining third was spent on testing and maintenance.

TECHNIQUE

NAME: SOFTWARE DATA BASE DOCUMENT

SUMMARY: The Software Data Base Document (SDBD) provided central viability to the Software Systems Engineers, Cognizant Engineers and Cognizant Programmers of the use of tables, buffers, files and constants. It provided the means of identifying common data and implementing a common data base internal to the operational software system. After development the SDBD provided centralized documentation and control of all common data and interface files used by the Viking Flight Team.

APPLICATION CONSIDERATIONS: A considerable amount of data was common to more than one Viking operational program. These data consisted of such items as computer turn-on times, one-way light time tables, Martian parameters, descent parameters, lander coordinates and antenna pointing parameters. Coordination of data that would be subject to little or no change was necessary to assure the integrity of the software system. In addition, files interfacing stand-alone software modules required centralized visibility to control their structure and provide a means of assessing system level impacts caused by changes to individual software modules.

RECOMMENDATIONS: The need for a centralized document is essential to provide management with the visibility to control and coordinate data and file structures internal to a software system. This holds true regardless of the method used to implement the data processing of the software system.

HISTORY: The Flight Operations Software Plan, issued in the spring of 1972, specified that all tables, buffers, files and constants used by the Flight Operations Software System would be compiled into a single document, entitled the Software Data Base Document (SDBD). The responsibility for coordinating and maintaining the document was assigned to the Integrating Contractor Software System Engineer (ICSSE). The SDBD was to contain data and files common to multiple programs, common to only one program, and an index of constants established as standards for all programs.

The SDBD was to be placed under the Viking Integration Change/Viking Change Summary (VIC/VCS) change control system beginning with the milestone for issuing the General Design Documents (GDD) of each program. In actual practice, each time an interface file description was added to the SDBD it became a baseline and was automatically placed under VIC/VCS control. Delivery of these descriptions did not necessarily correspond to the specified milestone.

Two documents were to be issued. These included an MMC SDBD under control of the ICSSE and a Viking Mission Control and Computing Center (VMCCC) document under control of the Data System Project Engineer (DSPE). The MMC SDBD contained the VL-VL, VL-VO, and VO-VO interface file descriptions. The VMCCC document contained the VL-VMCCC, VO-VMCCC, VMCCC-VMCCC and VMCCC-IPL (Image Processing Laboratory) interface file descriptions. The VL and VO programs were the batch portion of the Mission Operations software system, the VMCCC programs were the real time, near real time and institutional portion of the software system, and the IPL programs processed Lander and Orbiter imaging telemetry.

The Viking Lander and Viking Orbiter Software Systems Engineers (VLSSE and VOSSE) were required to concur on deliveries and changes to data and files affecting their software systems. Following milestone 12 (program delivery to VMCCC) the DSPE also was required to concur with changes to VL or VO interface file descriptions. The ICSSE was required to concur with all interface descriptions affecting VMCCC software.

The software plan therefore emphasized requirements for controlling interface file descriptions, but failed to make references to controlling what it referred to as tables, buffers and constants. Probably this was the reason why only tables and constants used by Lander programs were incorporated in the SD3D. The Orbiter software system did not include a common data base and the tables and constants used by each program were documented with that program. The Lander software system required the development of a common data base and associated read only file management software. Therefore, lander program documentation referenced all keys used to access the common data base. The common data base itself was documented in the SDBD.

DESCRIPTION: This description will be limited to the MMC SDBD. The development of the document was conducted by the Software Integration Group under the direction of both the ICSSE and VLSSE. The VOSSE supplied the VO-VO interface file descriptions. The VO-VL interface descriptions were negotiated by the affected programmers and engineers through the VOSSE and ICSSE.

Files interfacing multiple programs were identified in the Integrated Software Functional Design (ISFD). The Cognizant Programmers of each lander program identified all files that interfaced two or more load modules of their program. Each of the above files were assigned generic identifiers and listed for inclusion in the SD3D. The process of obtaining file descriptions then consisted of obtaining a detailed description of the purpose, format, data content, size (which could be variable), frequency of use and storage media from the CE or CP responsible for the program that generated the file. The WRITE statement for formatted files was also included. The file description was then taken to the CP/CE team of each program that accessed the file. If all parties agreed to the description, they signed their names to a concurrence form that was included with the file description in the SD3D. If there were disagreements, the Software Integration Group would call for a meeting among every CE and CP associated with the file. At that meeting an agreement as to the files description would be reached and

all parties would sign the concurrence sheet. If the resolution was that the file would have a slightly different format on the 360 than on the 1108, the differences in format were clearly identified and incorporated into the file description. After all CEs and CPs signed the concurrence form, the appropriate [illegible] sign the form, thereby authorizing the inclusion of the desc[illegible]n into the SDBD. The description now was under Viking Integration Change Control.

The tables and constants to be incorporated in the common data base were collected in a different fashion. Each CE/CP team identified the need for a table or constants to the Software Integration Group. This included only data that would be constant or relatively stable during development and operations, such as one-way light time tables, lander coordinates, the diameter of Mars, the time of separation or the value of pi. Each table, constant, or group of related constants would then be assigned a unique identification generic, called a key, by which it could be accessed through the file management software. A specific individual was made responsible for the values associated with each key. In the event that two programs requested different values for the same constant (time of separation, whether to represent the Mars diameter in meters or feet, etc), the affected CE/CPs would be contacted and an agreement would be reached. The values were then incorporated in the common data base. They remained under control of the ICSSE until the software system was placed on-line. At that time a computer printout of the common data base was reduced and incorporated into the SDBD. Any changes thereafter had to be approved by the directors responsible for affected programs. The users of the keys would be notified of the change in writing, and a new printout of the common data base would be taken and kept in a central location available for inspection by any Flight team member. The SDBD was not updated to reflect the change; rather it was the user's responsibility to attach the change notice to their copies of the SDBD.

The SDBD was also used as a central point to document the time utilities, methods for accessing the Parameter Passing File, and miscellaneous material on common utilities. It contained a cross index

section that showed every file a program used and every program accessed by a file. It also identified the five character generic that appeared in the header record of each file for data management and inter-computer transfer uses.

QUALITATIVE RESULTS: The interface file descriptions greatly simplified and facilitated the software system integration process. Despite the fact that every affected engineer and programmer concurred with the structure and content of a file, most interfaces did not work the first time they were tested. The SD3D made it easy to determine the reasons for the failures.

Because of limited manpower resources available to develop the SD3D, the document was somewhat lacking in respect to tables and constants contained in the common data base. Multiple entries of constants occurred and not all constants that should have been included were actually included. Some programs accessed tables and constants as input data rather than from the common data base. The potential for error was therefore greater than it should have been during planetary operations. Procedures had to be established to coordinate the use of multiple sources for one-way light time tables, the lander coordinates and the Flight computer clock counter.

The fact that the tables and constants were not published until the software system was delivered also caused some problems. During the early stages of program deliveries the common data base software was not fully tested. As a result some programmers had to build in the option to access common data through either input cards or the common data base in order to meet their acceptance test schedules. Because the data base had not been published, the CE/CP team sometimes had to guess at values for constants not yet agreed upon, such as time of separation. Then six months later when the test case was repeated to assure the program was still the same as had originally been delivered, the user would discover that the program output was different than expected. This then led to a failure report, and time and effort would be spent trying to locate the source of the error.

The two shortcomings noted for the SDBD were both in areas where people failed to recognize the importance of the document. The concept is extremely valuable, and should be emphasized, implemented and enforced in the development of any major software system.

QUANTITATIVE IMPACT: A four to five man year level of effort was made to develop and maintain the SDBD for the life of the project. In order to have assured compliance with the tables and constants portion of the document, an additional one to two man year effort would have been required.

The SDBD was issued in two volumes, which combined were about two feet thick. Volume I contained interface descriptions for 11 VO-VL files and for 104 VL-VL viles. Volume II contained interface descriptions for 48 VO-VO files, plus four common data base descriptions. They were the Viking Lander I and Viking Lander II common data bases as they existed on the 360/75 and 1108 computer systems.

TECHNIQUE

NAME: FLIGHT OPERATIONS SOFTWARE SUBGROUP

SUMMARY: The Viking Project placed the responsibility for planning, coordinating and monitoring the development of the Flight Operations software system in the hands of a multi-agency Software Subgroup. This group was composed of four Software Systems Engineers who were individually responsible for Lander software, Orbiter software, Institutional software, and Project software system integration.

APPLICATION CONSIDERATIONS: The multi-agency managers responsible for the operational software system were each members of a Flight Operations Working Group (FOWG). None of them were experienced in resolving technical problems relative to computer science or large data management systems. When it became evident that it would take a significant coordination effort to reach agreements, the FOWG elected to establish the Software Subgroup for that purpose. Only those problems that could not be resolved by the subgroup would then be presented to the FOWG with recommendations.

RECOMMENDATION: The resolution of tradeoffs between hardware and software requirements and the management of software resources and schedules is frequently handled by non-software oriented personnel. This will be successful only if management understands the software development process and realizes that software must be treated on an equal basis with hardware. The management of the software development itself requires an ability to resolve technical program level problems in a manner that will not impact system level performance. It also requires the ability to foresee potential performance deficiencies early in the development process. For these reasons experienced software systems engineers should be made responsible for software development at the system level.

HISTORY: It is obvious to make a Software Systems Engineer (SSE) responsible for the development process of a relatively small software system. Such was the case for the Flight and Test Viking software systems.

The problem of developing a large multi-agency, multi-faceted software system is quite different. The Mission Operational Software System contained engineering, telemetry, sequence generation, command generation, flight path analysis, science analysis and imaging programs for both Lander and Orbiter. In addition it contained mission planning, tracking data, ground resource and institutional software. Separate teams were established to develop the software for each of these functions. Cognizant engineers were made responsible for requirements and end product testing, and cognizant programmers were made responsible for design, code and implementation.

But this left unanswered such questions as what programs were needed, are redundant functions being developed, what standards and procedures should be followed, how will the system function, can the system be made to operate within available computer resources, how will the programs be integrated to form a system, and what assurance is there that the programs will be adequately developed and tested. To resolve these and associated questions, four SSEs were identified. They were not made responsible for the software itself; rather, they were made responsible to assure that a viable and efficient system would be generated on schedule.

DESCRIPTION: The Flight Operations Software Plan identified the Software Sub Group as follows:

Planning, coordinating and monitoring of development of the Flight Operations Software System is the responsibility of the Software Sub Group (SWSG) under the direction of the Flight Operations Working Group (FOWG). The SWSG shall provide guidance for the functional design of the system, and shall coordinate, integrate, review and advise on the design, development and implementation of the system. The SWSG shall emphasize Lander and Orbiter software integration and shall resolve any software interface problems that may arise. Problems that cannot

be resolved by the SWSG shall be presented to the Flight Operations Working Group with recommendations. Specific SWSG responsibilities shall include:

a. Review and coordinate FOS software schedules;

b. Participate in the evaluation and coordination of the Flight Operations functional requirements so that Viking software requirements can be developed;

c. Review the software design generated in response to the Software Functional Descriptions and the LFOS Functional Specification;

d. Review the Functional Requirements Document and the Software Requirements Document to assure that the requirements have been defined as necessary for software design;

e. Identify problem areas where analyses are required to design an integrated TDS/VMCCC/Project software system that will meet FOS functional requirements;

f. Provide guidance for software planning, design, and implementation;

g. Evaluate the readiness of the software system for flight operations;

h. Resolve or recommend solutions to software interface problems involving MMC, JPL-VOS, TDS and VMCCC;

i. Monitor the implementation of the overall FOS software design, development and testing to assure that all interfaces, design requirements, and schedules are correctly and completely satisfied;

j. Evaluate the readiness of the FOS software system for integration into the TDS computer complex;

k. Evaluate the readiness of the TDS/VMCCC Mission Independent Software System to support Project software testing and implementation;

l. Coordinate FOS ground software interfaces with the on-board software and hardware.

The members of the SWSG were then identified to be an Integration Contractor Software System Engineer (ICSSE), a MMC Software System Engineer (VLSSE), an Orbiter Software System Engineer (VOSSE), and a MCCC Data System Project Engineer (DSPE).

The ICSSE was given overall integration responsibility for the Viking Project Software System. Principal duties specified included coordinating Orbiter, Lander and TDS/VMCCC software interfaces and interfaces of ground software with on-board software, integrating and publishing software development schedules and status reports, controlling adherence of the software design to the planned design, implementing MMC software at JPL, and assuring software configuration management.

The principle duties of the VLSSE were to provide status and schedules to the ICSSE, coordinate Lander schedules, exercise Lander software configuration management, review the progress of Lander software implementation, assure Lander documentation and test data generation, and assure the readiness of the final program products for certification and user acceptance testing.

The VOSSE duties paralleled those of the VLSSE for orbiter developed software.

The DSPE was made responsible to assure MCCC constraints were not violated, compile computer usage estimates for development and generate them for integration, coordinate MCCC data system constraints upon Viking, participate in MCCC data system configuration control, prepare MCCC data system integration schedules, and assure generation of proper data system documentation.

QUALITATIVE RESULTS: The accomplishments of the Software Subgroup members played a major role in delivering an efficient operational software system to the Viking Project on schedule.

The SSEs resolved numerous inter-agency disputes and problems, a. developed and implemented the Integrated Functional System Design, the Software Data Base Document, the Viking Software Guide, the Lander Orbiter Software Test Plan and the VMCCC Data System and Integration Plan. They developed data management requirements, collected,

controlled and enforced interface agreements, assured software deliveries complied with procedures, supervised certification, conversion, and acceptance testing, negotiated computer time, held audits and reviews, maintained schedules, wrote procedures, resolved computer system problems, and conducted the Preliminary and Critical Design Reviews. Finally, they gained the confidence of the engineers and programmers, ran the software, coordinated failure reports and redeliveries and integrated the system.

Whereas the SSEs were given a fair degree of latitude in carrying out their duties, the most significant difficulties they encountered were caused by decision making policies of the non-software oriented management directly above them. Four examples will be given to impress upon the reader the importance of management understanding the software process before events unfold rather than after the fact.

Management did not initially understand the limitations of computer systems. As soon as the system design was formulated, the SSEs conducted loading analyses studies which showed that three or four large mainframe computers would be needed. Management held firm to the decision that the system would operate in one real time 360/75 and one general purpose 1108. The SSEs were therefore forced to assign programs to computers under these groundrules. The final system included two 360/75 computers, each operating a little more than half the time, for real time and batch operations, plus two 1108 general purpose computers operating full time with a third 1108 available for emergencies and peak loads. Had management faced this decision early, more efficient program loading could have been realized.

Management did not always treat software on an equal basis with hardware. The SSEs requested that telemetry formats contain some additional time tag words required for data analysis. The request was rejected, and complex and inefficient software functions had to be developed to resolve the situation. This increased the running time for the decalibration and decommutation software functions and caused problems beginning with the third week of planetary operations. Some science data was incorrectly time tagged, not because of software errors, but because of the complexity of the requirements for distinguishing

old data from new data. The problem was quickly resolved by modifying the requirements and then implementing minor software changes.

Management rushed software development. The SSEs provided schedules that took into account both the software development scope and permitted a top down approach to system integration. Management directed that several real time and batch programs would be delivered up to five months earlier than shown on the schedules to support Software System One testing. The SSEs argued that the batch programs were not needed (because Flight computer software would at most be only capable of producing a memory dump) and that early program conversion, acceptance testing and integration efforts would jeopardize final program delivery schedules. This advice was rejected, and the programs were delivered early and on schedule. They were not adaptive, were unreliable, and could only be run in a "canned" fashion, being very limited as to what data could be processed. No Flight software was available for Software System One testing, so not even a memory dump could be taken. For that reason, no one even bothered to run the batch programs. Management learned from this experience not to rush future software deliveries.

Management did not fully understand the software integration process. The SSEs originally specified in the software plan that Lander/Orbiter interface integration testing would be conducted prior to program delivery. The reason for this was that the SSEs knew that a large number of errors would be uncovered. Management changed the plan to require program deliveries be made before integration tests could be conducted. There is nothing wrong with this as long as one is willing to accept the fact that most programs will have to be redelivered. But when that happened, management jumped on the SSEs for making too many deliveries; each was costly in resources because of the involved procedures and retesting that had to be followed. The end result was that the SSEs ignored the plan and reverted to their original approach, thereby controlling the number of redeliveries required. It is important to realize that first deliveries were incentive deliveries, and redeliveries were not.

QUANTITATIVE IMPACT: The amount of support given to the Software Subgroup SSEs is shown in table 1. The numbers reflect the average manpower levels for the years shown. The figures include the SSEs and their staffs.

	1971	1972	1973	1974	1975	1976	Manyears
ICSSE	1	3	4	7	6	4	25
VLSSE	0	0	1	4	1	0	6
VOSSE	1	1	3	6	6	4	21
DSPE	1	1	2	2	2	2	10
Totals	3	5	10	19	15	10	62

Table 1. Software System Engineer Manpower

The figures reflect all activities stated herein for the SSEs. In addition the ICSSE manpower includes developing all VL data management software, maintaining two MMC computer consultants, developing the common data base, developing utility programs, and maintaining three lander programs. The VLSSE effort also includes developing the VL time utilities.

Flight Software Development Overview

1.0 Introduction

- 1.1 The Viking Lander Flight Software System
- 1.2 Software Development Responsibilities
- 1.3 Quantitative Software Description

2.0 The Requirements and Design Phase

- 2.1 Organizing for the Task
- 2.2 Defining the Software System
- 2.3 The Program Design Phase
- 2.4 The Development Cycle

3.0 The Test and Integration Phase

- 3.1 Module Testing
- 3.2 Subsystem Testing
- 3.3 Integrated System Testing
- 3.4 Postscript

4.0 Lessons Learned

Flight Software Development Overview

1.0 Introduction

The Viking Lander Flight Software System was developed over a five year time period. Significant difficulties were experienced during the major portion of its development caused by Lander science and engineering hardware subsystem design changes. This overview emphasizes the major problems, their solutions, and the significant accomplishments that ultimately led to an effective and efficient Flight Software System which worked well during the mission.

1.1 The Viking Lander Flight Software System

The Viking Lander Flight Software System consists of a set of software modules, called the Flight Program, which reside in a Guidance, Control and Sequencing Computer (GCSC). The GCSC interfaces with Viking Lander hardware subsystems thru input/output channels and interrupt registers. The Flight Program was the semi-autonomous controller of the Lander. It was required to perform prelaunch Lander checkout functions, control Lander science and engineering hardware subsystem activities during interplanetary cruise, and perform Lander checkout and calibration functions while in Mars orbit. During the descent to the Martian surface the Flight Program performed navigation, guidance, and steering functions, and controlled telemetry format modes, power management, pyrotechnic firings and upper atmospheric scientific investigation subsystems. Once upon the surface of Mars the Flight Program was required to control the various scientific investigation instruments, perform telemetry data management, and control all uplink and downlink communications activities.

1.2 Software Development Responsibilities

The Langley Research Center was responsible to NASA Headquarters for the management of the Viking Project. Contracts were awarded to the Denver Division of the Martin Marietta Corporation to develop the flight software system for the Viking Landers and to the Jet Propulsion Laboratory to develop the flight software system for the Viking Orbiters.

This narrative is limited to a discussion of the development of the Viking Lander Flight Software System.

1.3 Quantitative Software Description

Several versions of the flight program were developed for the 18432 word Guidance, Control and Sequencing Computer (GCSC). The version that was launched contained a pre-separation checkout code overlay strategy. Therefore it is estimated that the delivered flight program used to support mission operations contained 20000 instructions, developed at a cost of 1609 man months. It should be pointed out that this development effort permitted the Viking Flight Team to uplink 60000 code controlling data words to each Viking Lander during operations. In addition to the flight program development costs, 494 man months were required to produce approximately 200000 instructions of emulation, simulation and diagnostic support software.

The documentation that supported the flight software development consisted of a Software Requirements Document (1000 pages), a General Design Document (500 pages), a five volume Program Description Document (1500 pages), two timing and sizing reports (500 pages each), a PDR report (500 pages) and a CDR report (500 pages), or a total of approximately 5000 pages.

The estimated effort expended by development phase for the flight program is as follows

Definition	8%
Design	22%
Programming	25%
Test	23%
G&C Analysis	22%

2.0 The Requirements and Design Phase

2.1 Organizing for the Task

The original concept for Viking software development was that a single unified management approach would be applied to all of the software systems. A Viking Software Integration Group was formed to develop and document "Standards for Viking Software Development" which listed the documentation, flow chart and basic configuration management requirements for all MMC developed software. Subsequently, the multi-agency management coordination effort required to develop the operational software system led to the decision that that system would be placed under a Mission Operations and Design (MO&D) Directorate and the Test and Flight software systems would be developed under the Systems Engineering Directorate. Coordination between these directorates was established by means of a Viking Change Summary (VCS) procedure that was designed to assure system wide visibility into all hardware and software component change traffic.

The Systems Engineering Directorate was responsible for the Flight software group, the Lander hardware component groups, the Systems Test Equipment group, the Systems Engineering group, and the Guidance and Control group for descent. A Lander Software Integration group was chartered to monitor the development processes of the Flight and STE software systems and to write a Lander Software Development Plan. The software plan established a Software Change Board whose primary purpose was to assess the impacts that hardware component changes had on the growth of Flight software and to recommend solutions or courses of actions to be taken.

The Lander Software Integration Group within Systems Engineering was in existence for a relatively short time; as such it was essentially ineffective in monitoring the flight software development. This task was accomplished by the Systems Design and Integration Group until September 1974 and the MO&D Lander Performance Analysis (LPA) group thereafter.

2.2 Defining the Software System

The Flight Software System was initially only partially defined. It then evolved through an extensive trial-by-error iterative process. The reason for this can partially be traced to a lack of trained software leads and

software management personnel who inappropriately viewed Flight software as being one of several independent components that collectively formed the Viking Lander system. It was in part caused by the January 1970 discussion of President Nixon to postpone the Viking Mission from 1973 to 1975. But it was primarily caused by the difficulties encountered by the hardware designers to develop lightweight, compact and sophisticated science instrument subsystems.

The concept of the Flight Program for Viking 73 included two computers and two programs. A Guidance computer would be used to perform the descent phase of the mission and a sequencing computer would perform the landed science sequences. Each computer would have its own resident software.

The Guidance and Control (G&C) subsystem was designed based on the two computer configuration and requirements for the descent phase were levied. With the G&C subsystem being designed independent from the Lander science and engineering subsystems, a consistent and well defined subsystem, which included descent software requirements, evolved.

Problems arose from the fact that each of the other Lander subsystems were developing independently, and were experiencing serious hardware design problems in areas of power, thermal, weight and packaging. Systems engineering was greatly concerned about these problems, and concentrated their efforts in resolving them on an independent subsystem basis. The functional throughput for the entire digital system was therefore constantly changing.

With one subsystem evolving around one computer and all other subsystems evolving around an unidentified sequencer, when the decision was finally made to incorporate a single but block redundant computer for Viking 75, the Flight software system definition faced two serious problems. One problem was how to interface a computer to an existing and disjoint set of subsystems and produce a consistent set of software requirements; the other problem was how to choose a computer for the task.

The first problem was eventually resolved in an engineering sense, but maintained a consistent position of creating problems that needed resolution throughout design, development, test and operations. For example, the design of the Flight program to use a single register for both telemetry and science I/O caused conflicts that went undetected until late in the integrated system test phase which required costly changes to the Flight software to fix.

Attempts were made to standardize the system interfaces but the resulting mixture was far from standard, thus forcing a complex software interface and a complex computer I/O structure.

The problem of how to choose a computer was solved using a novel technique - buy a computer that fits the software. The concept of "software first" was important in the fact that an adequate computer could be specified with a high degree of assurance as to its capacity for the job. In addition "software first" provided early subsystem evaluation allowing design criteria changes prior to hardware build. The objective of "software first" was to perform a detailed computer timing and sizing task for the Flight Program and to define computer architecture adequate for the required accuracy and program control.

2.3 The Program Design Phase

The program design phase began with the "software first" criteria for selecting a Guidance, Control and Sequencing Computer (GCSC). An executable performance analysis was conducted with a "procurement language" Flight program to provide verification that the computer, the G&C subsystem, and the flight algorithms were a compatible set. The approach taken was a two-pronged assault geared to produce a solution. The first was to perform a design level sizing and timing analysis on Lander engineering and science software tasks. The second was to emulate the hypothetical Flight computer in real time by microprogramming the descent portion of the Flight program on a Standard Computer Corporation IC-7000 computer set.

These analyses led to firm minimal requirements for the GCSC hardware design. However, at this point in the development of the Viking Lander, weight was the most critical problem faced by management; almost every Lander hardware subsystem was too heavy. For this reason, the third best computer was procured from a software point of view. The selected lightest in weight computer was adequate, but was poor in its instruction set and architecture, which made coding inefficient.

Although "software first" proved system viability, software costs and design problems were not adequately considered in the system design and interface areas. The system interface requirements coordination was accomplished by the Systems Design and Integration Group within the Systems Engineering

Directorate. Although the flight software development group assisted and was involved in resolving interface problems, the System group provided adequate coordination and requirements definition. If fault is to be discussed in providing necessary coordination, it was that the original coordination/ requirement definition was given to the G&C group. This was a correct approach for descent requirements; however, for engineering and science requirements the G&C group was not structured to handle interface requirements. The Flight Software Integration group then assumed the requirement definition of the Systems Design group; however, the Systems Integration group continued requirement definition for most science and sequencing requirements. Not until the Systems Design and Integration group was dissolved was the total flight software coordination accomplished within one group, i.e., Lander Performance Analysis within MO&D. Even with the constant shift of responsibilities, flight software definition was accomplished effectively.

The software requirements were functional rather than specific. They consisted of Descent G&C requirements, a Sequence of Events (SOE) generated by the Lander Sequencing Group, and software specifications based on a specific sequence of events. The G&C requirements were specified by using blocks of FORTRAN statements that only partially defined the descent software requirements. The S.O.E. was a project controlled list of events, commands, and times that the software should meet. This was intentional, since a specific sequence was required and the Systems Design and Integration Group did not want a general capability designed into the Flight computer.

Although the software requirements document did define a set of requirements for the software design, none were included for a software executive. In this area it was up to the programmers to use their best judgement in assessing the needs of the science community of users. It should also be noted that many of the requirements evolved as the systems integration group and flight software development group became knowledgeable of the hardware design, including the GCSC.

The difficulties for the flight software group to obtain firm and complete requirements relative to the evolving Viking Lander system were complicated by physical considerations. The Flight software development activity was located in a laboratory one mile away from the rest of the project.

Although an extensive Mission Planning/landed science strategy activity was going on, the flight software personnel were almost never involved to obtain an understanding of how the flight software should interface with the Flight Operations software and users. If it were not for the efforts of one particularly dedicated and competent individual who worked this interface, the development of the Flight operations and GCSC software would have produced mutually incompatible designs. After both software systems were well into integrated testing, the Missions Operations management realized the severe limitations of the Flight software to the mission strategy and created a full-time group responsible for the integrated software design and test activity.

2.4 The Development Cycle

The development cycle took into account the inadequacies that existed during the software design phase. System integration held meetings between users and programmers, using the flow charts as the point of departure. In this manner, software capabilities could be discussed in a coherent fashion to iron out single level problems and inconsistencies. As the flow charts were accepted on a one by one basis, coding could begin.

The documentation requirements were standard in accordance with Viking software development, as specified in the Lander Software Development Plan. No software standards were imposed on the Flight Software Group, but guidelines were available. The Flight Software Group therefore organized their own standards relative to such things as labeling conventions, subroutine conventions and coding standards; however these standards were not consistently enforced.

To combat the changing and diverse requirements inherent with Lander hardware subsystem development, a modular system design had been generated that employed an Operating System as the module control manager. The Operating System allowed both absolute and relative time scheduling of modules. Modules were assigned one of five levels of execution priority. All I/O and interrupt service was handled by the Operating System routines. Module interface to these routines was controlled by Macro calling sequences. The levels of standardization allowed modules to be changed and modified with a minimum impact to the system design. Since changing requirements continued on into

operations, the decision to modularize subsystem and function software and provide standard Operating System sources proved to be highly successful in accommodating change with confidence.

The modular design also simplified the development cycle. The module design would be coordinated with the user, as described above. It then would be coded and debugged in the IC-7000 emulator. Unit testing then would take place on a module by module basis. Collections of modules would then be tested by interfacing hardware subsystem simulators with the IC-7000. Finally, the software modules would be placed in the GCSC when it became available, and integrated system level tests with Lander hardware components would be conducted.

The flight software development process was hindered by the lack of the assembler that was adequate for the task. The first assembler provided fixed address code. About halfway through the development phase, the Flight Software group made the decision to develop a relocatable assembler, which was needed to cope with the change activity and the multiple revisions of the flight code. Later on a decision was made to use the IBM 370 assembler, which was also being used for the Flight Operations Command software. After one or two assemblies were made with this assembler, the project manager made the decision that all future updates would be accomplished by manual "patches" using the basic GCSC octal code. This was done even through a great many changes had to be "patched" because of problems discovered in integrated testing (see section 3.3). The result was that a "listing" of the flight code had to be created post facto using the IBM 370 assembler, which was a laborious and costly process. Because of the patching, i.e. using "jumps" to unused memory and jumping back, the resulting listing of a contiguous function was not contiguous in core. This proved to be a burden for Flight Operations during the mission in reviewing Memory readouts and making software changes via uplink.

The assembler should have been developed for both the Flight and Command software systems and used through all development cycles. Major revisions after testing could then have been accomplished by proper reassembly of the flight program.

3.0 The Test and Integration Phase

3.1 Module Testing

The IC-7000 computer was subdivided into two sections. Each section was controlled by a separate programmable CPU. One CPU was microprogrammed to provide an emulation of the instruction set of the GCSC. The second CPU was microprogrammed to provide a separate and distinct control instruction set that was used to provide analysis, monitor and trace functions, and to simulate the discrete, interrupt and I/O register hardware of the GCSC.

By this technique the module testing could be accomplished by means of the control CPU which could activate the GCSC emulator CPU, monitor the emulation, and provide printed output status reports to a SC-4000. These status reports described the data flow and external responses of the emulation.

Thus each module could be tested individually under control of the Operating System to assure they met their design prior to testing them in conjunction with the hardware system they were to control.

3.2 Subsystem Testing

The IC-7000 computer resources were inadequate to permit the module testint portion of the development phase to overlap the subsystem test phase. These phases were required to overlap because of constant changing software requirements. A second IC-7000 computer system was purchased to solve the problem. This eventually provided a system for component integration testing and a system for module development testing.

The most significant problems encountered during subsystem component hardware/software testing were the lack of good development tools, testing aids and a stable laboratory environment. It was a circular problem that could be traced back to the lack of detailed software requirements.

Flight program subsystem testing would identify an inadequacy in the laboratory control software system that accessed and controlled hardware subsystem simulators. The laboratory software would be modified, and then the Flight program would identify another need.

Subsystem testing slowed to almost a standstill. No Flight program development was being performed because the development bed was in a constant state of flux, and the development bed could not be completed due to

continual changing requirements. By placing the laboratory software under strict change control and performing extensive laboratory qualification tests the process of conducting subsystem testing finally was able to proceed at an acceptable rate to support the development cycle.

3.3 Integrated System Testing

The Viking Lander Proof Test Capsule (PTC) was built as a third Viking Lander to support integrated system testing. It was similar in every respect to the two Viking Landers that were sent to Mars. An analog-digital Hybrid computer system was developed to simulate the Viking Lander descent through the Martian atmosphere to touchdown on the planet. During integrated system descent testing, the Hybrid computer both modeled and bench tested the descent science and engineering hardware subsystems to test the responses those subsystems would sense during actual operations. Control of the PTC could be accomplished only through the System Test Equipment at MMC. GCSC memory maps were generated by the Flight Operations Software System at JPL, sent over high speed data lines to Denver, and input to the PTC Flight computer via the STE.

Integrated system testing established that the computer was adequate for descent. The digital interface logic to the descent science and engineering hardware subsystems proved to be ideal for precise sampled data digital control. The Flight Program Operating System was also well designed in this area.

The computer was designed for power cycling control of I/O and memory when not in use. This proved to be an excellent concept and worked well during the mission.

Integrated system testing uncovered deficiencies in the computer hardware design which required costly Flight software changes and workarounds to be implemented as well as imposing mission constraints on the Flight Team. These deficiencies included subsystem conflicts caused by using a single I/O register for both telemetry and science control, and noise induced spurious false interrupts being sent to the computer during I/O switching.

Due to the lack of a fully integrated lander hardware/software design plan, the landed science and landed telemetry modules had not made adequate use of the priority and scheduling features provided by the executive. As a result of this expensive fixes to both the Flight and operational software

were required based on the results observed during integrated hardware/ software testing of mission sequences.

3.4 Postscript

For descent, a strong analytical systems group was available and developed the basic descent equations and logic in FORTRAN simulations prior to development of Flight code. As such, the descent design and testing went through a fairly orderly process (design - code - module - closed loop emulation testing using real time hybrid modeling of the vehicle - integrated hardware/software testing) with a minimum of problems. Plus it worked in perfect harmony with the Mission Operations Software System during the actual flight.

For landed operations, cruise and preseparation checkout, the systems engineering group was not software oriented enough. Because of this, not enough manpower of the right type was placed on insuring a total integrated hardware/software design between the flight vehicle, the Flight software, and the Mission Operations Software System. This led to problems in integrated testing with the Mission Operations software and, due to the lead time of launching the Flight software, placed a heavy burden on revising the Mission Operations software. Due to the integrated nature of the Flight and Mission Operations software a different management structure than was used should have been used.

As was done in descent, the system analysis group working with each subsystem or experiment should have developed a software requirements document which would have specified the functional flow chart level requirements of each function (for instance the Gas Cromatograph Mass Spectrometer or the Uplink process). At the next stage, the relationship of each program to implement these functions could be defined. In this manner two programs implementing the same function, such as the Lander Sequence of Event program and the Flight program, would derive their requirements and models from a common source.

4.0 Lessons Learned

The Viking Flight program development was much more costly than it should have been because of the initial lack of trained software managers, visibility into software problems, and an insufficient concern about these matters by management. Despite the many and significant problems faced by the Lander hardware component subsystem developers, software development and integration problems should have been brought to Director level attention for resolution at an earlier date than they were.

The Lander development organization was structured similar to the Viking Project organization, which lacked the technical ability to monitor and manage development to the root level. The problems generated by this organizational structure eventually led to a series of independent audits during the period in which subsystem testing slowed to almost a standstill. It was not until then that the difficulties encountered by the software developers came to the proper attention of the System Engineering Director. Only then were software problems treated on an equal basis with hardware problems, the result of which was that the ever changing requirements and lack of visibility received immediate response.

