SOFTWARE VALIDATION STUDY

D. Bruce Brosius

Autonetics

Prepared for:

Space and Missile Systems Organization

February 1973

SAMSO TR 73-69

SOFTWARE VALIDATION STUDY

FINAL REPORT

D. Bruce Brosius
Autonetics Division
North American Rockwell Corporation

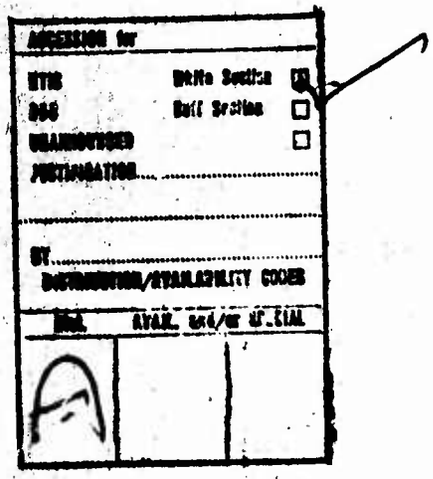
TECHNICAL REPORT SAMSO TR 73-69

February 1973

Headquarters Space and Missile Organization
Air Force Systems Command
Air Force Unit Post Office
Los Angeles, CA 90045

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.



Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author) Autonetics Division Rockwell International 3370 Miraloma Av, Anaheim, Ca 92803	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b GROUP None

3 REPORT TITLE

Software Validation Study Final Report

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Final Report. July 1972 to December 1972

5 AUTHOR(S) (First name, middle initial, last name)

D. Bruce Brosius

6 REPORT DATE February 1973	7a. TOTAL NO OF PAGES ~~230~~ 250	7b. NO OF REFS None
8a CONTRACT OR GRANT NO F04701-72-C-0370 b. PROJECT NO N/A c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) C73-11/201 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) SAMSO TR 73-99	

10 DISTRIBUTION STATEMENT

Qualified Requesters May Obtain Copies of This Report From DDC

11 SUPPLEMENTARY NOTES None	12 SPONSORING MILITARY ACTIVITY Deptartment of the Air Force, Headquarters Space and Missile Systems Organization (AFSC) AF Unit Post Office, Los Angeles, California 90045

13 ABSTRACT

The Software Validation Study was undertaken to provide an understanding and assessment of the process used to develop the software for a current ballistic missile system, the Minuteman III system, particularly the process of identifying software requirements and performing software testing. This information is intended to provide the foundation for development of advanced techniques to be applied to future software developments.

The development process identified as a result of the study exhibits two significant characteristics:

1. The requirements development activity is a continuous, iterative, evolutionary process dealing with broad conceptual requirements on one end and detailed programming level requirements on the other. Communication is the key to effective organization of this activity.
2. The software testing process lacks formalization in terms of identified objectives, test methods, test procedures, etc. This lack of structure and formalization represents a significant shortcoming in terms of test quality and efficiency.

DD FORM 1 NOV 1473

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
None						

SOFTWARE VALIDATION STUDY

FINAL REPORT

D. Bruce Brosius

FOREWORD

This report contains the descriptions and conclusions of the Software Validation Study authorized under Air Force Contract F04701-72-C-0370. The study was performed by the Autonetics Division of North American Rockwell Corporation, 3370 Miraloma Ave, Anaheim, California, for the Space and Missile Systems Organization (AFSC), Department of the Air Force. The Air Force program monitor for the study was Capt. Fergus Henderson, SAMSO/DYGS, Los Angeles, California.

Mr. D. Bruce Brosius, Autonetics Strategic Systems Engineering Department, performed the study during the time period of July 1972 through December 1972. He collected the files of many Autonetics engineers pertaining to the development activities of the Minuteman III Operational Ground and Flight Tape and analyzed these in detail. He also interviewed involved engineers as to their recollections of the events affecting the subject matter. No classified information was extracted from any of these sources. The resulting report has been disseminated internally at Autonetics under document number C73-11/201, "Software Validation Study Final Report."

This technical report has been reviewed and is approved.

Fergus Henderson

FERGUS HENDERSON, Capt, USAF
Project Officer, SAMSO/DYGS
Software Validation Study

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

SECTION I

INTRODUCTION

1.1 STUDY BACKGROUND

The motivation behind the Software Validation Study was concern over the effectiveness of the current approach to validation of software for spaceborne, ballistic, and avionic systems. The ability of the current procedures to meet the needs of future systems in terms of cost, development time, and software quality is in question. It was felt that an analysis of the "real world" procedures and problems evidenced by a current development would result in an objective assessment of the current state of the software development process. This information would provide the foundation for development of advanced validation techniques.

The objective of the Software Validation Study was to examine the software development process used on a current ballistic system development in order to identify the detailed process used and analyze its effectiveness. Of particular concern was the process of identifying/establishing software requirements and the methods used for software testing.

The Minuteman III Operational Ground and Flight software developed over the period 1967-1972 was chosen as the subject of the analysis. Several versions of the software were developed during this period, the most recent of which was the Wing VI Operational Ground Program, -11 revision (March 1971-March 1972). A significant change was made in the verification procedures for this program so its development was the primary source of information on the testing process. However, it was necessary to trace the requirements development process back to the early Minuteman III system development (1967).

1.2 STUDY APPROACH

The approach taken in conducting the study consisted of the following activities:

1. Data Collection and Evaluation

   The initial study activity was to collect, organize and evaluate the available data from which the development process could be reconstructed. Informal documentation consisting primarily of meeting minutes and internal letters together with more formal documentation of the verification process were the main source of written documentation. Due to the time period involved, and the lack of documentation in certain areas, considerable reliance was placed on the memories of personnel involved in the software development.

2. Identification and Description of Process Activities

   A survey of the collected data, software organization (company structures), published guidelines/procedures, and the opinions of involved personnel was conducted in order to identify and describe discrete activities evident in the Minuteman software development process. Activities were identified on the basis of specific disciplines or background required, organizational structure (Autonetics or overall Minuteman contracting), or method of assigning personnel.

In identifying/describing these process activities, a specific attempt was made to avoid preconceived ideas as to what functions were necessary to perform the software development. Any function, task, or activity which could be identified as a more or less discrete entity was included. In describing the activities, the following information was included: description of activity, product(s), relationship to software requirements development, relationship to software testing, primary interfaces, and pertinent comments.

3. Identification and Analysis of Program Examples

The method used to develop a thorough understanding of the development process involved detailed analysis of selected examples of specific software functions. It was felt that this approach of reconstructing the process from "the inside out" would result in a more accurate understanding and avoid errors due to overgeneralization or preconceived ideas. Two steps were involved in performing this analysis:

a. A tabulation was made of significant changes/problems/additions which occurred during the Minuteman III development process. Approximately 40 examples were identified and summarized from a requirements and testing standpoint.

b. Thirteen of the examples were selected for a more detailed analysis. The selection was based on an attempt to represent all the major program functional areas (IMU, Communications, Flight, etc) and/or particular significance in terms of requirements or testing (for instance, program errors which were not detected during software verification). The analysis consisted of tracing the particular software requirement/function through its development and evolution from initial requirements source to delivered program.

4. Process Description and Analysis

The data from the analysis of the program examples was used to refine and update the process activity descriptions and to identify the overall development process. This information was then used to construct the process description.

Analysis of the overall effectiveness of the identified process was conducted both as a part of the process identification activity and as a separate evaluation subsequent to the process description activity.

SECTION II

SUMMARY

2.1 APPLICABILITY

In using analysis of a specific software development to provide a summary of the development process in general, it is obviously critical that the particular development is generally representative of the current process and the future problems for the class of software required on ballistic, space, and avionic systems. In order to help relate the results of the study to other software developments, several significant characteristics of the Minuteman system/development are summarized below.

1. Organizational Structure

The Minuteman Weapon System development involves many different contractors and customer agencies. Though Autonetics is solely responsible for the onboard software, (as well as G&C hardware), this software interfaces with equipment/software from the following four contractors: Boeing, Sylvania - ground equipment and LCF software; General Electric - re-entry system; and TRW - targeting software. In addition, many customer agencies are involved in determining software requirements (TRW - guidance analysis and technical system engineering; SAMSO - direct customer; SAC - "user" command; NSA - system security.)

The organizational structure constrast drastically with a single customer/software contractor organization.

2. System Characteristics

The Minuteman software is part of an overall system development which involves many copies of the system (missiles) and continuous operation over an extended period of time.

This type of system contrasts significantly with a single or limited mission development. (A test flight or even space shot for instance).

3. Computer/Software Architecture

The onboard Minuteman computer is a complex, serial rotating memory organization. The software must be organized to utilize the memory rotation as the primary timing reference. The software is programmed in machine language and is severely constrained in terms of memory availability.

The primary effect of the memory constraints and serial organization is a lack of modularity in the software.

4. Contractor Orientation

The software development activity is heavily "system" oriented. Since Autonetics is also the G&C contractor for Minuteman, there is a strong "make the system work" (as opposed to "meet the contractual requirements") flavor to the software development. The attitude is further emphasized by the amount of total business which Minuteman represents to the software contractor. The "support the total program" attitude puts considerable pressure on supporting overall system schedules.

2.2 CONCLUSIONS

An assessment of the overall effectiveness of the Minuteman software development process must be prefaced by the statement that in general this process has supported the needs of a highly successful weapon system. In searching for shortcomings in the process, the primary motivation is to assess the adequacy of this process for future developments, presumably more complex and constrained. The major results of the current process assessment are summarized below.

2.2.1 Software Testing

1. The software testing process relies heavily on overall system testing, a luxury which may not be feasible on future developments.

2. The program testing process; i.e., specific "black box" testing of the software product, is relatively unstructured, undisciplined, immeasurable, and marginally adequate for even current needs.

3. Shortcomings in the software testing process result primarily from the lack of structure, failure to identify objectives, and lack of formalization of the process.

2.2.2 Requirements Development

1. The requirements development process is continuous from basic system concepts to detailed program structure; i.e., programming level details are as much "requirements" as system functions.

2. Communication and close interaction between the various disciplines involved (including programming) is the key to the effectiveness of the requirements development process.

3. With the exception of special analytical disciplines necessary, the requirements development process does not lend itself well to formalization.

4. Attempts to formalize the requirements process have resulted primarily in formalizing the documentation related to the process.

5. The primary shortcomings of the requirements development process are related to documentation, not the lack of it but primarily the inappropriateness of the particular documents for the functions they perform.

2.3 REPORT ORGANIZATION

The remainder of this report is divided into four sections and an appendix. Section III describes the background of the Minuteman Weapon System in order to provide perspective for the Minuteman III software development. Section IV contains the description of the Minuteman III software development process. Section V contains an analysis of the process identifying primary characteristics and evaluating overall effectiveness. Section VI contains definitions of the terms and acronyms used in this report. The appendix contains the detailed data from the analysis of specific examples of Minuteman software functions.

In general, this report assumes that the reader has some prior background and understanding of airborne software development.

SECTION III

MINUTEMAN SYSTEM BACKGROUND

This general description of the Minuteman Weapon System and its evolution is presented to furnish the background against which the Minuteman III Operational Ground and Flight software was developed. It is intended to show how such factors as hardware development, force deployment, test activities, operational philosophy, organizational interfaces, management policies and schedule environment influenced the development process of the Minuteman III Operational Tape Program.

3.1 MINUTEMAN WEAPON SYSTEM

3.1.1 Minuteman I Weapon System

The Minuteman I System was a solid propellant ICBM, developed as the primary strategic weapon system for the Air Force. Consisting of the flight missile and its ground support equipment, it was designated as the 133A Weapon System (WS133A).

The Flight Missile contained three stages of boost engines, an inertial guidance and control system and the re-entry vehicle. The missile was capable of delivering one warhead to one target with in-flight guidance limited to the boost phase of the flight, with limited target azimuth sector.