Although the flight software development had difficulties, it should be remembered that Viking was unique in its hardware complexity. Many hardware changes were required to meet weight limitations, additional redundancy and budget cuts. Even with knowledgeable managers in the area of flight software design, the hardware changes would have taken place and the flight software revised accordingly. Perhaps the lesson learned in this respect, is recognition that a flight software program must change as the hardware is revised. A schedule that incorporates flight software deliveries, reassemblies, etc. at the outset of the project was badly needed. A flight software development group must be structured to accommodate requirement revisions.

The flight software development group in Viking did accommodate numerable changes although there was a prevailing attitude that a change in requirements was indicative of someone not doing their job rather than an ongoing software development process that should be expected.

Another area that should be addressed is defining within the test program the integration of flight software utilization in a systems test configuration. Although the flight software executive was used in the test program,

the landed operations software was not used until the Plugs Out Test (POT). This created a situation where many people became familiar with only the test software, and after launch, were not aware of the limitations and constraints of the flight software.

In conclusion, even with alleged faults of poor requirement definition and lack of knowledgeable software managers, the flight software design was excellent, and, it resulted as a combined effort of the Flight Software Development Group, Systems Design, Systems Integration, Flight Software Integration Group and MO&D Lander Performance Analysis Group. Those using the software became aware of the capabilities and constraints, and utilized the flight software in its fullest to perform an excellently executed mission.

TECHNIQUE

NAME: EMULATED ON-BOARD COMPUTER (GCSC)

SUMMARY: A ground-based, microprogrammable computer was used to emulate the Flight Computer throughout all phases of GCSC software development. This software-first approach to the Lander computer system development facilitated computer timing and sizing specification, permitted early development of critical computer programs, and provided considerable visibility into test and evaluation activities.

APPLICATION CONSIDERATIONS: During the early phases of software and hardware definition it became evident that several problem areas could be satisfied by an emulation approach. Accurate sizing and timing estimates could be obtained by coding a hypothetical computer representing the class of available spaceborne digital computers. Computer memory capacity and speed have significant impact on power, weight, and volume; all were critical performance characteristics. In addition, we were committed to an Analog-Hybrid six-degree of freedom simulation of separation from the orbiter through soft landing on the surface of Mars. Integrated testing with real sensors and actuators was deemed important in this process. Emulated representation of the on-board computer would provide bit-for-bit fidelity. Unlike an "interpretive Computer Simulation" (ICS) approach, emulation offered the prospect of real time test and evaluation. Finally, most of the classic problems inherent with punched tape and limited visibility associated with on-board computer usage would be avoided.

RECOMMENDATION: Emulation is now a proven concept in computer system development. The approach offers the same type of advantages with any embedded computer system where the target computer is undeveloped or unavailable. The power of the technique is illustrated by the fact that the first flight computer was substituted for the Emulator and was running the descent program in a full simulated environment within just one week of computer delivery.

HISTORY: Conventional approaches to software development have utilized the "Interpretive Computer Simulation" (ICS) wherein the flight computer is simulated by interpreting its instruction set and action through programs written on a host computer. A typical response time, while quite variable, might be as much as 500 times slower than the actual machine. It is also difficult to anticipate, and thereby program, situations which can occur in a real time environment. Consequently, in order to develop software in a real time environment which could include actual devices, we decided against the ICS approach and pursued the emulation course.

Although the characteristics of the flight computer were not specified, sufficient general information was available early on the Viking project to convince us that microprogrammable computers then in existence could provide a real time emulation of the future flight computer.

It was recognized at the onset that an emulation would not, and could not, provide a one to one time relationship with the actual machine on an instruction by instruction basis. Indeed, for this to theoretically occur, the basic cycle time of the microprogrammable computer would have to be a submultiple of that of the actual machine. The stress, in a real time sampled data system, is upon accomplishing identical processing over a sample period.

The microprogrammable computer chosen was a Standard Computer Corporation IC-7000. It consists of two processors both of which are microprogrammable. One is the Central Processing Unit (CPU) and the second is labeled the Input/Output Processor (IOP). Each incorporates a control memory whose contents define the machine, i.e., the instruction set, interrupt structure, etc. for that processor.

The initial use of the complex consisted of an emulation for a hypothetical flight computer. Inasmuch as the actual machine had not been specified, it was felt that software development could proceed with a typical computer representing the class of candidates available. This emulation aided sizing and timing specifications for the GCSC, helped in early development of the man/machine and various system interfaces.

The characteristics of the microprogrammable complex dictated that flight computer emulation be done on the CPU and that the IOP should be used to implement I/O related aspects and to communicate with all

external devices including the man/machine interface and control. Consequently, a basic IOP language was defined and implemented in conjunction with development of the hypotehtical flight computer emulator.

The IOP language implementation proved to be an on-going, continual endeavor throughout the life of the project. A new instruction would be implemented as a need was generated. IOP microprogramming included implementation of an interrupt system. The IOP, in addition to providing the interface linkage for the CPU to implement flight computer I/O, also functioned as overall system problem control. Its language was used to perform tasks such as data recording, starting and stopping of runs, establishing an initialization state, etc.

Flight computer emulation began when selection of the flight computer was made. The emulation was mechanized in respect to the logic design of the GCSC and not in respect to the information found in what is commonly termed "programming manual". Design changes to the GCSC were tracked and implemented in the emulation.

As the overall system needs and requirements developed, additional requirements were forced onto the CPU microprogram over and above the basic microprogram related to real time flight computer emulation. The end result real time emulation bore little resemblance to the initial design because of these factors and changes in GCSC design.

Three emulations have been mentioned in the above - (a) the hypothetical computer; (b) real time GCSC; (c) IOP. Another version of the GCSC emulation was also developed and warrants discussion. This was called the "GCSC Trace" emulation. Its purpose was to provide a listing to the user to aid in program development. User control was provided to allow printout to occur only as desired. The printout provided machine state information following execution of each instruction and included the instruction, its location, various register contents, and time. TRACE, in actuality, consisted of a CPU microprogram and an IOP program. Whereas much of the GCSC instruction emulation was identical to that of real time, the interface to the IOP was drastically different. TRACE could be considered an analogous to the ICS type operation.

Two versions of TRACE were eventually developed. The preliminary version used IOP programming to keep track of time-related aspects,

e.g., instruction execution time, sample time, delay times, etc. As a result, the operation time was quite slow (about 300:1). A later version incorporated these features in the microprogram and resulted in a speedup (around 30:1 and in some unique instances could even run faster than real time). The latter microprogram was, of course, considerably more complex than the former.

As the characteristics of the GCSC became defined, some rather drastic incompatible features became apparent in respect to the microprogrammable machines capability. This was particularly true in respect to GCSC I/O. The GCSC design was driven by power requirements and interfaces were required with several unique external devices. The result was several I/O registers of varying length and unique time out periods. Also, the GCSC had partial power down or "go to sleep" capability. It was readily apparent once these characteristics became known that the basic microprogrammable computer could not handle the implementation requirements. Consequently, a hardware design modification was made to the computer to aid in the microprogram implementation.

DESCRIPTION: Four distinct microprograms were developed and are described separately.

A. HYPOTHETICAL COMPUTER EMULATION - The initial computer emulated was that of a "computer" typical of the class of computers available. The general characteristics were:

24 bit word
2's complement arithmetic
24 instructions

A simplified I/O and interrupt structure were included.

B. IOP LANGUAGE EMULATION - The instruction set incorporated into the IOP evolved over the life of the project and supported over 200K words of support software. The general features are:

36 bit word
2's complement arithmetic
163 instructions
interrupt system

Virtually the entire control core (2048 18 bit words) was utilized

in the process. The IOP micro language bears little resemblance to that of the CPU.

The IOP language supports all standard I/O peripheral device interfaces in addition to the unique devices associated with the Viking system.

C. GCSC REAL TIME EMULATION - The general characteristics of the GCSC are:

24 bit word
47 instructions some of which have several subsets
2's complement arithmetic
8 level priority interrupt system
10 I/O registers
18 K memory
2 K protected memory area
3 soft index registers
multi level indirect addressing
some error detection logic
Sleep mode capability
Programmable Timer

The real time emulation is bit-for-bit functionally equivalent with the GCSC. Emulation design was performed from the standpoint of the logical design of the GCSC. The priority interrupt system was emulated, for example, on a one-for-one equivalent of the flip-flop structure involved.

Several areas of incompatibility were evident between the GCSC and the microprogrammable computer as the GCSC design progressed. In many instances, use of the microprogrammable machine capability was compromised or bypassed in order to meet functional equivalence. Some examples are as follows:

(a) Index Registers - 3 soft index registers were utilized in the GCSC whereas the micro computer has several hardware index registers allowing indexing to be done as part of the instruction fetch with no additional time penalty. Use of this feature as is would result in a non exactness of memory contents between the two machines (the 3 memory cells assigned

to indexing). An exact equivalence could be achieved, of course, via appropriate microprogramming. However, by so doing, all instructions which are indexable must be tested, when executed, to determine if the feature is called for. The result is a critical time loss in addition to consumption of control core.

A compromise solution proved satisfactory in this case. Hardware indexing was utilized but the 3 memory cells were maintained via microprogram. That is, any instruction which modified these cells also modified the hard index registers.

(b) Multi-Level Indirect - This feature of the GCSC was not designed into the basic hardware capability of the micro machine. Although it could be accomplished via microprogramming, the same penalties exist as for index registers - excessive execution time and core consumption. This feature was discarded as a programmer option thereby eliminating the problem.

(c) Divide - The GCSC divide instruction contained some rather peculiar features (for example, the l.s.b. of the quotient was always "1") which disallowed use of the divide hardware in the micro machine and forced microprogramming of the GCSC divide algorithm. Consequently, emulation divide time was significantly greater than that of the GCSC. This condition was, however, of little consequence since flight programmers avoided the use of this instruction because of its pecularities. In fact, its main claim to fame was as a low power time killer - an instruction which consumes little power (only one operand memory reference) and takes a relatively long time to execute!

(d) I/O - Major difficulties in emulation of the flight computer via the micro program technique were associated with I/O. Problems therein are amplified when real time is involved. Consider the differences between the two machines. I/O in the micro computer is handled through

the Input/Output Processor which is linked to the CPU via common memory and through interrupts. That is, either processor may interrupt the other at the micro level. Interrupt interpretation and data transfers are mechanized through memory interrogation. Both processors have access to CPU control core. Thus, data transfer is parallel and a degree of handshaking is implied in mechanization. Once the information is obtained and interpreted by the IOP, it can then establish the necessary linkage with the external devices.

The flight computer I/O design, by contrast, reflects extreme sensitivity to power consumption. All data transfers are serial, register lengths are variable and transfer times are device dependent. A total of 10 registers, from 3 to 35 bits in length, are incorporated. Four of these relate to data transfers with external devices and are independent of one another. An "I/O complete" or "time out" interrupt is generated when a register is loaded or emptied. One register may connect to several external devices and, consequently, would have several time out periods. The other six registers relate to discrete registers, error detection, power control, and one to indicate which register has "timed out." Thus, more than one operation can be in process simultaneously and many distinct timing intervals are possible. The problem of using a single, parallel data linkage to simulate the four distinct serial linkages is complex and the timing requirements further complicate the problem. Once the flight computer I/O design began to crystallize, it became quite obvious that an I/O bottle-neck would exist if modeling were limited to the basic micro machine capability. This became painfully true in respect to the timing involved. An overwhelming amount of microprocessing time would be consumed in attempting to maintain the four channel simulation.

Additional hardware was designed and added to the micro machine to solve the problem. The various timing periods

involved were produced via hardware, thereby permitting the data transfers to occur in parallel (at the micro level) upon time out and, consequently, simulate the serial transfer.

The I/O instruction structure also created problems since several fields required decoding in determining instruction intent. This implies a considerable amount of time and core overhead in decoding. Additional hardware modifications were made to reduce the effect, especially in time sensitive areas.

(e) "Sleep" Operation - Because of power constraints, the flight computer allows a partially powered down mode to be established. It may then "wake up" as a result of external interrupt or termination of a specified time interval. This feature was emulated by suspending instruction execution during the sleep interval while remaining receptive, at the micro level, to those elements corresponding to the "awake" portion of the flight machine.

(f) Timers - In addition to a sampling clock and the discussed I/O time periods, a 1 ms. and a 12 ms. timing period were designed into the flight computer. These items were associated with the power conservation and "sleep" mode. Emulation of these features via the micro code consumed too much core and execution time. The problem was solved by augmenting the hardware to provide the capability.

Although the above items are significant and did have considersiderable impact on the emulation process, the achievements of the technique are a credit to its flexibility. Modifications were necessary only because of the real time aspects of the problem.

Table I is a timing comparison between the GCSC and the emulator. A variety of conditions are evident - from near exactness to a wide variation in both directions.

INSTRUCTION OR INSTRUCTION TYPE	FLIGHT COMPUTER	EMULATOR
ADD	8.68	7.70
SHIFT	4.34 - 23.44	8.41 - 17.34
DOUBLE ADD	13.02	14.52
MULTIPLY	83.33	31.35
DIVIDE	123.26	168.68
STORE	8.68	8.45
INDEXING	4.34	0 (H/W index registers)
INDIRECT ADDRESSING	4.34	1.4 (H/W index registers)
INTERRUPT	8.68	23.62
OUTPUT	8.68	17.15 - 113.2
INPUT	8.68	10.15 - 24.5

Table I. Emulation Timing Comparison (IN μs)

Input/output and interrupt aspects produced the greatest variation in timing and the poorest results. This was primarily due to two factors: (each flight computer I/O instruction was in reality a group of instructions, i.e., several subsets existed for each thereby requiring an extensive amount of decoding at the microprogram level; (b) communication linkages between CPU and IOP required servicing time.

Other items such as multiply, indexing, and indirect addressing tend to counteract the I/O timing. A reasonable degree of comparison exists for other instructions.

Again, individual instruction time is not the critical factor. The ability to perform the required flight program processing within a sample period is the critical item. For the instruction mix involved, emulation execution time was slightly faster than GCSC execution time.

Many features were also implemented in the micro program as an aid to overall system operation. For example, a read and write of flight program reset points was implemented. This feature allowed the user to run for any period of time, stop and store the machine state. Similarly, any stored reset point could be read and the

operation reinitiated. The machine state is determinable and may also be established via micro program whereas this may be impossible or difficult at a higher level.

D. TRACE EMULATION - This microprogram, in association with an IOP program, functioned to provide the user with a printout, or listing, of the GCSC state (instruction, location, register content, effective address and contents and execution time) after each instruction. Much of the microcode is identical to that of the real time emulation but many differences exist. TRACE must store all state conditions and make them available upon request. Also, the time related aspects have a different connotation than in real time and micro code in these areas differs considerably. TRACE was used for initial program development and in attempts to gain insight into problem areas.

QUALITATIVE RESULTS: The emulation approach enabled us to obtain reasonable timing and sizing estimates for the spaceborne computer prior to issuing its specifications. This was accomplished through emulation of a hypothetical machine. This approach also accelerated the laboratory system design and mechanization whereby interfaces between the various components, the man/machine interface, etc. were mechanized and operational early in the program. Also, operational runs were made to verify descent algorithms using this language before the GCSC specifics were known.

After the GCSC characteristics became known, the hypothetical emulation was phased out as the real time and trace emulation were developed. These items were operational at least a year in advance of GCSC availability. Consequently, program development was well underway when the actual machine became available.

Insight and visibility into program operation were possible through the approach which would not have been possible with the actual machine. Aerospace computers typically have little visibility provided to the user.

One aspect of the approach which proved quite useful as an aid to program debugging was the ability to examine conditions at a lower level than normal machine instruction level - the microcode level. This

technique was used many times to "look" at the internal machine state and thereby determine the problem cause.

Emulation also provides the capability to determine and thereby record machine state. Within the system framework, it often became necessary to stop the run after a long period of elapsed time and record all conditions as a reset point so that the exact state could be later reestablished without processing from the initial starting point. This is relatively easy to do at the microcode level but may be extremely difficult at the program level. Indeed, once the GCSC became available, a considerable amount of effort was expended in accomplishing the same task.

Emulation offers the user a large degree of flexibility as is illustrated by the changing instruction set of the IOP. As needs develop, changes can be made to accommodate that need. This capability is not realized without some expense as the user must develop his software and the personnel at the micro level must be intimately familiar with the machine.

In respect to the microprogramming effort per se, many difficulties were centered around a lack of control store. Both IOP and CPU control store consisted of 2048 18 bit words. The real time emulation was a constant process of fitting in system or GCSC design changes without exceeding core or execution time limitations. IOP control core was completely consumed via emulation of the IOP language. The final version of TRACE (Fast Trace) also consumed the entire CPU control core.

Some additional comments in respect to real time emulation are warranted. It should be understood that real time emulation is not automatically attainable. It was possible for Viking because the GCSC was a relatively slow machine. The microprogrammable machine must be a basically faster device than the computer to be emulated.

Careful attention should be given to the compatibility of the two computers if at all possible. Otherwise, considerable difficulties may appear as the emulation proceeds. This situation was painfully evident in respect to the GCSC real time emulation. Characteristics of the microprogrammable computer were completely ignored in respect to GCSC selection and its specifications. The result was a series of "find-a-

solution" processes culminating in many instances with hardware modification to the micro machine.

QUANTITATIVE IMPACT: Table II summarizes the manpower expended in the different Viking microprogramming tasks.

PROGRAM	MANMONTHS
Hypothetical Computer Emulation	4
IOP Language Microprogram	16
Flight Computer Real Time Emulation	20
Flight Computer Trace Emulation	8
Hardware Modifications	10

Table II. Manpower

Manpower consumed in the microprogramming effort includes a learning period. In respect to the IOP effort, the figure represents the manpower consumed over a five year project lifetime rather than an amount required to begin utilization of the resource. Basic operation consumed approximately six man months but, as the system developed, additions and changes were made to the instruction set and the operational philosophy. The result was a more or less continual, low key effort.

Flight computer emulation manpower consumption reflects the effort expended in learning associated with the flight computer - an understanding of its operation at the logic design level and in tracking the design and its changes from the conceptual to the developed stage over a three year period. As design changes occurred, changes were forced upon the emulation. Further, as previously stated, the micro program involved implementation of system functions. Probably 70% of the manpower would be attributable to emulation of the I/O instructions.

Manpower used for trace emulation reflects basic design and checkout and also tracking of changes made to flight computer design.

The figures presented also represent expenditures involved in generation and checkout of programs used as checkout drivers or "micro-program diagnostics". Approximately 15% of the manpower could be attributed to this activity, support of Flight Software development over a three year

period and documentation and support of Flight Software development over a three year period.

Hardware modifications relate to those items previously discussed which were incorporated to satisfy I/O requirements.

TECHNIQUE

NAME: VIKING LANDER COMPUTER EXECUTIVE PROGRAM (EXEC)

SUMMARY DESCRIPTION: This program was conceived and developed to act as a mini-operating system for the on-board computer. It was designed to accommodate and coordinate the variety of mission functions requiring computer services on board the Viking Lander. Common services performed include input/output, communications control, time reference, scheduling, and sequencing control. In essence the task programs could use the facilities provided by the virtual computer (EXEC) without regard for possible interference with concurrent tasks, or irrelevant detail of device operation.

APPLICATION CONSIDERATIONS: Several distinct mission phases were identified early: subsystem checkout prior to landing, control over the descent to the Mars surface, and science instrument operations while landed. Within each phase, several devices could be active concurrently and some devices were active over more than a single phase of operation. Sufficient commonality was observed pertaining to device services and scheduling functions that a centralized executive control program appeared attractive. Attendant overhead for a generalized executive had to be justified in an environment where low power and weight were prime considerations.

RECOMMENDATION: The Central Executive Program concept is sound and more widely accepted at this time (mid '76) than it was five years ago for space applications. The application program isolation provided in this approach facilitated necessary change of external functions. The 10% critical timing overhead and memory requirements (4.5K words) are now considered quite reasonable in view of the capabilities provided. Initial resistance has been overcome only gradually as the role and function of centralized management of computer resources has become revealed and appreciated.

HISTORY: The Viking Lander Computer (GCSC) is a general purpose digital computer with relatively small and conventional instruction set (27 Op codes with sub codes for I/O). The I/O is a special design that provides for serial data paths to lander subsystem, and accommodates several levels of interrupt.

Control of many lander subsystems was assigned to the computer (GCSC) at the outset: all of the guidance, navigation and attitude control; sequencing of separation, parachute deployment, radar activation, and rocket engine firing; and telemetry data acquisition control. Early in the project, control over the science instruments following soft landing was folded into the GCSC, thereby eliminating a separate controller/computer. Serial I/O through shared registers in the GCSC implied built-in conflict possibilities and required I/O register management. The flight control problem was based upon cyclic computations over 20 msec, 40 msec, and 1 sec periods, while the science devices require task scheduling with one second granularity. Once landed, control over the science and lander subsystems requires commanding each device individually with special control sequences spaced in time. The communication subsystem provides a means for modifying or changing these command sequences on a daily basis from the control center at JPL.

Initially, separate executive programs for each mission phase were considered. Analysis showed that there was considerable overlap at the phase transitions and that centralized control throughout the transition period would be beneficial. Thus, a single executive, providing general services as well as phase peculiar services, was chosen for development.

The EXEC program provided the necessary management of computer facilities so that tasks could be active concurrently without concern for one another. The resulting isolation provided considerable simplification of the application tasks, and assured centralized coordination of computer resources.

The EXEC program remained remarkably stable throughout the project development. In retrospect, some services provided were rarely employed. On the other hand, additional facilities to aid in specification of

event and time dependent actions would have aided understanding and facilitated change.

DESCRIPTION: The Viking Flight Program consists of a 'virtual machine' operating system known as the Flight Executive and a set of user application programs to control the Prelaunch, Preseparation, Descent, and Landed Phases of the Viking Lander Mission. The Viking Flight Program is permanently resident in the memory of the Guidance Control and Sequencing Computer (GCSC), and provides the capability to control Viking Lander functions from prelaunch GCSC memory load through the duration of the Landed Mission.

The Viking Flight Program is redundant in the sense that it resides in both memories of the block redundant GCSC. Both program loads are identical and provide full mission capability. The system is not dynamically redundant in that only one GCSC block has control at any given time and no computer-to-computer communication exists. Both computer blocks may be independently operated and checked out prior to separation from the orbiter, and either computer may be switched on or off during the Landed Phase; but only one computer block will be powered on at any time. The decision as to which GCSC block is to be employed during the Descent Phase will be made by Flight Operations and commanded via the Orbiter/Lander interface.

The Viking Flight Program provides command and control capability to the Lander subsystems employed for the following: communications with Flight Operations, experimentation, navigation, guidance, steering and control, data processing, power distribution, and pyrotechnic operations. Operating within the constraints of the Lander and its subsystems, the Flight Program can be activated at any time following load of the GCSC memory.

A significant portion of the Executive and Lander subsystem software may be viewed as an extension of the GCSC hardware. In effect, the code associated with I/O, flow of control, timing, and related basic functional elements of the computer, actually transform the GCSC into a distinct, but related processor. This new or "virtual" processor presents an altered interface to the user which at once facilitates requests

for scheduling, timing, and I/O service functions while coordinating like services, managing available resources, and coping with conflicts.

It is useful to examine and describe this transformed machine for at least two reasons:

First of all, the virtual machine is much simpler to employ from the individual user's point of view. Details of low level timing, hardware communication protocol, and accommodation of shared usage are of no concern to any one user - each views the machine as exclusively his. Programs describing particular application tasks are easier to understand and test since clarity of function is not buried in the extensive detail required of cooperating process implementation.

Secondly, the virtual or transformed machine may be treated itself as a device - a device possessing an augmented instruction set. Each pseudo-instruction or directive is characterized by a certain execution time, an assumed initial state of the machine, communication register utilization, the function performed, and the resulting state of the machine. The entire virtual machine may be designed, tested, and evaluated in a manner similar to that used for the hardware machine itself. This observation has important implications with regard to a basis for acceptance testing and selecting departure points for formal verification.

Continuing with the secondpoint above, a suitable virtual machine "integrity test" would be a program exercising the augmented instruc-set in a manner designed to check the extremes of virtual machine operation. Such diagnostics concentrate activity in both time and space in a manner much more severe than encountered in a typical operational environment. Successful completion, however, assures proper performance whenever these functions are employed within an actual application. Proper execution of hardware instructions, once evaluated, provides a basis for proceeding with software testing. In like manner, proper behavior of the augmented instruction set should be the basis for application programming and evaluation with respect to the virtual machine.

While it is possible and even beneficial to construct layers of virtual machines, each built upon the primitive operations of a more elementary machine, a single level will be described below for the

GCSC. Briefly, the augmented instruction set includes such operations as scheduling (ENQUE, DEQUE), input/output (IOR), power state control (TURNON, TURNOFF), and interrupt control.

The virtual machine is perhaps best understood from the application programmers' point of view.

Control of the virtual machine was based upon the hardware interrupt structure. At the top of this control is the highest priority and thus most urgent interrupt. Any unmasked interrupt will cause control to pass to that level and the associated processes initiated. Upon completion of that level's processes, the interrupt is cleared allowing control to pass to the next pending interrupt level. If no interrupts are pending, control drops through to the software controlled Schedule Stack, POP, which initiates the next appropriate task. Stacked tasks are established from an initiation time and specified priority determined during the scheduling processes of the Forty Millisecond and One Second Scanners. There is always one task at the bottom of the Stack, the Power State Switching Program (PSSP). This lowest priority task places the hardware into a "HALT" or "SLEEP" state depending upon the amount of time available until the next scheduled activity. Only external demands for attention will disturb the computer from the SLEEP state, while any unmasked interrupt may cause control transitions in any state.

Once POPped from the stack a user program has access to both the hardware Operation Codes and the commands provided by the virtual machine. As indicated, a task may be interrupted and suspended in favor of a more urgent task. In this event the suspended task is PUSHed into the stack until ultimately resumed via a POP.

The commands of the virtual machine provide services for the user program. In general these commands are requests for an operation that must be coordinated or scheduled along with a host of similar requests from other user programs for specified delays, task activation, serial I/O, and power group control.

The virtual machine manages the serial I/O registers. Peculiarities of register devices and sharing of facilities are accommodated. There was a close interplay of the cycle complete and real time interrupts (CCI and RTI) and associated processes. Timing and control requirements

dictate the Memory Readout and Telemetry Modes execute at this level of control. A Wait Bit Processor indicated how a user task may continue execution following completion of a specific I/O operation.

The user may be connected to specific external interrupts via the external interrupt handler. Here, the virtual machine filters all but the one of interest from the bits of the External Internal Register (R5), temporarily passing control to a special entry user task which completes appropriate housekeeping and initiates I/O and scheduling requests. The user is allowed a maximum of 500 microseconds to complete such operations at this level of control.

A third level of user control is provided by the RTI level processes. Up to five programs may be established at this level. This control level was provided to meet the requirements for tightly coupled CCI/RTI processes and for high frequency, periodic tasks.

External access to the virtual machine is provided with UPLINK. This process is itself scheduled or initiated in emergency conditions since it requires direct access to both R3 and R2. Uplink occurs at the EXI level and provides a means to update any specified memory location, an orderly alteration of the Mission Scheduled Event Table, and immediate ENQUE/DEQUE/IOR operations. The latter, thorough powerful, must be used with care since they perform isolated actions uncoordinated with on-going processes.

In this description, certain detail has been supressed in order to promote overall understanding. Within the context of this virtual machine any of the flight user programs can be considered independently with the assurance that necessary coordination and timing operations are properly carried out. In order to effectively utilize the virtual machine capacities, it is important to understand the function, limitations, and constraints associated with the services provided. These services, as seen by the user programmer, will be described in the following paragraphs.

The virtual machine processes exist to provide scheduling and I/O capability at the 'SINGLE INSTRUCTION' level as seen by the user programmer. From the programmer's point of view, the task is simplified because there is no need to generate redundant code or be encumbered

by various subroutine calling sequences in order to schedule a mission event or issue a command to an external device via the GCSC I/O registers; the problem of I/O timing is reduced to insignificance as seen by the user programmer. In addition, the problem of testing the flight program is greatly simplified; if the user programmer used the I/O or scheduling instruction properly, then the task to be performed will operate correctly. The virtual instruction set consists of twelve basic instructions divided into five classes: two device-power control instructions (TURNON, TURNOF), five interrupt control instructions (INTSET, EXIENB, EXIDIS, EXISAV, EXIRES), three event scheduling instructions (ENQ, DEQUE, EXIT), one I/O scheduling instruction (IOR), and one instruction to read input discrete register R6 (READR6). The device-power control instructions, the interrupt control instruction, READR6 and the DEQUE instructions always return control to the calling program, while the ENQ and IOR instructions return at the user's option. The EXIT instruction (as the name implies) does not return control to the user. All of the virtual instructions are assembler-level macro-instructions which generate subroutine calls to the executive routines which service them. The structures of these macros are explained in the code module descriptions of the service routines: the power control macros are serviced in the Power State Switching Processor; INTSET, READR6, ENQ and DEQUE are serviced by the Executive Utilities module; EXIENB, EXIDIS, EXISAV and EXIRES are serviced by the External Interrupt Module; IOR is serviced by the I/O Request Handler; and EXIT is serviced by the POP function in the Programmable Timer Interrupt Handler.

Power Control

TURNON or TURNOF DEVICE

These instructions cause the GCSC Power Groups which interface with the specified device to be enabled or disabled, respectively. The device itself is not switched on or off; power is merely made available to the interface circuits. Actual device switching is done by I/O commands via the Power Conditioning and Distribution Assembly (PCDA).

I/O Request Scheduling

IOR Address of I/O Data Block (IORB)

This instruction causes the specified IORB to be linked into the

EXEC's I/O request list. There are three lists, one for I/O Register R1 and two for R4. The lists are FIFO queues, so that the first IORB in the list is the first one processed. The registers can all operate simultaneously, but scheduling priority is done in R2, R1, R4 order. The desired register is specified in the IORB, as are the number of I/O commands and the command addresses. As soon as the IORB is linked (not after the I/O has actually been performed) control is given to the EXEC scheduler. If the programmer wishes to regain control himself, he appends an 'R' to the instruction:

```
IORR        Address of IORB
```

Event Scheduling

```
   ENQ      Address of event data block (ENQB)
DEQUEn      Event specification
  EXIT      A
```

The ENQ instruction causes the event described in the specified ENQB to be linked into the EXEC's Program Scheduled Events Table (PSET). The ENQB contains information regarding the priority and number of the event, when to initiate the event for the first time, how many times to reinitiate the event and at what frequency, and the entry address of the event. PSET is scanned for events every 40 msec.

After the link is made, control is normally transferred to the scheduler. If the user desires instead to return to his own code, he appends an 'R' to the instruction.

```
ENQR        Address of ENQB
```

The DEQUEn instruction causes the specified event(s) to be removed from PSET. The n can assume a value from 1 thru 4, and the event description changes in each case. If n=1 (DEQUE1), all events with the specified event number AND priority are removed from PSET. For n=2, all events with the specified event number ONLY (any and all priorities) are removed from PSET. For n-3, PSET is completely cleared. And for n=4, all events with the specified priority, event number AND entry address are removed from PSET.

After the deletion is made, control is always returned to the user.

The EXIT instruction causes a transfer of control from the user's

program to the scheduler. The general form of the instruction is:

EXIT A (A = blank, 0, +, or -)

Where A = Blank results in an unconditional transfer; A='0' results in a transfer only if the A-Register contents are zero; A='+' transfers if A-Register bit 0 is off; and A='-' transfers if bit 0 is on.

The following table is provided to summarize the virtual instruction set:

Instruction	Operands & Effect
TURNON	Device Name Enable GCSC Power Groups for specified device, and return.
TURNOF	Device Name Disable GCSC Power Groups for specified device, and return.
IOR	Address of IORB. Link this IORB to the I/O chain for the appropriate I/O register for processing, and exit to scheduler.
IORR	Address of IORB. Same as IOR, except return to caller after linkage complete.
ENQ	Address of ENQB. Link this ENQB to the appropriate PSET priority chain, and exit to scheduler.
ENQR	Address of ENQB. Same as ENQ except return to user instead of scheduler.
DEQUE1	Event Priority & Number Remove all events with this event number from this priority level of PSET, and return.
DEQUE2	Event Priority & Number (Priority ignored) Remove all events with this event number from ALL priority levels of PSET, and return.
DEQUE3	No Operand Required Clear PSET & PSET1 (Remove ALL scheduled events), and return.

Instruction	Operands & Effect
DEQUE4	Event Priority, Number & Address Remove all events with this priority, number and entry address from PSET, and return (more selective than DEQUE1).
EXIT	A Transfer to Scheduler If condition 'A' is true. A = Blank: unconditional. A = 0: transfer if A-reg is zero. A = +: transfer if A-reg is positive. A = -: transfer if A-reg is negative.
INTSET	Desired interrupt service status. Enables or disables the specified interrupt.
EXIENB	Mask to enable desired external interrupts. The user's mask is logically OR'ed with the executive external interrupt mask to allow processing of the user's EXI's.
EXIDIS	Mask to disable Desired EXI's. The user's mask is logically AND'ed with the executive EXI mask to ignore the specified EXI's.
EXISAV	Mask indicating which EXI's to save for later processing. The user's mask is logically OR'ed with the executive EXI save mask to delay processing of the specified EXI's.
EXIRES	No operand required. Clears the executive EXI save mask and immediately processes all saved EXI's.
READR6	No operand required. Read I/O register R6 and return contents to user in the A-Register.

QUANTITATIVE RESULTS: In spite of early criticism from certain flight S/W experts (who were G&C oriented) the Flight Software Executive structure and design was excellent for Viking. By using a central master scheduling technique, priority and interrupt structure and standardized I/O it allowed orderly development through many iterations of the application's

programs for descent, communication, landed science, etc. The overhead was a bit heavier than required for descent only, but the common executive minimized memory required to do the total flight job. The concept of using standard scheduling (ENQUE) and I/O (IOR, IORB) functions has allowed many mission functions not previously planned to be implemented very simply by the Flight Team with no software change outside the normal process of altering data base through commanding.

In face, the ground software is more restrictive than the flight software for mission operations.

The advantages of centralized resource management achieved through the EXEC pseudocode operations were several. They can be summarized from two points of view. First, the EXEC service calls appear as virtual machine instructions at the source code level. These "instructions" are high-level in that they invoke whole sequences of code at the machine language level, yet appear as single lines compatible with the rest of the source text - even to the point of providing the standard assembly level indirect and indexed addressing options. We are confident that many potential errors were avoided through use of these MACRO calls which accurately and mechanically generated the required detailed calling sequences. The mnemonic macro names and operands additionally provide clarity and understanding of the intended sequence of operations as distinct from the information hidden in the expanded sequence of assembly level instructions. The power of the technique can be illustrated by the fact that some science application programs consist of nothing but a series of IOR's (input/output requests) and a concluding exit.

The second advantage, closely related to the first, is the isolation afforded to application programs through the EXEC services. All questions of conflict resolution, device peculiarities, such as redundant transmission of individual commands spaced precisely in time, and I/O completion interrupt clean-up are handled by the EXEC routines and hidden from the application programmer. His view is that of the sole system user. Again, the clarity of intent and avoidance of error was significant.

Finally, the virtual machine concept led to a third advantage not originally anticipated. Verification of the EXEC services was viewed

as an extension of the hardware self test program. The EXEC commands were tested as though they were individual instructions of the virtual machine on an absolute basis. That is, the full range of the implied service was evaluated on a stand-alone basis as distinct from the instances of its use. Once verified independent of the application class, the EXEC services could be used with confidence in modified or new application sequences. Re-verification involved only the application program instructions and EXEC interfaces - the virtual machine instructions, per-se, required no further testing.

QUANTITATIVE RESULTS: The overhead required by the executive was directly related to the amount of I/O activity required. Timing was only critical during descent when I/O had to be serviced, sequences scanned and calculations made in less than real time. A sequencer would have been approximately twice as fast to accomplish this, but would have had to been recoded everytime a requirement changed. For single I/O operations, the executive required two milliseconds of a ten millisecond cycle. This represents the greatest overhead burden.

The development and maintenance of the executive required a continuous two man level of effort for five years. This effort covered requirements, design, code, test, integration, maintenance and mission operations.

TECHNIQUE

NAME: HARDWARE/SOFTWARE INTEGRATION LABORATORY

SUMMARY: In order to assure compatibility and performance between the flight program and the Guidance and Control (G&C) subsystem an Integration Laboratory was built. It provided capability for stand-alone calibration, diagnostics and test of G&C hardware as well as to operate that hardware with Flight software in a simulated real-time environment. This Integration Laboratory was extremely useful in early identification of system interface problems as well as acting as an independent source to compare hardware build signatures with realistic responsiveness. As a result, the reliability of the system integration process on flight vehicles was greatly increased.

APPLICATION CONSIDERATIONS: A hardware/software test facility which provides a high degree of visibility and control has usually only been implemented on manned type missions where a very high reliability is required. In the case of Viking the extremely difficult descent presented an equally complex situation that could not accept the classical approach to hardware/software mating. With the Viking approach to laboratory integration the hardware was mated and analyzed with the software at each step in the system build.

RECOMMENDATION: Although this approach has been used in the past only on expensive and complex developments, its yield in reliability has been shown to be cost effective. With new advances in computer and hardware integration techniques this type of facility can be utilized at a significant decrease in cost to provide real advances in test and reliability for smaller programs.

HISTORY: The Viking Hardware/Software Integration Lab was conceived at the beginning of the Viking project as a Guidance and Control (G&C) subsystem test facility! The Viking G&C system consisted of an Inertial Reference Unit (IRU) composed of gyros and accelerometers, a Radar Altimeter (RA), a Terminal Descent and Landing Radar (TDLR), Reaction Control System Thrusters (RCS), Terminal Descent Engine Values (TDE), Terminal Engine Shutdown Switches (TESS), and the Guidance Control and Sequencing Computer (GCSC). The original intent of the laboratory was to test the hardware and software interfaces of the G&C subsystem. Since a GCSC delivery was not due until very late in the program and a specific computer had not even been chosen, an IC-7000 microprogrammable computer built by Standard Computer Co. was installed in the facility to act as a flight computer for the preliminary G&C subsystem development and integration. The IC-7000 had a microprogramming capability that was used to emulate the flight computer and to provide an interface with the G&C hardware. As an interface tool the IC-7000 was found to be adequate, but it was soon realized that it had the potential to also support the entire flight software development and test activity. This activity of hardware/software integration and software development was to continue for the entire Viking Lander development.

The IC-7000 is a dual microprogrammable processor machine in which one processor (the CPU) was used to emulate the GCSC (refer to Emulated-On-Board computer technique) and the second processor (the IOP) became the interfacing I/O to the "environment". The "Environment" was a multitude of possibilities including an Analog/Hybrid Simulation of the Martian atmospheric condition and flight dynamics with Analog/Hybrid models of the G&C hardware, digital models of the G&C hardware, or the actual G&C hardware. With these capabilities actual hardware performance could be compared to software models and be played against various "worst case" flight conditions.

In addition to playing hardware controlled by Analog/Hybrid simulations against the flight software, the hardware could be calibrated, tested, and interfaced in a stand-alone condition. With such a configuration, software verification, system validation, and engineering performance testing were all performed within the bounds of the same lab

with closed loop tests being able to be performed with or without hardware in the loop. Eventually the lab was connected to a 4.8K bit high speed data link with the Viking Mission Control Center at JPL. With such an interface Mission Control could send test data to the lab, a test be performed, and the results could be sent back to JPL for flight control analysis. All in all the lab supported the project from inception through operations.

DESCRIPTION: The lab is represented in block form in figure 1. Each of the block components is described as follows.

DATA CONVERSION EQUIPMENT (DCE):

The DCE is the system interface controller. It is an addressable serial and parallel channel controller as well as the data formatter for each interface. The prime interfaces were between each external component and the IC-7000.

STANDARD COMPUTER CORP IC-7000:

The IC-7000 is a parallel processor machine. Each processor is a micro-programmer with 2K words of control store. The processors share a common memory of 64K 36-bit words and have peripherals including, 1200 LPM printer, 2311 disk, card reader/punch, 3 tape drives. One processor (the CPU) is used to perform the emulation of the GCSC. When an emulation level test is being performed the CPU operates on the Flight Software. When a GCSC hardware level test is being performed the GCSC is utilized as host for the flight software. The other processor (the IOP) acts as host for the Run Time Operating System, the Open Loop Vehicle Models, and as the Interface Control Monitor. The IOP was also host for system interface diagnostics and G&C hardware diagnostics. In addition to being used to directly support test and integration efforts, the IC-7000 was host for all flight software development tools, including data recording, post run analysis programs, assemblers, and file management utilities.

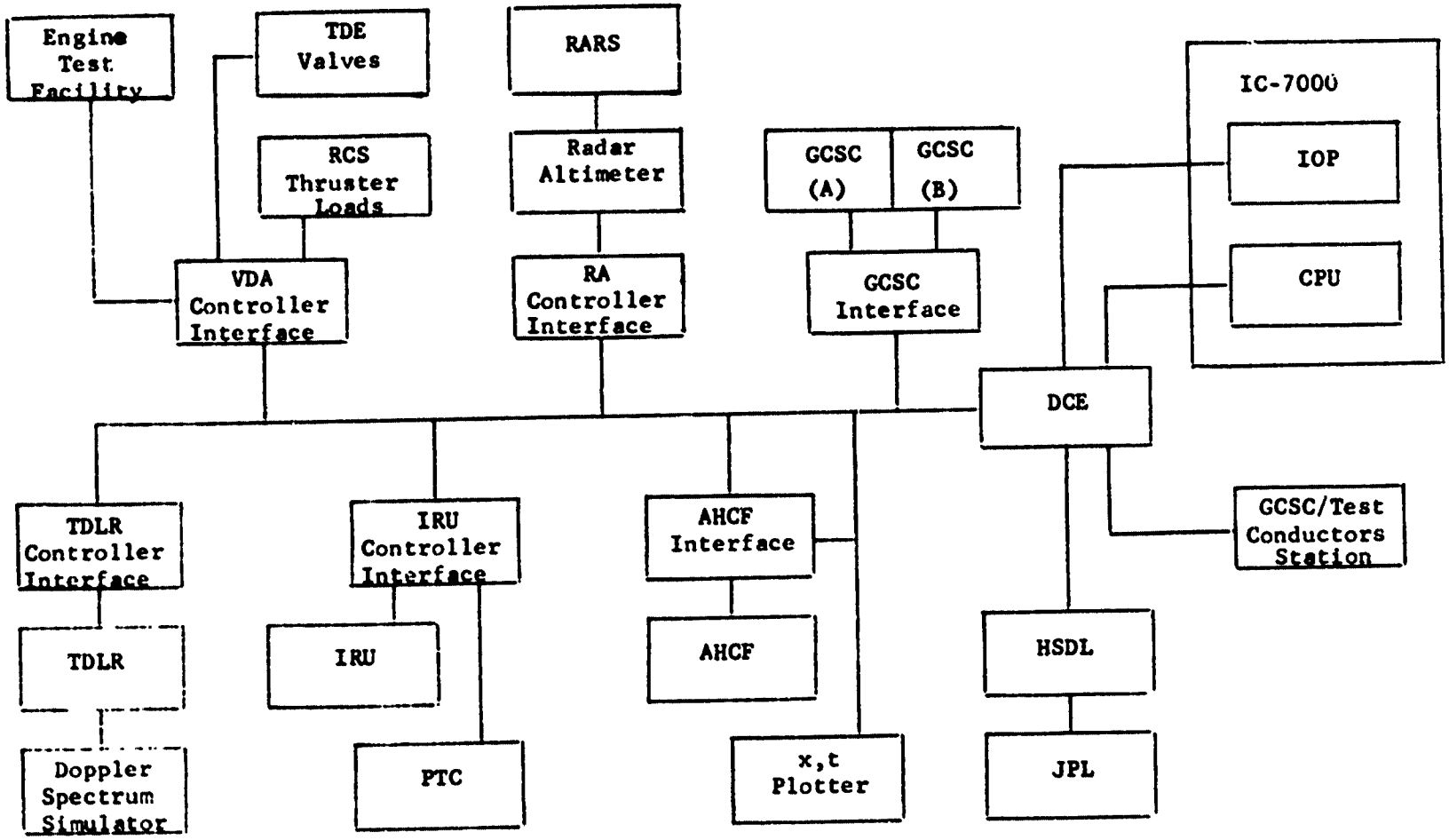


Figure 1. Hardware/Software Integration Laboratory

HIGH SPEED DATA LINE (HSDL):

The HSDL is a 4.8K bps dual line full duplex interface between the DCE and the Jet Propulsion Lab (JPL) Real-Time 360-75 Command Center. The interface was defined at JPL to be an additional Deep Space Network station so that transmissions to and from JPL could look like spacecraft communication. In addition card image data could be transmitted. Thus in support of operations JPL could transmit final vehicle conditions to the Lab prior to beginning the descent to the surface of Mars and a real-time simulation could be performed with analysis and results transmitted back to JPL for final verification. For flight crew training simulated vehicle telemetry data could be transmitted directly to JPL telemetry displays where flight crews would be required to respond to anomalies.

GCSC/TEST CONDUCTORS STATION:

The Test Conductor's Station acts as a computer display panel for the GCSC as well as displaying vehicle simulation specific data such as power status, operational modes, engine states, etc.

ANALOG HYBRID COMPUTING FACILITY (AHCF) INTERFACE:

The AHCF was used to simulate the Mars environmental condition, the Viking Flight Dynamics, and the G&C equipment. The G&C equipment simulations, the actual G&C hardware, or combinations thereof could be used as stimuli to the GCSC or the GCSC simulator. As an example, the RA, TDLR, and RCS thrusters were modeled in the AHCF with the actual IRU being torqued by the AHCF and being read by the GCSC. TDE commands were laser linked to the test stand to fire actual flight type engines with the engine responses fed back to the AHCF models. The result was a closed loop test of actual G&C hardware, modeled G&C hardware, and actual flight software. The AHCF interface is a serial channel controller that interfaces the DCE to the AHCF. AHCF data could be acquired from the interface and fed to the X, T, plotter.

INERTIAL REFERENCE UNIT (IRU):

The IRU consists of a four axis gyro system - three cardinal axes and one skewed - and a four axis accelerometer system - three along the cardinal axes and a redundant one along the X-axis. The IRU is cabled

to the IRU controller and then either to the interface to the DCE or to the Proof Test Capsule (PTC) which is a third Viking Lander built for systems testing. The gyros can be torqued either by the AHCF or by commands from the GCSC.

TERMINAL DESCENT LANDING RADAR (TDLR):

The TDLR is a four beam doppler radar used to provide lateral velocity data for vehicle landing. The Doppler Spectrum Simulators are used to drive returns for each beam. The TDLR is connected to the control and interface unit which connects to the DCE.

VALUE DRIVE AMPLIFIER (VDA) CONTROLLER & INTERFACE:

The VDA controller and interface serves as interface for the Reaction Control System (RCS) thrusters (which are in effect only electrical loads), as interface for the Terminal Descent Engine (TDE) values, and as controller and interface to the 96K bps OPCOM lazer communications link to the Engine Test Stand. The actual engines can be fired, the laboratory values fired, or AHCF simulations used for closed loop testing.

RADAR ALTIMETER (RA):

The RA is a doppler Radar Altimeter which is used to update the inertial navigator during atmospheric descent. The Radar Return Simulator (RARS) is used to provide stimulus to the RA. The RA is connected to the control console which is interfaced to the DCE.

GUIDANCE, CONTROL & SEQUENCING COMPUTER (GCSC):

The GCSC is a block redundant flight computer. Either GCSC(A) or GCSC(B) may be powered on. No computer to computer interface exists. The GCSC interfaces to the DCE for all I/O data and can be interrogated and controlled via the GCSC/Test Conductor Station. Either GCSC or the GCSC emulation may be the host for the flight program.

QUANTITATIVE RESULTS: The hardware/software integration lab proved to be a valuable tool for building and testing reliable software. The successful operation of two Viking Landers on the surface of Mars verifies this fact. The lab provided a common tool for software development, software

testing and verification, G&C systems analysis and integration, G&C system performance testing, off-nominal and failure mode analysis, and operational support and crew training. This amount of flexibility using a system where run controls and data visibility are the same whether using hardware or models yielded a very efficient test and development facility. The problem areas associated with the Lab were:

1) Trying to use the facility too early in the facility development cycle while extensive changes were in progress - this was resolved when the development ended;
2) Limited computer resources for the amount of software and hardware development and integration required - this was resolved by installing a second IC-7000 for software development purposes; and
3) Modeling difficulties due to lack of a High Order Language processor - this problem was not solved using the IC-7000 complex for Viking.

QUANTITATIVE IMPACT: The hardware cost of the lab excluding G&C Flight hardware was approximately three million dollars. The analysis, flight software development, simulation software development, and test and diagnostic software development cost was approximately seven million dollars. For a ten million dollar cost a highly reliable software system was produced for a one billion dollar project, which was not an unreasonable percentage of the project cost. Today, the Lab and its development costs would be considerably less due to the advancements in microprocessors, associated interfaces, and microprogramming skills.

TECHNIQUE

NAME: INDEPENDENT VERIFICATION OF ON-BOARD PROGRAMS

SUMMARY: The on-board computer operations were so critical to mission success that several means were employed to assure error-free software. A separate contract was established to evaluate in detail the algorithms, implementation, and testing of the Lander on-board computer system. The independent verifier performed analyses and tests, using their own tools and interim development products. The overall experience was favorable in helping to identify possible difficulties and in imposing additional discipline on the software development process.

APPLICATION CONSIDERATIONS: Independent verification means redundant verification, and therefore is usually considered only where software error implies high risks and costs. Redundant activities may include the entire spectrum of development - and, in fact, are considered more effective where this is done. Rederivation of equations; independent scientific simulations; in-depth analysis of concept, design, and code; and finally exercise of developed code and data may all be productive in identifying discrepancies and producing error free code. The full spectrum of independent verification was employed for the Viking Lander on-board software - beginning with the Software Requirements Definition (SRD).

RECOMMENDATION: Independent Verification and Validation (V&V) has long been standard for certain Air Force contracts where software criticality is a prime consideration. Multiple viewpoints throughout evaluation has proven useful in identifying both specific discrepancies and classes of software errors. Improving automated tools and introduction of formalized development disciplines affect independent V&V in two ways: first, the costs associated with the activity are declining, and second, the probability of initially producing error-free software is increasing.

HISTORY: The Viking Project Office (NASA, Langley) determined that due to the criticality of the Viking Lander Flight Software an independent V and V effort would be made. TRW Systems, Redondo Beach, Calif. was given that contract. The contract included responsibility to verify the Flight Software Requirements, the Command List, the Guidance and Control Analysis, the Analog/Hybrid Vehicle and environmental models, and the Flight Program. The effort lasted for over three years on-site at NASA Langley under direct NASA supervision.

DESCRIPTION: The independent verification was accomplished in four major areas: Documentation Review, Analysis and Modeling Review, Dynamic Interpretive Simulation of the Descent portion of the Flight program, and static interpretive simulation of the landed and on-orbit flight program functions. The documentation and analysis reviews were performed by senior TRW personnel on-site at NASA Langley. The simulations were performed on the NASA computers at Langley free-of-charge to the contractor. Although the simulations were accurate, they were extremely time consuming and often took many days to set up and perform a single execution.

QUALITATIVE RESULTS: Probably the most significant qualitative result of the independent verifier was the confidence it gave to the Project Office that every step that could be taken to assure a correct and error free flight program would be launched was accomplished. No significant technical problems were uncovered by the independent verifier. Many documentation errors were identified and code problems in preliminary software versions were noted. There were discussions and debates about development and design techniques but all in all no system, software, or operational procedural problems were encountered.

QUANTITATIVE IMPACT: MMC believes the cost for independent verification of the flight program was in the neighborhood of 2 to 3 million dollars for the three year effort accomplished by TRW.

TECHNIQUE

NAME: ON-BOARD COMPUTER TIMING AND MEMORY SIZE MONITORING

SUMMARY: Classically, memory size and worst path timing are critical in aerospace applications. A 50% margin for each was allocated at preliminary design time. Accepting the fact of inevitable change, a margin allocation curve was also established at preliminary design time in order to control margins throughout the project development phase. The plan originally called for a few hundred words margin at launch time to accommodate last minute changes during operations.

APPLICATION CONSIDERATIONS: The high relative cost per change with low margins has been well established. Changes made with limited spare room or time often lead to redesign of existing code with attendant ripple effects. Some published results indicate that relative costs begin to rise where margins are less than 50%. With many of the Viking Lander devices and subsystems at the state-of-the-art, 50% margins at preliminary design time was deemed essential. Perhaps more important, was the realization that margin monitoring was also essential. Where continuing system change can be anticipated, software changes rapidly consume margins unless they themselves become a part of the change controls.

RECOMMENDATION: Management is well advised to pay close attention to timing and memory size when a relatively small computer is to be used to perform a significant real time task. This requires a considerable effort to obtain accurate estimates of the impact of proposed changes. The 50% growth margin used by Viking was not great enough to avoid the necessity of optimizing algorithms and designs already coded.

HISTORY: During the development of the Viking Lander on-board program, constant control was exercised over the growth of the program within the time and space domains of the computer hardware. This was done because there were several precedents of program development problems due to unchecked growth. In order to achieve good control several tools were developed and used throughout the life of the project.

The first problem with respect to obtaining good control was to establish realistic values for the memory and time margins. The approach used was to define a hypothetical computer and then to program the descent guidance and control equations for it. The descent phase was picked for two reasons. First, a significant amount of analysis had yielded a set of descent control equations which could be coded. Secondly, the descent phase represented the major area of concern about timing. Based on this exercise, the number of instructions to be executed was obtained along with their frequency of execution.

The sources of good size values were: first, the code size defined by the descent software; and second, the code size estimated by coding the flowcharts established for the remainder of the on-board program. Together, these produced the initial program memory size requirement. To this memory size was added a 50% growth margin. That value was defined to be the limit for program size growth. The two values defined a linear growth curve, starting with the program size at the date the analysis was made and terminating with a full computer at program delivery. This growth curve defined, at any given time when an audit was made, whether the program growth was being contained.

The hypothetical computer characteristics were included within the computer procurement drawing as a statement of desired instruction set and timing. This also allowed for the generation of memory and timing impact summaries for each of the prospective computer vendors. Then once the vendor's machine characteristics were known, the impact to timing was well defined. In addition, there are instruction set versus memory requirement relationships defined by information theory. Using them, a memory impact was defined for each of the possible computers. This exercise informed the project management that the selected computer would have little impact timewise but that the memory would

have to be 2000 words larger if the initial 50% margin was to be maintained.

Following selection of the on-board computer, a software change control mechanism was installed. This provided a definition of what changes were outstanding and their associated time and size impact. Management could then weigh changes against any possible growth violations. In order to make this accurate, all changes were forced thru this control system.

During the course of the on-board program development there were several major stresses on the control mechanism. Each of these involved the definition of a significant violation of the size or time margin curves. The first was the incorporation of a generalized on-board executive. The design change was necessitated by the total program requirements, but presented an unknown risk to the time margins. In order to gain an insight into the timing of the then current sequencing algorithm and the proposed processing, a discrete simulation model was constructed. The model was used to define the worst case time consumption by the executive. With this known, it became apparent that there was minimal risk to incorporate the generalized executive.

The second major stress occurred when the development of coded modules for nearly all of the on-board program was completed. The resultant code size violated substantially the size growth curve. As a result a set of code reduction changes were proposed. Incorporation of the changes brought the program size back within the established growth curve. However, they necessitated substantial changes to the already coded and partially tested program.

The third major stress was due to the violation of time margins by the descent control code. This was isolated thru the use of an emulator for the on-board computer. The descent code, run in an emulated mode, was shown to take too long during certain descent phases. As a result, the descent control equations were changed in order to reduce the number of calculations required. In addition, some algorithms used during descent were optimized with respect to time. This problem had been brought to management's attention quite early, due basically to the enforcement of regular time and size audits. Because of this,

there was ample time to analyze and correct the problem.

DESCRIPTION: The components of a good timing and memory size monitor system are: accurate software audit reports, timely report generation and total software change control. One cannot emphasize enough the importance of any of these components. Without any one of them, the process of monitoring is susceptible to failure. In describing each, one can take the Viking on-board program development and show why each is required.

The generation and reporting of the current program size and time requirements was basically an audit. For the audit to be effective, the process of generating realistic data had to be accurate. In the case of the on-board software, this was done directly by timing the known worst case loops via an emulator. The availability of an emulator greatly simplified and improved the accuracy obtained from this task. Because the landed phase development was significantly behind the descent development, the size values obtained were prone to error. However, by coding candidate modules from flowcharts relatively good size values were obtained. In fact, some module size values as originally defined were within ten percent of the final size of the coded flight module.

Since the on-board program was constantly changing, the size and time requirements were audited by management monthly with approximate values maintained between audits. This provided, given an accurate audit process, an actual input. The input was then used to update the graph for memory growth. Using this graph management readily established trends of rapid growth. When they were recognized, the change traffic was interrupted and a status meeting held to define which changes to reject, together with requirement changes or design changes to incorporate.

The audit process depends upon an accurate sampling process. This in turn depends upon accurate reporting of current size, impact of changes in progress and impacts for proposed changes. Software changes were forced to proceed through a control system for the on-board program. This entailed initially the complete definition of the change requirements. The on-board software development group used these requirements

statements to define the impact of the proposed change on memory usage and, for descent control changes, the impact on timing. The proposed change and associated impact was then presented to a management team for consideration. The team could then approve or reject the changes as too large an impact, or as an unnecessary requirement. In this manner the modifications to the memory and time requirement were made by one group. This caused the auditing of the memory and time margins to be quite accurate.

QUALITATIVE RESULTS: As a result of timing and sizing monitoring, the on-board program was developed and delivered successfully. The concept of regular and highly visible audits seemed to allow for ample time to recover from major stresses. In addition, management was provided sufficient information to control a highly volatile software development task. A time when the system seemed to fail was when the entire software group was devoted to the development process to the detriment of the audit process. As a result, the audit would encompass a very long time period and nominally would define a significant change to the size and time margins. In addition, continual auditing seemed to force a more disciplined development.

The growth constraint curve was a linear line connecting a 13K memory size in July 1971 to an 18K memory size in October 1974. Twice large accummulations of new code caused the current memory size estimate to violate the constraint curve. The first occurred between March and May 1973, when as built code rapidly grew from 15.5K to 18.5K. The second occurred between February and June 1974, when as built code grew from 16.5K to 19K. On each occassion management was forwarned that an unacceptable growth was taking place, thus permitting them time to assess the need for and ramifications of candidate redesign efforts. On both occassions management required the implementation of agreed upon design changes that brought the as built code below the memory size constraint curve. These two instances demonstrate the practicality of using the technique.

QUANTITATIVE IMPACT: The process of monitoring the memory size and timing margins of the flight computer was a planned event that did not require additional staffing. As such, there was no manpower cost impact.

TECHNIQUE

NAME: REQUIREMENTS GENERATOR FOR FLIGHT HARDWARE AND SOFTWARE

SUMMARY: The Viking Lander Guidance and Control system in the flight environment was extensively analyzed via simulation and other supporting techniques before hardware and software specifications were written. This led to the generation of detailed specifications for hardware and software which produced results that were compatible in later integrated software/hardware closed loop tests and in actual flights with minimal revisions. An auxiliary technique of this process was the FORTRAN specification of the Flight software routines to be consistent with the overall system analysis and simulation models.

APPLICATION CONSIDERATIONS: This approach has been used generally on aerospace guidance and control contracts, but more often than not the simulations evolve too late to influence the hardware design or significantly change the software. By having a complete six-degree-of-freedom simulation early in the program, models for all components (inertial sensors, radars, actuators, aerodynamics, etc.) could be evaluated closed loop with preliminary flight software control laws and algorithms for overall system accuracy, stability and response.

RECOMMENDATION: An algorithm design and specification language is one of the missing gaps in software development technology. The use of FORTRAN for this purpose can be helpful, but should only be used as an overall statement of a suitable solution. It should not be used as an implementation requirement on logic control or computational sequences.

HISTORY: In the development phase of Viking, a simulation was used to analyze different radar designs, variations of parachute and aeroshell mechanics, propulsion dynamics, and flight software algorithms. The models were all written in FORTRAN. These formulations, after considerable experience, were directly translated into models for an analog-digital hybrid real-time simulation which was used for closed-loop testing of the descent phase flight software and guidance and control hardware. Having the FORTRAN flight software models in the hybrid simulation allowed a three-way comparison of the FORTRAN program, the hybrid real-time simulation, and the hybrid real-time simulation mixed with Flight software/hardware combinations. Because of this testing, timing and phasing problems were identified and corrected very early in Viking development.

DESCRIPTION: The six-degree-of-freedom simulation (MOD6MV) was developed from the MOD6DF program developed by Litton, Incorporated for the Air Force. The program provided a modularized structure which handled dynamic integration of differential equations, inter module communication through a common array, random noise generation, standard data input, standard print output, vehicle dynamic staging control, and plots. Therefore within this structure it was a simple process to develop the Viking peculiar models, using Guidance & Control engineers who only had rudimentary FORTRAN experience. This program later evolved into the Flight Operations Lander Trajectory Simulation (LATS) program used for pre-separation analysis of candidate landing trajectories.

The Flight Software Requirements Document (descent phase) was written directly from the FORTRAN algorithm models. This had to be transposed by hand to a typed version using greek letter symbols, etc., because the customer objected to the style of FORTRAN type equations.

QUALITATIVE RESULTS: FORTRAN is one of several machine independent programming languages. With its algebraic statement capabilities, the computation problem can be stated succinctly and with precision. FORTRAN has several shortcomings as a specification tool for flight computations, however. FORTRAN is not a sufficiently high order language for

specifying conditional computations, identifying accuracy constraints, and performing vector and matrix operations. Thus, these operations tend to force undesired detail on conditional statements and matrix element calculations, and allow accuracy specifications to be ignored or only implied.

Complete, precise, consistent, and concise requirements are highly desirable but difficult to obtain. Typically, the programmer must carefully analyze a partial problem statement, ask questions, fill in missing detail, and resolve inconsistencies. Some of these problems were solved through use of the FORTRAN simulation and could have been expanded to integrate all descent phase functions with minor additional effort.

The program documentation (SRD, Flight S/W flows, etc.) should use a consistent nomenclature which can be related directly to the simulation model. FORTRAN provides this (upon agreement between the system analysts and flight programmers) with some limitations which can be worked around. Furthermore, it is highly desirable to use nomenclature common to standard keypunches and typewriters as opposed to special greek letters and math symbols. As such it would have been preferable to structure the SRD routines directly from the FORTRAN listings using comment fields to specify accuracy and range of variables because many typographical and reproduction errors resulted from the Greek letter translation.

Other lander system requirements for sequencing during the descent phase which did not affect guidance and control were not included in the simulations. Therefore, these requirements were confusing to the flight programmers since they were not integrated into the G&C specifications. For consistency, the simulations should have included a FORTRAN sequencing module which would have been a direct analog of the Flight program version.

It is very important to have a system analysis group which is capable of integrating, simulating, and specifying all requirements in a consistent language. The language should have one to one correlation with the simulations and the actual software/hardware. Although this is commonly done in Guidance and Control, it could and should be extended to any spacecraft flight software operations. On Viking, many

hardware/software and software/software incompatibilities resulted from the lack of end-to-end simulation modeling of portions of the system, and the lack of an analytical systems group working those mission phases.

QUANTITATIVE IMPACT: The manpower costs involved in developing the six-degree-of-freedom MOD6MV simulation to analyze and generate descent flight hardware and software requirements were approximately 40 man months. The additional effort required to develop a 12169 source card Lander Trajectory Simulation program for Flight Operations was 73 man months, which, with overtime, amounted to more like 80 man months.

The total effort required to develop the G&C hardware and software requirements was approximately 25 man years of which 12 to 15 involved the development and use of MOD6MV.

System Test Equipment Software Development Overview

1.0 Introduction

- 1.1 The STE Software System
- 1.2 Software Development Responsibilities
- 1.3 Quantitative Software Description

2.0 The Requirements and Design Phase

- 2.1 Organizing for the Task
- 2.2 Defining the Software System
- 2.3 The Software Design Phase
- 2.4 The Development Cycle

3.0 The Test and Integration Phase

- 3.1 Module Testing
- 3.2 Subsystem Integration Testing
- 3.3 Integrated System Testing
- 3.4 Maintenance and Operational Test Support

4.0 Lessons Learned

System Test Equipment Software Development Overview

1.0 Introduction

The Viking Systems Test Equipment Software System was developed to control and monitor Viking Lander flight article hardware checkout and verification. It was the first of the three software systems built by Martin Marietta for the Viking Project. This narrative discusses in chronological order the process followed to develop the system.

1.1 The STE Software System

The Viking Systems Test Equipment (STE) Software System was developed to provide the means to check out and analyze the performance of the Viking Lander hardware component subsystems and the Automated Ground Equipment (AGE) hardware. It was placed on-line at the start of lander system integration testing and continued to support verification of the integrity of each lander subsystem up until launch. During cruise and planetary operations it was used by the Viking Flight Team to simulate anomolous conditions on the Proof Test Capsule.

The system was comprised of a Honeywell H-632 computer set, computer peripheral equipment, and MMC designed and built hardware test equipment. Both manual and computer hardware test equipment were employed. The manually controlled hardware was used primarily for RF alignment and circuit test purposes. The computer controlled hardware consisted of analog to digital converters, discrete output circuits, discrete scanning circuitry, and telemetry monitors. In addition, both digital and RF links were available to control information transfers between the STE computer and the Guidance, Control and Sequencing Computer (GCSC) in the lander.

The STE Software System was comprised of three major software subsystems. A pre-test software subsystem prepared interface data files and Viking Test Language test sequences for on-line, real time execution. An on-line software subsystem supported real time execution of the test sequences to conduct the actual checkout operations, provided interaction with an operator for control and status of test operations, processed discrete, digital and

telemetry data received from the lander subsystems, and generated both printed logs and a comprehensive tape history to record the proceedings. Finally, a post-test software subsystem provided a means for extracting data from the history tapes and producing selected plots, printouts, and limited mathematical analysis of the data.

1.2 Software Development Responsibilities

The Langley Research Center was responsible to NASA Headquarters for the management of the Viking Project. A contract was awarded to the Denver Division of Martin Marietta Corporation to develop computer controlled systems test equipment to checkout and verify Viking Lander hardware components and subsystems. No other agencies or manufacturers were directly involved in this task.

1.3 Quantitative Software Description

The STE software system contained 133,000 words of instructions written in assembly language code. The system was developed at a cost of 600 man months. Supporting documentation included a Software Requirements Document (200 pages), a General Design Document (400 pages), a Program Description Document (1000 pages) and Users Guide (400 pages).

The estimated effort expended by development phase is as follows:

Definition	10%
Design	20%
Programming	35%
Test	35%

2.0 The Requirements and Design Phase

2.1 Organizing for the Task

The Systems Test Equipment Group was organized in 1970 as an independent entity within the Systems Engineering Directorate. The purpose of the system was to provide a means to exercise various Viking Lander hardware components then passively monitor and record the resultant Lander responses. Therefore, the system design would be adaptive to, rather than impacted by, changes to Lander hardware. The Viking Integration change procedure under the control of the Project Change Board was the means by which the STE and lander developments were coordinated.

A Software Chief was appointed by the STE manager to be responsible for the design, development, control, implementation and maintenance of the software portion of the STE.

2.2 Defining the Software System

During the period in which the STE Software System had to be designed neither requirements nor a Lander Software Development Plan had been generated by Systems Engineering. The STE Software Chief therefore formed a software team with individuals who had experience in developing similar computer controlled test systems.

Requirements for the overall design were obtained through consultation with systems engineering groups, who had solely hardware backgrounds. Some detailed information was available in areas of interface signal characteristics and testing descriptions. All requirements concerning the software control system, the selection of the computer set and peripheral equipment, and the man-machine interface had to be developed by the software team.

The approach taken was to outline the design of a flexible software system, wherein specific requirements relative to hardware components and interfaces could be treated as data. In this way if the requirements should be found to be inadequate at a later date, only a minimal amount of rework would be required in complex areas.

The software team identified the need for three software subsystems. An off-line pre-test software system would translate Viking Test Language test sequences, nomenclature data, signal address data, conversion and calibration data, status and criteria data, decommutation data and configuration data into object code to be stored in tables and files in mass storage. An on-line software subsystem would provide STE software support during testing. An on-line control language would be generated to permit a test operator to input instructions via a CRT keyboard or a cardreader. The on-line software subsystem would interpret the control language, locate and input the appropriate test sequences and support data from mass storage, and execute the test sequences. Discrete and digital stimulus and control data would be output as directed by the test sequences. Telemetry and hardline analog, discrete and digital monitor data would be input to the STE computer from the Viking Lander and STE hardware via a data bus controller. On-line evaluation would consist of change detection and limit checks. Data would be time tagged and placed in mass storage in an Operations Test Log for post-test analysis. Finally, an off-line post test software subsystem would be developed to provide additional processing of data recorded during testing. This processing would include the preparation of reports, sorting and recording of special data tapes, and limited processing of engineering and science data from the Operations Test Log.