The Guidance and Control subsystem, housed in the trapezoidal shaped section in the fourth stage of the missile frame, consisted of the Inertial Measurement Unit (IMU), its digital computer and some of the flight control electronics. As its name implies, the IMU sensed missile motion along the axes of an inertial coordinate system and transmitted the information to the airborne computer. In the computer, the IMU signals were mathematically transformed into a launch centered earth fixed ballistic coordinate system to calculate the missile's position and velocity along the desired trajectory. The IMU was designated the NP10 Guidance and Control system.

The Digital Computer, called the D17, was a serial, rotating disk machine with a memory capacity of 2560 words. It also had the capability to respond to several kinds of special hard-wired signals from the ground support equipment. Besides solving the guidance problem, the computer processed the control signals for the engines and for stage separation. It also provided status checks of various elements of the airborne equipment.

The Ground Support Equipment controlled the pre-launch operations of the missile system, checked the status of the system and displayed that status. The first operational ground equipment was the C53P Coupler Console. It was used in conjunction with the C24 Targeting Console to control and align individual missiles in the silos of the Wing I Launch Control Facility (LCF). Wing I was the first of six such installations planned by the Air Force to deploy the Minuteman Weapon System. Figure 1 illustrates the composition of a wing.

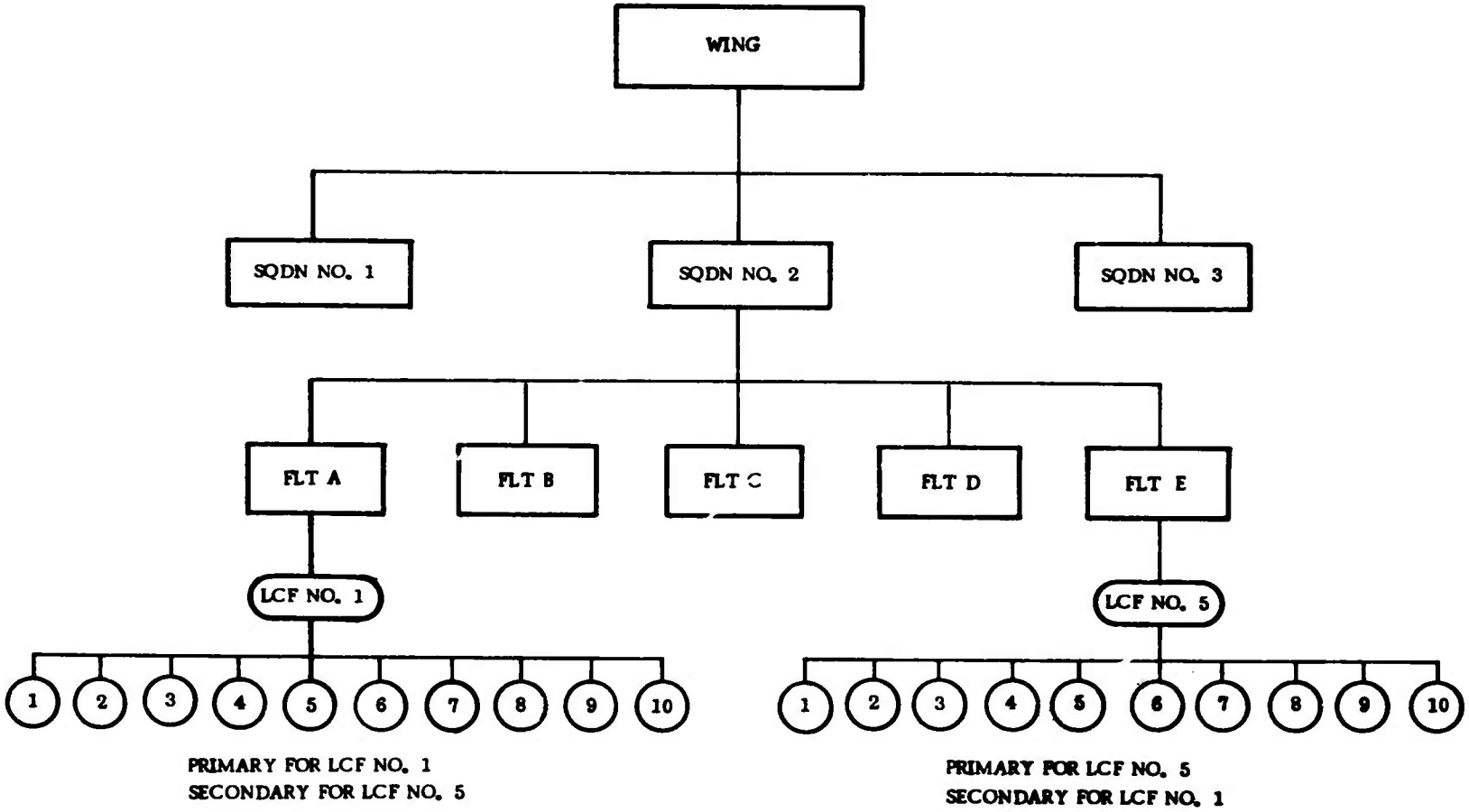


Figure 1. Typical Wing Organization

After Wing I was deployed, the C53P was redesigned to absorb the C24 and was called the C53D. The C53D ground equipment was developed for, and installed in the Launch Facilities (LF's) of Wing II through V. The LF's were each connected to the LCF with redundant underground cabling. The new equipment provided the capability of greater status checking of the missiles during pre-launch operations.

The R&D program for Minuteman I involved both the inplant test activities of several associate contractors and the flight test program at Cape Canaveral, Florida (presently Cape Kennedy or Easter Test Range (ETR)).

Various ground equipments were used for these tests, including the C18 and C19 consoles at ETR. The C18 and C19 performed many of the same functions as the C53D.

3.1.2 Minuteman II Weapon System History

Before Wing VI was deployed with the planned Minuteman I Weapon System, it was decided to develop an improved second-generation weapon system.

This second-generation system, called the WS133B, was developed to provide a hardness capability to withstand attack and to retaliate with a larger payload at a greater range, with 360^{0} target azimuth sector capability. The missile contained the same elements as the Minuteman I, but the second-stage engine was more powerful to accommodate the heavier re-entry vehicle. The movable attitude control nozzles were replaced with fixed valve type controls. The overall missile envelope was the same as the Minuteman I.

The attendant new development in the guidance and control subsystem resulted in the NS17 Missile Guidance Set (MGS). The NS17 incorporated more accurate accelerometers, a self-contained azimuth reference for IMU alignment and a new computer. It also utilized miniaturized solid state integrated circuits (IC's) in its electronics package. A new digital computer was developed for the NS17 employing the new IC's and a rotating disk memory. Designated as the D37C, it had a memory capacity of 7225 words. This greater memory provided more operational capability in flight and more WS status checking during ground operations. It also had the capability of accepting commands and transmitting status messages via cable and radio communications.

Following a different system design approach, the functions of status monitoring and command and control were transferred from the ground equipment to the D37. With this shift in operating philosophy, the airborne computer controlled much of the system pre-launch operations.

The WS133B operational ground equipment was redesigned to eliminate much of its status checking operation and to provide greater command and control capabilities in the new system. This equipment, more sophisticated and efficient, and utilizing the IC's in its construction, was designated the C163.

To apply the WS133B improvements to the Minuteman I force, it was necessary to embark on a force modernization (Force Mod) program for Wings I through V. The Force Mod program consisted of redesigning the WS133A equipment to accommodate a Minuteman II missile. The C53D operational ground equipment was redesigned to incorporate the new communications link and some of the C163 Command and Control capability. It was redesignated the C53E. Other additions and modifications have subsequently been made in both configurations of the Minuteman II systems. Among these were the Penetration Aids (PENAIDS) subsystem, the Data Transrecorder (DTR) equipment, and further G&C reliability improvements.

3.1.3 Minuteman III Weapon System

The third generation of the Minuteman system was developed to maximize the retaliatory capability of the deployed forces. Without increasing the number of deployed forces, this objective was achieved by giving the Weapon System greater capability to survive an attack and to subsequently launch more warheads per missile. To provide these capabilities required development of a more comprehensive circumvention system with PENAIDS and a system of Multiple Independent Re-entry Vehicles (MIRV) utilizing a Post-Boost Control System (PBCS) for the fourth stage of the missile. The PBCS consisted of the missile guidance set and the post-boost propulsion system (PBPS). A block diagram indicating the functions performed by the new PBPS is shown in Figure 2. The pictorial overview of the missile, shown in Figure 3, illustrates the relative position of the PBCS in the missile system.

The NS17 Missile Guidance Set (MGS) was redesigned to satisfy the post-boost control requirements of the Minuteman III mission. These involved maneuvering the PBCS to independently deploy the elements of the re-entry vehicle. Because of the emphasis on this operation, the early MGS development effort was referred to as the post-boost system development rather than Minuteman III. The MGS was later designated the NS20 Missile Guidance Set. A functional block diagram of the guidance set is shown in Figure 4 indicating the various functions performed by it.

In addition, a new third stage was developed to provide greater range for the system.

Components of the NS17 system were redesigned to give the NS20 more operational capability while making it more radiation resistant and accurate. This included a larger digital computer, radiation shields in the MGS housing, non-radiation sensitive electronics circuits, new alignment equipment and more powerful platform torquer motor.

The redesigned computer, designated the D37D, doubled the size of the D37C computer memory to 14,137 words and increased the number of input and output discretes to 110 and 74 respectively. It still utilized the serial, rotating disk memory organization. A comparison of the parameters of the old and the new computer is shown in Figure 5. The new computer also employed radiation resistant electronics and radiation shields over its housing.

Since the NS20 missile guidance was to be used in both Force Mod and Wing VI sites, attendant redesign to the operational ground equipments was required. Both the C53 and the C163 were modified to interface with the D37D Computer and to incorporate the hardened (radiation resistant) electronics. They also accommodated new modes of system operation as part of the circumvention capability of the Weapon System.