The STE Software Chief selected the PDP 11-45 computer; however, the decision was reversed by management and the Honeywell H-632 computer set was selected as the hardware that would support the software system. The primary rationale was based on initial cost estimates; the H-632 was less expensive. Other factors that supported the decision included its I/O bus capability, a FORTRAN compiler was available, it had a byte/bit instruction set, the CPU speed was reasonably good, and it appeared to have sufficient tape and direct access disk mass storage capabilities.

As it turned out, the H-632 was removed from Honeywell's product line within two years, the I/O portion of the computer had design problems, the FORTRAN compiler contained so many errors as to make it useless, the bit/byte instructions worked so slowly that only limited usage was allowable, the instruction speed turned out to be 50 percent slower than advertised,

the magnetic tape controllers were prone to data dropout, and the disk data was easily destroyed whenever power transients occurred. Honeywell cooperated with MMC by supplying onsite field maintenance personnel (at considerable cost to the Project) and assisted MMC with top engineering support for major fault isolation. But as a discontinued product line, all activity terminated in correcting the H-632 hardware and software design problems. This forced the senior STE programmers to support computer troubleshooting. This support requirement was to continue through launch of both Vikings. The costs involved easily offset the initial lower price of the H-632.

2.3 The Software Design Phase

Initially the software team collectively outlined a top level software system design. Then the Software Chief subdivided the team into four individual sections, each responsible for a major software component. Technical lead programmers were made responsible to develop more detailed requirements and an intermediate design for the specific tasks their section had been assigned.

One section was responsible for the file management functions and Viking Test Language processing required by the pre-test software subsystem. The other three sections were responsible for the executive, display and monitor processing functions required by the on-line software subsystem. The display function included the definition of the on-line control language. The executive function required the development of a real-time control system and an input/output control handler. The monitor function included discretes, analog to digital data, digitals and telemetry.

The Software Chief considered the post-test software subsystem processing requirements to be too vague and incomplete for an intermediate design to be developed in parallel with the other two software subsystems.

The STE Software Requirements Document (SRD) was written by Systems Engineering during the period in which the intermediate design efforts of the pre-test and on-line software systems were being conducted. As such it was coordinated with the Software Chief in order to be consistent with the fait accompli design. It served merely to document the details of the hardware component interfaces, the on-line control languages, the input/output

bus handler, and the Viking Test Language. It contained only a passing reference to the post-test processing software subsyst. . Furthermore, Systems Engineering did not provide personnel to maintain the SRD after it was issued. By default the STE Software team was made responsible to gather and define the post-test processing requirements. Nothing more was done in this area until after both the pre-test and on-line software subsystems became operational.

2.4 The Development Cycle

The STE Software System was scheduled to be designed, coded and checked out in a span of approximately eighteen months. In most areas of the on-line software development, schedules were generous enough that the software system was not a pacing item of the project, and these schedules were met.

There were some unforeseen areas which required additional tasks not originally planned. It was discovered that the vendor supplied disk file management system was inadequate, so a new design was implemented to manage all areas of the disk, except for a small region to contain the vendor supplied batch operating system.

The technical leads monitored the progress of their workers and provided the Software Chief with status reports. The Software Chief held weekly design reviews to allow every member of the team the opportunity to assess the developing system design. A technical lead would discuss the design status of a single subsystem at each of these meetings. The technical leads were rotated so that by the end of a month the team had reviewed the entire system. As the intermediate designs were completed, the Software Chief scheduled design reviews to permit management, system engineering, STE hardware, and system integration personnel the opportunity to understand and critique the software system development process. Critical areas of some programs were analyzed for speed and core usage predictions to assure safety margins. However, no meaningful computer loading analyses could be performed since there was no way to assess the resources that would be required by the post-test system.

The Software Chief relied heavily upon the integrity of the individual technical leads for the adequacy of the design and the completeness of the coding and checkout.

When the coding for each individual software function was completed, a test demonstration was held to assure the Software Chief and management that the software would perform its required tasks and was ready for test and integration.

The development cycle was therefore a straightforward process managed by the Software Chief. No serious impacts were caused by hardware changes, primarily because of the flexibility built into the Software system design. The technical leads had control over specific portions of the system, and were free to modify or change their code as they saw fit to the extent that no requirements were overlooked or violated. The Software Chief maintained visibility over the entire process through the eyes of the technical leads and was solely responsible for coordinating and resolving system level software problems within the STE.

3.0 The Test and Integration Phase

3.1 Module Testing

The module test phase was very informal. No test requirements, test plans, test procedures, test schedules or test results were formally documented. It was the duty of each technical lead to perform unit checkout for each of the subsystems for which he was responsible.

Checkout driver programs were written by the individual programmers. In addition a standardized driver was available for all on-line packages. The actual test cases were designed solely at the discretion of the programmers and the technical leads.

The module tests for on-line programs were conducted primarily for the purpose of demonstrating how the software worked to a system integration lead, who had been made responsible for the integration of the individual pieces of that system. The system integration lead had the authority to modify or change any software part so as to improve overall system efficiency.

3.2 Subsystem Integration Testing

After each program was module tested it was turned over to the system integrator, who attempted to integrate the program into the on-line system. This was physically accomplished by turning program decks over to the integrator. The integrator would then sit down with the individual programmers to learn the procedures for running the software.

The initial integration of the on-line software subsystem with the STE hardware in the System Test Bed consisted of a fairly thorough checkout of the analog and discrete systems, but the telemetry system and digital down-link and up-link channels to the Lander Guidance, Control and Sequencing computer were not verified.

The technical lead responsible for the on-line monitor functions had assigned programmers the tasks of developing the discrete, analog to digital and digital functions, and had taken sole responsibility for the telemetry system software. When the telemetry system software was not turned over to the system integrator in a timely fashion, the Software Chief required that the technical lead demonstrate that the software functions had actually been

developed and coded. In particular, the Software Chief wanted to be assured that the as delivered software would be capable of monitoring every specified lander telemetry format.

This points out the degree to which management's visibility as to what was happening was limited to relying on the integrity and competence of the technical leads. In the instance of the telemetry system software it proved to be a serious mistake.

From all outward appearances the monitor function technical lead was experienced and competent for the task. The software design for the telemetry monitor had been reviewed and appeared to be sound and reasonable. During subsystem testing, the technical lead had, when ordered by the Software Chief, demonstrated that the software code could process each telemetry format correctly. However, for that demonstration the technical lead had generated special code to make the telemetry model appear to work, knowing full well that it could only handle one of the formats and could not meet the requirements of the STE. It wasn't until after the software was added to the on-line system that the Software Chief became aware that a problem existed. Coincidental with this, the technical lead tendered a resignation and left the company.

3.3 System Integration Testing

During the first attempts to interface the STE with Viking Lander hardware in the System Test Bed it was discovered that the telemetry monitor software code could not process the various formats of telemetry frames. It is academic as to whether or not the design could have been salvaged had the programmer not quit the company. The problem was a real one and a serious one.

The effort to develop a new telemetry monitor function was further impacted by management. The Viking Lander Software Plan specified validation and verification requirements for the STE hardware/software system. To accomplish this six months of integration testing prior to beginning vehicle sequence validation in the System Test Bed had been scheduled. To save costs, management elected to modify this approach by ignoring the Software

Plan and deleting the six month STE integration effort. The net result was that the software was never validated, formally or otherwise. The development process merely continued until the system worked. In addition the STE programmers had to support vehicle sequence validation during regular shift time, and perform integration, maintenance and the telemetry function development on second shift. It proved to be an inefficient use of people, and as they grew tired they became very error prone.

System integration of the non-telemetry portions of the on-line system was conducted in parallel with the crash effort to redesign and recode the telemetry monitor function. It amounted to a two to three man effort, working 16 hours per day, seven days a week for a month to produce a modified system. By the time the situation was corrected, 24 man months had been wasted. The most irritating part of it all was knowing that the problem could have been avoided, or at the least minimized, had management required a visible means of establishing adequate criteria and procedures to monitor the various stages of software development.

During the checkout of the modified telemetry software monitor function a new problem was uncovered. It was found that the N35 interface console, which was the main STE hardware to Honeywell H-632 computer interface, did not react to commands as planned. A sequence of operations had been adopted which worked within the timing requirements of the on-line system and provided adequate I/O in the telemetry area. However, when all systems were run concurrently, the I/O Bus transfers became confused and caused the on-line system to abort.

This problem was never completely solved but eventually was managed. After a schedule slippage of approximately a month, it was found that two pieces of hardware logic sections in the N-35 that should have been slaved to one another, actually were independent of one another. As information passed through the N-35, a race was on in the hardware logic. When the bad guy won the race, the system would abort. Hardware corrections were made to attempt to correct the situation, and some software modifications were incorporated to attempt to avoid it, but no one was ever able to identify all the causes. Even though no permanent solution was found, the impact was minimized by adopting procedures designed to control the problem.

By October of 1972 both the pre-test and on-line software subsystems were operational and could support Lander systems testing. About this time efforts to develop the post-test software subsystem were initiated. The reasons for this late start were a continuing lack of meaningful requirements plus the impact caused by schedule slippages in integrating the vehicle with the on-line system.

The initial design of the post test system was based upon what the lead programmer thought would be logical requirements. Before design could be implemented, it was disclosed that the post test system would have to process telemetry stream data recorded on analog tapes. The design under development did not accommodate such an ability, and had to be scrapped. It was then decided that the post test system would be a modification of the on-line system so that it could access the hardware interface drivers. This decision greatly reduced the development time of the post test processor, and retained the flexibility to use any previously developed code that could function within the on-line system. By late 1973 the post test system was finally operational.

3.4 Maintenance and Operational Test Support

The STE Software System functioned adequately to support Lander testing, first in Denver and then at Kennedy Space Center. The procedures adopted to handle the N-35 I/O Bus problem reduced the number of test aborts to less than one per week, which did not prove to be a serious handicap.

One problem occurred that can be blamed on the lack of an initial overall systems design, which would have defined every function that was to be developed and permitted computer loading analyses to be performed. The pre-test software system was slow and consumed most of the off-line processing hours available each day. Because of this post test off-line processing was frequently delayed or cancelled because of a lack of computer time.

The only other problem that merits mention was the amount of computer set and peripheral equipment down times. Whereas the H-632 computer set had been selected for cost effective reasons, it increased costs during the maintenance and operational test period. The design errors inherent in the computer forced entire crews to wait, sometimes for several days, while trouble shooting efforts took place.

Originally it had been planned to slowly take STE programmers off of the Viking Project. By the time the STE was moved to KSC for operational test support, there were to be no programmers left. As matters turned out, the number of programmers assigned to STE peaked during this period. This was only partially due to the problems with maintaining the H-632. New requirements for software functions constantly arose, both in Denver and at KSC. Two new programs were written at the Cape to assure that Flight loads would be correct.

4.0 Lessons Learned

Software design should be made flexible when requirements are weak and incomplete. This was done for the STE software system and was the main reason for experiencing as few problems as occurred.

The lack of good system requirements at the start of software development will lead to wasted manhours downstream in the development.

Computer loading studies should be conducted early to aid in developing efficient software designs from a user point of view.

A strong standardized executive system was designed that relieved the module programmers of the necessity to address problems inherent in I/O timing conflicts. This permitted reliable code to be developed with less effort than otherwise would have been required.

Schedule programmers to remain on a project beyond software delivery dates. Unforeseen maintenance problems can arise that require thorough familiarization of the software to resolve. Also, new requirements should be anticipated.

It is a risk that can have serious repercussions when a single programmer is permitted to develop an important piece of software even with some kind of technical monitoring. An obvious way to avoid this is to require that at least two programmers be assigned to each program. This is not always practical from a personality point of view, and sometimes is not possible due to manpower resource considerations. The STE Software Chief had in fact assigned two programmers to develop the original telemetry function. But the technical lead turned out to be a loner who did not cooperate with his assistant.

The selection of the least expensive computer that was adequate to support STE software requirements proved to be a costly mistake when the product line was dropped by the manufacturer. Future projects should keep this fact in mind.

TECHNIQUE

NAME: TEST DATA BASE STRUCTURE

SUMMARY: The Viking Project was characterized by extensive use of automation during all phases of test and flight operations. A major task was the collection, management, and configuration control of the information required by the various computers serving the project. An equally important task was the definition of the interfaces between the various information subsystems, including man-machine interfaces.

APPLICATION CONSIDERATIONS: The generation of the basic data files had to be completed prior to committing the system to test. Each file played an important part in linking the System Test Equipment (STE) software, application programs and test equipment to the test task. The data had to be put into a form that was immediate recognizable and usable by design personnel, sequence writing and test operations personnel. It had to be put in a form that was compatible with the software interface and applications programs (test sequences). Since the flight computer would play an important role in Lander tests, the mechanisms for data inputs and table inputs to that computer had to be implemented.

RECOMMENDATION: The Viking data base structure concepts and implementation techniques are highly recommended for use on programs similar in complexity to Viking. They proved to be viable, visible, efficient, and easily manageable. MMA is adapting the Viking Data Base Structure to their PACE program.

HISTORY: Some concepts of technique and data base structure evolved from previous MMA programs and studies such as the MMA Computerized Aerospace Ground Equipment (CAGE) design for the MOL program; the bulk of Viking data base structure design techniques, concepts and implementation were developed during the on-going Viking program. The resultant system was derived thru extensive give and take during the evolutionary process, with special significance given to synergistic effects between STE SOFTWARE, FLIGHT COMPUTER TEST SOFTWARE (STACOP), Test Sequence, Test Equipment Vehicle designs and test requirements. The basic data base structure (due to its impact on software and test sequence designs) was developed relatively early in the Viking schedule. Modifications to technique and additions to the data base files continued throughout the program.

The Viking test data base was structured to provide a realistic tie-in between all of the various components that made up the test environment. These components were:

1. Test Vehicle
2. Test Equipment
3. Test Interface
4. Data
5. Software
6. Test Sequence
7. Management and Control
8. Test Environment

DESCRIPTION: The test data base structure can best be described in its relationship to the total command and information system.

The basic task of the information system was to provide the STE computer with sufficient software and data to enable it to control and monitor the VLC/STE system during all phases of Viking system level testing. The methods of implementation were influenced by the considerations below.

Volume of Data

The information required by the STE computer included detailed data on each stimulus and monitor point in the total hardware system. It

includes all the standard messages and commands that were sent over all the digital interfaces. In this context 'data' included automated procedure so it included functional information on all STE and VLC operations that were visible at the system level.

Continuous Monitoring of All Data

It was a requirement of the STE that all data be monitored continuously, i.e., each data sample arriving at any interface had to be compared with predefined criteria and appropriate displays made or alternate action taken in the event of a criteria violation. The success criteria had to be dynamically modified as the system responded to test stimuli.

Testing Multiple Vehicles in Multiple Configurations

Each of the Viking test articles had to be tested in all mission configurations; landed, entry capsule and spacecraft. Each vehicle, in each configuration, represented a unique data environment which had to be exactly defined for the STE computer. Most of the data required was common to all test configurations.

Incremental Availability of Data

The information required to support Viking System level testing became available at different points in time. The first was available when design criteria was released. The final parametric data was not available until just prior to the time a test was to be run.

Commonality of Data

A single data item might be common to all test articles and to many test procedures for each test article. A change to such a data item could have a significant impact. This was compounded by the fact that the test articles and associated information libraries were in varying states of completion.

Data Supplier and User Interface

Most of the originators of the information required by the system were not computer or programming oriented. To the maximum extent possible, all input and output data was to be in English or in standard engineering format.

Operator Interface

The ON-LINE test operator had to be provided with 'selected' data on the test operation in progress. In the event of a problem, the operator had to have the capability to request additional data and/or implement workaround procedures.

Conceptual Design

The Viking STE information system was structured on design concepts intended to solve all the explicit and implicit problems identified above.

The characteristics of the information required for Viking System Level testing was such that three separate but interrelated regimes were required, the data regime, the test sequence regime and the software regime.

Data Regime

The data files contained all the static and descriptive information about the test articles and their interfaces with the system test equipment, and as appropriate, interfaces with the facilities and associate contractors. A special class of data was included in the same data base to support general simulation activities. Design goals driving the system were:

a. To accept data in engineering terms and formats;
b. Require that each data item be entered only once;
c. Accept data as soon as it became available;
d. Provide for retrieval of selected data subsets;
e. Accommodate data changes with a minimum of impact, but provide knowledge of what that impact was to other information elements or subsets.

Test Sequence Regime

The test sequences contained the "tests". Derived from test requirements, and utilizing the data base and Viking Test Language, the automated test sequences defined, scheduled, and set criteria on the functional happenings to occur aboard the Lander during the tests, the STE-SET UPS for the test and the required interaction between the Lander and the STE. Requirements for the test sequence subsystem were:

a. Compatibility with data base files;
b. Compatibility with STE software off-line system (file management, translator and load algorithms);
c. Compatibility with the on-line system;
d. Knowledge of test article functional design;
e. Knowledge of functional interaction between the test article and STE;
f. Knowledge of STE SET-UP requirements;
g. Knowledge of configuration control methods;
h. Provision for modularity in sequence preparation and execution;
i. USE OF VTL language.

Software Regime

The software was designed to accomplish all the processing necessary to support on-line testing, off-line data management, and to provide workable Man-Machine interfaces for all processing activities. Design goals were that the software subsystem should: provide maximum diagnostic capability to detect as many errors in data and sequence input as possible, be unaffected by changes in the VLC or VLC/STE interfaces, provide the necessary visibility to the operator to allow proper decision-making during all processing, and provide for the interaction capability in that the decisions could be implemented.

The Viking on-line operating system has to be considered as a data driven system. As such, it is necessary to understand some facets of implementation of Viking data handling techniques. The Status and Criteria Table (SACT) is the single most important item in the understanding of the system. It is the common interface between most of the elements in the total system. In a programming sense, the SACT is an interrupt table. Each data or criteria change represents a potential interrupt. The conditions under which the interrupts are to be honored, and the action(s) to be taken in response to the interrupt, are controlled by the test sequence writer, and to a limited extent by the test operator.

The physical structure of the SACT may vary from system to system, but the function of the SACT remains the same.

The SACT is an array or table consisting of a line entry for each interface point in the system for which dynamic status must be maintained. Fields are allocated in each line entry for many types of data. The format and content of the SACT entry is dependent upon the type of interface point which is being represented. Typical entries in the SACT are:

A. Status - Latest significant data sample received;
B. Care Bit - Criteria open or closed;
C. Inactive - Should data be ignored;
D. Type Info - Continuous, momentary, telemetry, etc.
E. Display Info - CRT, line printer, console light, etc.;
F. Criteria - As appropriate to signal type;
G. Aperture - As appropriate;
H. Source of Criteria - Initial conditions, seq. TB, etc.;
I. Alternate Action - Stop, Go To, Abort, etc.;
J. Special Bits - As required;
K. Spare;
L. Address in mass storage where other information concerning the interface point is stored;
M. Simulator Peculiar Bits;

SACT Usage

The SACT is constructed by the data loading software when a specific sub-set of data is selected from the master file and placed in mass storage for use by other software, thereby accomplishing an interface function with the on-line software subsystem and with other off-line software modules.

The on-line software subsystem uses the SACT as a storage facility for dynamic data and as a source of 'instructions' as to how the data is to be processed. It should be noted that these instructions may be modified by the test sequence in progress, or by the on-line test operator.

Other usages of the SACT can be inferred from the contents of the SACT as presented above.

Description of Basic Information System Elements

The STE on-line and off-line software subsystems are an integral part of the information system. The software provides the capability to store, convert, correlate and display the large volume of data flowing into and out of the information system. It is analogous to a communications system in that information may be input in one form, stored in another form, transmitted in another, and output in still another form (or in several forms to suit the various users or receivers). The function and scope of the software is determined by the information structure and its interfaces, and by the hardware elements of the project.

STE Software

It is important to realize that the effectiveness of the software is determined by the extent to which the information needs and interfaces are properly defined early in the program. In many cases, the same considerations apply to the proper selection of hardware.

Basic Data Files

There are three basic files in the information system. These are the interface file, the group file, and the decommutation file. They are classed as 'basic' because each entry is a direct representation of a fixed characteristic in the hardware or software. The exceptions to this are the criteria entries in the VAIF.

Each building block or sub-structure within a composite structure must have a fixed point of reference or 'anchor' that 'locks' it into place in higher level structures. The 'interface point' provides that reference point for all data files.

The Viking AGE Interface File (VAIF)

As implied above, the interface points tie the entire system together. The ID symbol positions the IPDS in the file and 'connects' it to a hardware subystem or interface, as well as serving as a shorthand address for all man-machine interfaces.

The 'descriptor' provides functional information about the interface point in standard engineering terms. The interface point data set contains all of the information required by the sequence writer, the test

operator, and the STE software.

In addition to the above, the TSS Logic in the IPDS defines the functional relationship of the IP to other interface points in the VAIF.

VAIF Structure

The VAIF is structured on the 'page' principle in that each interface point has a page or set of data associated defining it. This consists of characteristic, parametric and functional information.

Each page is made of 'lines', where each line is input on a punched card. Effectivity is implemented at the line level. (Each card has its' own effectivity codes).

Information in the VAIF is organized by Interface Point Data Sets (IPDS's). Each IPDS is a uniquely identified block of data containing enough information to describe one interface assignment in the system to both the STE computer and the human user.

The VAIF is maintained as a Source File (card image) on magnetic tape. Although information from the VAIF is essential to almost every step of system operation, only two programs interface directly with the VAIF tape. The pretest File Manager is used to update the VAIF, to provide copies of the current VAIF tape, and to make listings of the VAIF. The Data Base Manager uses card numbers and effectivity codes to select only the information needed by the computer for a particular application from the VAIF, then formats this selected data into the BIDs and the SACT. The Binary Interface Data Set (BIDS) is a disk-resident file used primarily by the Sequence Translator to convert an IPDS name into either a command word or an address recognizable by the On-Line operating system.

Every electrical interface between the Viking STE and the Lander, either command or monitor (digital analog, and discrete), is represented in the VAIF as an IPDS.

The format of the VAIF allows single lines or entire blocks of comments to be inserted among the IPDS's. The 'file loader' will ignore an IPDS without a zero card and any card numbered 900 to 999.

Included in the comments at the front of the file are definitions of symbols and abbreviations, descriptions of formats, sample IPDS's; in general, notes on the structure of the VAIF and the relationship of

the VAIF to the entire system. Following are examples of VAIF IPDS's:

EXAMPLE OF TM MONITOR IPDS

```
0001    B4002    INTL EQPT PLATE NEAR BAT ASSY 1 TEMP                  000 BTE
0003    B4002    TYPE=AM TM                                  AVL=VOP   001
0004    B4002    SACT=0868                                             002 BTE
0007    B4002    APER=2              LIM=50/100                        084 COM C
0008    B4002    RANGE=0/300                     UNIT-DF     ACC=3.61  100 BTE
0010    B4002    AMB-50/100                                            108 COM C
0012    B4002    OPLIM=50/105                                          109 COM C
0013    B4002    SC142,0,0,300,255                                     110 BTE
0016    B4002    120,03377,152.7,.040                                  112 COM C
0017    B4002    CHN-R4              EXC=    PART=SV74D12-1            600
0020    B4002    OR(C8001,C8002,C8003)                                 831 COM C
0022    B4002    C=OPLI A=STOP       D=                                850 COM C
0023    B4002    AND(C4001M)                                           861 COM C
0024    B4002    * REV-329,343,378,403                                 990
```

EXAMPLE OF DISCRETE COMMAND

```
0001    S1418    STE DOU-3-018                                         000 BTE
0002    S1418    FIRE B/S CAP SEP CUT VO CMD 75J                       000 COM 567
0004    S1418    TYPE=CON DIS STM                            AVL=V     008
0005    S1418    DQU=3     CHANNEL=018                                 031 BTE
0007    S1418    STE RELAY TURNS AROUND VLC SIG PWR 2                  951
0008    S1418    P20, REG=01 BIT=18                                    952
```

EXAMPLE OF DISCRETE MONITOR

```
0001    S2418    STE DMC-3-018                                         000 BTE
0002    S2418    FIRE B/S SEP CUT VO CMD 75J          (J29-GG)         000 COM 567
0003    S2418    MONITORS S1418                                        000 TE1
0004    S2418    TYPE=DM HL C                         AVL=VOP          0001
0005    S2418    SACT=0361                                             002 BTE
0006    S2418    TYPE=CON DIS                         AVL=V            008
0007    S2418    DSU=3     CHANNEL=018                                 032 BTE
0008    S2418    OFF                                                   082 BTE
```

EXAMPLE OF DISCRETE MONITOR (Continued)

0009	S2418	F304 AWS FUNCT CODES	700
0010	S2418	C=OFF A=STOP D=P CATEG=	850 BTE
0011	S2418	AND(S1418)	861 COM 567
0012	S2418	P20	951

The Viking AGE Group File (VAGF)

The VAGF contained groups of related IPS to provide a short-hand mechanism for all user's of the VAIF. This might be the test sequence writer, the on-line test system operator, or the requestor for post-test data reduction.

The Viking AGE DECOM File

The Viking AGE Decom File contained the data required by the telemetry docommutation software for decommutation of the various Lander telemetry formats. This data included information such as format identification, discrete or analog data, word length, most significant bit first or lease significant bit first data, and format length.

QUALITATIVE RESULTS: The Viking Data Base as structured worked very will in support of the Viking program. Since the total software and data system was a new design, compatibility with the total information system was "designed in". That is, the STE software system was driven by the data base. This design made it easy to change requirements without being required to change code. Defining all interfaces (commands, monitors, telemetry, etc.) with appropriate associated parametric information in one central file was of specific value. The ability of this file to provide for man/machine interfaces greatly enhanced understanding and control of the system during test operations.

QUANTITATIVE IMPACT: The Viking software data file system required 168 engineering man months to develop. This development process contained throughout the life of the STE.

The Viking AGE Interface File (VAIF) contained 1289 line printer pages. Each page contained from 3 to 10 IPDS's. Four hundred forty revisions were made to this file. The Viking AGE Group File (VAGF)

contained approximately 40 pages. The Viking AGE Decom File (VADF) contained 16 decommutation files. The latter were incorporated in the Mission Operational Software System and used to decommutate the telemetry received from the Viking Landers on Mars.

TECHNIQUE

NAME: VIKING TEST LANGUAGE (VTL)

SUMMARY: Viking Lander hardware, subsystem, and system integration and test was performed by a computer based system, the System Test Equipment (STE). The STE software design gave the Test Engineer access and control over the testing process via a relatively simple test language. Test sequences prepared by test engineers were carried out in two stages: translation and checking of the near English sequence into an intermediate form, and subsequent interpretive execution by the STE computer.

APPLICATION CONSIDERATIONS: System/Subsystem test and evaluation requires considerable on-the-spot flexibility. Preconceived test sequences while adequate for fully understood and working hardware, can rarely cope with special tests and malfunction isolation. Moreover, the test support software had to be conceived, designed, built, and verified long before flight versions of the hardware devices would be available. The total volume of test sequences employed grew considerably over the final integration and test phases. Individual test sequences, written in a programming language, and modified in response to a fluid testing environment would have presented an unmanageable software development problem.

RECOMMENDATION: User oriented languages, and test languages in particular, are a proven concept. Martin Marietta has employed the concept on several large projects over the past decade. The particular variations employed on the Viking Project are deemed worthy of wider application. These are: (1) a very simple language that is easy to learn and use with confidence; and, (2) a tightly controlled database that contains all the detailed configuration differences and details - all invoked by device name in a specific test sequence.

HISTORY: The Viking Test Language (VTL) instruction set was conceived and finalized early in the program. This was necessary due to the close tie-in between ground checkout hardware, system data base, and checkout software. No new elements were required during the program. New requirements in test were reflected in data base changes, ground computer checkout software changes, and Flight Computer System Test and Checkout Operation Program (STACOP) software changes. New capabilities were provided thru the use of the VTL such as being able to reset test time by test sequence and the capability of calling from disk a set of pre-established command uplinks by VTL. These capabilities provided timing synchronization between Viking Flight Computer operations and ground computer operations.

The VTL was utilized from the beginning of test. Buildup of the system test bed throughout all phases of integrated system testing proceeded as follows:

a. System Test Bed - hardware, software, sequence debug and test
b. Proof test capsule testing (thermal VAC, VIB Acoustics, etc.)
c. EMC testing
d. Lander buildup and integrated system test
e. Plugs out testing
f. Flight compatibility testing
g. Lander orbiter interface tests
h. Launch pad tests
i. System test bed tests with PTC in support of cruise descent and landed science evaluation operations.

DESCRIPTION: Test Language Overview - The Viking Test Language (VTL) is a high order computer language which gives the test engineer access and control over the testing process. The Viking Lander and STE command and monitoring operations were performed by test sequences constructed from the VTL elements. These sequences resided in computer core during on-line operations and were essential computer subroutines.

The test sequences written in the near-English symbology of the VTL were keypunched into symbolic source card input to the computer's off-line assembly/translation process. The output of the assembly/

translation process was an expanded object file which in turn was loaded to the on-line operating disk. The sequence resided on disks until called by the test operator for execution.

The test sequence (TS) is analogous to a book. It has a beginning, an end, and in the middle there is a general theme. A test sequence is written, reviewed, controlled and executed as an end-to-end operation. The scope and length of a given TS is determined during the initial planning and integration activities, but is primarily a function of the test task. The test sequence is called, initiated, and executed whole or in part and stopped as required for manual test operations. The test operator controls the test via CRT/Keyboard and manual pushbutton controls.

A subset of the Viking test sequences was the 'abort' sequences. The abort sequence is analogous to an "alternate ending" for the story that the normal test sequence has started.

The Abort Sequence is written using the same symbology and form as a normal TS. The abort sequence is different only in the way it is called into execution. The call statement is a default condition occurring within the normal TS.

If, during test sequence execution, an abnormal status is encountered, and if the TS writer has chosen an Abort Sequence as a secondary action, the normal sequence will immediately terminate and the on-line software will call and begin executing the named Abort Sequence.

The Abort Sequence is the only real branching capability of the VTL.

The test sequence consists of a sequence identifier and descriptor (title) and a main body. The identifier is an alpha numeric number in the order of 4 X01-A which provides the calling and configuration control mechanism. The main body of the sequence contains the VTL components required for performing the required test function.

COMPONENTS OF THE LANGUAGE

Test Block - A Test Block (TB) is analogous to a paragraph in a TS. As a general case a TB contains a single stimulus statement, or a closely related set of stimuli, some time delay, and the necessary criteria statements to provide the evaluation of the resultant responses.

The TB is assigned a number (TBNNNNN) and a 32 character descriptor

by the TS writer. The TB number serves to identify any displayed data or criteria violation during TS on-line execution. The data message contains the TB number relative to when the criteria was established and the TB number in which the criteria was violated or requested for display. This feature provides for quicker analysis by reminding the analyst of the headings or original purpose of the test.

The TB 32 character descriptor also serves as an outline to the writer of the TS and as an aid to the reviewer of the TS. This descriptor appears in English form in TS translated listings and is displayed during the on-line execution of a TS. The on-line automatic display feature enables the operator to track the real time execution of the TS.

Alternate Action "STOPS" - If during execution of a TB, a criteria violation is sensed for which the secondary verb "STOP" has been specified, sequence execution will stop at the end of that TB.

The test block is a sequence entry and exit point.

Prior to initiating a sequence in the Automatic Mode, "START ON" and "STOP ON" points may be specified. These points are identified in terms of TBs, i.e., SSB, TB01052, TB20510.

TEST LANGUAGE STATEMENT

A test statement is a line of unexpanded (VTL) source code. It will generate one or more VTL elements during translation. The VTL statement exists before translation to machine instructions and the VTL element exists after translation. To the TS writer, the two terms are almost interchangeable.

A test element is analogous to a single action in a conventional handwritten test procedure, such as pressing a button, reading a meter, etc. Some VTL statements result in single actions. A repeat statement (REPT) is expanded by the translator into more than one element.

Two forms of Test Language Statement exists: the executable and the non-executable. The executable VTL statement refers to its requirement to be acted upon during the translation process by the software and being placed in an "object" form in the object sequence created.

The test language statement may take several forms but in general will always consist of a verb, a noun, and an adjective. In addition, the statement may contain time, either to initiate a timer or a time

as to when the statement is to be executed. It may contain a command to cause a display (CRT and/or line printer) or it may contain a secondary verb such as an "abort" sequence identifier or a 'STOP'. Certain types of VTL statements will contain effectivity nomenclature which is used during translation and loading to disk operations. The effectivity function provided a capability to run a given sequence against one or more vehicles where unique differences in instruments would call for different commands to be executed. For example, camera gains and offsets.

STATEMENT ARGUMENTS

The VTL arguments are analogous to the noun, verb, adjective, etc. used in normal sentence structure and will be explained in those terms.

VTL Test Element Format - The general form of the EXECUTABLE test element/statement is shown below. Each modifier field is also defined. Executable test elements perform actual test operations. The numbers listed below are the column numbers on an input card or on a coding form.

TIME	VERB	NOUN	ADJ	EV TH	D	VERB 2
1-	9-	17-	25-	49-	52-	57-

TIME - The Time entry allows an element to be executed at some increase of time. The time entry is optional. The first character must start in Col. 1 and the entry must not exceed Col. 8.

1) Tn + i This is the general form for referencing an event timer where n = 1, 2, 3, and i is an increment of milliseconds or seconds. The "+" means at the specified time increment, execute this element. See WHEN element for additional information.
2) Event timers provide relative timing between the various executable elements within a test block.
3) Event timers are reset and started by a T1, T2, or T3 in columns 49 and 50 of a VTL statement.
4) Event timer references must allow 5 milliseconds for each executable element.
5) Event timers can be reset and started as often as required in a given test block.
6) Timing increments can be in milliseconds or minutes and seconds.

VERB - Each executable test element statement requires a VTL primary verb in this field beginning at Col. 9. The verb specifies the element action, such as command a relay closure or establishing monitoring criteria. The other element statement fields shown above provide for element modifiers, as dictated by each verb and its sequence function.

NOUN - The nouns (beginning at Col. 17) declare the object of the primary verb action and consist of identifiers for interface points, commands, and measurements.

ADJECTIVE - The adjectives define the primary verb action; such as condition, limits, values, and references. The adjective field starts at Col. 25.

1) ON, OFF - self explanatory discrete conditions:
   OPEN - don't care condition.
   CLOSE - reverts to the condition established before the last OPEN statement.
2) OPLIM - operating limits as separately defined for analog voltage measurements: i.e., 23/32 VDC.
3) AMB - ambient condition: opposite ot OPLIM where measurement is OFF/false, and also ambient temperature, pressure type measurements.
4) LIMITS - For analog monitors, actual limits and engineering units can be written on the card in the form lower/upper, units. Both limits and units must be entered.