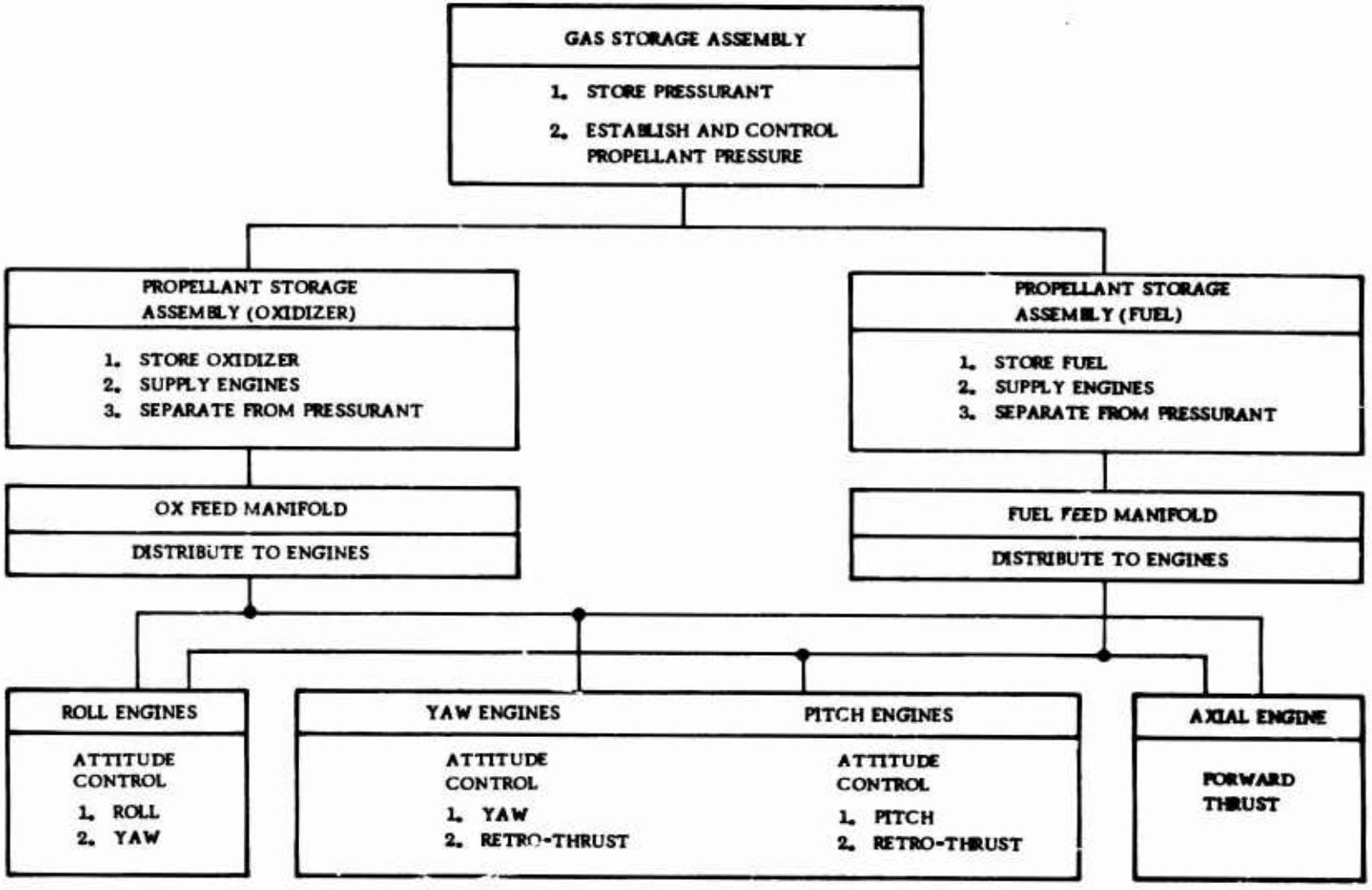


Figure 2. PBPS Functional Block Diagram

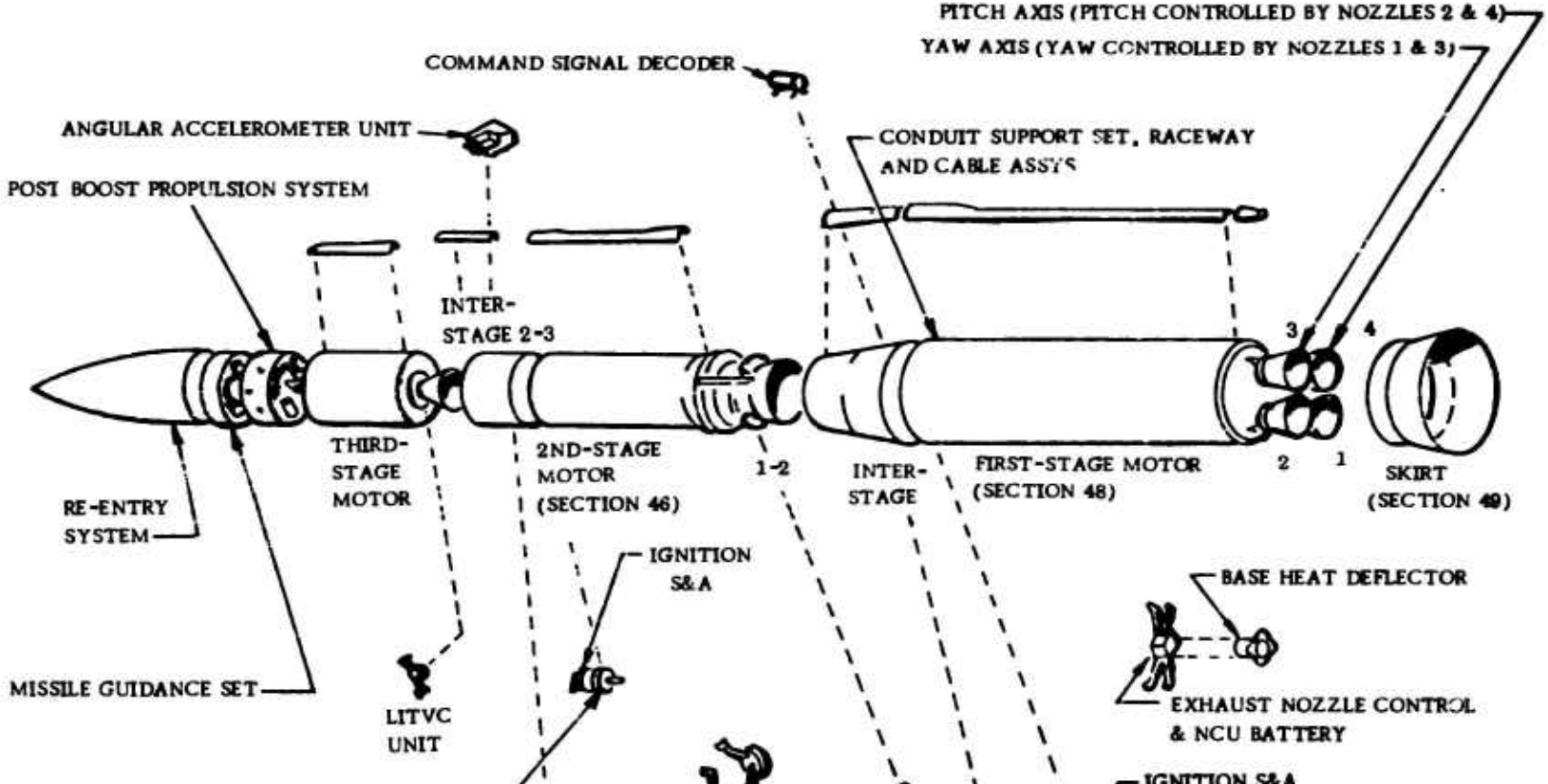


Figure 3. Minuteman Missile Exploded Conceptional View

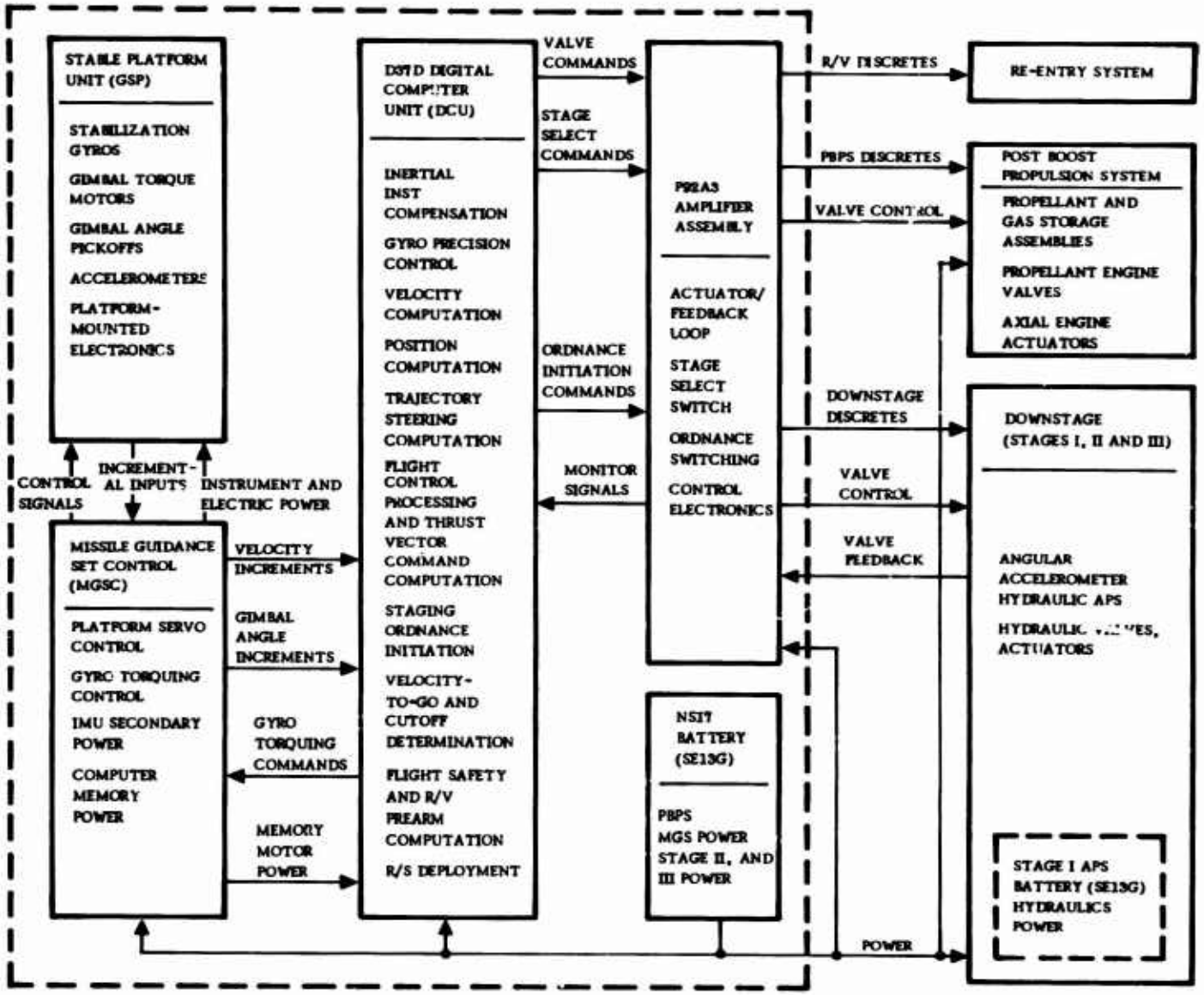


Figure 4. G&C Functional Block Diagram

	MM III D37C	MM III D37D
SIZE	20.9 X 6.9 X 9.5 IN.	22.3 X 7.4 X 11.0 IN.
WEIGHT	39.9 LB	42.0 LB
VOLUME	0.43	0.60 CU FT
POWER (COMPUTER)	315 WATTS	402 WATTS
SELF-CONTAINED POWER SUPPLIES	YES	YES
COOLANT	LIQUID	LIQUID
PARTS COUNT	6475	7086
MEMORY	DOUBLE HEAD PLATE	DOUBLE HEAD PLATE
MEMORY WORDS	7225	14,137
HOT AND SHORT LOOPS	313	569
COLD	6912	13,568
LOGIC STRUCTURE	NAND	NAND
DIVIDE COMMAND	1 BIT ON LINE	1 BIT OFF LINE
MULTIPLE COMMAND	2 BIT ON LINE	3 BIT OFF LINE
LOW LEVEL READ SWITCHING	YES	YES
ELECTRONIC MEMORY LOOP ADJ.	YES	YES
CONDITIONAL FILL	YES	YES
GYRO TORQUING	INDEPENDENT	INDEPENDENT
RESOLVER RATES	3 AT 6400 PPS 4 AT 3200 PPS	3 AT 6400 PPS 4 AT 3200 PPS

	MM II D37C	MM III D37D
INPUTS		
DISCRETES	72	110
PROGRAM (LOAD)	8	8
INCREMENTAL	7 PAIRS	7 PAIRS
DC VOLTAGE	32	32
COMMUNICATIONS	6	5
EXTERNAL CONTROL	6	5
DISPLAY CONTROL	10	11
OUTPUTS		
TELEMETRY	29	31
COMMUNICATIONS	3	3
MONITOR	20	20
DISPLAY	1 PAIR	1 PAIR
SINGLE CHARACTER	7	7
BINARY INCREMENTAL	4 PAIR	4 PAIR
DISCRETES	47	74
DC VOLTAGE	4 (4 LINES EA)	4 (4 LINES EA)

Figure 5. D37C/D37D Comparison

The Minuteman III was developed on an accelerated schedule which necessitated flight testing of the various new subsystems as they became available. To define these R&D versions of the system and what was contained in each, four increments of system development were identified. These configurations were called Minuteman III Block I, II, III and IV.