EVENT TIMER - Tn is the general form for reinitializing an event timer, where n = 1, 2, 3, or 4. The event timer field starts in Col 49.

1) Timers specified in this element field are initialized to begin timing after the primary action specified in the element is complete.
2) Timers provide a maximum time of 99 minutes and 59 seconds.
3) Timers are reset by each test block and event time does not carry across TB boundaries. A timer can be reinitialized by its entry in this field on succeeding elements.
4) Refer to TIME field for event timer functions.

DATA DISPLAY - The D field, Col 52, provides the capability to display data to the line printer and CRT. When used, the value of the NOUN is

displayed.

1) In the case of a criteria violation the value of the noun shall be automatically displayed by the on-line system whether or not the test writer used display for the failed element.

2) L - display to the CRT.

   C - display to the line printer and CRT.

VERB 2 - Secondary verbs allow for alternate actions in case of criteria violation. These actions are only allowed for monitor functions i.e., DISPLAY, WHEN, ESTAB and CHECK elements.

1) GO TO - The GO TO secondary verb allows the test writer to abort the present test sequence when the primary action fails. The GO TO secondary verb is an implied GO TO inasmuch as the words GO TO are not written. The test writer simply writes the name of the sequence to which control is to be transferred of the form PBNN-A (for Planned) GO TO requires off-line translation and cannot be used as a primary verb. Execution of the GO TO causes the on-line interpreter to initiate lockout flags which prevent the resumption of the test sequence. The GO TO sequence is loaded for execution by the on-line system and provides all hardware backout provisions. When the GO TO verb is executed, its execution results in an immediate and complete abort of the sequence and no return is possible, except by recalling the sequence and specifying a "start on" point.

2) ABORT - Same as "GO TO" sequence except as follows:

   A. There is only one location in CPU memory for one ABORT sequence. This means that special provisions must be made by the on-line operator to load the correct ABORT sequence for the normal test sequence which is to be running. On-line control language will be used to load the ABORT.

   B. The ABORT sequence when called by a test sequence criteria violation, will not require access time delays such as loading from the disk to the computer memory. By having placed the ABORT sequence in memory it will be

accessed and executed with the minimum time delay possible.

3) STOP - The STOP secondary verb allows the test writer to cause the sequence to pause after execution of a display element or to pause after completion of the test block when the primary action fails. The sequence can be resumed and thus is not aborted. STOP cannot be used as a primary verb and requires translation by the off-line system. The STOP secondary verb is written STOP. If the STOP verb is executed, the test will be discontinued after completion of the Test Block being executed. At this point the test operator may request that a Recovery Sequence be executed, or another TB selected as a starting point, or the test can be continued from the point at which it stopped by depressing the "P", then escape on the CRT keyboard.

The general form of the NON-EXECUTABLE test element statement is shown below with field definitions. Non-executable test elements command or control the off-line translator. Comments, sequence identification, and REPT, the repeat code name, and ENDR verbs are non-executable elements.

IDENTITY	ELEMENT TYPE	MODIFIERS. DATA, LISTS
1-	9-	17-

IDENTITY - A name or label which identifies data, listings or a routine to the translator. An asterisk in Col 1 of this field identifies comments.

ELEMENT TYPE - This field defines the element type (REPT, GROUP, LIST, etc.) starting in Col 9.

MODIFIER - Data and listings are located in this field as necessary for each identifier and verb, starting in Col 17.

DEFINITION OF VTL ELEMENTS

The BEGIN element marks the beginning of the first executable statements in a test sequence. It is an executable element and requires translation by the off-line system. When executed, the on-line interpreter is reinitialized for a new test sequence and all flags, counters and sequence timers are initialized. In addition a display is made to CRT and line printer according to the on-line requirements. The BEGIN verb

does not require any modifiers. The Test Block or Critical Block element must follow immediately after the BEGIN element.

The CHECK test element interrogates the value or state of any currently monitored parameter. It is an executable element and requires translation by the off-line system. Execution of the CHECK element will not change the status and criteria tables. It evaluates the parameter with respect to the limits or conditions specified in the adjective field.

The DISPLAY element causes up to 40 characters to be displayed to the LINE PRINTER and/or the CRT. The display area reserved on the CRT screen for this type of display is one line long. The element does not utilize an adjective or initiate a timer and the second verb is optional. This form of the DISPLAY element specifies messages (not data) to be displayed to the test operator on the CRT and line printer. It is an executable element and requires translation by the off line system.

The ESTABLISH element is used to establish monitoring criteria on discretes, discrete groups and analogs. The ESTABLISH test element changes the status and criteria tables in the Monitor system for discretes, and analogs. It is an executable element and requires translation by the off line system. When executed by the on-line interpreter, the criteria will remain as changed until another ESTABLISH element is executed for the same monitor parameter. This is true even though the sequence has reached the End element. In all cases, if the adjective used is OPEN, the secondary verb must be blank. If the adjective OPEN is used, the adjective CLOSE may be used. CLOSE will restore the adjective state to that state which exists just previous to the OPEN state.

The WHEN element provides the capability to synchronize or pace the test sequence based on external discrete monitor occurrence. This is accomplished by a three step execution of the element. Step one "opens" the criteria on the monitor being addressed. Step two alerts the system to an expected change and the expected state of the discrete monitor(s). Step three is completed when the discrete monitor(s) change(s). Step three would also start event timers and displays if these options were invoked. Step three would be accomplished immediately if the discrete monitor(s) was already in the expected state.

Failure of the element occurs when the time period specified elapses and the discrete does not change to the expected state. The WHEN element is an executable test element and requires translation by the off-line system. All WHEN elements require use of the time modifier.

The END test element marks the end of a test sequence. It is an executable element and requires translation by the off-line system. When executed, the on-line interpreter displays and records completion messages to the test operator. It also puts the on-line software into the monitor mode. The END verb does not require any modifiers.

The SET element enables control of stimulus discretes and commands to be implemented via hardware. It is an executable element and requires translation by the off-line system.

The REPEAT element is a nonexecutable test element and is not part of the translated object code. It serves as a translation control command only. It is used to define repeat blocks in the REPEAT input section (before the Begin element) of the test sequence source input decks. It is also used to identify where in the sequence code the repeatable code should be inserted (after the BEGIN element). Repeatable coding features allow the test writer to reduce the burden of rewriting the same or similar string of coding. These features provide the capability to write a string of coding once and to give it a name. Whenever this string is repeated the name of the coding string becomes the primary verb. The translator will substitute the repeatable code for the name at translation time. Repeat tests can be referenced for use as many times as required in the body of the test sequence. If RCBs are not needed, the Repeat input section is left out of the sequence.

The ENDR element terminates a repeat code block. It is not executable and is not included as part of the translated object output. It serves as a source input control card only and as such will appear as part of the source listing output only. The ENDR verb does not require any modifiers.

A Sequence ID (SID) card is used to identify the sequence to be translated. The SID card must always be the first card of each Test Sequence. Translation will not be performed without the SID card.

The sequence descriptor appears at the top of each listing page and is displayed on the CRT during on-line execution of the sequence.

Test Block Numbers (TBN) are provided to allow sequence writers to identify blocks of test elements which perform modular test functions. TBN is used during execution to inform the test operator of the test block to be executed in the test sequence. Test Block Numbers will be followed by a descriptive text identifying the test block which is to be executed. During translation the Test Block Number and its corresponding text are formulated into a special display command element. When executed by the on-line system, the TBN is displayed on the CRT and line printer.

QUALITATIVE RESULTS: Utilization of a relative simple test language for test control of the Viking Lander system proved invaluable all during the program, expecially in the final stages. The relatively easy "tie-in" between test data results (line-printer printouts) and the test sequence provided hardware and science oriented engineers (not necessarily with any computer software background) the capability of understanding test aspects and test results. The "QUICK-LOOK" of data results were enhanced to a great degree.

The positive aspects of the VTL as verified during Viking test are:

1. Design and control of tests by hardware oriented engineers;
2. Test sequence/procedure English Form lent itself to checking by engineers and checkers not software oriented;
3. Capability for rapid turnaround of test sequences for malfunction isolation, sequence updates and alternate mode testing;
4. Capability for entry points for running a sequence in parts or in jumping from parts of one sequence to parts of another sequence;
5. Capability for changing abort sequences easily;

6. Capability for changing data acquisition modes quickly and easily (significant due to the many TM formats and data rates of Viking);
7. Easy change of software decommunication of TM data compatible with 6 above;
8. Easy tie-in between test data results and test sequences;
9. Follow-on testing with reduced manpower;
10. Versatility of VTL with the other system components for running a great variety of tests;
11. Capability for sectionalized testing.

The difficulties encountered with the VTL were more related to the implementation of the VTL concept rather than in the language itself. These difficulties probably were no more or less than would be encountered with any new system introduced to engineering and test personnel.

It was found through trial and error, that a test sequence writer must have the same basic logic talents that a software or hardware designer has and should be system oriented. To write a Viking test sequence, required knowledge of the test article (operation and interfaces), the system test equipment operational setups and man-machine interfaces, the software operational interfaces, the system test and checkout operational programs, the data base and the test language plus the other associated disciplines such as test system simulator, configuration control etc.

Probably the most significant item in the use of a VTL language for a system as complex as Viking is the selection of the test sequence writers/designers.

QUANTITATIVE IMPACT: The responsibilities of the personnel who developed the Viking Test Language sequences differed relative to the test stage. The responsibilities, and approximate effort for each are as follows:

1. Test sequence design and generation - 50%
2. On-line test support for sequence debug - 10%
3. On-line test support - 20%
4. Data review - 10%

5. Working anomelies - 3%

6. Identifying data file changes - 2%

7. Inputs for software changes - 2%

8. Procedure development - 3%

The time required for application program test sequence design decreased exponentially as the Viking test program progressed. This can be attributed to the human learning process plus the establishment of a data bank of sequences which could be drawn upon for major or minor portions of new test sequences. Early in the program a 3000 line sequence took two to three weeks to develop. Later on a similar sequence typically could be generated in 8 hours. Completely new sequences still required approximately 100 manhours to develop.

During Viking system test bed operations in support of cruise and Landed science few sequences took more than 8 hours to develop. More time was spent in establishing requirements and writing test procedures.

A total of 1565 test sequences, containing 974144 lines, were generated for Viking. Of these 332 exceeded 1000 lines and 88 sequences exceeded 3000 lines.

The development of the Viking Test Language translation software subsystem took an effort of 52 manmonths to generate 11486 lines of code.

TECHNIQUE

NAME: TEST SYSTEM SIMULATOR (TSS)

SUMMARY: The Test System Simulator (TSS) was conceived and developed to provide the capability to debug the Viking automated test sequences prior to using them to actually test hardware. The TSS is a general purpose data driven software simulator that operates in the Honeywell H-632 computer system. No hardware other than the computer set and its standard peripherals is required for simulation. The data required and simulation statements became an integral part of the master data file used to support translation of automated test sequences and computer controlled testing.

APPLICATION CONSIDERATIONS: Rationale for developing the TSS was influenced by the necessity to have test sequences designed and ready prior to hardware availability. Factors considered were: adequate debug of sequences could preclude hardware damage; sequence timing problems could be resolved; possible hardware problems could be detected early; sequence debug could save many manhours of time in translation; and the simulation would increase sequence designers understanding of the Lander system.

RECOMMENDATION: The test system simulator technique as used on the Viking Project is a proven concept. It provided a valuable tool in debugging automated test sequences for a relatively low cost. A simulator of this design can be used to support a parent project in sequence and procedure debug, sequence of events verification, failure modes and effects analysis, power profile, hardware/software design verification, and crew training and certification. The use of the simulator technique will not eliminate the need for bread boards, prototypes or test beds.

HISTORY: The Viking Test System Simulator was formulated in early 1972. It was recognized that a simulator could be of decided value; however, due to the pressures on the program, little time or money was available for accomplishing the task. The basic design philosophy of the TSS, simply stated, was to provide a mechanism whereby design and test engineers could incrementally add bits and pieces of data concerning any part of the test or mission hardware, software or operational considerations, to the computer memory bank. The computer would integrate all of the data into a single unified model, and use this model to assist the engineers as much as possible throughout the program. The first cut at the approach was to provide a simple functional simulator which included simulated vehicle discrete command and discrete responses with appropriate timing. Additional logic was added thru data base simulator inputs as time permitted and as Lander component and science instrument final designs were completed. The power and pyro subsystem sequences were the first sequences run thru the simulator. With this came the task of debugging the simulator software and simulator logic statements in the data base along with the sequences. Some debug of software and simulation logic was required when new logic was added to the data base. As the power, pyro, vibration acoustic, pyro shock and initial mission sequence of events were designed, they were debugged by the simulator. The simulator proved effective in this respect.

At this point in the program many parallel sequence design efforts were in progress or beginning to develop, such as combined system tests (vehicle health checks for the proof test capsule during thermal vacuum tests), electromagnetic compatibility test, science-end-to-end evaluation tests, etc. Manpower to continue logic development and training time was limited. Logic for meteorology and x-ray was developed as time permitted with additional work being performed in other system areas. The above sequence types were run against the simulator with respect to power up, pyro, and basic discrete logic functions.

Beyond the discrete stimulus and response stages, it became increasingly difficult to provide the simulator logic.

The advanced design of the System Test and Checkout Program (STACOP) which was the test software loaded in the flight computer made the testing more autonomous and less dependent upon commands initiated by STE. Commands were initiated by table look up in the flight computer or by macro programs.

The Gas Chromatograph Mass Spectrometer and biology instruments were driven by internal programs, in themselves, as commanded by single initiate commands.

Each of the science instruments had many modes of operation which changed data response timing.

Testing evolved to the use of actual flight software for system test where all sequencing of events were initiated by the loaded program.

By the end of the Viking sequence generation, approximately 50% of the vehicle logic and about 95% of the supporting hardware had been simulated.

DESCRIPTION: The test system simulator (TSS) is a general purpose, data driven simulator that runs in the Honeywell H-632 computer system. It can simulate the operation of any system where the relationship between the elements of the system can be described by a logical expression. No hardware other than the computer and its standard peripherals are required for simulation. The simulation is time and resource oriented in that a response to a stimuli may occur immediately or at a specified time after the pre-requisite conditions for the response are established. When a condition exists that supplies or depletes a resource, such as electrical power, the composite rate is calculated and the total is integrated as a function of time. When the system being simulated includes a computer, the programming of the computer is simulated by appropriate entries in the data base.

The TSS consists of the three basics: the input, the operating system, and the output.

The input stimuli, environment definition, and criteria are input to the simulator as a series of time oriented control statements (procedures) from mass storage (or in the form of a card deck). These may be input in parallel, the control statements from each being interleaved as a function of time.

The logic for the TSS resides as a subset of the Viking STE data base. The logic is designed to provide the TSS with the 'transfer function' of the component, sub-system, or system. The logic consists of two general types. The first is a direct representation of hardware circuits. The second is 'functional' logic. Each type has its own special considerations. The logical unit in the TSS is the 'terminal'. Terminals are classified in several ways. The most general distinction is between status and message terminals. A status terminal is used to 'remember' the state of a hardware point or a software flag or buffer. All status terminals have an entry in the VIKING information system status and criteria table (SACT).

Messages are not true terminals. They are stored in mass memory and accessed when 'SET' by the test language or the on-line control language. A message may be a pulse or a digital word. A special class of message is the delay type. A delay terminal is used to cause an event to occur at a pre-determined time after a specific set of conditions exist. All delays are momentary. Most messages do not have an entry in the SACT. A SACT entry is required if the message is generated automatically.

All TSS logic is expressed as 'AND' functions or 'OR' functions. 'Nesting' of functions is not allowed; the equivalent of the nesting function is implemented thru the use of sub-terminals. 'NOT' functions are available and timing is accomplished thru the use of the 'delay' terminals.

TSS peculiar terminals were created, as required, to provide a complete model of VLC or STE functions. The I D symbols were structured so as to position the IPDS in the appropriate part of the VAIF (adjacent to related IPS in the real system).

The hardware logic (power buses, relays, circuit breakers, switches, push buttons etc), are directly represented by TSS terminals. A magnetic latching relay is represented by a 'memory' terminal. A command may be a monentary command, or a continuous discrete stimulus. A normal relay may be a continuous terminal or a memory terminal (if it was 'latched' by another terminal). The capacity of the TSS, with existing

computer memory is approximately 10000 terminals. This capacity could be signifi(antly increased with a nominal software effort without an increase in computer hardware.

Much of the logic is functional in nature. A command may establish a mode directly without any representation of the individual circuit elements involved in the mode. The same is true of most software logic in that individual bits and instructions are not represented.

A powerful mechanism is the inclusion of configuration items in the logic. This means that the configuration could be changed on-line. Special Interface data points are assigned for this application.

The operating system of TSS includes three major software subsystems; the sequence expander (TSSE), the binary file management system (TSSB), and the run-time programs (TSSR). In addition, various input/output, scanning and conversion subroutines are used.

The sequence expander performs a function equivalent to the VTL translator in the real system. The TSSB is the simulation language 'compiler'. It reads the source statements (VAIF DATA CARDS) and performs the conversions, table construction, and linkages required to construct the model of the system to be simulated. The data for the specific model desired is selected based on the load effectivity statement. The first function of the TSSB is to read the load statement to set up the selection process. After a card has been selected, control is transferred to the appropriate card processor. The 'model' constructed by the TSSB is a combination of the data stored in the SACT and in the Viking Binary Interface file.

The load algorithm controlling the data selection process is written specifically to satisfy the requirements of the Viking program.

The simulator run time software utilizes the real system status and criteria table which is a key element. All of the run-time routines use the SACT in the same way. The run time software includes the run time executive, which provides an environment wherein the tasks that comprise the TSS run-time simulator can be executed according to their respective states and priorities. The six priority tasks are the Timing System Transmitter, Dynamic Status Change, Dynamic Response Simulation,

Test Language Processor, Control Language Processor and Timing System completion. Since the simulator runs according to pseudo-time rather than real time, there is no need to respond to interrupts. Hence, all processing can proceed at one CP level.

The timing system transmitter (TCB) task resides at a priority that is higher than all tasks which process jobs that are required to be performed at a specific pseudo-time. Because of its high priority, it is able to transmit jobs to the executive without interruption. It then deactivates, allowing tasks of lower priority to execute the jobs which have been held for them.

The Dynamic Status Changing (DSC) task processes all changes-of-status that occur to any Status and Criteria Table entry. The Dynamic Status Changing Task is activated by any task with a properly encoded job word in the TCB.

When the task gets control a test is first made to determine if this is a bits-per-second return. If so, then bits are added to the specific bit buffer and the task deactivates. If it is not a BPS return, then a test is made to determine whether the new status is different than the current status. If no change is required, this task deactivates.

If a change is required, a test is made to see if the Status and Criteria Table (SCT) entry is failed in its current status. If so, a message is formatted and output. If the SCT entry is not failed, a check is made to see that it has not already changed at the current pseudo-time. Then its status is changed and corresponding power level changes are made.

The Dynamic Response Simulation Task (DRS) task simulates the responses to all change-of-status that occur to any Status and Criteria Table entry or command. The DRS task is activated only by the Timing System Completion Task when it has determined that the pseudo-time must be changed.

When this task is activated, one pass is made over the Status and Criteria Table, searching for those entries which have been marked as "Affected" by change of status or command. When such a SCT entry is

found, its proper status is computed based upon the logic equations in the Binary Interface Data Set (BIDS). This new status is compared with the current status; if they are the same then no change-of-status is required so the next SCT entries are checked. If the new status and current status differ then the new status is sent to the Dynamic Status Change Task, which resides at a higher priority, for final processing. If the SCT entry is of the "Delayed" type, the job to the DSC task will be held in the Hold-Chain until the proper pseudo time.

When the DSC task makes the change of status, it may also mark additional SCT entries as "Affected", and will again set the response-simulation-required flag. After completing one pass over the SCT, the flag is tested to see if further responses are required. If so, another pass is made over the SCT. This process is repeated until all responses are complete for the current pseudo-time, then this task deactivates.

The Test Language Processor Task processes Test Language Sequences from the disk as specified by the Control Language Processor. When this task gets control the first thing it checks is to see if an external wait is set so that control cards can be read and processed by the CLP in parallel with the sequence. If it is time to honor the wait, control is passed to the Control Language Processor task. Otherwise, updating of flags and timers in the current sequence buffer is performed and the time field on all executable elements is processed to check if a delay is required. If no delay is required, the element is formatted and displayed.

The proper processor for the current element gains control now and processes that element. On return any errors found by the processor are flagged and displayed. An update to the sequence pointer is done so that the next element will be processed when this task again gets control. If an error has occurred while processing the current element, control is given to the Control Language Processor so that the operator can decide what corrective action needs to be taken. Otherwise a delay of 5 milliseconds pseudo time is performed by suspending the task for this amount of time.

The Control Language Processor Task processes all control statements for controlling the simulation run. The control statements are

divided into four basic categories: sequence control, statements that are equivalent to test language statements and general control statements including statements beyond the test language capabilities. The control statements belonging to each category are:

Sequence Control - CALL, EXEC, MODE, PROCEED, RESTART, RESUME, RUN, START ON, STOP ON, TEST.

Test Language Equivalent - ACQUIRE, ESTAB, SET

General Control - CLEAR, CARD, DUMP, FAIL KEY, INACTIVE, MOD, REMOVE, RESTORE, SAVE, SETDEL, SETMAX, STOP, TIME, TSTV AT, WAIT

Display - CANCEL, C PRINT, DISPLAY, GROUP, STATUS, VIEW

When the Control Language Processor Task is activated it inputs a control statement from the card reader or the CRT keyboard, outputs the control statement to the CRT and line printer. If the statement is EXEC, PROCEED, RESTART, RUN, or RESUME the necessary processing is performed followed by a call to the Executive to deactivate the CLP task and activate the TLP task. If the control statement is not one of the above, the necessary processing for that control statement is performed and a new control statement is input.

The Timing System Completion Task resides at a priority that is lower than all tasks which process jobs that are required to be performed at a specific pseudo-time. Hence, control falls to this task only when all jobs for the current pseudo-time have been processed. The main function of the Timing System Completion Task is to make sure that all jobs for the current pseudo-time have completed and then advance the pseudo-time clock. When all jobs for the new pseudo-time are complete, control again falls to this task. Hence, this task never deactivates.

The Man-Machine interface is an important part of the operating system. It is essentially the same during simulation as it is during on-line operation. Input is thru sequence tapes, CRT Keyboard, and card reader. Output is CRT display and line printer tabulation. Hardware functions such as circuit breakers, pushbuttons, connection of

test equipment, etc., are accomplished by control statements (since no project hardware, other than computer hardware is involved in the simulation).

The TSS operator has a 'Hands on' capability for total control of the system during any point in simulator operation. Since the TSS operates in 'pseudo-time', and controls this time, the operator may 'freeze' the system while he evaluates the data to determine a course of action. If alternate courses of action appear appropriate, he may save the status of the total system. After each alternate is tried, the system may be restored to the saved point in preparation for the next alternate.

The basic time increment is the millisecond. The on-line system stops the clock while it accomplishes all the operations necessary at that time. The time is then advanced to the next time in the timing system 'delay line'. The next time may be a VTL element, a delayed response, an automatic stimulus, or the expiration of the time delay specified by a wait command.

On-line error and special messages provide the operator with information on the functioning of the TSS, the operator/software interface, and interaction of the various logic terminals. They consist of eight characters, and are displayed in the first field of the CRT data lines, and the fifth field of the line printer display.

Simulator output is in the form of line printer tabulation and real-time cathode ray tube display. The level of detail to be displayed may be controlled by the operator. Several mechanisms exist to specify critical functions to be 'high-lighted' on both the line printer and CRT. All output is in engineering terms and all changes are in the form of a measurement or terminal number with a 40 character, English language descriptor.

QUALITATIVE RESULTS: The data that drives the TSS is compiled using a simulation language so simple in structure and use that it is hardly recognizable as a new language. The simulation statements become an integral part of the master file used to support translation of automated test sequences and computer testing and data monitoring and display.

Three major characteristics of the TSS distinguish it from other forms of computer modeling techniques. These are its commonality with the system being simulated, the universality of its capability to support the parent project and the ability to operate on incomplete data.

An important benefit is derived from using the TSS to debug Viking test sequences in that some sequence errors are detected that do not show up as errors when the sequence is run against actual hardware. Some of these errors may result in overstressing mission hardware in a manner that is not visible, even during post test data processing.

As the mission hardware and software is exercised in its various configurations and modes, the adequacy of the design is established. The TSS does not elminate the requirement for integrated hardware/software validation, but it can provide early visibility and can detect some types of design inadequacy.

The commonality of man-machine interface between TSS operation, and operation with the real system, makes the TSS an effective training tool. Training may be accomplished concurrent with sequence debug, with no use of, or risk to mission hardware. Selected failures can be simulated to provide a realistic certification environment.

As the data file is loaded into mass memory, the TSS software constructs a 'model' using whatever data is available from the file. Several pre-run and 'on-line' techniques exist to bridge the gaps in the logic until complete data is available. The significant point is that the TSS may be used very early in any new project.

The Test System Simulator does not replace intelligence sequence design. There is no way for it to determine what tests or commands should have been run or put in the sequence without unnecessarily complicating the logic.

Viking testing both at KSC and in Denver's system test bed permitted jumping from test block to test block in different sequences. The basic TSS did not catch the synergistic effects resulting from this type of operation. However, sufficient experience had been accumulated by then so that little risk existed.

QUANTITATIVE IMPACT: The cost to develop the TSS was considerably reduced by taking advantage of existing STE software. That portion unique to the TSS was developed in 9 manmonths. To develop all of the software from scratch would probably have required a one to two man year level of effort.

Fourteen manmonths were consumed in generating inputs, running sequences and reviewing outputs. Computer resources required up to 30 minutes set up time per sequence, processing 30 to 100 sequence lines per minute. Approximately 5 percent of the 974,144 Viking test sequence lines were processed by the simulator.

TECHNIQUE

NAME: FLIGHT SYSTEMS TEST AND CHECKOUT PROGRAM (STACOP)

SUMMARY: The Viking Lander hardware was configured such that the only feasible communication and control paths to on-board subsystems and science experiments were through the Flight Computer and Telemetry Systems themselves. A special computer program (STACOP) was written for the onboard computer to facilitate the extensive checkout operations required for system integration, test, and prelaunch operations.

APPLICATION CONSIDERATIONS: Since the on-board equipment was largely controlled by the on-board computer, there was no need to complicate interfaces by having separate STE connections with all devices and subsystems. Several operational paths existed to the flight computer (GCSC) from which the desired sequencing and control could be accomplished. Capabilities provided via STACOP included: simple one-by-one command thruput to a specified device, and return of status words; stored list of commands to be issued to specified devices at specified time intervals; accommodation of individual computer/device interaction for selected subsystems; management of several subsystem interactions where thruput/response activities could not provide timely and coordinated combined testing operations. The first three general capabilities were originally implemented. The fourth was added later in the development as more complex system testing requirements were identified.

RECOMMENDATION: STACOP provided the needed mechanism to perform required testing via the on-board computer. It was based upon the existing GCSC operating program services so that relatively little new code was needed. Careful consideration must be given to possible timing and interaction requirements as part of complete system checkout operations. Delays associated with uplink commanding, STACOP interpretation, on-board commanding telemetry return, and STE checking are too cumbersome to accommodate realistic interaction of several on-board subsystems throughout system level exercises.

HISTORY: The STE was developed to test the Viking Lander hardware component subsystems. Since the Guidance, Control and Sequencing Computer (GCSC) could access and monitor these subsystems directly via I/O, interrupt and discrete registers, it was natural to consider developing a general purpose test program that could execute in the GCSC under control of the STE. In that way the STE could exercise sequences that would test the hardware interfaces between the GCSC and the Lander subsystems. The Viking Systems Test and Checkout Program (STACOP) was defined to meet this requirement.

It was envisioned that STACOP and Flight software could be completely checked out and verified using the Standard IC-7000 GCSC emulator and that no significant problems would be encountered when the software was loaded into the actual Lander GCSC. This proved to be overly optimistic, since the very first time that a GCSC load was attempted via the STE, the attempt failed and the load was not realized. GCSC test support equipment (TSE) was made available on a temporary basis to resolve this problem. The permanent solution was to develop a Computer Control and Display Unit (CCDU), of which three were produced.

The CCDU was connected directly to the Lander GCSC by means of cables routed through the bottom of the Lander equipment bay. The CCDU was used to monitor the STACOP program during tests that the lander bottom plate was not required to be installed. During tests in which the plate was installed, GCSC visibility was reduced to merely a power on/power off prediction. A GCSC memory readout function was added to STACOP to provide greater trouble shooting capability when the CCDU was not available.

STACOP was used during Lander integration, subsystem verification, and pre-launch checkout for two and on-half years prior to the Viking launches. It was a general purpose test program in the sense that it was not a canned sequence of pre-programmed events which drove the Lander through a rigid series of operations.

DESCRIPTION: The System Test and Checkout Program (STACOP) was operated in the Guidance, Control and Sequencing Computer (GCSC) in a Viking Lander

Capsule configuration which included an interface with the System Test Equipment (STE) Honeywell 632 computer. A modified version of the Mission Executive of the Viking Flight Program was employed to perform input/output processing, scheduling and interrupt processing. The executive acted as a resident Operating System for the GCSC with STACOP being a collection of functions to be performed under direction of the STE and using the services of the executive.

The STACOP functions consisted of an initialization program, a controller program, an external interrupt processor, a discrete input register monitor, a telemetry program, a priority interrupt processor, a command storage common routine, the Honeywell GCSC self-test program, a command processor, an output subroutine, and a terminate STACOP common routine.

The STACOP functions were controlled by the STE via directives transmitted to the GCSC through a STE/GCSC interface. The directives contained information specific to individual functions and included processing requests to schedule or terminate stored sub-programs, input or output serial or guidance and control data, read or issue discretes, store data into memory, and downlink memory data. The STACOP program could communicate with the STE by transmitting downlink messages via the STE/GCSC interface.

The STACOP program was initialized following GCSC power-on. During operations, STACOP would process priority interrupts, external interrupts and status changes obtained by monitoring a discrete register. These events would occur during the normal performance of the various Viking Lander hardware subsystems. In response to the STE Data Ready external interrupt, STACOP would receive and process directives from the STE. Thus the input parameters received by the STACOP program were interrupts, discretes and uplink directives. The output parameters of the STACOP program were commands and data issued to the VL hardware subsystems and STE downlink messages.

The STACOP program and data consisted of 7724 GCSC memory words. The program loader was 139 words. The executive, including the GCSC self-test program, was 4522 words. No flight software other than that mentioned herein operated in the GCSC when STACOP was run.

QUALITATIVE RESULTS: STACOP provided a very flexible means of controlling Lander operations. Test Sequences, written in the easy-to-use Viking Test Language, were translated by STE into "Directives" which were then uplinked to STACOP for processing. Hundreds of test sequences were used during Viking Lander development. Some tests were very simple and focused upon only one lander component, while other tests were designed to test many components simultaneously. An important advantage of this philosophy of testing is the Engineer involved with the development of a particular Lander component could also be intimately involved in the testing of that component without having to be well versed in the area of computers and software. Also, during Lander integration, testing could proceed even though many of the Lander components were not yet installed.

STACOP served as a test-bed for Flight and Flight-like programs. The STACOP program consisted of several actual Viking Flight Program modules including the Executive Program with its Interrupt Processor, Input/Output Processor and Task Scheduler, the GCSC Self-Test (Diagnostic) Program, and the Telemetry Program. Other programs within STACOP while not being actual Flight Program modules, were written following the same programming philosophies, techniques, limitations, and constraints as those used to develop the Flight Program. In this way, many thousands of hours of execution time had already been logged by the time the Viking Flight Program was first loaded into a real GCSC.

It should be emphasized that visibility into a system central computer, like the GCSC, provides much more than just a programmer tool to monitor software performance. It can also provide visibility into the performance of other components and even the entire system under actual operating conditions. If the CCDU would have been available during all Viking Lander testing, some software problems, hardware problems and other problems which, with limited visibility appeared to be software or computer problems, could have been identified and corrected sooner, resulting in less down time and less risk of equipment damage.

QUANTITATIVE IMPACT: The original STACOP program was developed with a one man year effort, largely because many functions were available from Flight code. Thereafter, an additional man year development effort was required to handle new requirements and correct errors.

TECHNIQUE

NAME: SCIENCE INSTRUMENT PERFORMANCE VERIFICATION

SUMMARY: The science instrument data recorded during the STE checkout and verification testing of the Viking Lander subsystems were dumped to magnetic tape. This magnetic tape was then used as input to large scale computer systems to provide realistic test data for the development of the science instrument Mission Operational Software System analysis programs, which in turn provided verification of science Flight Article subsystem instrument performance.

APPLICATION CONSIDERATIONS: During Post-test analysis the STE was required to analyze and verify the performance of the Viking Lander science instrument subsystems. The Honeywell 632 Computer Set used by the STE could not compile the FORTRAN science instrument analysis programs. In addition, no realistic data were available to support the development of the science instrument analysis programs. Therefore, the technique of taking the raw data gathered by the STE to the science cognizant engineers for analysis offered a cost effective means to resolve both of these problems.

RECOMMENDATION: The technique greatly aided two areas of Viking software. It helped the STE area by providing evaluation of the recorded science instrument performance data, and it provided realistic, rather than hand generated, test data to the Viking Flight Team science engineers. The technique is applicable to any project developing both vehicle checkout systems and operational support systems.

HISTORY: No formal requirements were generated for the STE post-test software subsystem during the early development phase. However, at that time plans were formulated to compile the developmental versions of the Mission Operational science analysis programs on the Honeywell H-632 computer system. In that way the programs could be used to analyze the performance of the Viking Lander science instruments.

No more consideration was given to this subject until late in the STE software system development period when work finally began in earnest to develop the post-test subsystem. At that time the X-ray fluorescence science analysis program, ICAN, was converted to run on the Honeywell computer. It was then discovered that the FORTRAN compiler available on the Honeywell system, which was an unfinished, one-pass compiler with 45 open hooks, was incapable of producing executable code. In addition, no floating point hardware or software capabilities existed. The options available to carry out the original plan were therefore limited to developing a FORTRAN compiler that would work, or writing a floating point software function and develop assembly language versions of the science analysis programs.

The programmers that were developing the STE post-test software subsystem suggested that a better approach would be to develop a capability to allow the science instrument performance data gathered in the STE to be made available to the science analysis programs that were operational on a CDC 6500 computer set. This met opposition from two sources. The STE Software Chief opposed the concept, believing that the STE would lose control over the process. The Viking Flight Team Lander science team leader took the position that it was an additional resource consuming task done purely to support the STE and of no value to the science team. Resolution of the problem was further compounded by the fact that two separate directorates were involved, and no system integration team existed that could have forced a resolution.