3.2 MINUTEMAN SOFTWARE

The Minuteman Software evolution must be described from both the procedural and the technical aspects of computer program development. Generally speaking, the process by which tapes (software programs) were developed became more formalized as the weapon system matured; and technically, they became more complex and sophisticated. Many factors caused changes both in the way programs were developed and their functional content. And, as time passed, more people and separate engineering facilities were dedicated specifically to software development.

The software development process was affected by such things as the number of people involved and how they were organized, customer policy decisions, the sensitive nature of the weapon system, the types and locations of test activities, and the weapon system master implementation schedule. The sources of program requirements increased and the communications concerning them became more difficult and formal, as more people from more organizations became involved. The Air Force established the policy early, that operational tape programs would be treated as hardware, and their configuration controlled accordingly. This decision was made because strict control was needed to insure weapon system security and to guard against an unauthorized/inadvertent launch.

The types of testing at the various locations forced a priority system for developing certain portions of the program. Those functions required by factory and experimental engineering tests were developed first, next, those requirements for other in-plant and off-site tests, then the flight test functions and finally, the operational functions. These activities were not completely serial since several groups of engineers were involved. This order of development quite naturally follows the master development schedule for the weapon system with information gathered from earlier activities being integrated later. Since the test activities are widely scattered across the United States, supplying documented programs to them and integrating the test results affected the development process.

The technical evolution of the Minuteman software, predictably, paralleled the Weapon System development. Each redesign and innovation in the hardware usually brought about corresponding change activity in the tape programs. Often, tape changes were made to avoid hardware modifications, i.e., as "workarounds".

Technical development was also affected by the factors discussed in the procedural evolution, especially test activities and schedules.

The development history of the Minuteman operational tapes is summarized in Figure 6. Each family of operational programs is depicted by a bar superimposed on a time scale. Major revisions and deliveries are indicated by arrows placed at the appropriate points in time on the bar. The dash numbers represent the configuration part number of the tape.

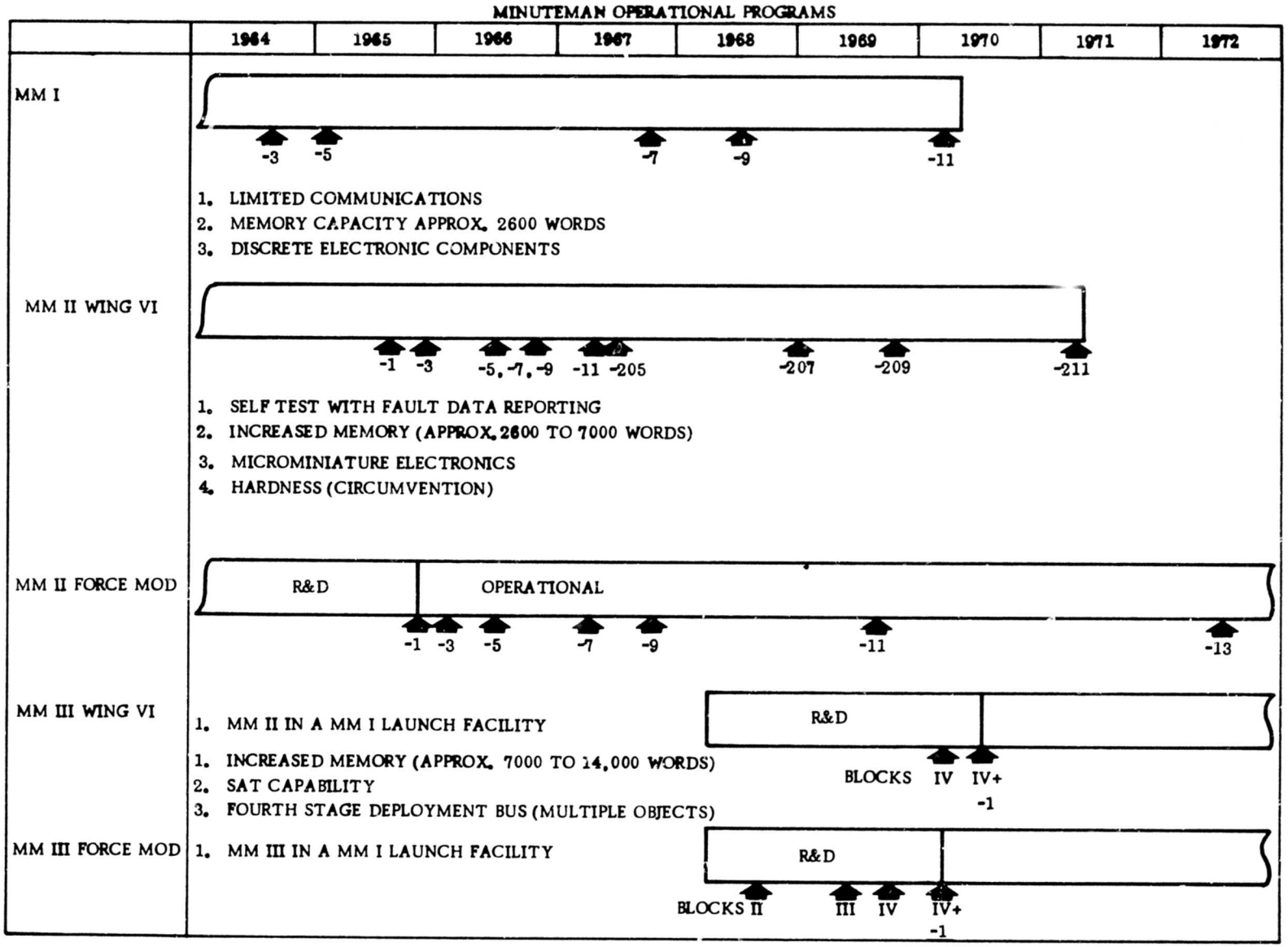


Figure 6. Historical Summary

3.2.1 Minuteman I Software

In the early days of the R&D program for the Minuteman I G&C system, a comparatively small number of people were involved in developing the tape programs. They were primarily the engineers and programmers assigned to develop specific subsystems as reflected by the organizational lines of the company. The navigation engineers were responsible for the IMU functions; flight control personnel for the downstage control and flight tape; the ground equipment developers handled the system status checking; and the system assembly engineers developed the integration and interface software. Although each group worked fairly independently, they used what was available from the other groups and applied it to their activity.

This method of operation took advantage of the technical experience gained in designing the hardware to develop the software. It also allowed close-knit informal working relationships with considerable person-to-person contact. This minimized the amount of formal communication required. Often a requirement or mechanization was unilaterally decided upon within the group and implemented. Customer involvement in the early development was very limited.

The Minuteman I R&D activities began in 1957 and continued for approximately three years. The factory G&C Subsystem Acceptance Test Programs were developed and then modified for various other in-plant engineering test activities. This exercising of the various functions of the tapes had the effect of additional checkout. From the in-house design proof testing to integration testing to associate contractor compatibility testing, many basic operational requirements and mechanizations were developed and refined. Because of differences in interfacing test equipment and test emphasis, numerous variations to the basic mechanizations were programmed. The final phase of the R&D effort was the flight test activities at Cape Canaveral, Florida and Vandenberg Air Force Base in California. The D17 Flight Test Programs were written to be used with the C18 and C19 ground equipment at Cape Canaveral. The system to be flown, the ground equipment and the software were tested for compatibility in the Autonetics engineering laboratory. Upon completion of the compatibility tests, the whole installation and the people involved were moved to Cape Canaveral to support the flight.

The Vandenberg AFB flight tests were conducted using both operational ground equipment and flight missiles. The tape for this application was the operational program which interfaced with the C53P equipment. These Assembly and Checkout (A&CO) tests were the final confirmation of the complete hardware/software system prior to operational deployment.

The advent of the Minuteman I operational tape brought about formalizing of the software interfaces within Autonetics and with the Air Force. The Air Force was concerned about the control of tapes since they governed the operation of the entire weapon system. Their concern resulted in the policy decision to classify them as hardware contract end items. This decision placed them under the same configuration control provisions as the rest of the weapon system. It also influenced the software development process more than any other decision or policy in the Minuteman program.

To meet the requirements of the hardware configuration control system, the Air Force directed Autonetics to establish administrative procedures and appropriate forms of software documentation. Autonetics was also directed to create an office to coordinate and manage tape development. Figure 7 is a sample PERT type chart showing the flow of software activities under the resultant system. With few additions and variations, this system has been used for all subsequent Minuteman operational tape development. The preparation of the formal documents and the flow time to obtain their approval lengthened the development process time by several months.

The technical development of the Minuteman I programs was accomplished, to a large extent, by code checkout at the programmer's desk and then laboratory checkout on an actual system. Programming aids were limited to the machine language assembler. Limited flight simulation was accomplished for the flight program.

3.2.2 Minuteman II Software

The Minuteman II software development followed the same general course as the Minuteman I software. With the Minuteman II R&D program beginning in 1962, mechanizations were developed to test the basic functions of subsystems; used with additional control functions for inplant system testing; modified and iterated upon for flight testing and finally, refined for operational use. Procedural differences were noted in the relationship of the organizations performing these activities and the documentation which they produced. Both were more formal, with more people involved at each level.

Separate organizations were established to manage the software development effort, to generate software requirements, to develop the programs, to operate software checkout laboratory sites and to conduct full-time flight test operations at both the Eastern Test Range (ETR) and the Western Test Range (WTR). Special disciplines were also established such as flight safety engineering, security software developers and targeting specialists. The need for communications increased considerably; Figure 8 indicates the complexities of the organizational interfaces.

The introduction of the new WS133B hardware caused the establishment of a new family of operational tapes. Since their first application was at Wing VI with the WS133B hardware, they became known as the Wing VI Tapes.

The principal Minuteman II software developments were made to implement the new D37C Computer, the C163 ground equipment and the new IMU instruments (accelerometers and gyrocompass). The D37C memory had three times the capacity of the Minuteman I Computer, allowing many more system functions to be accomplished and controlled by the operational program. The added input-output capability of the new airborne computer caused many of the system status-checking functions to be transferred from the ground equipment to the operational tape.

The greater command and control capability of the C163 ground equipment along with the D37 I/O characteristics (including radio communications) required completely new communications processing functions in the programs.