The result was that the problem was resolved at the worker's level behind management's back and without management approval. The science cognizant engineers supplied the STE programmers with the necessary

requirements to develop the technique, and the STE programmers did the rest. Once it became a fait accompli, it was reluctantly accepted by management. Eventually it became the approved method of validating science instrument performance.

DESCRIPTION: The requirements to accomplish this technique were:

(1) to record on the STE system all science instrument data during checkout tests;

(2) record this data on magnetic tape by instrument; and

(3) process the magnetic tape to produce data files in a format acceptable to the various Viking Flight Team science programs.

The following discussion describes how each requirement was met.

The data recording requirement on the STE was specified in the STE Software Requirements Document, and had been developed before the post-test processing problem arose. However, telemetry data were recorded in a serial manner as they occurred which resulted in all types of data (e.g., science, engineering) being interspersed on the recording media. The recording media was the Operational Log Tape (OLT) which was always on-line during Lander System. The OLT could span over several reels during a lengthy test.

Given that the data were recorded on the OLT, the problem was to strip a desired data type from the tape and produce files containing only one data type. The Viking STE Post-Test System had the capability to strip data of a specific type from the OLT and place it in a disk file. This satisfied the first requirement. After the data were in a file on the disk a systems utility (DISKS) was used to write the data on to magnetic tape. This satisfied the second requirement.

In order to process the magnetic tape written by DISKS some software had to be developed for the large scale CDC 6500 computer. This software had to be programmed to read the tape, recognize the data type (e.g. seismometry, X-ray, etc.), and construct an input file that the science analysis program could process.

The file formatting software that was generated was a FORTRAN program that incorporated available CDC system supplied library routines to perform the bit manipulation. It worked out well and satisfied the requirement.

QUALITATIVE RESULTS: The primary benefits derived from this technique were as follows:

1. The verification of science instrument performance in the STE environment was accomplished.
2. Realistic data were provided for the development of the science analysis programs. Without using this technique, the Lander science team would have had to hand generate test data, greatly increasing the chances for error.
3. Programming time and effort were saved in trying to find a method to process science data in the STE.
4. Coordination was established between the ground testing of the flight articles and the Viking Flight Team science members.

The serendipitous fallouts realized by using the technique were far more interesting and proved to be of value to the science team.

The first of these fallouts was the discovery that some of the formats for science program files were not as documented (e.g. time words, ID bits, data start locations, block sizes, etc.).

Next it was discovered that the algorithms developed for the seismometry analysis program were wrong. This was verified by the scientists who had developed the algorithms. The impact of not discovering this until integration would have been much more costly to correct.

Finally, when the uplink/downlink Viking Lander science sequence tests were conducted, the science cognizant engineers were able to process the downlink telemetry in Denver. Then when the same telemetry data passed through the Mission Operational Software System and was processed by the science analysis programs at JPL, they were able to make direct comparisons of printouts to verify that the system was operating correctly.

QUANTITATIVE IMPACT: Because the technique was developed in a clandestine manner, it cost very little to implement. No documentation or formal testing were required. Implementation of the technique required four man months and three hours of CDC computer time. These costs are very economical, considering that the alternatives were to write a new FORTRAN compiler or write all of the science analysis programs in assembly language.

When the sequence tests were first run at JPL the science engineers were able to immediately detect that the answers were wrong, which reduced the effort required to locate the error sources.

Finally, Viking experience was that programs that were tested with hand-generated data typically failed when realistic data became available. This phenomena did not occur with the science analysis programs. The net savings to the project by using this technique can therefore be estimated at 5 to 10 man months.

TECHNIQUE

NAME: VIKING TEST SEQUENCE GENERATION

SUMMARY: Viking Test Sequences written by test engineers were prepared using two basic methods. They could be written on Viking Test Language (VTL) coding forms, key punched on 80 column cards, and then input into the STE pre-test file management system by a card reader. The second method was to generate them by use of the MARTIN-DIGITAT computer system and TOPS programs, which consisted of a CRT/Keyboard driven on-line batch computer system that provided for file creation, editing and file dump to tape.

APPLICATION CONSIDERATIONS: During later stages of the Viking program, the second method of test sequence generation was used almost exclusively. It was found that the keyboard/CRT editing capability provided for a much greater flexibility for update, modification, and merge of sequence elements. Sequence generation and modification time could be reduced from the keypunched card method on the order of eight to one. In addition, the CRT/Keyboard driven computer system contained sub-programs that were aids to sequence generation. Foremost of these were a diagnostic program that permitted a quick check of sequence element noun/verb/adjective compatibility, sequence timing and test block numbering. The program provided diagnostic messages to the sequence designer for correction iterations. Another sub-routine provided for test block renumbering. The CRT/Keyboard driven capability was preferred by sequence designers from a human engineering point of view.

RECOMMENDATION: It is recommended that a computer file generation and editing system be provided for generating application programs (test sequences) for any system comparable in complexity to Viking.

HISTORY: The use of the computer file editing system for sequence preparation evolved naturally from other usages. These usages were system statusing and special purpose programs which could produce listings of Direct Communication System Uplink Commands and Surface Sampler Commands and responses.

This technique was used in support of the Viking Test Language technique described elsewhere in this report. The detailed description of the test sequence language is presented there.

DESCRIPTION: The on-line SIGMA 5 file generation, editing and management system is an established file management system available at MMC. Its application to the Viking sequence or application program generation task provided the following capabilities:

1. File Creation
2. Write Line Commands
3. Read Line Commands
4. Move and Delete Line Commands
5. Move and Keep Line Commands
6. Find Text Commands
7. Replace Text Commands
8. Merge Lines from one Sequence to Another
9. Copy a Sequence to a New File Number (Identification)
10. Test Block Renumbering
11. Quick Check Capability
12. Copy to Line Printer
13. Copy to Tape

QUALITATIVE RESULTS: The benefits derived from use of the computer file generation and editing technique were significant. Sequence design could be accomplished eight times faster than by using the card generation technique. Updates and modifications could be made quickly. Finally, it saved translation time by providing a diagnostic capability for detecting errors.

QUANTITATIVE IMPACT: Approximately 95 percent of the 974144 sequence lines, plus 90 percent of the modifications, were generated using the MMC Sigma 5 computer system.

TECHNIQUE

NAME: SOFTWARE CHANGE REQUEST/IMPACT SUMMARY

SUMMARY: A Software Change Request (SCR) was prepared and processed to secure authorization to the method of implementing changes to released software. The SCR also provided for control, coordination, and scheduling of the proposed software changes into the Viking Change Summary/Viking Integration Change (VCS/VIC) system. An SCR impact summary was used to collect pertinent impact information to support evaluation of proposed changes.

APPLICATION CONSIDERATIONS: Software change request procedures were well established at MMC and JPL prior to the Viking Project. Only minor modifications were needed to adapt them to Viking's needs. The flight computer was resource limited relative to the amount of potential software involved. Impact summaries were defined to estimate memory sizing and computational timing impacts that would result from software changes.

RECOMMENDATION: It is well known that hardware wears out with time. It is not too widely known that software wears out with change. Given sufficient change traffic a software system will eventually become inefficient and error prone. Viking Lander and Orbiter software adapted from existing programs was in general less efficient than Viking software developed from scratch. Any software change process should attempt to protect software efficiency and flexibility when authorizing new requirements to be implemented. A useful technique for accomplishing this is to review and approve the method of implementation separate from the requirements request.

HISTORY: The go-ahead to implement requirements changes to MMC developed hardware or software was authorized by the approval of a Viking Change Summary (VCS). In the event that the change affected the Viking Project Office, Viking Orbiter, Viking Mission Control and Computing Center, Tracking Data System and/or the Deep Space Network, a Viking Integration Change (VIC) also had to be approved by all affected parties.

The VCS/VIC forms were color coded in green, blue, pink and white. The green form was used to coordinate the change, the blue to obtain Project approval for out-of-scope changes, the pink to allocate additional costs, and the white to show final approval. A Project Control Board (PCB) was established to control and monitor all VCS/VIC traffic, assuring that all potentially impacted parties were aware of the proposed requirements changes.

Software Change Request (SCR) forms were used to respond to the VCS/VIC system in the same sense that the design process responds to the requirements process. Impact summaries accompanied the SCRs to assess delta changes to core memory, drum, and disk space, computational timing, schedules and manpower.

DESCRIPTION: The method of responding to change requests originating through the VCS/VIC system differed among the Systems Engineering Directorate, the Mission Operations and Design Directorate, and Mission Operations at JPL. However, each followed the same basic philosophy that the method of implementation should be reviewed and approved separately from the requirements request.

The Systems Engineering Directorate documented the change procedure for Flight and STE software in the Viking Lander Software Plan. It included both a SCR form (Figure 1) and an Impact Summary form (Figure 2). The primary reason for including the Impact Summary form was to maintain visibility and control over the growth of Flight software.

Any member of the Viking Flight Team could originate a Flight or STE hardware/software requirements change by preparing a VCS and taking it to the Project Control Board. Typically, software changes were needed to support hardware changes. If the PCB felt that the change

PAGE OF	SOFTWARE CHANGE REQUEST	SCR/DCN NUMBER
TITLE OF CHANGE		ISSUE DATE

CHANGE SOURCE	REFERENCE	CATEGORY

CHANGE IS ☐ RECOMMENDED ☐ MANDATORY	ORIGINATOR	SECT/UNIT	PHONE

SOFTWARE SYSTEM AFFECTED

Yes	No		Yes	No		Yes	No		Yes	No		Yes	No	
☐	☐	GCSC	☐	☐	STE	☐	☐	FLT OPS	☐	☐	DSN	☐	☐	OCIS

SOFTWARE PROGRAM(S) AFFECTED: (Name and Identification Number)

DESCRIPTION OF CHANGE

REASON FOR CHANGE:

ANTICIPATED SCHEDULE	DATE
REQUEST TO INITIAL SCB REVIEW	
REQUEST TO FINAL SCB REVIEW	
GCSC/IOC INPUT	
COD	
CHANGE NEEDED	
CHANGE COMPLETE	

ORIGINATING DEPT APPROVAL:

SOFTWARE CHANGE COORDINATOR APPROVAL:

SOFTWARE CONTROL BOARD APPROVAL

Figure 1

PAGE OF	SOFTWARE CHANGE REQUEST - IMPACT SUMMARY	SCR/UCN NUMBER
TITLE:		DATE:

PREPARED BY:	SECT/UNIT	PHONE	DATE	REVIEWED BY:	SECT/UNIT	PHONE	DATE

DOCUMENTS AFFECTED: (Identification Number and Name)

MEMORY SIZING IMPACTS: (Core Memory, Drum, Disc)

PROGRAM						
SUBSECTION						
△ WORDS						

COMPUTATIONAL TIMING IMPACTS:

PROGRAM						
TIMING AREA						
△ TIME						

REWORK IMPACTS: (Schedule, Man months, Hardware, Procedures, etc.)

SOFTWARE CONTROL BOARD RECOMMENDATIONS:

Figure 2

request merited consideration, the VCS was turned green and sent to the appropriate hardware and software groups for further assessment. The software group would then prepare a SCR to describe how the change could be implemented and an Impact Summary showing estimated delta impacts to computer resources. The change package was then taken to the Software Change Board (SCB) for review and approval/disapproval. The SCB had been created by the software plan expressly for this purpose. It was particularly reluctant to approve changes that would significantly reduce Flight computer memory or computational timing margins. When such cases arose, the Systems Engineering director or his appointee would attend the SCB review and make the final decision.

Following the SCB action, the VCS change package was reboarded with the PCB for close out action. If the SCB had disapproved the change, the VCS was cancelled. If the SCB had approved an in-scope no cost change, the VCS was turned white permitting the change to be implemented. If the SCB had approved an out-of-scope or cost impact change the PCB forwarded the change package to the Viking Project Office for approval/disapproval.

The MO and D directorate followed the change procedure specified in the Flight Operations Software Plan, which included an SCR from similar to that shown in Figure 1, but did not include an impact summary form. Because MO and D was represented on the multi-agency Flight Operations Software Subworking Group, the PCB required that MO and D coordinate all VCS/VIC traffic within Flight Operations before boarding changes. Therefore, MO and D appointed two change board representatives (CBR) to the PCB to review each VIC/VCS for possible impacts originating outside of MO and D. VIC/VCSs which were determined to have a possible impact upon MO and D were delivered to appropriate technical leads within MO and D for further action. Technical leads were appointed for software changes, simulation system changes, science instrument changes, test and training changes, DSN, VMCCC and VL changes, Mission design changes, Flight path analysis changes, Flight hardware/software changes, and Viking Integration Changes.

The technical leads coordinated the changes with MO and D personnel, obtained SCRs, and prepared functional task descriptions (FTDs) which summarized cost and schedule impacts for the MO and D director to approve/disapprove. All significant c' ges were discussed at MO and D Technical Staff Meetings, which were held weekly. The CDRs then reported the MO and D director's decision to the PCB.

Changes originating within MO and D were coordinated by MO and D with all affected agencies and approved by the MO and D director prior to boarding a VCS and VIC before the PCB.

When all Viking Flight Team members had permanently located at JPL, the process for changing Mission Operations Software was modified. The originator of a requi.ements change filled out a VIC/VCS and took it to the Integrating Contractor Software Systems Engineer (ICSSE) for coordination. The ICSSE distributed the VIC/VCS to the appropriate SSEs for further action. The SSEs obtained SCRs from the appropriate VFT technical personnel and returned the change package to the ICSSE, who then scheduled a meeting between the Mission Directors and the technical personnel involved. At that meeting the Mission Directors either approved or disapproved the change, after which the VCS/VIC was turned white or cancelled. Since VPO was represented by the Mission Directors, the blue and pink portions of the VIC/VCS process were no longer required.

QUALITATIVE RESULTS: The creation of the Systems Engineering Software Control Board to monitor and control Flight Computer memory sizing and computational timing margins was instrumental in preventing the Flight computer from becoming overloaded. The SCB could not have accomplished this task without the visibility available through the Software Change Request and Impact Summary forms. There were cases in which multiple SCR/Impact Summaries were submitted to the SCB to judge the impact of full and partial compliance to the requirements change request.

By contrast, the process used by Flight Operations, whether at MMC or at JPL, rarely allowed management to understand the impact changes would have on computer loading or program performance. Since

the programmers were aware of the constraints on program size and run times, they could merely create a new load module to perform a new task, thereby hiding the impact to the system. Flight Operations managers attempted to control the growth of the system by refusing to authorize any changes other than "make play". Therefore, the only changes that came before them were marked "make play", after which they were invariably approved. Fortunately, many of the changes were designed to improve program performance and reduce computer loading requirements.

That is not to say the Flight Operations change process did not work. In fact, it did. The overriding reason for this was that the as built software system was extremely large compared to the change traffic that impacted it. If you add a cup of water to a half filled quart bottle, you will notice the change immediately. If you add a gallon of water to a swimming pool, you won't see the change. In that sense it is unlikely that impact summaries would have served a useful purpose to Flight Operations management.

QUANTITATIVE IMPACT: Typically, a Software Change Request and an Impact Summary could be filled out in an hour or two. Some of the larger changes may have taken from a day to a week to work out. However, this should be viewed as a zero cost impact, since Viking did not have to add any unplanned personnel to handle the task. Furthermore, most of the work would have had to be done eventually since the vast majority of changes were necessary and were approved.

The time consumed in processing a change fluctuated considerably. No accurate manpower estimate can be made for the average change. Most VIC/VCS traffic was on the books for one to two months before being cancelled or turning white. Some went through the system in a day. One VIC, which attempted to standardize Viking Orbiter and Lander file headers, took a year to finally resolve (and then no single standard was reached). Keeping the above in mind, the manpower required to coordinate a VIC/VCS, respond with an SCR/Impact Summary, review and approve the response, and close out the VIC/VCS is estimated at one to two weeks.

TECHNIQUE

NAME: VIKING SOFTWARE STANDARDS

SUMMARY: Documentation and flow chart standards were specified for all MMC Viking software very early in the life of the project. No further standards were imposed on Flight or STE software other than those adopted by the software groups themselves. Mission Operations issued a Viking Software Guide that listed standards, procedures, guidelines and constraints to be followed. Responsibility for adhering to the guide rested in most cases with the individual programmers.

APPLICATION CONSIDERATIONS: Management considered that controlled, uniform documentation was the key element needed to establish visibility and understanding of the development process. American National Standard flowchart symbols were adopted to provide project wide consistency. The multi-agency development of the operational software system required nomenclature, naming, labeling and coding standards be adopted to coordinate the effort. In addition, the project was required to adhere to computer usage guidelines and constraints that had been established at JPL.

RECOMMENDATION: Standards adopted at a management visibility level can be effective and are enforceable. Standards set below that level are of little value. On Viking, programmers tended to ignore guidelines and non-enforceable standards. Documentation standards make reviews easier to accomplish and lead to greater thoroughness. Labeling and naming standards are a convenient tool to avoid confusion. Coding standards are of dubious value and can have negative effects when computer run time and program size are required to be significantly constrained. This remark should not be interpreted relative to Structured Programming standards, which were not employed on Viking.

HISTORY: Early in 1971 MMC formed a Viking Software Integration Group (VSIG) for the purpose of monitoring the development of Flight, STE and Mission Operations software. The first task assigned to the VSIG was to define and document a uniform set of standards. In October of that year the group issued "Standards for Viking Software Development" which set documentation, flow chart, identification and handling standards for all MMC developed software.

Shortly thereafter a Mission Operations and Design Directorate was formed separate from the Systems Engineering Directorate, at which point the VSIG was abolished. In May 1972 the "Flight Operations Software Plan" was issued that set standards for Mission Operations software. The plan incorporated the earlier documentation standards, added the Software Functional Description document to the list, imposed labeling standards, and specified the requirement that a Viking Software Guide be issued to establish standards, procedures, guidelines and constraints.

A Lander Software Integration Group (LSIG) was formed in the Systems Engineering directorate. The LSIG issued a Viking Lander Software Plan in September 1972 that incorporated only the original documentation standards. No other standards, guidelines or constraints were imposed on Flight or STE software by the LSIG, which slowly was depleted by the attrition and transfer of its members. It would have been too late by then to impose further standards on the STE, since the system was about to go on-line.

In January 1973 the Viking Software Guide was issued to incorporate administrative, integrated, lander, orbiter, and VMCCC memos. The document was distributed to every programmer and engineer responsible for an operational software program.

DESCRIPTION: Standards for Viking Software Development specified the documentation responsibilities shown in Table 1.

	FO	STE	G&C		
			V	L	E
Functional Requirements Document	P				
Software Requirements Document	P	P	P	I	I
General Design Document	P	P	P	I	
Software Data Base Document	P				
Program Description Document	P	P	P	I	
Source Listing	I	I	I	I	I
Users Guide	P	P	P	I	I
Users Acceptance Test Plan	P				
DSN/Project Compatibility Test Plan	P				
Development Test Plan		I	I		
Validation Plan		P	P		
Programming Handbook	I	I	I	I	I
Software Design Handbook	I	I	I	I	I

Key:

V: Vehicle programs written in GCSC assembly language

L: Lab support IC 7000 programs written in IOP assembly language

E: Micro-language programming of the IC 7000 CPU and IOP

P - Project Controlled Document I - Informal

Table 1

The document specified that it would serve as the central requirement collection point for software standards until the requirements were incorporated into documents that cover procedures. Briefly, the documents were defined as follows:

1. The Functional Requirements Document functionally describes a program or system of programs.
2. The Software Requirements Document specifies the software requirements corresponding to an FRD.
3. The General Design Document responds to the SRD and outlines how the program or systems of programs will be developed.

4. The Software Data Base Document formally documents the detailed data base for all programs.
5. The Program Description Document describes the final program.
6. The User's Guide describes how the program can be used.
7. The User's Acceptance Test Plan describes the procedures that will be used for module testing on JPL computers of ground based Lander Flight Operations programs.
8. The DSN/Project Compatibility Test Plan describes the method of verifying the integration of ground based Lander and Orbiter Flight Operations software in the internal JPL Computer Systems.
9. The Development Test Plan describes how testing will be conducted at each stage of program development.
10. The Validation plan describes the method followed to validate to customer satisfaction and MMC/QC that MMC developed software/hardware systems meet all specified requirements.
11. The Programming Handbook describes the programming conventions imposed by the specific computer system or systems for which software is to be developed.
12. The Software Design Handbook records do's and don'ts for good programming practices discovered during program development.

The organization of each of the first six documents listed above was described on a paragraph by paragraph basis. The basic outline for these documents was required to be followed. Subsectioning was permitted to the degree that it was consistent with the needs for documenting the information. A single outline was shown for all test documents. The purpose of showing the outlines was to provide uniformity and completeness of information content. A consistent outline was generated for both requirements and design documents. The SRD outline was as follows:

1. Introduction
   - 1.1 Scope
   - 1.2 Problem Statement
2. Applicable Documents

3. Requirements
    3.1 System
        3.1.1 Processing
        3.1.2 Input/Output
        3.1.3 Interfaces
        3.1.4 Data Base
        3.1.5 Limitations and Constraints
        3.1.6 Diagnostics
    3.2 Subprograms
        3.2.N Function "N"
            3.2.N.1 Processing
            3.2.N.2 Input/Output
            3.2.N.3 Interfaces
4. Hardware Environment
    4.1 External Interfaces
    4.2 Hardware
    4.3 Man/Machine Interfaces
5. Verification
6. Miscellaneous
10. Appendix

Sections 3.1.4, 3.1.5, 3.1.6 and 10 were not applicable to the FRD. Section 3 was titled "Description" for design documents, which also included sections 3.1.7, Storage Allocation, and 3.1.8, Flow Charts. In addition, each subprogram described under section 3.2 of the PDD included the eight subsections shown under section 3.1.

A paragraph was included to describe the information required in each subsection. The documentation standards stated that a cognizant programmer would support the cognizant engineering in developing the SRD, primarily because the document would significantly influence program design.

The Programming Handbooks and Software Design Handbooks never materialized, although the intent of the former was adopted by Mission Operations in the Viking Software Guide.

Flow chart standards specified in the document were based on those published by the American National Standards Institute. The coding guidelines listed were relatively brief and generally ignored. The software handling standards merely addressed maintaining duplication in master file storage, the intent of which was followed by the Viking Program.

The Flight Operations Software Plan added the Software Functional Description document and specified its organization (refer to the Integrated Software Functional Design technique). It also specified the numbering system that would be used to identify and control all MMC and JPL documents.

The Viking Software Guide contained standards for Viking software symbology notation, use of non-minimal language, Viking Lander terminology, Viking Orbiter terminology and Viking acronyms. The symbology notation standardized the first two characters of every program name, subroutine and data table name that would be delivered to the Mission Control and Computing Facility (i.e. - IBM 360/75 computer system) for incorporation on the Mission Build. The non-minimal language standard required programs that operated on two or more different computer systems use comment cards to show the exact coding differences required by each system.

Guidelines included descriptions of minimal languages, language comparison charts and programming style. Procedures addressed program conversion, documentation production, test activities and program delivery.

File naming standards were incorporated in the Software Data Base Document. Every permanent file used by a Viking program was assigned a unique five alpha character designator. MMC expanded this to a twelve character string that included spacecraft, mission, date and version designators. No file naming standard was ever adopted for Viking Orbiter programs.

The Mission Control and Computing Center (MCCC) documented guidelines and constraint standards required to be followed by all projects

using their facilities. These standards imposed limits on core allocation, contiguous core allocation, CPU time, amount of print/plot output, number of tape drives, large-capacity storage, direct access storage, and sizes of program card decks.

QUALITATIVE RESULTS: The documentation standards were basically a sound idea. They simplified the task of the author to organize and present material. They undoubtedly led to more complete documentation than otherwise would have been realized. The process of reviewing the documents was also simplified.

There were differences of opinion over the value of the FRD. The Mission Planning and Flight Path Analysis groups liked the idea, carried it out, and felt it was a worthwhile exercise. The remaining groups considered it an unnecessary additional step and developed their SRDs directly. The principal difference between the two approaches was that one provided for an intermediate requirements review and the other did not.

The level of detail indicated for the SRD was needed by Viking. The principal disadvantage was that the organization of the requirements implied some design criteria. This had a negative effect when cognizant programmers were not available to support the document generation.

The use of external file naming convention standards made it very easy to checkpoint and recall data recorded by a particular instrument on a particular Martian Sol (day). The standard also permitted an efficient automated file management system to be developed (refer to technique on-line data file management system).

MMC management did not place proper emphasis on the MCCF symbology notation standard and the lander programmers did not understand it. This could have had a very serious cost and schedule impact on lander software had it not been for JPL's willingness to help resolve the problem. The standard required that all subroutine names begin with the characters LM and all data table names begin with the characters LZ. The reason for the standard was to prevent MMC developed software from conflicting in name with any other software on

the Mission Build. By not following the standard, MMC was faced with the task of changing literally thousands of call statements to subroutines. However, JPL came to their rescue by creating a private Viking subroutine library (VIKILIB), which made it possible to waive the standard. This example stresses the importance for a developing agency to know and understand any enforceable standards set by a user agency.

The use of flow chart standards simplifies the design review process. Consistency is more important than the particular selection of symbols. The Viking Lander Flight software group developed their own standards in this area, rather than adopting the ANSI standards used by the rest of the project.

The setting of a standard for the use of non-minimal language effectively reduced the minimal language standard to a guideline. Although it was not enforceable (no standard set below management's visibility level is enforceable) it was followed because it eased the conversion task of the programmer.

QUANTITATIVE IMPACT: The documentation standards cost four to five man months to develop. The Viking Software Guide cost an additional four to five man months. The ANSI flow chart and MCCC guidelines and constraints standards were available at no additional project cost.

The automated on-line data file management system and the inter-computer transfer function were developed at less cost because file naming conventions were standardized. The estimated savings in these areas is one man year. Even then each individual file would have required some form of standardization.

TECHNIQUE

NAME: FLIGHT OPERATIONS SOFTWARE PLAN

SUMMARY: The Flight Operations Software Plan was the controlling document for Viking operational software development. It established management roles and responsibilities, the software design and development process, and the methods by which management would control and implement the software system.

APPLICATION CONSIDERATIONS: Development of an integrated software system requires that compatible standards, procedures and processes be established to minimize interface problems, ambiguous terminology and communcations problems. This was particularly essential to Viking, since several agencies and contractors were responsible for various portions of the software system. A unified plan agreed upon by all parties that clearly and consistently outlined the method by which the software system would be developed and implemented was considered mandatory by the Viking Project. No consideration was given to permitting each operational software developer to independently manage their development processes.

RECOMMENDATION: The first step taken in developing a software system should be to write a software plan. The plan should define management roles and responsibilities, specify documentation requirements, establish milestones by which progress can be measured, define configuration management control, and describe the development process to be followed from initial design through system integration. Once a plan has been formalized and agreed upon, management should take appropriate steps to assure that it will be followed. If this is done, schedules and costs can effectively be controlled.

HISTORY: The Viking 75 Project Flight Operations Plan was prepared under Contract Number NAS1-9000 by the Martin Marietta Corporation, Denver Division. The Integration Contractor Software System Engineer (ICSSE) was responsible for its generation and publication.

The plan was written during the latter part of 1971 and early 1972. It was concurred upon by Flight Operations managers at the Viking Project Office, the Jet Propulsion Laboratory and Martin Marietta Corporation. On 15 May 1972 it was approved by the Viking Project Manager, at which time it became the controlling document for the development of the Viking Operational Software System.

Eight minor revisions were made to the plan using the Viking Integration Change Control system. The final revision was incorporated on 19 June 1974.

Copies of the plan were distributed to all Flight Team members, programmers and engineers.

DESCRIPTION: The plan consisted of five sections and three appendicies, organized as follows:

1.0 Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 General
- 1.4 Acronyms and Abbreviations
- 1.5 Definitions of Terms

2.0 Applicable Documents
- 2.1 General
- 2.2 Reference Documents

3.0 Management of the Flight Operations Software System
- 3.1 Introduction
- 3.2 Software Sub-Group
- 3.3 Integration Contractor
- 3.4 Design Responsibility

4.0 Software Design and Development Process

- 4.1 Introduction
- 4.2 Software Functional Description and Integrated Software Functional Design
- 4.3 Functional Requirements Document
- 4.4 Software Requirements Document
- 4.5 Software Data Base Development and Definition
- 4.6 General Design Document, Schedule and Work Plan Development
- 4.7 Program Development, Testing and Release
- 4.8 Test Plan Development
- 4.9 Testing
- 4.10 Project Software Delivery
- 4.11 Software Maintenance and Support

5.0 Management Control Method

- 5.1 General
- 5.2 Milestones
- 5.3 Documentation
- 5.4 Requirements Definition
- 5.5 Design Monitoring
- 5.6 Review
- 5.7 Approval of Detail Design
- 5.8 Software Control Board
- 5.9 Software Handling and Labeling
- 5.10 Computer Program End Product
- 5.11 Software Development Progress Monitoring
- 5.12 Programming Guidelines and Conventions
- 5.13 FOS S/W Interfaces with On-Board S/W

Appendicies

- A Documentation
- B Software Change Control - Flight Operations
- C. Program Labeling

The plan established roles and responsibilities for the ICSSE, MMC software System Engineer (VLSSE), VO Software System Engineer (VOSSE), Mission Computer Control Center Data System Project Engineer

(DSPE), Cognizant Engineers (CE), Cognizant Programmers (CP), and Flight Team members who were users of a particular computer program. The primary responsibilities of the SSEs were to plan, coordinate and monitor the development of the Operational Software System. Cognizant Engineers were made responsible for program requirements and testing, and their associated documentation. Cognizant Programmers were made responsible for program design and development and their associated documentation. Flight Team members were made responsible to support and review development of program and test requirements.

Documentation requirements were specified relative to organization, responsibility, review, concurrence, approval and change control. This included program documents (functional, requirements, description, test and user), the Integrated Software Functional Design (ISFD), the Software Data Base Document (SDBD), and the Lander/Orbiter Software Test Plan.

The principal management control methods employed were: establishment of milestones to support schedules; preparation of documentation, monitoring of design response to requirements; formal reviews of documentation, software end products, and test results; approval of detailed design; change management; software handling and labeling; release of software end products; software development progress monitoring; and establishment of programming guidelines and conventions.

The software development progress monitoring required the SSEs to maintain and publish detailed schedules and to hold weekly telecons to discuss progress and resolve problems. The CEs were required to maintain Schedule and Work Plans by which the SSEs could measure progress. Each SSE was to establish a Software Design Team (SDT) composed of CPs and CEs to review schedules, identify interfaces and review design concepts. The CPs were required to review test results with the SSEs at critical points during program development. Computer usage reports were required to be sent weekly to the SSEs. The SDT was required to review General Design Documents, working flow charts, SDBD inputs, test reports, Program Description Documents and User Guides.

The SSEs were required to review program listings on an intermittent basis to verify conformance to guidelines and conventions, completeness of the program, and, in critical areas, the coding logic. Finally, the SSEs were required to monitor corrective plans and actions for program errors uncovered after program delivery.

QUALITATIVE RESULTS: The strongest feature of the plan was that it specified the means by which it could be made to work. That was by creating roles and responsibilities for SSEs which made them responsible for the development process. The overall plan was successful because it led to an orderly development of the software system.

The principal reasons that programs had to be modified and redelivered were due to new requirements, poor designs, program to program interface errors, and errors detected due to the lack of good test data. How the plan treated each of these subjects, and how it might have been improved to lessen the impact of these redeliveries will be discussed in the following paragraphs.

Requirements were written by a cognizant engineer (CE) and documented in a Software Requirements Document (SRD). The CE was supported by a cognizant programmer (CP) concerning programming techniques, feasibility and I/O formats, and by an SSE concerning interface requirements. The SRD was reviewed by appropriate Flight Team Members, technical personnel, the SSE, the Data System Project Engineer, and by a VPO technical monitor. A formal review was then conducted by the SSE to assure communication and resolution of all comments, after which each of the reviewers concurred with, and the responsible director approved, the SRD. The definition of concurrence was explicitly stated relative to each of the reviewers, covering such items as satisfaction of functional requirements, consistency with system design, technical correctness, and conceptual correctness. The plan should have also required the reviewers to estimate SRD completeness in terms of potential new requirements. This would have had the effect of forewarning those programmers who were developing operational software in parallel and dependent on Flight hardware or software that significant change traffic was potential.

As such, during the design phase, those programmers would have tended to model requirements as data rather than as part of the program structure, thereby giving up (in some cases) efficiency for flexibility.

Program designs were developed by a CP and documented in a General Design Document which was reviewed at a meeting attended by the CE, CP, SSE and technical personnel. The deficiency here is that the plan did not take into account the CPs experience, the fact that the CE and technical personnel usually will not understand the ramifications of the design relative to the computer system, nor the fact that the SSE had too many responsibilities to be able to give adequate thought to design ramifications. The plan should have specified a criteria for the selection of CPs, which it did not. It also might have required that one or two outside programmers attend the review to question the CP as to how the design was to be implemented, and comment on what they thought of the approach.

The plan identified the SDBD to document all interfaces and required that each interface be tested by Software Integration. The wording here was perfectly adequate. However, the milestones section of the plan should have required interface testing to be conducted as an extension of Users Acceptance Testing, which would have been prior to placing the software under strict change control. This would have forced a top-down approach to integration, wherein programs would be delivered in subsystems, rather than by the "as available/as needed" approach actually used by integration. This would not have reduced the number of errors detected, but would have significantly reduced their impact on redeliveries.

Finally, the plan did not address the subject of test data, other than to mention a CE or a CP was responsible for its generation. The plan should have specified requirements for the development, review, documentation and approval of test data, stressing the completeness of the data to assure thorough program testing.

QUANTITATIVE IMPACT: The software plan was negotiated and written at a cost of six man months. Beyond that it is difficult to describe it in quantitative terms, other than to state that it contained 99 pages. The true impact of the plan was that it established the basis for successfully developing a million plus source card software system on schedule. The impact of techniques specified or implied by the plan are described separately in this report. They include HOL utilization, different development sites, the Integrated Software Functional Design, Cognizatn Engineer/Cognizant Programmer, SCR/Impact summary, software standards, Management Visibility, Flight Operations Software Subgroup, and the Software Data Base Document.

TECHNIQUE

NAME: SOFTWARE DEVELOPMENT MANAGEMENT VISIBILITY

SUMMARY: Progress was monitored by maintaining five levels of schedules based on a series of significant milestones. Program and system level requirements, design, interface and test documents were reviewed and approved. Software change traffic was closely monitored, widely reviewed and well documented. Weekly meetings were held to air problems. Open items lists were maintained. Software Systems Engineers were established to monitor the implementation of the software design, development and testing and to assure that all interfaces, requirements and schedules were correctly and completely satisfied.