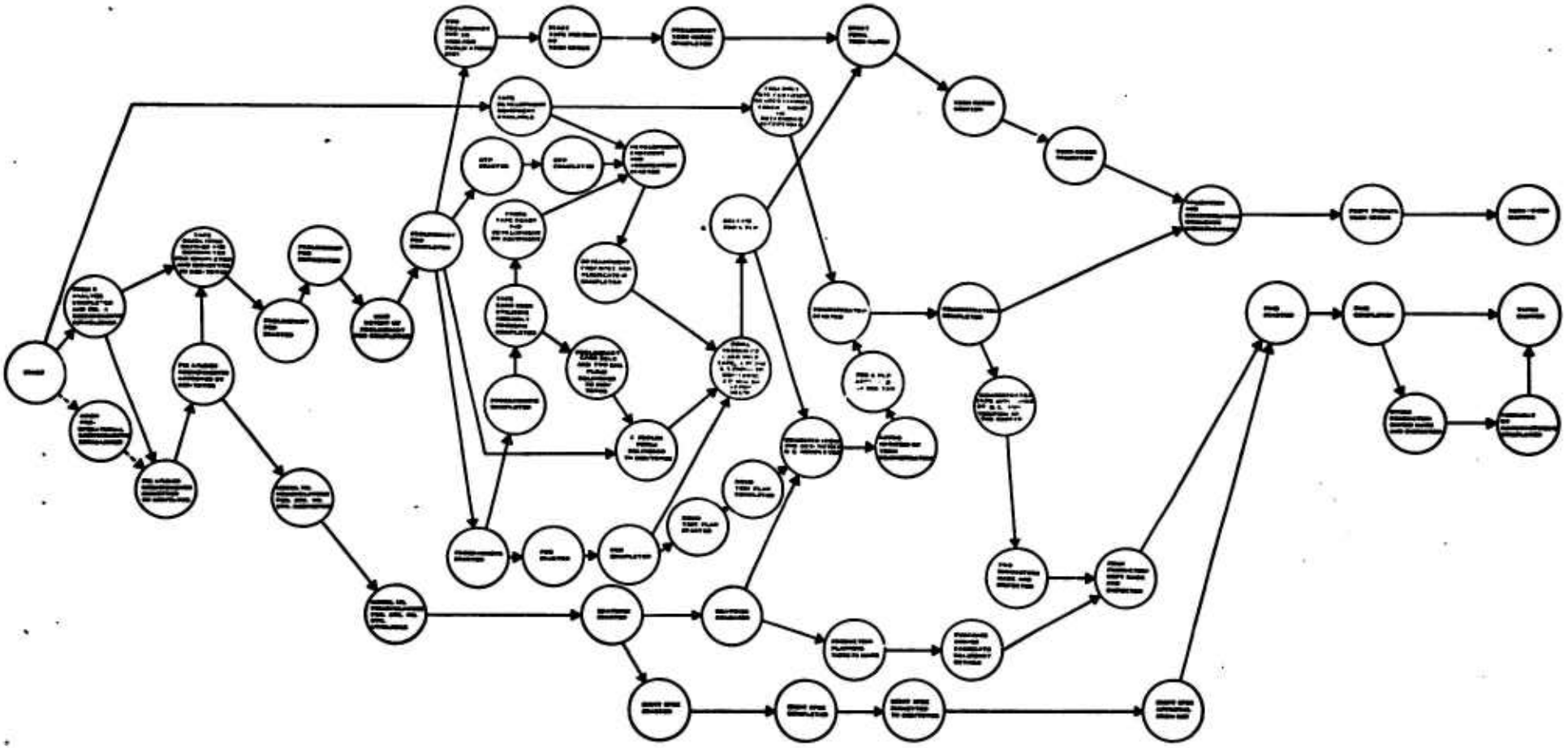


Figure 7. Typical Hardware End Item Tape Flow Chart

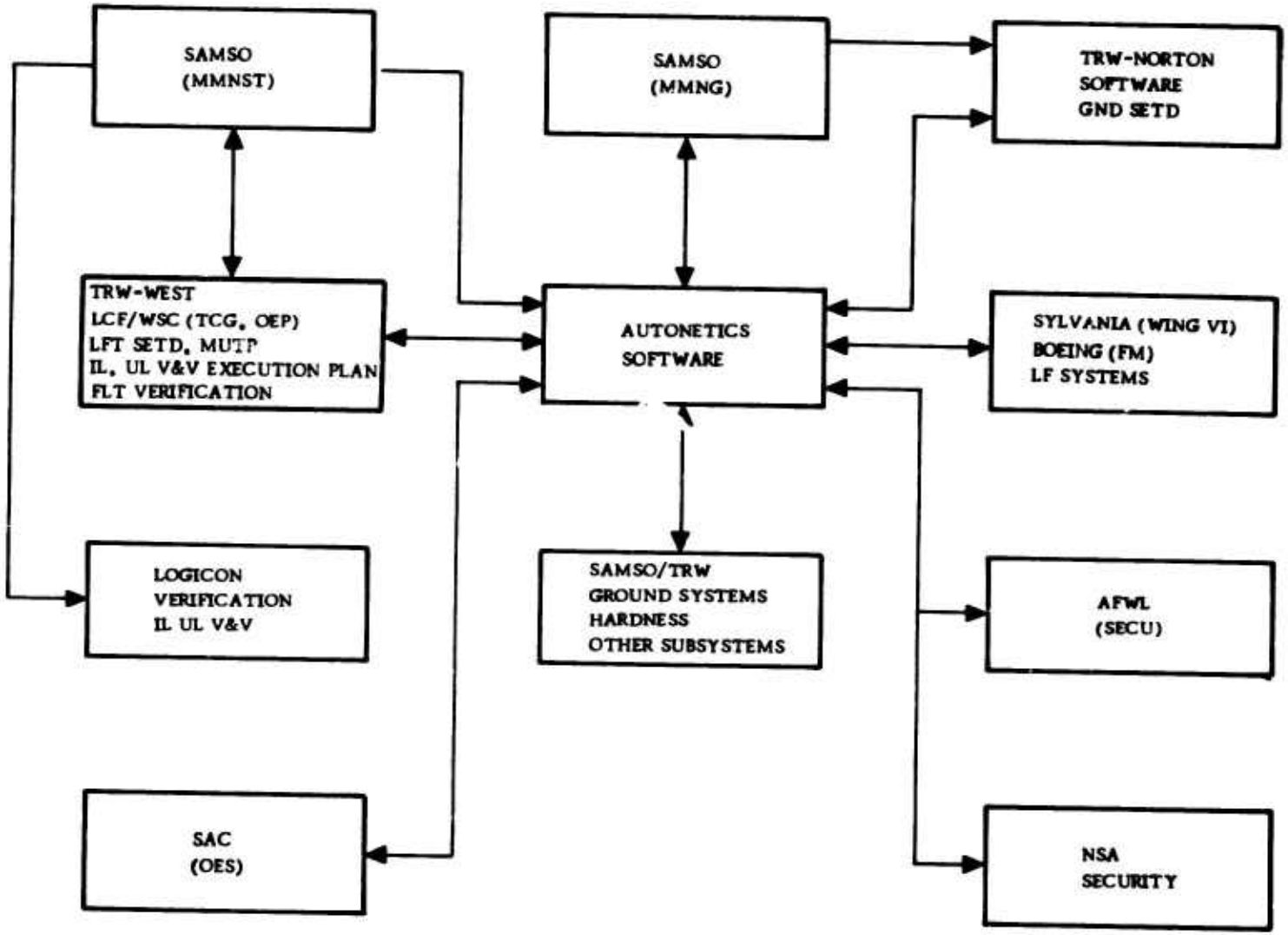


Figure 8. Typical Minuteman II Organizational Interfaces

The NS17 IMU contained new accelerometers (PIGA's) and a new Gyrocompass Alignment set (GCA). New mechanizations were developed to calibrate the PIGA's and to process their outputs for vehicular velocity. System azimuth alignment was remechanized since the integrally mounted GCA gave the MGS a self-contained azimuth reference, which was used as a backup for the external reference, an autocollimator.

The Force Mod operational tapes incorporated the Wing VI IMU mechanizations into the WS133A-M environment. The portions of the program dealing with system status checking, message processing and the radio communications were developed to accommodate the D37/C53E interface.

Both families of Minuteman II tapes were revised later on several occasions to accommodate such weapon system additives as the Penetration Aids (PENAIDS) subsystem, seismic and radiation environment requirements, the MGS reliability improvements and the Data Transrecorder (DTR) equipment.

The advent of the Minuteman III R&D program in 1966 literally created a Minuteman software "explosion." The explosion more than doubled the number of people, organizations and computer programs required to support the widespread weapon system activities. With hundreds of Minuteman I and Minuteman II systems still actively operational, and extensive tape development being performed to improve the reliability of Minuteman II, Minuteman III struck with full force. Not only were computer programs needed for the regular factory, engineering and flight tests, but the accelerated operational deployment of Minuteman III dictated additional versions of tapes to support added test locations. It forced development of the Minuteman III operational tapes concurrently with the many R&D flight tapes needed for wide variations of flight test missions. New subsystems were flight tested as they became available, resulting in tapes being built for each block of R&D hardware. These were identified as Blocks I, II, III and IV.

The situation was further aggravated by the addition of new radiation environmental testing and the expansion of programming aids to facilitate development. All of these factors combined to cause the software explosion.

The effects of the explosion on software development were apparent in: (a) the specialization of personnel and organizations to specific families of programs; (b) the utilization of various off-site hardware test sites to checkout software as well as hardware; (c) the expanded use of software and hardware simulators for program checkout; (d) the changing of control procedures to allow delivery of tapes on a less-formal basis; and (e) naturally a great increase in the number of personnel involved.

The Minuteman programming organization at Autonetics was increased. Units were formed to specialize in the development of Minuteman II operational ground flight programs, Minuteman III experimental and operational ground programs; Minuteman II/III experimental and operational flight programs, Minuteman II/III factory test programs and programming aids/tape verification. These units were organized into teams to work specific families of tapes, i.e., C18/C19, Minuteman III FM, Minuteman III Wing VI, etc.

The in-house Autonetics engineering laboratory test facilities experienced a similar reorganization and expansion. Operational hardware sites for several configurations of Minuteman II and Minuteman III equipment were established specifically to checkout software. The system experimental test sites were converted to support Minuteman III flight testing, and the System Simulation Laboratory was expanded from the limited capability of the Minuteman II Direct Simulation Laboratory.

The role of the off-site test facilities was altered/modified by the software development situation. During Minuteman II development, the associate contractors, doing weapon system integration and other system level hardware testing, received formally validated and demonstrated tapes to conduct their tests. This allowed them to concentrate on hardware problems and give less consideration to software problems. The Minuteman III situation required them to use early cuts of tapes with deficiencies. In essence, more off-site checkout was accomplished than had previously occurred. The same climate existed with the Eastern and Western Test Ranges during their early ground testing activities.

Although the nature of flight tests and launch facility operation require strict control of the tape programs, the stress of the Minuteman III development schedule necessitated some changes to the configuration control procedures of hardware tapes. Time consuming administrative procedures along with peripheral documentation specified for hardware tapes were streamlined or deleted. New classes of tapes delivered to these revised administrative requirements were identified as prototype and interim operational tapes. The prototype and interim operational tapes were handled less rigidly paperwise, but no degradation in technical and functional integrity was allowed.

This approach allowed many versions of flight tapes to be delivered to support the blocks of R&D systems. Several tapes within a particular configuration block were delivered. It also allowed timely delivery of the initial version of the Minuteman III operational ground tape.

The expansion of the System Simulation Laboratory (SSL) and the development of several software simulators were required because of the software explosion. Simulations were developed to speed up tape checkout as well as to test mechanizations that previously could not be exercised. Mathematical models of the missile system in flight and during ground operation were refined and used extensively in program checkout.

The same families of operational tapes were developed for Minuteman III as had been used in Minuteman II. Both FM and Wing VI operational programs incorporated many weapon system improvements made possible by doubling the memory. The mechanizations tested and flown for each block of R&D hardware were analyzed for incorporation into the operational programs.

To allow concurrent development of both the flight and ground operational programs by independent units, each program was kept as an entity. The interface between these programs in terms of memory usage had to be tightly controlled. Each program used approximately half of the D37D memory of almost 14,000 words. The flight portion increased in size by 300 percent to accommodate the new re-entry system requirements.

More customer involvement at all phases of program development has been the general trend. It became especially noticeable on Minuteman III in the early phases of requirements definition and program design. The verification by an agency other than Autonetics became established practice. This additional phase of development was started in a very limited sense on Minuteman II to provide an added measure of security against unauthorized/inadvertent launch and flight safety. It was expanded on Minuteman III to include other aspects of the program.

SECTION IV

PROCESS DESCRIPTION

The primary purpose of the study was to develop a detailed understanding of the process involved in developing the Minuteman III Operational Ground and Flight Software. Particular emphasis was placed on the process of establishing software requirements and performing software testing. This section describes the development process which was identified. Figure 9 shows the time frame and main program revisions involved in the subject development.

4.1 GENERAL

Figure 10 is a conceptual diagram of the overall development process, showing the primary flow and identifying the major activities involved. It illustrates the classic development phases; i.e., requirements development, program development (design and coding), and program testing. Activities which, while involved or related to the software development process, were not in the "main stream" of requirements development, program development or direct program testing are omitted from the diagram in order to emphasize the primary activities. These peripheral activities are described in Section 4.2 and referenced where appropriate in subsequent sections. Notice that particularly in the requirements development phase, the specific activities are somewhat unique to the Minuteman system/organization though they are representative of similar activities on other systems.

Size of the bubbles on this diagram represent a highly subjective indication of the relative significance of the activities and the positioning is a rough indication of the overlap of activities. Note the extreme overlap of the requirements and program development phases. The three phases will be described in detail in subsequent sections.

4.2 PROCESS ACTIVITIES

The following paragraphs describe activities which were identified in the Minuteman III software development process. Each of these activities was directly identifiable in the process, either because of its specific purpose/function or because of the way the process was organized. As previously indicated, the process of identifying these activities involved surveying the organizational structure of the software development, the task assignments of personnel, the stated approach and techniques, and the generally recognized functions involved in the Minuteman III software development. Any activity which either appeared somewhat distinct or was thought of/or organized as a distinct function was included in the list. The activities are limited to those in which Autonetics participated, though some mention is made of activities performed by other Minuteman contractors. In identifying these activities, preconceived ideas of what functions are necessary for software development or how software development should be organized were specifically avoided.

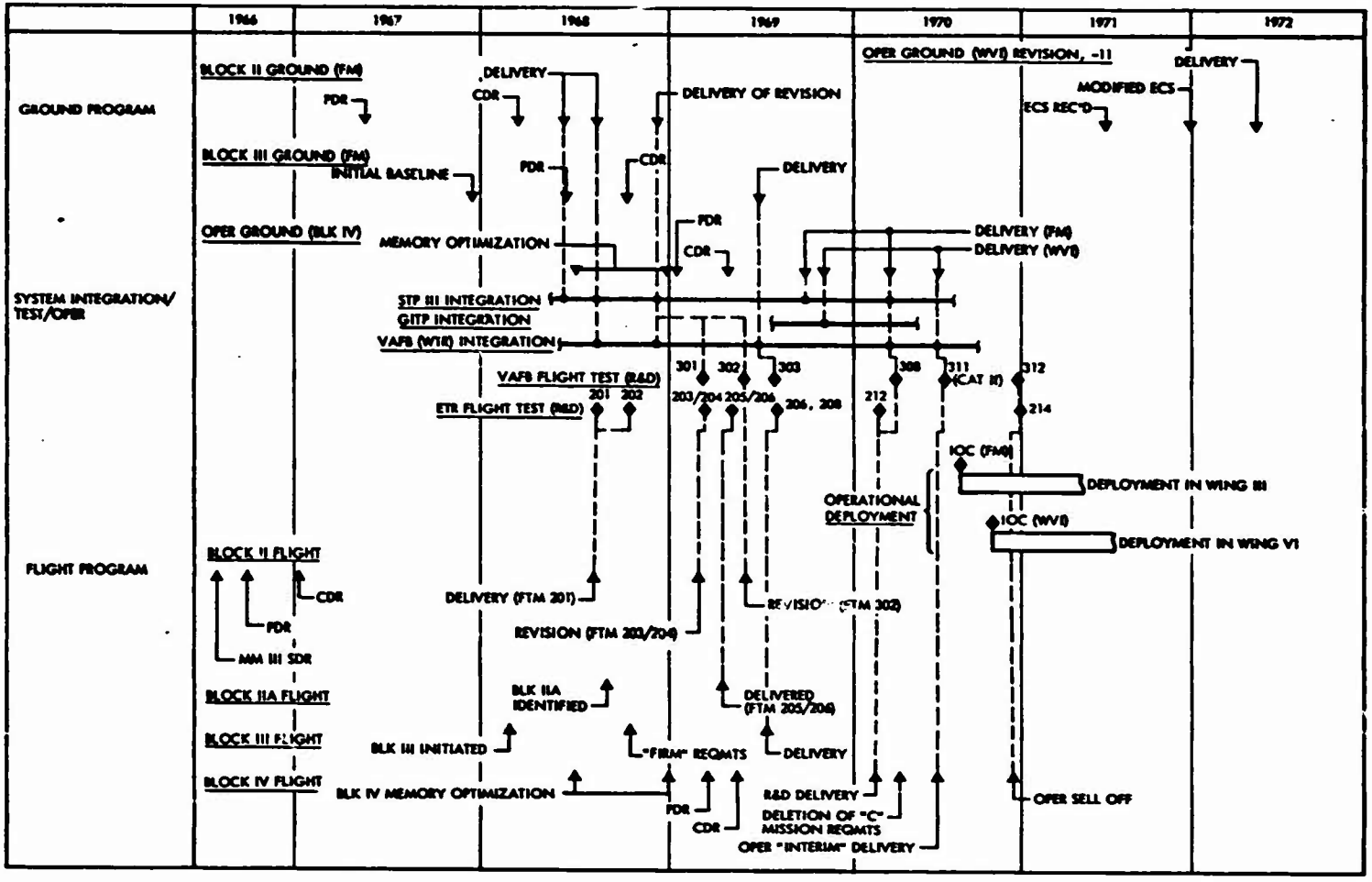
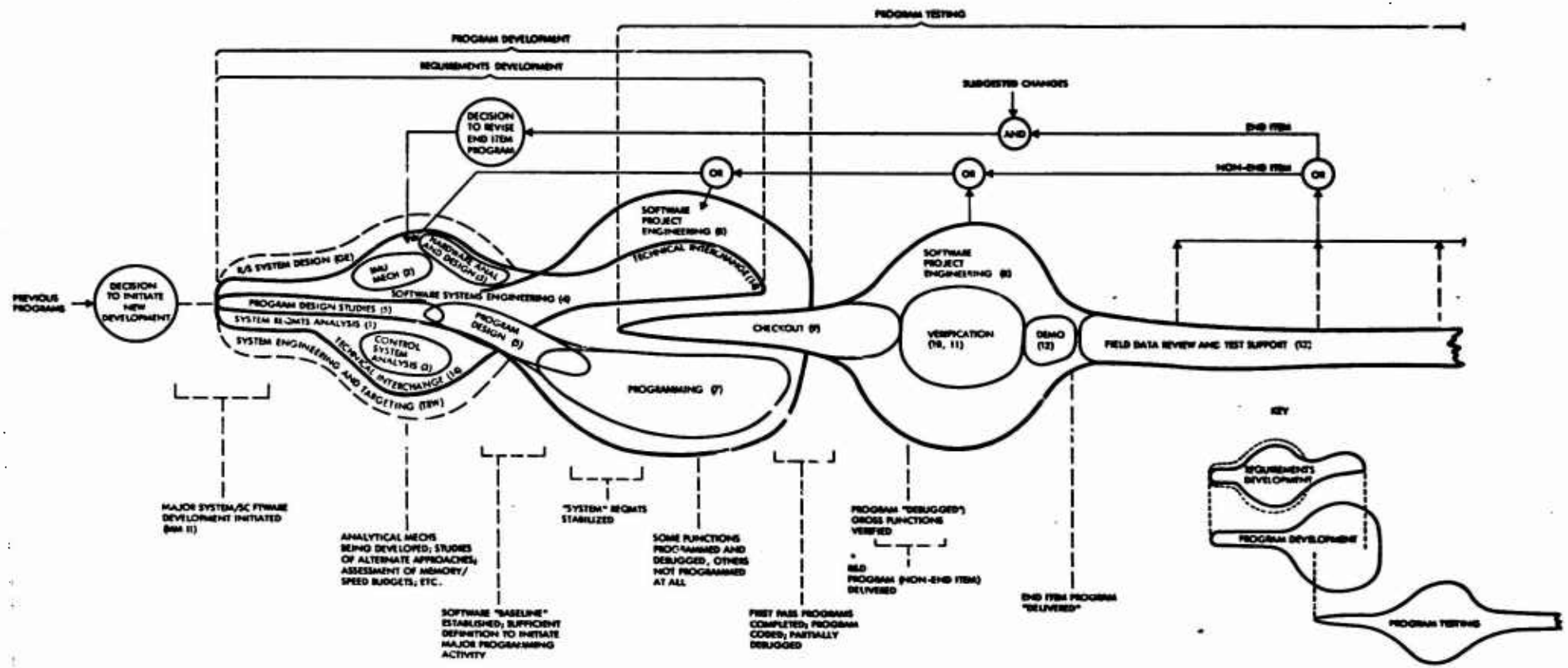


Figure 9. Minuteman III Development Schedule



NOTE: NUMBERS IN PARENTHESIS REFER TO ACTIVITY INDEX IN SECTION 4.2

Figure 10. Primary Software Development Flow

The activities are described individually and the following information is provided: description, products, relationship to requirements development and software testing, primary interfaces, and comments. References to these activities will appear throughout the report, but for convenience, the descriptions are all included in this section. No attempt is made in this section to relate all of these activities. The interrelationship of activities and their function in the overall process will be discussed in subsequent sections of the report.

4.2.1 Index to Activities

4.2.2 Activity Descriptions

ACTIVITY: Item 1, System Requirements Analysis (SRA)

DESCRIPTION:

This is a formalized process for proceeding from a system concept to detailed requirements for hardware/software end items and interfaces which was conducted for the Minuteman III System (as an update of the Minuteman II SRA). The process involves primarily developing functional flows of the total system with an accompanying description for each block of the flow. The flows start from the most basic functional requirements and work downward; necessary end items (hardware and software) are identified to implement the functional requirements and the requirements are partitioned among the end items. Interfaces between end items are also generally identified, but design and documentation of specific interfaces is not a part of the SRA process though some of the same personnel may be involved.

PRODUCTS:

Functional Flows, Forms B (narrative describing requirements for each block of flows).

SOFTWARE REQUIREMENTS:

Since the Minuteman III SRA was an update of the Minuteman ' baseline, it was in some areas, a documentation rather than a requirements development effort. However, ground program requirements related to interfaces or the logic/procedures of system operation were in many cases developed as a result of this activity. Requirements related to IMU mechanization and flight were generally not developed as a part of this activity but were documented (after the fact) in the SRA data package which includes the Figure A, Part I (Part I software spec.).

SOFTWARE TESTING:

No direct relationship. It seems reasonable that analysis and "testing" of system/software requirements would occur during this activity but no specific examples were identified.

INTERFACES:

This activity is conducted jointly by all Minuteman Associate Contractors and TRW. The documentation is published and maintained by Autonetics, but the flows and narrative are iterated on a block-by-block and word-by-word basis at review meetings.

COMMENTS:

This activity appears to be effective in the organization and initial design of a large, complex system; particularly in identifying "black" boxes, interfaces, and system modes/functions. At some point, after initial system development and operation, the "front end" portion of this activity; i.e., updating the flows and narrative, is normally discontinued and the Part I end item specs become the highest requirements documentation. This point occurred in late 1968 on the Minuteman III system.