APPLICATION CONSIDERATIONS: Software systems are frequently delivered late, not documented accurately, contain unidentified risks, overrun costs, are unreliable, and fail to meet mission objectives. This was of particular concern to the Viking Project, since the launch windows were narrow and planetary operations had to be completed during a four month period between Mars Orbit Insertion and the conjunction of Mars with the sun. Therefore, not only was a well defined software development cycle established, but a great deal of emphasis was placed in providing management with sufficient visibility to assure that the plan was carried out.

RECOMMENDATION: The need for management visibility into any development process is obviously necessary if costs and schedules are to be controlled. The use of documentation, milestones, reviews, presentations, meetings and change control are necessary but not sufficient to assure a reliable system that meets the mission objectives will be delivered on schedule. Root level program schedules and software systems engineers, used effectively, can significantly enhance management visibility.

HISTORY: The degree and type of visibility management had into the Mission Operations software development process varied as a function of the development phase.

During the software system definition phase visibility was very good. The software plan and the integrated software functional description were carefully reviewed at the director and Project manager levels. Milestones were well defined, and detailed schedules were developed. The quantity of software to be developed and the results of computer loading studies gave top management visibility to allocate resources on a realistic basis.

This was followed by the requirements phase, wherein management had the least visibility at any point in the development cycle. It was restricted to monitoring schedules and reviewing each Software Requirements Document (SRD) prior to its release.

Visibility improved during the design and code phases. The SRDs were under rigid change control so that impacts caused by requirements changes were reflected in weekly updates to schedules. Reallocations in personnel assignments were made to prevent serious schedule slippages. Management could also monitor the development of the Software Data Base Document, which defined interfaces and the common data base, and the Lander/Orbiter Software Test Plan, which showed the system integration process and indicated the resources that would be required to carry it out.

The certification, conversion and user acceptance test phases permitted the lander, orbiter and integrating contractor software systems engineers to assess which requirements and constraints had been met in test and which had not. The latter were placed on waiver lists, the resolution of which could be monitored by all levels of management.

The software was now placed under rigid change control, and errors detected during the unit verification and system integration phases were made highly visible by means of a very efficient failure report system. Corrections to these errors could be controlled at the director level and implemented through an Integration Change Control Board.

During the spacecraft compatibility test phase, the failure report system was changed to a Viking Incident Surprise Anomaly (VISA) system, which gave management the added visibility to assess the projected impact that software errors and requirements changes would have on specific operational phases.

DESCRIPTION: Sixteen milestones were identified to provide management visibility into the development process of each program. In addition, the cognizant engineers for each program were required to keep weekly assessments of percentage completion during the design, code, debug and programmer test periods. If the percentage completion estimate was not compatible with the schedule for an ensuing milestone, the cognizant engineer was required to change the projected completion date shown for that milestone to an earlier or a later date, as appropriate. Three columns were maintained for the schedules of each milestone; planned, projected and actual. Therefore, when a projected schedule date was changed, management could assess the impact it would have relative to the entire software system development process. Manpower and resources were reallocated, as required, to maintain a consistent overall schedule.

The impact of software requirements changes was handled by scheduling phased deliveries for programs. Separate schedules were maintained for each phase of program delivery.

The milestones used to monitor Viking program software development progress were:

1. Release of Software Functional Description
2. Functional Requirements Document sign-off complete
3. Software Requirements Document available for review
4. Software Requirements Document sign-off complete
5. General Design Document sign-off complete
6. Program design and initial module code complete
7. Module testing complete
8. Certification Test/Users Acceptance Test Plan available for review

9. Certification Test/Users Acceptance Test Plan sign-off complete
10. Program development complete
11. Certification test complete
12. Acceptance test complete
13. Program delivery to VMCCC for integration
14. Unit Verification test by VMCCC complete
15. Program integration tests complete
16. Project software delivery

The program delivery milestone required delivery of all program documentation, including the Program Description Document and User's Guide for which no earlier milestones existed. Therefore, projected and actual schedules were maintained for each deliverable item for this milestone.

Weekly meetings were held between upper management, team leaders, and the software systems engineers. Progress, problems and conflicts were aired at these meetings. Open item lists were issued and monitored.

A Viking Integration Change/Viking Change Summary/Software Change Request (VIC/VCS/SCR) system was established to monitor all changes to software requirements. A VIC impacted more than one agency and required concurrence of all parties before it could be approved. The VCS was used for Lander hardware or software changes. The SCR was used for lander or orbiter software changes implemented at JPL. A Project Change Board met weekly to discuss the status of all open change traffic. Representatives from each subsystem were required to attend these meetings to assure that any impact to their subsystem would be recognized and taken into proper account.

Lectures and presentations were held at frequent intervals wherein all programmers and engineers were briefed on management concerns, the development cycle, and items pertinent to the software system and its status.

Preliminary and Critical design reviews were held on the integrated software functional description of the software system.

The integrating contractor software systems engineer was required to review and approve the results of all users acceptance tests and provide upper management with written reports of those reviews.

The Software Subworking Group, which was composed of the integration, lander, orbiter and institutional software systems engineers, was required to issue monthly reports to upper management stating work accomplished, problems encountered that required resolution, and work planned for the following month.

Changes to the common data base required formal written approval by upper management before they could be implemented.

An Integration Change Control Board met weekly to discuss the status of all programs scheduled to be delivered or redelivered to the software system. The board was chaired by a Project Manager and attended by the software systems engineers and representatives of software programs to be discussed. The need for and impact of changes or waivers were aired at these meetings. The ICCB assured that software deliveries complied with established procedures.

Several documents played a key role in providing management visibility throughout the development stage. The Software Functional Descriptions and Integrated Software Functional Design permitted management to determine which programs were required, how they would interface, what utility programs would be needed, conduct computer program assignments and loading studies, hold a system critical design review, and determine integration requirements. The Lander/Orbiter Software Test Plan proved to be an extremely useful tool to the project. It allowed management to schedule manpower and resources, and assured them that inter program communications would be thoroughly tested. The Software Data Base Document gave management a single source by which they could be assured that all program interfaces and the common data base would be visible and controlled. This provided the visibility by which management could determine if a change to one program would in any way affect the operation of any other program in the system.

The JPL Mission Build process allowed management to know what software was in the system at all times.

QUALITATIVE RESULTS: The visibility techniques management employed during the software development process were sufficient to permit a reliable software system that met all mission objectives to be delivered on schedule. The use of documentation, milestones, reviews, presentations, meetings and change control were standard management visibility tools. The concepts of the root level schedules and the software systems engineers roles were innovative, and worked well.

A great deal of expense and effort could have been saved had management demanded greater visibility early in the development process. Programmer selection was essentially ignored, assignments being made on an availability basis, rather than by experience and computer understanding. Critical reviews were not held to challenge requirements especially where they invoked design. Nor were critical reviews held to challenge the program designs themselves. As such, management control over the design and code phase was limited to monitoring schedules, so that design inadequacies did not become visible to the software systems engineers and systems programmers until the acceptance testing and integration phase. Costly modifications and workarounds were then required to force the program designs to meet mission timelines and computer constraint requirements. This occurred with about one-third of the programs.

QUANTITATIVE IMPACT: Ten percent of the programming and engineering manpower effort was consumed in activities directly related to providing management visibility. The primary efforts of the software system engineers and their staffs were devoted to directing and monitoring the development process and maintaining software schedules, thereby providing management with central sources by which progress and status could be measured. These staffs varied in size during the development stage (see Flight Operations Software Subgroup technique).

The manpower and computer resources that might have been saved had proper attention been paid to programmer selection and critiquing requirements and designs is estimated to be in access of three man years.

TECHNIQUE

NAME: COMPREHENSIVE END-TO-END SYSTEM LEVEL TESTING

SUMMARY: A series of integrated tests were performed using the Flight Operations ground data system, a Viking Lander on-board computer and its Flight Software, and the Viking Lander science, telemetry, communications, and power subsystems. Ten different tests, covering the four major mission phases were completed over a 22 month span to verify compatibility of the various programs and demonstrate representative Viking mission sequences. Approximately 156 software design changes were implemented as a result of this integrated series of tests.

APPLICATION CONSIDERATIONS: The integrated testing described herein were derived to insure and enhance mission success. The major factors that dictated this comprehensive testing were: a) error free Flight Software was required for the on-board computer; b) scientific and engineering mission design features had to be accurately translated into uplink commands to the Flight Software and the effect of these commands on the lander functions had to be predicted with certainty by the uplink programs; and c) the downlink programs had to provide accurate scientific and engineering data in a timely fashion so that new uplink commands could be generated based on actual conditions encountered at Mars.

RECOMMENDATIONS: The Viking system is a large one with many interrelated software and hardware elements whose mission application required virtually error free performance. Mission success is strongly dependent on comprehensive testing of the critical system elements where the emphasis is placed on testing mission level functions and performance requirements on an end-to-end basis. This provides a necessary check and balance against the design requirement testing done by the developers of the individual hardware/software elements of the system. The class of integrated testing described herein was a mandatory part of the Viking development.

HISTORY: The Flight Operations/Flight Software integrated test concept was not part of the original Viking development plans. The concept was recommended by several people on the project and further emphasized by some audit/review committees in the late 1973, early 1974 time period. Detail planning began in August 1974 and an initial series of tests were approved by the customer by early September 1974. As the benefit of the tests became apparent more were defined and approved for implementation.

DESCRIPTION: The Flight Operations/Flight Software Integrated Tests (FOFSIT) were a series of tests conceived to satisfy the following objectives:

a) Demonstrate compatibility between the Viking Lander on-board computer Flight Software and selected portions of the ground Mission Operations Software System (MOSS);

b) Demonstrate this joint set of software is compatible with "flight-like" Viking Lander hardware; and

c) Demonstrate this combined set of software and hardware is compatible with selected mission design requirements and representative Viking mission sequences.

Each test involved running a series of the Flight Operations programs in the Viking computer environment at the Jet Propulsion Lab (JPL) in Pasadena to produce uplink commands for the Flight Software in the on-board computer (GCSC). These commands were transferred to Denver by tape or data line where they were loaded into the on-board computer via the operational uplink communications port. For tests involving the interplanetary Cruise, Preseparation Checkout, or Landed surface operation mission phases, the Flight Software/on-board computer were run in a test bed composed of a full up, operational Viking Lander. This non-flight article lander with its power, telemetry, communications, science, and G&C subsystems was made available after an environmental qualification test program had been completed. For the test involving Descent to the surface of Mars the Flight Software/on-board computer combination were coupled to a hybrid computer facility programmed to simulate the descent vehicle and Martian environment.

When the Viking Lander or the descent trajectory simulation were run under control of the Flight Software a lander level downlink telemetry data stream was recorded. This data was then transferred to JPL by magnetic tape or data line where it was processed through a series of downlink programs to produce science and/or engineering outputs.

Table 1 provides a summary of the integrated test program composed of 10 individual tests covering the four major mission phases. The general test configuration for these mission phases is shown in Figures 1 and 2. The 22 individual software programs involved in the integrated testing included:

a) twelve batch Univac 1108 programs with a total of 147,000 source cards;

b) seven batch IBM 360/75 programs with a total of 88,000 source cards,

c) two real time IBM 360/75 programs with a total of 230,000 instructions for telemetry and image processing; and

d) one Flight Software program with a total of 18,432 words.

The following mission and spacecraft design constraints had a significant influence on the design and implementation of the integrated tests described herein:

a) The lander downlinked data to earth, directly and/or relayed through the orbiter, only once a day over a 20 minute one way light time path;

b) After the lander was separated from the orbiter the descent program in the on-board computer could not be modified by uplink;

c) During surface operations the lander was in a near autonomous operating mode running a preprogrammed mission in the on-board computer that could be modified by uplink only once every two days except in emergency conditions (once a day maximum);

d) The mission was designed to be adaptive and the adaptive reaction time, as measured from on-board downlink data generation to implemented adaptive uplink commands, varied from 3 days for emergency conditions to a maximum of 21 days; and

Table 1

Flight Operations/Flight Software Integrated Test Summary

Mission Phase/ Test Name	Test Description	Test Profile Time	Programs Tested	Test Period
Cruise	Maintenance functions for batteries, tape recorder, & 1 science instrument computer overlay & memory readout functions	10 hrs	8	Feb thru Jul 75
Presep C/O (FCT-3)	Precursor to launch pad test of lander C/O functions done prior to descent in Mars orbit during the mission	6 hrs	9	Dec 74 thru Apr 75
Descent (FCT-4)	Precursor to launch pad test of compressed descent sequence using modified G&C equations/ constants. Included landed initialization sequence (1st post land function)	3 hrs	7	Dec 74 thru Apr 75
Descent	Nominal full descent in a simulated Mars environment for a 2 burn deorbit mission. Included full uplink of G&C constants	5 hrs	14	Jan thru Sept 75
Landed (FCT-1)	Precursor to launch pad test 2 short landed segments involving real time imaging over the direct downlink, a tape recorder playback over the relay link, and command updates via the uplink	2 hrs	9	Dec 74 thru Apr 75
Landed (Multiday)	A fully integrated science/telemetry/communication sequence with 2 uplink periods. Based the planned mission sequences for 6 out of the first ten days on Mars. Tested 58 of 104 Flt Software major capabilities	154 hrs	15	Apr thru Oct 75
Landed (Multiday Extension)	A fully integrated sequence with three uplink periods. Tested an additional 30 of the 104 Flt Software major capabilities	62 hrs	15	Sept thru Dec 75

Table 1 (continued)

Flight Operations/Flight Software Integrated Test Summary

Mission Phase/ Test Name	Test Description	Test Profile Time	Programs Tested	Test Period
Landed (FOS-VER)	A test of the first 17 days of the landed pre-programmed mission (PPM) stored in the on-board computer. No uplinks	419 hrs	15	Nov 75 thru Apr 76
Presep C/O (FOS-VER)	A test of the lander C/O functions and uplinks done during the 52 hr period prior to descent of the lander	52 hrs	9	Nov 75 thru Apr 76
Landed (Switch-over)	Test of four uplinks derived to reinitialize the lander for mission operations if a switch to the backup on-board computer had occurred. Test included uplink processing and lander operations; no downlink processing at JPL	7 hrs	2	Feb thru Jun 76

Notes: 1) Test profile time covers the duration of Viking Lander operation; JPL computer processing time is excluded

2) Programs tested include only those involved in Viking Mission operations; test support programs not included

3) Test period includes time to run the uplink programs at JPL, run the Flt Software in Denver, and process/analyze the downlink data via the programs at JPL

4) A total of 22 different Flight Operations/Flight Software programs were used in the overall test program

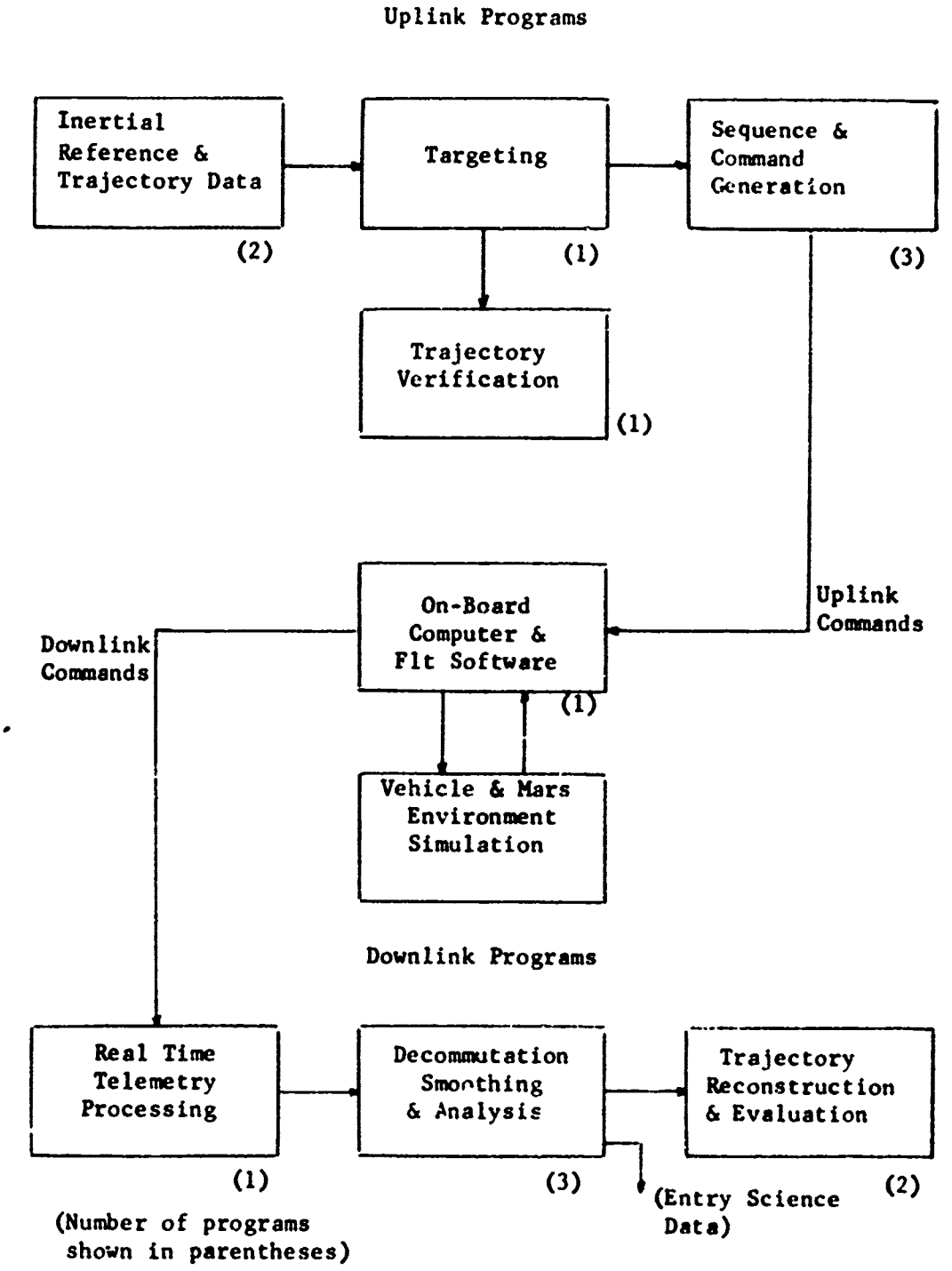


Figure 1. Descent Test

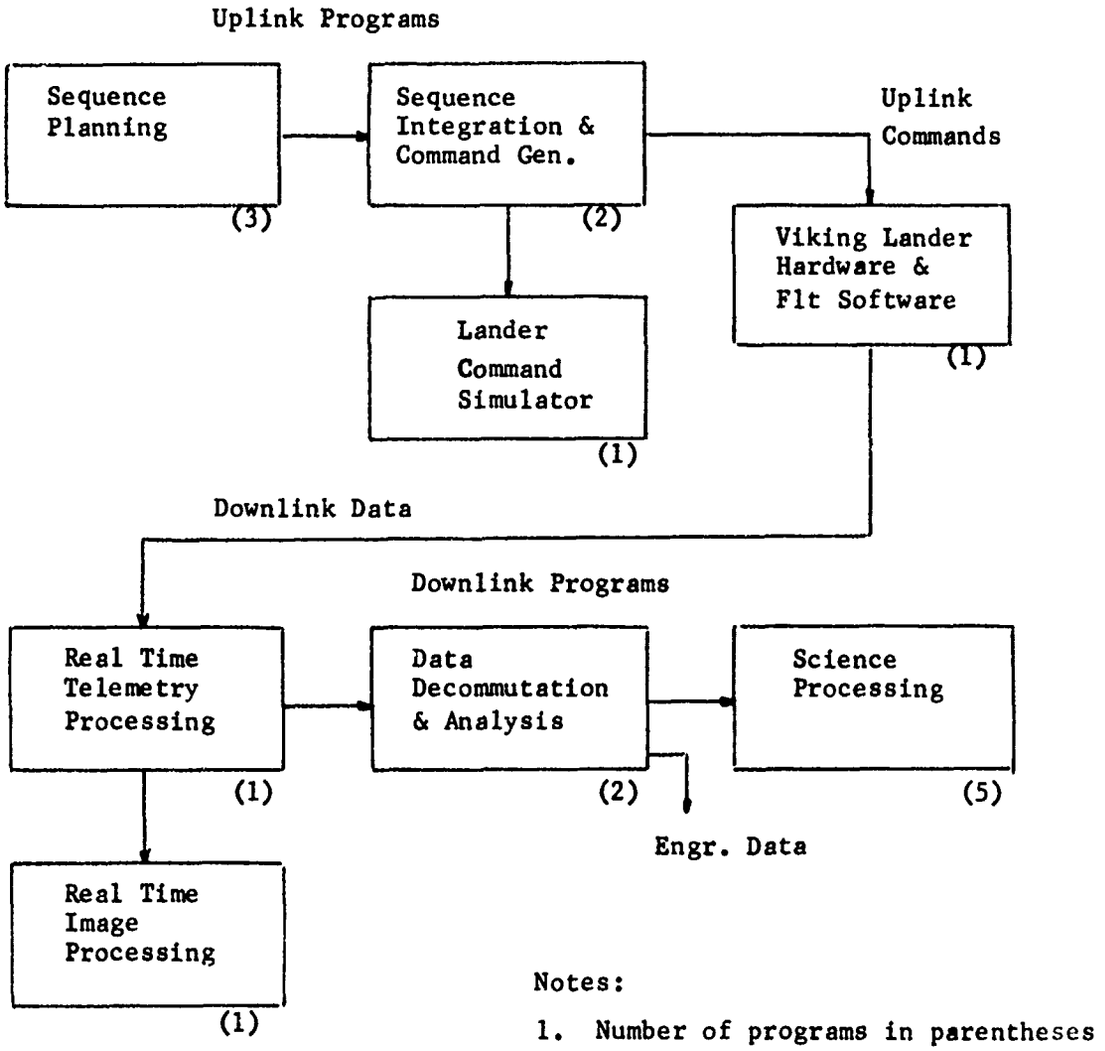


Figure 2. Landed, Cruise & Presep C/O Tests

e) The on-board computer was the central controlling source for performing, on a time shared basis, all lander functions including science control and data acquisition from seven experiments, direct and relay communications to earth, data storage management via tape recorder and core memory, and lander power management.

As a result of these constraints the integrated testing emphasized, in order of priority: a) uplink programs, b) Flight Software and lander operations, and c) downlink programs. Tests involving extended periods of lander operations were required to verify compatibility of the many interactive functions. The greatest emphasis was placed on full validation of the uplink programs that simulate lander operations including those of the Flight Software. This validation was done by direct detail comparison of the uplink predictions with the actual functions performed by the lander and its Flight Software.

The overall integrated test program covered a 22 month period beginning in August 1974 with a planning phase. In December 1974 a 200 page test plan was produced, covering the first six tests listed in Table 1. Nine of the ten tests involved running the Flight Software in the on-board computer in a full up Viking lander. For this portion of the testing additional detail test procedures were developed to provide a means to a) monitor, in real time, proper lander sequencing and hardware safety, and b) control the ground support equipment to acquire downlink data generated by the lander. Although all the tests were run in a fairly informal "engineering test" atmosphere, a significant portion of the overall effort was expended in the development of test plans and procedures (approximately 10-12% of the total manpower expenditures).

Generally, each test required several iterations (average of 3 to 5 passes) through the applicable uplink programs before a safe and adequate set of uplink commands were available for use with the Flight Software/on-board computer in the lander or descent test beds. Although the downlink data, in general, had to be processed more than once (average of 2 to 3 passes) to obtain satisfactory results, fewer

iterations were required than in the uplink area. The uplink and downlink data processing for most tests was accomplished by a small group assigned to the integrated test program. The time spans to accomplish the iterative uplink/downlink testing at JPL for an individual test ranged from a few weeks to a few months. During the actual mission operations the equivalent processing was accomplished by a large team of people in a few days in a single pass.

Table 2 provides a summary of the equivalent test iterations for the portion of the test involving the Flight Software/on-board computer in the descent or Viking lander test beds.

Table 2

Integrated Testing - Test Bed Utilization

Test Name (Ref. Table 1)	Test Bed	Number of Iterations	Total Hours
Cruise	Viking Lander	2-1/2	25
FCT's	Viking Lander	4	52
Descent	Descent Hybrid Sim.	3	15
Multiday	Viking Lander	2-1/2	385
Multiday Extension	Viking Lander	1-1/4	77
FOS-VER Presep C/O	Viking Lander	2	104
FOS-VER Landed	Viking Lander	1	419
Switchover	Viking Lander	3	21
			1098

QUALITATIVE IMPACT: The qualitative impact of the testing described herein can be summarized as follows:

a) Immature software - In terms of meeting the mission objectives, many of the programs tested were found to be immature. In particular, the three programs that 1) integrated science experiment/engineering subsystem sequences and modeled the various lander functions, 2) generated lander uplink commands, and 3) simulated Flight Software functions/commands required the majority (i.e. 48%) of the design changes implemented.

b) Lander "signatures" - Quicklook analysis of the downlink data from the integrated test revealed many anomalies. Extensive analysis indicated most of these were not caused by any design problem or failure in the hardware or software (i.e., they were signatures of a normal system). The acceptable items were cataloged and the availability of this knowledge during mission operations avoided unnecessary problem solving and delays.

c) Mission design validation constraints - The design and implementation of many of the integrated tests were based on various of the Viking mission design strategies; this provided a means to test the validity of each strategy. The testing identified many mission design problems and constraints. As a result revised strategies were developed and validated in the integrated test series. The accepted constraints were documented and served to guide the development of related mission design strategies.

d) Flight Team training - Initially the integrated tests were performed by a centralized group assigned to the task. Most of the group members were on the Flight Team but their integrated test assignments were not necessarily related to the functions they were to perform during the mission operations. During the last three tests (i.e., Multiday Extension, FOSVER, and Switchover) tasks such as uplink generation, downlink processing, etc. were performed by the appropriate Flight Team groups. This experience proved a valuable aid in training people for their actual mission assignments with realistic data.

QUANTITATIVE IMPACT: The quantitative impact of the integrated testing described herein can be summarized as follows:

a) Duration of integrated test program: 22 months;

b) Programs tested: 22 programs involving approximately 483,000 source cards/instructions;

c) Number of tests: 10 tests covering 4 mission phases;

d) Hours of real time lander (Flight Software) operation: tests ranged from 2 to 419 hrs; 1098 hrs total time;

e) Software design changes derived from test program: approximately 156 changes affecting 20 of 22 programs tested (Reference Table 3 for details);

f) Manpower cost: approximately 380 manmonths; peak loading 26 to 28 people for a 6 month period;

g) Computer Cost:

Univac 1108 @ JPL	275 hrs
IBM 360/75 @ JPL	2300 hrs
IBM 370/155 @ Denver	250 hrs
CDC 6400 @ Denver	40 hrs

h) Loss of mission critical data or objectives due to software design failures during actual mission: None.

Table 3

Software Design Changes

	Program Type	Programs Affected	Changes	% of Total
1.	Uplink sequence integration and command generation/simulation	3	75	48.1%
2.	Flight Software	1	16	10.3%
3.	Downlink real time processing and data decommutation/analysis	3	27	17.3%
4.	Science analysis	4	23	14.7%
5.	Descent unique - uplink & downlink	4	10	6.4%
6.	Miscellaneous	3	5	3.2%
7.	Programs with no impact	2	0	
	Totals	22	156	100.0%

TECHNIQUE

NAME: TECHNICAL AND MANAGEMENT AUDITS

SUMMARY: The Viking Project Office (VPO) formed a group of NASA software managers, known as the Tindall Committee, to review the Viking software development approach and design. Following this, the VPO conducted three independent audits by software experts from around the country. MMC held a semi-independent audit led by the head of the Systems Development Corporation. Technical audits by MMC management were used to measure progress and maintain schedules.

APPLICATION CONSIDERATIONS: The successful development of any major software system is a considerable task for experienced professionals. Neither the VPO nor MMC had ever built a system as large as required to support the Viking Mission. Therefore the desire to have experienced software development managers review and comment on the Viking approach manifested itself in the Viking managers minds early in the program. During the coding phase of development MMC managers knew their visibility into the process would be limited. Technical audits offered them the opportunity to assess progress, thereby enhancing the chances for schedules to be met.

RECOMMENDATION: Management is typically reluctant to shift significant amounts of resources to accommodate obviously well intentioned and reasoned recommendations originating from within their working ranks. It is difficult for them to weed out the good ideas from those that are necessary. By bringing in experienced experts from around the country to audit the development process, management can obtain a basis for making such decisions. Equally important is the fact that such audits are likely to produce recommendations concerning areas or concepts that have been overlooked.

HISTORY: In 1971 the Viking Project manager formed a committee of software managers from Johnson Space Center, Goddard Space Flight Center, Ames Research Laboratory, Marshall Space Flight Center, and NASA headquarters. The committee eventually became known as the Tindall committee, named for its leading spokesman, H. W. Tindall of JSC. The purpose of this committee was to attend monthly Viking management status reviews and make comments, assessments and recommendations to the Project manager relative to the Viking approach. The committee stayed in existence for approximately two years, monitoring progress up to and including the critical design reviews for Flight and Mission Operations software.

In late 1973 MMC brought in three software managers with different backgrounds to perform a semi-independent audit of the software development process. This included the head of Systems Data Corporation, an IBM executive and a member of a software consultant firm.

Shortly thereafter the Viking Project Office conducted an independent audit of Flight software. For this purpose they brought in experts from both industry and government.

Three software managers from JSC were brought to Denver at the request of the Viking Project Office in the spring of 1974 to audit the Mission Operations software system.

Finally, the Viking Project Office arranged for a group of experts from GSFC to attend a presentation of the Flight and Mission Operations software development process late in 1974.

In addition to these outside audits, the VPO, MMC and JPL managers held monthly meetings to review progress. This included audits of the major problem areas encountered by the Software System Engineers.

Finally, the MMC Mission Operations and Design director conducted individual indepth audits on each program for which he was responsible.

DESCRIPTION: During the tenure of the Tindall Committee its members would sit in and passively monitor the proceedings of the Flight Operations Working Group, which consisted of the managers responsible for the Viking Lander, Viking Orbiter, Viking Mission Control and Computing Center, Tracking Data System and Deep Space Network software development.

In addition, the committee attended the preliminary and critical design reviews for both Flight and Mission Operations software.

An early finding of the committee was that the multi-agency Viking managers could not resolve problems amongst themselves. Month after month the same problems remained unresolved. Furthermore, too many of the problems were technical in nature. One major recommendation made by the committee at this time was that management should concern itself primarily with handling schedules and resources and make the SSEs responsible for the design and development process. In that way, management would only be required to resolve those problems that the SSEs could not resolve, which should greatly reduce their task. A second recommendation was that the multi-agency Viking Flight Team be organized immediately in order to develop a working rapport long before it was needed to support the mission. Both recommendations were adopted.

The committee offered numerous suggestions to help resolve inter-agency problems, influenced the Integrated Software Functional Design, recommended that schedules be carried to several levels of detail (five were adopted by Viking), and pointed out the need for a computer loading study that covered the entire planetary operations phase on a day by day basis.

The semi-independent audit sponsored by MMC in late 1973 led to the decision to reallocate resources to accommodate end-to-end tests for the cruise, descent and planetary operations of the mission. The auditors flatly told MMC management that there was no way to know whether or not the system would work without such tests, and none had been scheduled. The idea was not new, since it had been recommended earlier by MMC software developers. The fact that it was repeated by an accredited outside source provided the straw that broke the camel's back in this area.

The independent auditors brought in by VPO to review Flight software saw the need for the Systems Engineering director to place software on an equal basis with hardware.

The audit by JSC might have been more fruitful had it occurred a year earlier. They recommended that individual program requirements be constantly reviewed to try and weed out any unnecessary ones. Most of the software had been developed by that time, so it was impractical to make extensive use of the recommendation. Two programs were reviewed with only minor success. JSC expressed concern over the inter-agency integration task, but could offer no constructive comments on the subject. Finally, JSC stated that they believed the programmer, rather than the engineer, knows best how to test a piece of software. MMC did not accept this recommendation since they were primarily concerned with the function the software was to perform, and the engineer knew the functions. They required that the programmers deliver working software to the engineer who then was required to acceptance test it.

The final audit conducted by GSFC came toward the end of the software development process. For that reason it amounted to more of a review than an audit. About all GSFC was able to comment was that a sound approach had been taken and no major item had been overlooked.

The VPO, MMC and JPL managers frequently required that the SSEs make semi-formal presentations both before them and before the cognizant engineers and programmers. The intent of these presentations effectively made them status and design audits by management. The audience would comment, criticize and raise questions following each presentation. Action items would be assigned at these presentations to resolve problems.

During the coding phase of operational software programs, the MO and D director would notify the cognizant engineer of a program that an audit of the program would be held in three days before the Mission Director. At these audits the engineer had to demonstrate what had been accomplished, what remained to be done, and how the schedule would be met. On a few occasions, when the Mission Director was in Denver, the engineers were given only a two or three hour notice of such an audit. On some occasions the directors were not satisfied that the schedules supported the work to be done, based either on what had been accomplished or on the amount of new requirements facing the engineer.

In these instances the engineer was required to maintain a level 6 schedule, which broke the work assignments down to a daily basis over a period of about a month.

QUALITATIVE RESULTS: The audits were extremely valuable to the Viking Project and contributed directly towards the success of the mission.

Technical inter-agency software problems were resolved much easier when the responsibility for handling them was shifted from the managers to the SSEs.

The establishment of the Viking Flight Team early proved to be a sound idea. The members of the team quickly realized that their responsibilities lay within the directorate and group to which they had been assigned even when the group leader or director was from a different agency. By the time the VFT was needed, responsibilities were well established and understood, which resulted in smooth operations during the mission and quick response to anomalies.

The implementation of five levels of schedules played a major role in developing the system on schedule.

The computer loading analysis study, conducted at the recommendation of the Tindall committee, led to the realization that the Viking Project would have to install a third 1108 computer at JPL to meet mission timelines. The computer was installed, and subsequent events proved that it was needed.

The scheduling of end-to-end tests for the Flight and Mission Operations software systems may have made the difference between mission success and mission failure. Although management was not convinced that the tests, which were an unscheduled and expensive resource drain, would do no more than give them a warm feeling that the system would work, they nevertheless accepted the auditor's recommendation. When the tests were finally conducted, they revealed literally hundreds of incompatibilities, errors and misunderstandings. Had the audit not been made, the tests would not have been conducted. In that event the problems most likely would not have surfaced until planetary operations, at which time they would have been extremely serious.

By auditing software at the program level, the MO and D director was able to reallocate resources to maintain schedules when it was evident that an engineer had underestimated the scope of a task. Such discoveries were made at these audits.

The value of audits by outsiders is that they will feel compelled to find something that you are doing wrong. Therefore, if you conduct such an audit and fail to get any recommendations of significance, the probability that you are on the right track and doing a good job is extremely high.

QUANTITATIVE IMPACT: The direct costs of the outside audits was equivalent to hiring consultants for a few days. The indirect costs of the audits was the time spent to prepare for and conduct them. Preparation was generally easy, because the speakers had merely to discuss their accomplishments, plans and problems, all of which were very familiar. On some occasions the speakers had to spend two or three days preparing slides and viewgraphs. Since relatively few audits were held, the total time consumed by them could not have exceed more than a few man months.

TECHNIQUE

NAME: GROUND DATA SYSTEM (GDS) TEST PROGRAM

SUMMARY: The GDS Test Program was a subset of the Flight Operations (FOS) Test Program. The objectives of the GDS tests were to verify and demonstrate the capability of the GDS to support Flight Operations Personnel Test and Training (FOPT&T) and to verify the capabilities committed to the Viking Project to support missi n operations. The GDS tests were end-to-end tests of all earth based facilities which were required to support mission operations and personnel training. The tests involved all operational subsystems of the GDS including telemetry, tracking, command, and monitor and operations control. Both real time and non real time functions were verified, including the generation of VO and VL command uplinks, products required for operational decisions, and all simulation data required for VFT training.

APPLICATION CONSIDERATIONS: The GDS Tests proved to be invaluable in preparation for the personnel training program. Numerous problems were exposed in spite of the fact that, in theory, all capabilities which were tested during GDS tests had been previously tested in some other element of the FOS Test Program. Furthermore, though these tests were classified as "engineering" tests, the training benefits which were realized far exceeded expectations. The personnel training program which followed was far more successful than anyone had expected.

RECOMMENDATION: The GDS Test Program exposed numerous technical problems that required resolution prior to the start of the personnel training program. It also provided invaluable training benefits for the Viking Flight Team working in concert with the personnel of the institutional facilities. The GDS Test Program was thus a necessary bridge between the development tests and the personnel training tests. Without this bridge, the personnel training tests would have had to deal with numerous technical and procedural problems which could have jeopardized the capability of the Viking Flight Team to adequately prepare for the planetary operations.

HISTORY: The need for a GDS Test Program was recognized in mid 1972. The objective of the GDS tests was to verify and demonstrate the readiness of the GDS to support key milestones in the development of the FOS. The tests were scheduled to occur in late 1974, following delivery of all the software and hardware required to conduct Viking mission operations. The GDS tests would be preceded by various engineering and operational tests conducted by institutional personnel of the Deep Space Network (DSN) and the Viking Mission Control and Computing Center System (VMCCCS) including interface tests between the two institutions. The GDS tests would be followed by the FOPT&T program in early 1975. The FOS would then be operational prior to the launch of the first spacecraft in August 1975. Further studies over the next year revealed that this ambitious FOS development program could not be implemented prior to launch. The delivery of the FOS was then divided into two phases. The phase one delivery would occur prior to launch, and would include all hardware, software, personnel and procedures required to conduct launch and cruise operations. Phase two delivery would occur after launch but prior to the start of planetary operations and would include the remaining capabilities required to conduct planetary operations. The FOS Test Program, including the GDS Test Program, was suitably modified to be compatible with the new FOS development approach. The schedule risk implied by the phased development was recognized. Accordingly, a subset of the planetary capabilities was scheduled to be delivered in phase one, and the GDS Test Program was amplified to include precursor tests of the planetary design of the GDS during the prelaunch period in order to identify basic problems and constraints which may require a long lead time for correction. This subset included DSS-14 in a planetary-like configuration (some planetary capabilities were not available), and some elements of the Project software and Simulation System.

The objective of the GDS Test Program thus evolved to the following: To verify and demonstrate the readiness of the GDS configurations to support key activities in the development of the Viking FOS. The activities are:

a. Flight Article Compatibility Tests (FCT) with a Viking Lander, a Viking Orbiter, and the compatibility test station MIL-71 at Kennedy Space Center (KSC).

b. Launch and cruise training exercises and flight operations with the 64 Meter Deep Space Stations (DSS), i.e., DSS-14 at Goldstone, California; DSS-43 at Canberra, Australia, and DSS-63 at Madrid, Spain and the prime 26-Meter DSS net; i.e., DSS-11 at Goldstone, DSS-42 at Canberra and DSS-61 at Madrid.

c. Cruise flight operations with the secondary 26-Meter DSS net; i.e., DSS-12 at Goldstone, DSS-44 at Canberra and DSS-62 at Madrid.

d. Launch and cruise training exercises and flight operations with the Near Earth Phase Network (NEPN) including facilities of the Air Force Eastern Test Range (AFETR) and the Goddard Space Flight Center (GSFC) used to acquire spacecraft data prior to the initial acquisition by a DSS.

e. Planetary Verification Tests with DSS-14 prior to launch.

f. VFT planetary training exercises and mission operations.

DESCRIPTION: The Flight Operations System (FOS) Test Program for Viking was defined in the FOS Test Plan, PL-3713006. This plan established the overall objectives, purposes and scope of FOS testing. The Ground Data System (GDS) Test Program, a subset of the FOS Test Program, was defined in the GDS Test Plan, PL-3720313. This plan provided the objectives and description of each GDS test and established the facilities required to support each test. The detail test objectives, acceptance criteria, test approach and Viking Flight Team (VFT) support requirements for each test were defined in a GDS Test Script. The step by step sequences required to execute each test were defined in the test Sequence of Events (SOE).

Tests conducted prior to launch were designated "launch and Cruise (L&C) Phase Tests" while those conducted during cruise to prepare for the planetary operations were designated "Planetary Operations Phase Tests". The Launch and Cruise Phase Tests included .. ' tests in

support of FOS milestones (a) through (e) inclusive, as defined in the previous section. These tests were divided into four distinct categories.

The conduct of the tests in all four categories were chronologically interwoven because of the planned delivery schedule of the various GDS facilities. The more complex all-up system tests of each type were, in general, preceded by prerequisite tests of reduced complexity. The specific tests in each category are defined below:

Category 1 - Tests of the FCT Configuration

A GDS test (designated GDS 7.0) was performed to verify and demonstrate the readiness of the GDS to support FCTs. Simultaneous telemetry and command functions were successfully performed with MIL-71 during this test. The sequence of telemetry data states (data rate and format) was representative of the sequences planned for the FCTs.

Category 2 - NEPN Tests

Two tests (GDS 8.1 and GDS 8.2) were performed with the NEPN to support launch. The first test demonstrated data flow from SIMCEN at JPL to the compatibility test stations, STDN MIL and TEL-4, via MIL-71, and the return of data to the VMCCC and MSA. The second test attempted to demonstrate telemetry and tracking data flow using the NEPN down range stations for a representative launch sequence. It was discovered that the telemetry data could not be frame synched in the Mission and Test Computer Facility (MTCF). Post test investigations revealed that a configuration change was required at MIL-71 to correct the problem. This configuration was made and successfully demonstrated in a special test conducted in parallel with the Flight Operations Personnel and Training test DT-1. This configuration was then utilized with success in the Operational Readiness Tests (ORTs) and launch.

Category 3 - Cruise Tests

Thirteen tests were performed to verify and demonstrate the readiness of the launch and cruise configuration of the GDS to support VFT launch and cruise flight operations and training exercises. These tests were conducted with the nine DSS facilities of the DSN. The test designation, test type and applicable DSS are identified in table 1.

Table 1

Launch and Cruise GDS Tests

GDS Test Designation	Test Type	Deep Space Station (DSS)
1.1	Telemetry	DSS-11
1.2	Command	DSS-11
1.3	Tracking	DSS-11
2.3	Tracking	DSS-14
1.4	Combined Systems	DSS-11
3.4	Combined Systems	DSS-42, DSS-43, DSS-44 DSS-61, DSS-62, DSS-63 and DSS-12 individually
3.5	Combined Systems & Multiple Stations	DSS-42, DSS-43, DSS-61 DSS-63, DSS-14 and DSS-11 collectively with realistic overlap

Tests 1.1, 1.2, 1.3 and 2.3 were performed to prepare for the more complex Combined Systems Tests. All tests were successfully performed and all test objectives were met. Five tests were performed to verify and demonstrate the operation of the planetary design of the GDS. The test designations, test type, applicable DSS, and the mission phase simulated during the test are identified in Table 2.

Table 2

Planetary Verification Tests

GDS Test Designation	Test Type	DSS	Simulated Mission Phase
2.1	Telemetry	DSS-14	Planetary-1 VL and 1 VO serially
2.7	Combined Systems	DSS-14	Planetary-1 VL and 1 VO serially
2.6	Telemetry	DSS-14	Planetary-2 VOs and 1 VL simultaneously
9.2	Combined Systems	DSS-11, DSS-14	DOY 175 of the Primary Mission Design (PMD)
9.3	Combined Systems	DSS-11 DSS-14	Planetary-2 VOs and 1 VL simultaneously

Tests 2.1, 2.7 and 2.6 were performed to prepare for the 9.2 and 9.3 tests by sequentially increasing the complexity of test functions. These initial tests were not totally successful; however, judgement was made that the problems encountered were understood well enough to permit implementation of the 9.2 and 9.3 tests. Tests 9.2 and 9.3 were accomplished successfully with most test objectives satisfied. The tests did demonstrate the integrity of the Planetary Operations design of the GDS; however, GDS 9.2 demonstrated that the VO simulation math model (OSIM) required significant development to be useful for Planetary Operations training exercises, and that improvements were required in usage of the General Purpose Computer Facility (GPCF) in order to accommodate the anticipated software loads. GDS Test 9.3 demonstrated the need for analog recordings to recover data which would otherwise be irretrievably lost in the event of

failure of critical, non redundant equipment at the 64-Meter Deep Space Station. This capability was added to the 64-Meter Stations in time to support planetary operations and was utilized on numerous occasions with varying degrees of success.

There were eight GDS tests performed during the Planetary Operations Phase Tests. The objectives of these tests were to verify and demonstrate the readiness of the planetary configuration of the GDS, including the Viking Project Simulation System (VPSS) to support the planetary FOPT&T, and the readiness of the GDS to support planetary mission operations.

GDS Test 10.0 - This test was performed to verify the operations of the institutional portion of the VPSS (exclusive of the project supplied spacecraft simulation models) interfacing in the long loop mode with the compatibility test station (CTA-21) located at JPL. Telemetry data states representative of planetary operations were demonstrated. The responses of the DSN portion of the VPSS to control messages generated in the VMCCCS were verified.

GDS Test 5.1 - This test demonstrated that the VPSS, including the institutional portion (MSIM) and the Project supplied spacecraft simulation models, OSIM and LSIM, could interface in the long loop mode with a 64-Meter DSS to simulate a mission segment encompassing the Mars Orbit Insertion (MOI) for spacecraft B. This test was accomplished with DSS-14.

GDS Test 5.2 - GDS Test 5.2 verified the capability of the GDS to process the spacecraft X-band products. This test was conducted during scheduled Viking passes.

GDS Tests 5.31 (DSS-14) and 5.32 (DSS-43 and DSS-63 Individually) - These three tests were structured primarily to verify readiness of the planetary configuration of the GDS to conduct planetary flight operations. The maximum design loading conditions were imposed consistent with the committed capabilities of the VMCCC and the DSN. Telemetry, tracking, command and monitor functions were performed simultaneously. Real time displays were generated in the MSAs. The generation of telemetry data records up through and including Experiment Data Records

(EDRs) and first order VL imaging products were verified. These data records were generated from Intermediate Data Records (IDRs) which were prepared by the DSN using interim IDR software. First order Viking Orbiter (VO) Visual Imaging Subsystem (VIS) products were not verified because the capability was not yet delivered.

GDS Test 6.0 - This test was structured primarily to verify readiness to support the planetary phase of VFT test and training. A segment of the Primary Mission Design (PMD) (as reflected in the Flight Operations Personnel Test and Training (FOPT&T) test scripts) was selected for simulation activity. The math models OSIM and LSIM were used to provide the simulated telemetry data for this mission segment. The test involved DSS-14, DSS-43, and DSS-63 in accordance with a representative 24 hour time-line. Batch processing of software programs representative of the selected mission segment was accomplished in the Mission Control and Computing Facility (MCCF) 360/75 and GPCF 1108 computers for downlink processing activities.

GDS Test 11.0 - This test was incorporated into the GDS Test Program in order to verify capabilities which were not available for GDS Tests 5.31 and 5.32 and 6.0 to verify the untested capabilities (except VIS first order processing) and to demonstrate the readiness of the planetary configuration of the GDS to support real time and near real time planetary operations. The requirement to test VIS processing had not been incorporated into this test in view of the extensive VIS data processing performed as a normal part of cruise operations. This test verified the suitability of special GDS configurations which had been devised to support three critical mission activities - (1) Mars Orbit Insertion (2) Viking Lander Direct Link Transmissions and (3) Transmission of critical data over the Viking Orbiter high rate subcarrier.

QUALITATIVE RESULTS: This section describes the specific benefits derived from the GDS Test Program, and a critique on the successes and difficulties encountered.

1) Numerous hardware and software design deficiencies were exposed during the tests. In most cases, these deficiencies were corrected prior to the start of VFT training. The training exercises were thus conducted with a minimal number of incidents caused by design errors, permitting the VFT to concentrate on their training objectives.

2) During conduct of the launch and cruise phase GDS tests, a list of liens was compiled for unexpected characteristics of the GDS. Each lien was dispositioned in one of two ways: the lien was removed by corrective action for the observed characteristic, or the lien represented a constraint or characteristic which must be observed by the VFT in planning and conduct of mission operations. As a result of periodic reviews on the status and disposition of these liens, a decision was made to generate a document which compiled all guidelines, constraints and limitations on the operation of the GDS which were identified during all Ground Systems Test (including the GDS tests and all other tests of the FOS test program) and which must be observed during Viking flight operations. This document, VFT-003 Viking 75 Project Guidelines for the Operation and Use of the Viking Ground Data System, was published on 24 June 1975 and was continually updated throughout the planetary development phase and planetary operations of the Viking Project.

3) Due to limitations of the simulation system, some capabilities which were required for mission operations were not tested during the GDS test program. The simulation system was adequate to perform end-to-end tests of the real-time portion of the GDS, but the data was not adequate in some cases to generate non real time products. In one particular instance, orbiter image data (VIS) processing, the lack of adequate testing resulted in the discovery of serious VIS processing problems during cruise operations. Fortunately, the extensive processing of VIS data during cruise operations enabled

most of these problems to be solved prior to the start of the planetary operations wherein the VIS Data was critical for selection of a landing site. This panic could have been avoided if the simulation system requirements were more carefully reviewed early in the development cycle and all deficiencies eliminated. The importance of an early and sustaining interraction between the developers of the simulation system and the users (the VFT, including the GDS Test planners) cannot be overemphasized. In the Viking program, this interaction was not as thorough as it should have been which not only caused some test deficiencies as described above but also led to the necessity for a multitude of changes in the simulation system during the GDS Test and FOPT&T period.

4) Numerous GDS tests, special tests conducted by GDS personnel, and tests conducted by the VMCCCS personnel were run to ascertain Flight Support 360/75 computer loading guidelines. (The multimission real time computer) Each of these tests had slightly (or markedly) different results because the loading on the computer proved to be sensitive not only to the input data and the data quality, but also to the manner in which the computer is utilized by the various users. As a result, the loading guidelines changed through an evolutionary process by the identification and measurement of the loading parameters until finally a set of guidelines was established that satisfied both the needs of the Viking Project and the MCCC.

5) During the course of preparation for planetary operations, the GDS configuration (hardware configuration and MOSS) changed as the detail planning matured. The GDS Test Program was initially scheduled to be complete prior to the start of the VFT planetary test and training program. Before the program was complete, it became clear that additional tests were required, and hence GDS Test 11.0 was incorporated into the program. Fortunately the resources, both GDS and personnel,

could accommodate this addition. In the future, a GDS test should be planned as a contingency just prior to the start of operations to verify the final hardware/software configuration.

6) During the planetary GDS test period there were three major activities competing for the GDS and VFT resources:
   a) Conduct of the GDS tests including post test data processing;
   b) Preparation for the VFT Planetary test and training program;
   c) Conduct of Viking cruise operations.

   These three competing activities taxed some elements of the VFT to the limit. As a consequence, some of the post test data processing activities as defined in the GDS test scripts were not completed prior to the next milestone (although, in most cases, complete enough to make meaningful conclusions). Moreover, when incompatibilities were discovered fixes had to be devised in an unexpectedly short time period to prepare for the next activity. The GDS and VFT did achieve the state of operational readiness on schedule. However, in retrospect, the planetary GDS test program should have been started one month earlier to reduce the conflict for resources and permit the completion of all planned test activities. This more conservative approach would have increased the benefits of the GDS Test Program.

7) One of the fundamental objectives of the GDS Test program was to verify the readiness of the configurations of the GDS to support the VFT test and training. Hence, the VFT had to implement the test sequences under the direction of GDS personnel without the prior benefit of training. This proved to be a difficult challenge to which the VFT favorably responded. The training benefits were important not only during the Launch and Cruise Phase GDS Tests, but also during the planetary phase wherein the VFT had experiance in cruise

operations. The planetary configuration of the GDS was far more complex than the cruise configuration. Furthermore, in many instances, the planetary modes were contrary to the cruise experience of the VFT, thus leading to confusion with regard to specific instructions in the test SOEs. For example, during cruise the HSDL was customarily used for VO high rate telemetry data at 1 Kbps and 2 Kbps. In the planetary operations VO high rate data is normally routed over the WBDL, irrespective of data rate. Thus, an invaluable by-product of the GDS Test program was the experience gained by the VFT and these benefits were realized during the very successful test and training program which followed.

8) The test schedule philosophy required scheduling retests for all of the complex combined systems tests. This provided the assurance that resources would be available for a retest in the event that (1) the test was not successful, or (2) problems were identified in this test which required fixes or workarounds. In this manner, difficult scheduling perturbations on short notice were avoided. If the retest was not required, then the facilities were released back to the institutions to support other activities. In the main, this philosophy proved to be prudent. More than half of the scheduled retest periods were actually needed. In two cases, additional retests were required beyond the planned retest periods. These two retests were easily accommodated, because there remained some unused retest periods from other tests.

QUANTITATIVE IMPACT: There were 19 GDS tests and eight retests conducted during the launch and cruise phase, not including the two tracking tests conducted with a flight spacecraft. The total test hours corresponding to these 27 tests was approximately 260 hours, not including countdown activities or pre and post-test data processing. Three of the 19 tests involved the generation of non-real time products following the tests. The launch and cruise phase tests were conducted

between February 1975 and July 1975. During this period, the manpower from the GDS test organization was nine engineers and one aide. There were approximately eight VFT personnel that supported each test, in addition to the dozens of institutional personnel (DSN and VMCCCS). The time required for each of the VFT support personnel to prepare for the test was approximately eight hours. The manpower required to generate the post-test data procucts was not significant.

Following completion of the launch and cruise phase tests, the GDS test organization was reduced to three engineers and one aide to prepare for the planetary phase tests. These tests were conducted between November 1975 and May 1976, with the peak load occurring between December 1975 and February 1976. During the planetary phase, seven tests and four retests were conducted. The total test time was approximately 140 hours. The manpower required for post-test activities was significantly greater than that required for the launch and cruise tests. An estimate of the number of manhours is not available, since no accounting records were kept. A reasonable guess is two to three manmonths for VFT personnel. As noted in the previous section, the GDS test activity competed for resources with other activities, and not all planned post-test activities were actually accomplished. Two of the Planetary GDS Tests utilized the spacecraft simulation models, OSIM and LSIM. The time required by the model operators to prepare for the tests was significant; however, there are no specific accounting records from which to generate actuals. A reasonable guess is four manmonths.

In summary, the GDS Test Program expenditures are estimated as follows:

1) Twenty-six tests plus 12 retests for a total of 400 hours of test time;
2) Approximately 600 hours of DSS time including 400 hours of test time and 200 hours of countdown;
3) Approximately 1000 hours of 360/75 computer time including 780 hours of test time, 50 hours of countdown, and 120 hours of batch processing and math model preparation;

4) Approximately 720 hours of Univac 1530 computer time including 620 hours of test time, 50 hours of countdown, and 50 hours of non-real time processing;

5) Approximately 670 hours of Univac 1219 computer time including 620 hours of test time and 50 hours of countdown;

6) Approximately 50 hours of Univac 1108 computer time for batch processing;

7) Approximately 92 manmonths for GDS Test Engineers;

8) Approximately 18 manmonths for VFT support personnel.

TECHNIQUE RELATIONSHIPS TO STRUCTURED PROGRAMMING SERIES

The techniques described in this report have at most an indirect relationship with the Structured Programming Series, as described in RADC-TR-74-300 (Volumes I through XV). The primary reason for this is that Viking did not require that structured programming techniques be followed. Top down modular designs were used for the Flight and System Test Equipment software systems, each of which operated in a single computer. In addition, these systems were developed in assembly language code. A top down integrated functional design approach was taken to structure the multi-computer/operating system/program Mission Operations software system. Although most of this software was written in FORTRAN, structured programming was not invoked because a significant number of programs were obtained by modifying existing code and because the guidelines and constraints imposed by the Viking Mission Control and Computing Center were restrictive relative to program size and run times. Tables 1 through 4 cross references the techniques described herein with appropriate sections of the Structured Programming Series.

Table 1

Mission Operations Techniques	Structured Programming Series Volume/Sections
Overview	I/2; IX/2, 3
High Order Language	I/App A
Dif. Dev/Int Sites	I/App A
Computer Loading Pred.	IX/3, 4
Lander Command Simulation	
Prog. & Data Base I/F Mgmt	I/2
On-Line Data File Mgmt	I/2
Int. S/W Functional Design	I/2; IX/3
Mission Build Process	V/2, 3; VI/4
Cog Eng/Cog Prog.	I/2; X/1, 2
S/W Data Base Document	I/2
FO S/W Subgroup	I/2; IX/2; X/1, 2

Table 2

Flight Software Techniques	Structured Programming Series Volume/Sections
Overview	IX/3; X/1, 2
Emulated On-Board Computer	II/1
VL Comp. EXEC Prog.	
H/W S/W Int. Lab.	
Ind. Verifier	
Timing/Sizing Monitoring	IX/4
Reg. Gen. for Flt H/W & S/W	

Table 3

STE Software Techniques	Structured Programming Series Volume/Sections
Overview	IX/3; X/1, 2
Test Data Base	
Viking Test Language	II/1
Test System Simulator	II/1
Flt Sys. Test & C/O	
Sci. Inst. Perf. Verif.	
Viking Test Seq. Gen	II/1

Table 4

Management Techniques	Structured Programming Series Volume/Sections
SCR/Impact Sum.	IX/3
Viking S/W Standards	I/3; VII/3
FO S/W Plan	IX/2
S/W Dev. Mgmt Visibility	I/2; IX/2
Comp. End-to-End Testing	XV/2
Tech & Mgmt Audits	IX/2
Ground Data System Testing	XV/2

COMMENTS RELATIVE TO STRUCTURED PROGRAMMING SERIES

Volume I, Section 2: Viking interfaces were worked early and did not cause a delay in integration. Test data often was poor. A modular integration approach was taken. The Mission Build process provided the Programming Support Library function.

Volume I, Section 3: Standards relative to code were not enforced.

Volume I, Appendix A: Code efficiency was a function of programmer skill and clarity of requirements. Compilers caused some inefficiency problems.

Volume II, Section 1: Emulation and special processing techniques were employed to check Flight Code and STE test sequences before they were used in test.

Volume V, Sections 2, 3: The Mission Build process provided for Programming Support Library type functions.

Volume VI, Section 4: The Mission Build process provided a library organization similar to that described.

Volume VII, Section 3: Viking found it necessary to develop each type of document described.

Volume IX, Section 2: Viking applied the four management functions described. Auditing played a major role in supporting the control function.

Volume IX, Section 3: Viking followed the basic development cycle described. The reasons for source code updates was primarily caused by requirements changes. Program improvements and program errors also contributed significantly.

Volume IX, Section 4: Data was collected relative to computer loading and associated human activities; it proved to be very valuable. Timing and sizing data was also needed to prevent overloading the Flight computer.

Volume X, Sections 1, 2: A process similar to the Chief Programmer Team description was used to develop both the STE and Flight software systems. This tended to reduce upper managements visibility into the development process. The Software System Engineer/Cognizant Engineer/ Cognizant Programmer philosophy used to develop the million plus source card Mission Operations software system significantly differed from this approach. It proved to be highly successful.

Volume XI, Section 2: Viking experienced most, if not all, of the estimating problems listed. Because the development period spanned a long period of time, the Project Manager applied a rather interesting approach to controlling costs. Every six months or so, the budget allotted to managers would be ordered cut. The managers then would reassess their resources and drop efforts that had earlier appeared sound, but with time did not prove to be too practical. In that way, Viking was able to adapt to changes in the state of the art and maintain sufficient resources to implement them without experiencing serious overruns in most software areas.

Volume XV, Section 2: The end-to-end and ground data support tests, considered together, provided the actual validation tool for Viking. This was not foreseen until midway through the development phase, and was not recognized until the tests were actually performed.

RECOMMENDATIONS FOR AIR FORCE APPLICATION

Each of the techniques described in this technical report are applicable to specific situations that can arise during the development of a software system. The primary value of the report should be to increase the ability of the Air Force to understand the nature of software development and to apply this knowledge to recognizing those items that can and have impacted costs and schedules. The following recommendations are made relative to software management functions:

1) Follow a top down development cycle that includes the following phases

   a. Mission Definition
   b. System Requirements
   c. System Design
   d. Module Requirements
   e. Module Design
   f. Code and Debug
   g. Module Test
   h. Subsystem Integration
   i. System Integration
   j. Mission Test

2) Write a software management plan that defines and can control the development cycle from requirements through final system delivery. The plan should be geared to the software task at hand. At a minimum it should specify management roles/responsibilities, documentation requirements, developmental milestones, any standards that will be imposed, reviews that will be required, baselines that will be established, the software control process, the change control procedure, the level of testing that will be required, and delivery procedures for the software end product.

3) Establish meaningful milestones that can be measured. Items such as documentation release, approvals, reviews, baselines, deliveries and tests are useful in this context.

4) Don't wait until the test and integration phase to find out if the software system will support the mission. Determine as early as possible if operational problems will exist. Computer loading and operational analyses can be used to support this function.

5) Stress the importance of requirements to be complete, accurate and precise. They should go beyond the technical needs for the software and address such items as all known constraints, human engineering problems and test considerations.

6) Establish baselines to control requirements, design and end products.

7) Place requirements under control and don't permit changes to them to be approved until their impact on costs, schedules and resources are understood.

8) Stress the importance of designing to meet the requirements. Also stress the importance of designing to take advantage of the target computer characteristics.

9) Don't permit coding to begin until the design has been approved.

10) Establish an independent test and integration team to test the software against requirements. Let the programmers test the software against its design.

11) Maintain central sources for requirements and data.

12) Gear configuration management to bringing software under control as soon as practical. Do not begin formal integration until the software is under control (i.e. - out of the hands of the programmers).

13) Stress the importance of test data. Begin the effort to collect it early in the development cycle. Avoid the use of scaffolds (i.e. - fake, hand generated type data) wherever possible. If they are necessary, have them developed independent of the programmer responsible for the software that will process them.

14) Plan on uncovering errors during every phase of test and integration. Greater emphasis on the requirements/design phase should reduce the number and seriousness of errors.

In addition to the above it is recommended that the Air Force adopt a policy of hiring independent software experts to audit the development approach and problems faced by contractors responsible for building large and/or complex software systems. The auditors should have experience in developing similar systems, and the same auditors should not be used over and over again. The audit report should be made available both to the Air Force and to the contractor.

No recommendation is offered relative to which is best - the chief programmer approach, the Software Chief approach (refer to STE and Flight overviews), the SSE/CE/CP approach (refer to Flight Operations Software Subgroup and Cognizant Engineer/Cognizant Programmer techniques), or the software pool approach. The size and/or nature of the software system will influence most contractors as to which approach is most appropriate. In any event, the Air Force should recognize that each of these approaches are sound, and the selection of the particular approach should be left up to the software developer.

Since Viking did not require structured programming techniques be followed, no recommendation can be made relative to their value. However, some techniques associated with structured programming were followed. These included modular design, in-line procedures, minimizing the use of unconditional transfers, and code walk-throughs. All of these features tended to improve software reliability.

RECOMMENDATIONS FOR FURTHER STUDY AND ANALYSIS

1. The data driven design concepts used for the flight and test software systems proved operationally to be extremely practical. Modifications to lander hardware components could easily be tested by merely changing data base items. Significant and safe changes to landed operations were available to the Viking Flight team, who uplinked 60000 words of code controlling data modifications to each of the Viking Landers. It is therefore recommended that the Air Force study the influence that designing requirements to be data rather than code has on system reliability, schedules and costs.

2. The issues involved with software portability need further study. Emphasis should be placed on systems as well as programs. Topics that should be addressed include programmer education, pathfinder studies, design standards, language choices, phases of testing, data base management, interface requirements, translation techniques, and methods that can maintain near optimal performance across differing computer capabilities.

3. The concept of attempting to write a Users Guide as part of the requirements phase should be studied from human engineering and software reliability point of views. Such a technique could prove to be cost effective from a software change point of view.

4. The relative values of software documentation should be studied. This would include establishing minimal documentation requirements, concepts for reducing the amount of documentation and increasing reliability through centralization, determining how long documents should be maintained by type, and setting standards for the content, control and organization of the documents.

5. A study to determine the value of a Chief Programmer approach as a function of the size and scope of the software task, especially as it relates to software only or software/hardware development, should be made. Particular emphasis should be placed on determining its impact on management visibility during the development phase. Methods for improving the ability of the software developers to understand technical and human engineering requirements should be addressed.

6. A study should be conducted to determine aids, mechanisms, tools and procedures that can be used to provide for early software control to improve system reliability.

7. A trade-off study on the types of functions where assembly language is cost effective over HOL should be made. The average assembly language programmer is of higher quality than the average HOL programmer. In all probability, the bit manipulating Viking Lander decalibration and decommutation program would have been better designed, smaller and more efficient if the original requirement had been to write it in assembly language rather than FORTRAN.

ACRONYMS AND NON-STANDARD ABBREVIATIONS

AFETR	Air Force Eastern Test Range
AGE	Automated Ground Equipment
AHCF	Analog/Hybrid Computing Facility
BCD	Binary Coded Decimal
CCDU	Computer Control and Display Unit
CDC	Central Data Corporation
CE	Cognizant Engineer
CP	Cognizant Programmer
CPU	Central Processing Unit
CRT	Cathode Ray Tube
DART	Dynamic Algorithm Replacement
DAS	Direct Access Space
DCE	Data Conversion Equipment
DSI	Data Systems Integration
DSN	Deep Space Network
DSPE	Data Systems Project Engineer
DSS	Deep Space Station
EDR	Experiment Data Record
FCT	Flight Article Compatibility Tests
FIFO	First In First Out
FOPT&T	Flight Operations Personnel Test and Training
FOS	Flight Operations System
FOWG	Flight Operations Working Group
FRD	Functional Requirements Document
GCF	Ground Communication Facility
GGSC	Guidance, Control and Sequencing Computer
GDD	General Design Document
GDS	Ground Data System
GPCF	General Purpose Computing Facility
GPSSM	General Purpose Simulation System Model
GSFC	Goddard Space Flight Center
HOL	High Order Language

HSDL	High Speed Data Line
ICCB	Integration Change Control Board
ICS	Interpretive Computer Simulation
ICSSE	Integrating Contractor Software System Engineer
IDR	Intermediate Data Record
IPDS	Interface Point Data Set
IPL	Image Processing Laboratory
I/O	Input/Output
IOP	Input/Output Processor
IRU	Inertial Reference Unit
ISAM	Index Sequential Access Method
ISFD	Integrated Software Functional Design
JPL	Jet Propulsion Laboratory
KSC	Kennedy Space Center
L&C	Launch and Cruise
LCOMSM	Lander Command Simulation (program)
LOL	Low Order Language
L/OST	Lander/Orbiter Software Test (plan)
LPM	Lines Per Minute
LRC	Langley Research Center
MCCF	Mission Control and Computing Facility
MMC	Martin Marietta Corporation
MO&D	Mission Operations and Design (directorate)
MOSS	Mission Operational Software System
MSA	Mission Support Areas
MTCF	Mission Test and Computing Facility
NEPN	Near Earth Phase Network
NOCC	Network Operations Control Center
OCOMSM	Orbiter Computer Simulation (program)
OLDFMS	On-Line Data File Management System
ORT	Operational Readiness Test
OS	Operating System
PMD	Preliminary Mission Design
PTC	Proof Test Capsule
RA	Radar Altimeter

RARS	Radar Return Simulator
RCS	Reaction Control System
RTPM	Real-Time Program Management
SACT	Status and Criteria Table
SCR	Software Change Request
SDBD	Software Data Base Document
SFD	Software Functional Description
SOE	Sequence of Events
SRD	Software Requirements Document
SSE	Software System Engineer
STACOP	System Test and Checkout Program
STE	System Test Equipment
SWSG	Software Sub-Group
TDE	Terminal Descent Engine
TDLR	Terminal Descent Landing Radar
TDS	Tracking Data System
TRB	Translation Control Block
TRCB	Transfer Control Block
TSE	Test Support Equipment
TSS	Test System Simulator
UAT	Users Acceptance Test
UVT	Unit Verification Test
VADF	Viking AGE Decommutation File
VAGF	Viking AGE Group File
VAIF	Viking AGE Interface File
VCS	Viking Change Summary
VDA	Valve Drive Amplifier
VFT	Viking Flight Team
VIC	Viking Integration Change
VIS	Viking Imaging Subsystem
VL	Viking Lander
VLC	Viking Lander Capsule
VLSSE	Viking Lander Software System Engineer

VMCCC	Viking Mission Control and Computing Center
VO	Viking Orbiter
VOSSE	Viking Orbiter Software System Engineer
VPO	Viking Project Office
VPSS	Viking Project Simulation System
VSIG	Viking Software Integration Group
VTL	Viking Test Language
VTOC	Volume Table of Contents
WBDL	Wide Band Data Line

METRIC SYSTEM

BASE UNITS:

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

SUPPLEMENTARY UNITS:

plane angle	radian	rad	
solid angle	steradian	sr	

DERIVED UNITS:

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

SI PREFIXES:

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

* To be avoided where possible

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.