ACTIVITY: Item 2, IMU Mechanization Development

DESCRIPTION:

This activity involves developing the software mechanizations for alignment, instrument biasing, calibration, leveling, torquing, etc., of the Inertial Measurement Unit (IMU). Since Autonetics designs and manufactures the IMU (including the instruments) this activity was an integral part of the IMU system development. The basic software mechanization had to be developed and programmed for in-plant IMU testing and normal factory operation. Since there was considerable commonality between Minuteman II and Minuteman III IMU functions, the Minuteman II mechanizations as reflected in Minuteman II factory support software became the baseline for the Minuteman III development. This activity preceded the ground tape development and hence provided a test bed and test data to develop the initial requirements for the ground software. Later in the evolution of the ground program this activity tended to become largely separated from factory operation and became a software system engineering activity for IMU-related functions.

PRODUCTS:

Equations, logic, and parameters associated with IMU mechanizations; technical direction relative to IMU software functions.

SOFTWARE REQUIREMENTS:

The then-current Minuteman III factory support software became the IMU baseline requirements for the initial Minuteman III factory software. These requirements were not published as such but the programmers were told to develop a program for the D37D computer "like the _____ version" of the Minuteman II (D37C) program. This factory program in turn became the baseline requirements for the IMU mechanizations in the initial Minuteman III ground program. Documentation of ground program requirements for the IMU functions was strictly after the fact. The programmers got "requirements" from the listings of factory programs and Minuteman II ground programs with some support by IMU system engineers. As the Minuteman III system development progressed, a software system engineering function/organization became established which was generally responsible for subsequent modifications and additions to ground program requirements in the area of IMU functions.

SOFTWARE TESTING:

The factory support programs provide significant testing of the software mechanizations. Also, statistical/parametric data derived from factory operation is used in deriving IMU mechanization requirements for the operational programs.

INTERFACES:

IMU factory/factory support personnel, IMU design personnel, programmers.

COMMENTS:

This activity showed an interesting (and apparently representative) trend in terms of organizational history. During initial Minuteman III development, the IMU software mechanizations were the responsibility of an IMU (hardware) system organization with no direct tie to the software development organization. This organization was primarily devoted to engineering in support of design and manufacture of the IMU.

During later Minuteman III development (subsequent to initial IMU factory start-up) this activity shifted both organizationally and in terms of emphasis, to a "software" system engineering function whose primary concern was with IMU software mechanizations in operational programs. To a large extent, the same personnel were still performing the activity. See further discussion under Software System Engineering.

ACTIVITY: Item 3, Control System Analysis

DESCRIPTION:

Through systems analyses and simulations, flight control system equations and programming requirements were developed for the Minuteman III systems. The control equations included commands for pitch, yaw, and roll loops, digital compensations, gain changes, control logic and switching functions, control interfacing with the steering scheme used, and other associated control functions.

The programming requirements also included required computational delays, accuracy of computations, maximum expected values for control parameters, values for fixed control gains or constants, and other associated programming requirements for the flight control system.

System analyses and simulations were also performed to verify adequacy of the flight program mechanization. System analyses verified stability margin requirements for the flight control system. Hybrid simulation trajectory studies using the actual airborne Minuteman computer, including mechanizations from the flight program when available, were performed to verify compliance of the flight control system with systems requirements and design criteria. These simulations included system nonlinearities and verified satisfactory control system stability and performance for a nominal system and expected off-nominal system parameters.

PRODUCTS:

Internal letters documenting flight control programming requirements.

SOFTWARE REQUIREMENTS:

The flight control program requirements for the early flight programs were supplied in a relatively informal manner (internal letters) to the programming area. Later in the flight program evolution, these requirements (or what these requirements had become) were incorporated in the design criteria and the Part I software specification.

SOFTWARE TESTING:

During the flight control system analysis, a hybrid simulation using the Minuteman airborne computer is utilized. In order to perform this simulation analysis, the flight control equations must be programmed on this computer. When convenient/possible the "current" version of the flight control program was used in this simulation. However, since much of the simulation must be performed prior to establishing programming requirements, the flight program development generally lagged this simulation activity sufficiently to make it difficult to use the actual flight program.

Simulation was also performed to verify control system performance of the "operational" flight program as a part of the formal software verification.

INTERFACES:

TRW - (redundant control system analysis and guidance interfaces), programmers.

ACTIVITY: Item 4, Software Systems Engineering

DESCRIPTION:

Common to most usage of the term "system engineering", this is a difficult activity to describe. Generally speaking, this is a continuing activity concerned with developing/documenting software requirements, resolving reported system anomalies which may or may not be software related, participating in TI meetings to discuss system software requirements/changes/problems, and interpreting system requirements/subsystem requirements/hardware characteristics in terms of software requirements. The disciplines involved vary considerably from straight documentation to mechanization development to circuit analysis.

Generally, this activity is subdivided according to functional areas of the operational programs. For example, a given person may work primarily flight program-related activities, another IMU-related activities, and a third, missile test activities. However, a single person may be assigned to coordinate all the changes for a particular program revision.

PRODUCTS:

Figure A, Part I software specs; technical direction relative to software mechanizations.

SOFTWARE REQUIREMENTS:

This activity is directly involved in the day-to-day requirements evolution process. Programmers generally rely on this activity to resolve problems/conflicts/incompatibilities related to software requirements.* However, depending on the personalities, disciplines, schedule, etc., the people who actually perform the system engineering activity may vary. In fact, it is not unusual for the programmers themselves to perform the system engineering activity directly once they become experienced with the system and mechanizations.

SOFTWARE TESTING:

This activity is not directly related to software testing except in coordination (and in some cases performance) of "tests" to determine feasibility of implementing suggested mechanizations. Note that the same personnel are involved in test planning for the ground program verification activity.

INTERFACES:

TRW and other associate contractors (coordination of system/software requirements), hardware design areas in Autonetics (hardware characteristics/performance), programmers (software requirements).

*See COMMENT

COMMENTS:

There has been considerable discussion, change, and variation in organizational structure with regard to this activity. The most significant change which occurred during the Minuteman III development was a major company reorganization.

During initial Minuteman III development, all "systems" activity was the responsibility of a lead division (the Minuteman Division). This responsibility included customer interface as well as all system requirements development, including software requirements. The actual software development, was the responsibility of the Navigation Systems Division, a "product" division which manufactured the Minuteman IMU.

Within this structure, a distinct area in the systems division was assigned responsibility for publishing Minuteman software requirements. However, in the area of IMU-related requirements for instance, the actual source of the mechanizations (which were evolving more or less continuously) was in the product division. Therefore, the task of software system engineering for IMU functions was for all practical purposes performed in a hardware related area in the product division and the systems division organization documented requirements after the fact. This situation was also true for flight program requirements since the guidance mechanizations were developed by TRW, the control mechanizations by another Autonetics product division (the Data Systems Division), and the guidance error analysis by an Autonetic's support division (Research and Engineering Division). On the other hand, functions related to external (to Autonetics) interfaces, ground equipment, test functions, system modes, etc. (essentially non-IMU and non-flight) were primarily developed in the systems organization.

It should be recognized that, while this organizational structure caused some problems for software development it was effective in some other aspects of the Minuteman program.

Approximately 1969, the software related organization was completely restructured and combined within one division. In general, the personnel who had been actually performing the software system engineering functions within the various divisions were transferred into a central organization which had total software responsibility.

ACTIVITY: Item 5, Hardware Analysis and Design

DESCRIPTION:

This activity as related to software development involved the hardware design/analysis disciplines applied to development of the D37D computer, IMU, PBPS, silo ground equipment, and flight control electronics (P92). As related to the software development, this activity involves design/identification of hardware operating characteristics (timing, sequencing, etc.), test conditions/sequences, interface characteristics, and specific mechanizations necessary to operate the hardware within the overall system. On Minuteman III, the primary software related activity was centered around functional/test requirements for the PBPS and new GCA mechanizations for the IMU.

PRODUCTS:

Reference documentation; hardware specs; technical assistance.

REQUIREMENTS:

This activity is the direct source of software requirements related to hardware operation, test and interface. Many of these requirements are implicit; i.e., software must function properly with the hardware. Others represent specific constraints, particularly timing sequences, which are directly imposed on the software. Test requirements were generally not fully developed as a result of this activity but rather as a part of the Software System Engineering activity (see separate description); however, provisions for testing contained in the hardware design obviously influence detailed test requirements.

TESTING:

Software support to hardware development activities provides early means of testing hardware/software mechanizations and potentially even actual software routines. This was particularly evident with IMU functions and to a lesser extent D37D factory support provided early familiarization with computer operation.

INTERFACE:

Software system engineers, programmers.

COMMENTS:

a. In areas where hardware functional operation is not straightforward, such as IMU operation, the initial testing (and use) of software mechanization is almost indistinguishable from the software requirements development activity. Data from operation/test of hardware/software functions is essential to verify and refine the mechanizations; hence, software requirements. This activity is not a result of poor or inadequate previous analysis, but a necessary supplement to achieve an optimum hardware/software mechanization. To a considerable extent this activity continued during the system level integration and testing of the Minuteman III system.

b. An important role of the software system engineering activity is to bridge the gap between hardware design/development and software disciplines; i.e., between hardware designers/subsystem engineers and programmers. Performing the role requires an understanding of both disciplines and some knowledge of detail in both areas in order to identify meaningful tradeoffs and achieve a reasonable "system" solution. This is a highly creative function and therefore its performance varies considerably between personnel assigned to the task.

ACTIVITY: Item 6, Program Design

DESCRIPTION:

This activity involves determination of the software structure and program organization necessary to implement specified system/software functions. In particular, this activity involved designing the executive structure of the program partitioning system functions into program organization, and analyzing the timing relationship of the detailed program functions as well as the overall program flow.

The overall design for the Minuteman III software was carried over from Minuteman II with the primary new activity occurring in the area of post-boost flight functions. During early Minuteman III system design, this activity involved studies to assess the feasibility and impact (memory and computer execution required) of implementing various proposed mecahnizations. Later in the software development, this activity involved primarily "fitting" new or modified functions into the existing software structure.

PRODUCTS:

Program organization (flow and timing structure); program memory and speed budgets; Level I/II flow charts. *

REQUIREMENTS:

a. This activity is a "user" of system-level requirements, but for new functions it is directly involved in the process of deriving/selecting specific mechanization details. The process of establishing the detailed programming requirements involves an iterative process in which mechanizations are proposed, program designs/impacts determined, tradeoffs made, and a compromise mechanization generally selected. Later in the system development, as the program/system became more firmly established, the framework in which this iterative design process occurred grew much smaller, but it is still evident.

b. Often a translation of software requirements occurs during this process. Requirements identified at a functional "system" level must be converted/expanded/modified to adapt them to implementation in the software. In the process of making this translation, errors, inconsistencies, or oversights may be discovered in the "requirements" which are then modified to reflect the problems. The result of this activity is really the software "requirements" identified/described at a more detailed, programming level.

*Three levels of program flow charts were prepared for the Minuteman III software. Level I is a one page, overall program flow. Level II shows all program modes and functions. Level III shows detailed program decisions/flow.

TESTING:

Program design can be considered a form of software testing in that it evaluates the feasibility and desirability of the chosen mechanizations. The primary area of concern in this type of testing is the program impact in terms of available memory and execution speed. However, assessment of numerical precision and selection of numerical techniques also provides a "test" of the analytical adequacy of particular mechanizations.

INTERFACE:

Software system engineers; hardware designers.

COMMENTS:

a. The distinction between this activity and the Programming activity (see separate description) is somewhat arbitrary. The primary significance of drawing a line between the two activities is that for a new program (or new functions of an old program) considerable program design activity is specifically involved in the process of determining system/software mechanizations. Much of the design activity does not flow directly into the programming activity and hence, appears as a relatively distinct function.

b. The unique program timing constraints imposed by the serial rotating memory architecture of the D37D computer adds another dimension to the program design (and programming) activity which is not apparent in the general case. However, these constraints are more a matter of degree since program timing is a significant design problem regardless of the computer organization.

ACTIVITY: Item 7, Programming

DESCRIPTION:

This activity involves the actual determination of the computer instructions which comprise the program. As defined here, this activity includes only what is normally called "coding". The program design and checkout activities which are often included under the term programming, are defined separately.

The computer instruction sequence is determined by utilizing a knowledge of the computer/interface hardware operation to implement the mechanizations resulting from the program design and requirements identification activities.

PRODUCTS:

Computer program (card deck of computer instructions, data, comments, etc.); Level III flow charts.*

*Three levels of program flow charts were prepared for the Minuteman III software. Level I is a one page, overall program flow. Level II shows all program modes and functions. Level III shows detailed program decision/flow.

REQUIREMENTS:

This activity is the ultimate "user" of software requirements. Omissions or oversights in requirements, as stated or as understood by the programmer(s) are filled in at this point either by explicitly identifying and resolving them, or by implicitly completing the mechanization as a result of constructing the program. Inaccuracies or errors in stated or understood requirements may or may not be recognized by the programmer and hence corrected prior to software testing.

TESTING:

This activity represents a type of testing of the requirements in that mechanization problems are sometimes identified as a result of the examination necessary to produce the coded program.

INTERFACE:

Software system engineers.

COMMENTS:

See comments under Program Design.

ACTIVITY: Item 8, Software Project Engineering

DESCRIPTION:

The software project engineering activity involves primarily the task of coordination. As currently organized (see Comments) this activity does not include monitoring budget and expenditures. The coordination task involves the following:

a. Coordination and preparation of development milestone charts which reflect interchange agreements, support to field activities, TXO site and SSL allocation, etc.

b. Coordinate, conduct, and document interchange meetings, briefings, and design reviews.

c. Monitor status of software development activities. Coordinate and report problems to management and other involved organizations. Expedite resolution of problems.

d. Coordinate and direct the administrative functions required during the software development. These involve such things as processing, submittal, and obtaining approval of identification specifications, nomenclature requests, SAC numbers, Federal Stock Numbers, FAI data packages, etc.

e. Coordinate and perform the administrative functions necessary for program demonstrations including TXO site availability and certification, notification of Customer, expediting problems, etc.

The software project engineer generally acts as a point of contact within the software development organization.

PRODUCTS:

Milestone charts; status reports; briefing; minutes of meetings; coordination letters; various customer notifications.

REQUIREMENTS:

This activity involves some coordination of software requirements particularly scheduling and identifying unique requirements to support various test activities. However, this would not be considered a significant activity in terms of establishing software requirements. In coordinating and documenting technical interchange meetings, the project engineering activity plays a role in documenting and communicating software requirements (see Comments).

TESTING:

Software project engineering plays a role in coordinating and expediting problems associated with testing activities.

COMMENTS:

a. The project engineering activity for software seems to differ somewhat from other project engineering functions. The difference is primarily in the degree of "total system" involvement and amount of coordination required on a project which is essentially development of a "subsystem". The software product has such a great degree of total system implication that the software project engineering activity requires more interfaces and concern with overall system considerations than a typical hardware subsystem.

b. The software project engineering function underwent a significant organizational change during the Minuteman III development. In part, this change was the result of a major company reorganization which combined the Minuteman software development activities into one division where previously the activity had been spread between a management/system division and two "product" divisions. (See Comments under software system engineering). Prior to the reorganization; the project engineering organization(s) was distinct from the line organizations and consisted of a central software project office in the management division, Minuteman Division, which had overall project authority for Minuteman software (including dollars). Each product division in turn had its own project engineering organization which directed the relevant activities within each division. Under this organization, the Minuteman Division Project Office was the central point of contact for customer, TRW, and associate contractor interchange. The lines of communication within Autonetics were obviously fairly long and considerable conflict, power struggle, loss of communication, etc., were in evidence. Under the current organizational structure, established in 1970, the software project engineering activity is a line function and the software project engineers report to the software development management. It seems to be generally agreed that this is a much more effective and efficient organization particularly in terms of the technical software development activities.

INTERFACES:

Software line management; programmers; software system engineers; test engineers; logistics personnel; SAMSO project offices; TRW project/technical personnel.

ACTIVITY: Item 9, Program Checkout

DESCRIPTION:

This activity includes all testing performed (or at least directed) by the programmers to satisfy themselves, the responsible programming engineers a and the supervisor, that the program is sufficiently free from errors to warrant proceeding with program verification. The steps involved in this process are as follows:(1)

a. Coded program segment is assembled. (A machine language assembler is used which executes on the central data processing system.)(2)

b. Program Listing is examined to check for errors detected by the assembler.

c. Listing is compared with coding flow chart; branches are traced manually to verify coding.

d. Flow charts are analyzed to determine functional paths in program.

e. Determine means of forcing execution of functional path on DSIM ("Stand alone" computer simulator), SSL (hybrid simulation lab), or hardware TXO site. Decide on tests to be run.

f. Conduct test(s). If expected response is not received, select one of the following procedures:

(1) Rerun test.
(2) Run similar test on program known to be "checked out".
(3) Study program listing/flow charts.
(4) Attempt to verify subportions of the program path being tested by devising additional tests.
(5) Seek assistance from another programmer.
(6) Record pertinent data, and consult with system engineer, programmer, or analyst.
(7) Proceed to another unrelated test.

PRODUCTS:

Final product is a program considered by the programmer(s) to be "ready" for verification.

(1) See COMMENTS

(2) Generally the ground program is assembled all together because of the tight memory interleaving required.

SOFTWARE REQUIREMENTS:

This activity is more oriented to verifying that the program structure is consistent, implemented as planned, and generally meets the desired performance requirements rather than toward explicitly verifying conformance to all requirements.

SOFTWARE TESTING:

This is the first testing activity conducted directly on the software product. The extent of the testing is not well defined and varies between programmers, programs, and program functions.

INTERFACES:

System Engineers and analysts to resolve requirement/mechanization problems, TXO site and SSL personnel to assist in conducting tests.

COMMENTS:

a. This process varies considerably from programmer to programmer and from program function to program function, but it varies primarily in the sequence of steps and extent of effort expended in each step.

   The point which this process/testing is considered completed is not well defined. Generally, it is at the discretion of the programmers or programming responsible engineer/supervisor to decide when "sufficient" checkout has been performed.

b. This process is not documented to any extent and no explicit identification of the tests performed is used in planning subsequent testing.

ACTIVITY: Item 10, Ground Program Verification

DESCRIPTION:

This was the primary means of ensuring that the software product satisfied its specific requirements. The following steps were involved in this process:

a. A Program Requirements Document (PRD) was prepared. This was done to establish a single, consistent reference for the verification such that a one-for-one correspondence could be established between requirements and testing.

b. Tests were planned to verify each of the PRD requirements. Test construction involved a combination of system performance observation (functional) and program mechanization analysis/testing (structural) at the discretion of each test planner.

c. Test plans were reviewed by another person for traceability and completeness.

d. Tests were performed on the Hardware Tape Checkout (TXO) site and System Simulation Laboratory (SSL). The results were recorded and analyzed by the test planner.

e. When errors were discovered, they were analyzed by the test planner and the programmer and appropriate changes made to either the program or the test plan or both. If the program was changed, the test planner and programmer would determine what tests to re-run based on the portion(s) of the program which were changed.

PRODUCTS:

Detailed Test Plan (DTP), PRD, Verification Test Summary Report, Test Data.

SOFTWARE REQUIREMENTS:

The inherent philosophy behind this method of program testing is the ability to identify and document all program requirements since the PRD is used as the reference for determining test requirements.

SOFTWARE TESTING:

This activity is intended as the primary means of formal testing of the software product as a separate entity.

INTERFACES:

Programmers; Site Personnel.

COMMENTS:

a. The real-time nature of the program causes potential interaction between functions (requirements) which do not necessarily appear explicitly in the PRD description of the requirements for either function. The requirement to perform a given function during specific modes of system operation implicitly requires that other functions being performed "at the same time" do not interfere with the function. Planning tests to explicitly verify each requirement does not guarantee verification of these "implicit" relationships. In planning the tests for program verifications, mechanizations were analyzed to select potential "danger points" from a function interaction standpoint. Tests were constructed to cause interrupts, mode changes, etc., at these points. This is, however, a rather subjective process and does not lend itself well to evaluation and measurement.

b. This process underwent a significant change during the Minuteman III development. See Section 4.5 for discussion.

ACTIVITY: Item 11, Flight Program Verification

DESCRIPTION:

The verification activity for the Minuteman III flight program(s) consisted of "flying" various trajectories and mission configurations using the MFS simulator. This verification was conducted almost entirely by the flight programmers.

For R&D flights*, a separate flight program was generally delivered for each flight. Verification of these programs consisted of simulated flights using the actual trajectory to be flown as defined by a targeting tape supplied by TRW. Both "nominal" (various instrument biases/nonlinearities and missile dependent characteristics set to zero or nominal case), and perturbed (instrument biases, hot or cold thrust profiles, variation in missile mass properties, etc. introduced) flights were generally simulated. The extent of the activity for each flight was at the discretion of the programmers.

Verification of the operational* flight program was somewhat more formal. A set of five trajectories/missions were selected by Autonetics, TRW, and SAMSO to be used to verify the flight program. These trajectories/missions were specifically designed to exercise flight conditions, mission options, trajectory shapes, etc., which "covered" the operational envelope. Various perturbations on such things as center of gravity, thrust profiles and winds were also introduced. An attempt was made to introduce these perturbations at points in the flights where flight conditions would tend to accentuate any adverse effects of the perturbed conditions. Targeting data for these five missions was developed by TRW, specifically to support the flight tape verification.

PRODUCTS:

Simulation Data Package (listings, plots, tabs, etc.).

REQUIREMENTS:

As a part of the verification of the operational flight program, a Test Requirements Cross Reference Index (TRCRI, see separate description) was prepared and used to provide assurance to Quality Assurance Personnel that the software requirements were met. However in general, specific tests were not designed to demonstrate conformance to each requirement. Rather, data from the five simulated flights was referenced to "demonstrate" conformance to various requirements.

*Only the operational flight program, i.e., the software which is used for operational deployment of the weapon system, was treated as a "hardware" end item on the Minuteman III system. The R&D flight programs were treated as engineering tools and hence did not have formal demonstration/documentation/sell-off requirements.

TESTING:

The verification activity is the primary testing of the software product as a separate item. In conducting this testing, the main criteria for success is stability during simulated missile flight, execution of mission events (staging, thrust segments, attitude maneuvers, etc.) and impact accuracies. In some cases, data is compared with similar simulation data obtained from TRW's targeting simulations, but this tends to be done only to debug problems or as a gross check when a new function is first programmed. The inherent philosophy of this type of testing is the verification of software functions, implicitly through observation of system performance during "normal" operation. This approach is somewhat different than is used for ground program verification where in most cases an attempt is made to explicitly verify each specified program requirement through individual tests.

INTERFACES:

TRW (Targeting data/problems), software system engineers.

COMMENTS:

The implicit verification technique used for the Minuteman III flight software is a natural choice, particularly since this philosophy was carried over from Minuteman I. The Minuteman I flight mission was straightforward and hence the flight program flow was reasonably "straight line". For this type of program, a given operational "mission" tends to exercise a very high percentage of the program functions and hence provides a reasonably good verification tool.

The post-boost mission of the Minuteman III system, requiring deployment of multiple vehicles with attendant maneuvers, thrust segments, engine inhibit/delay logic, etc., requires considerably more complex software from the standpoint of program decision points, flow paths, and mission configurations/alternates. This added program complexity makes it much more difficult to select a test mission(s) which will exercise all aspects of the software operation. Additionally, it becomes more difficult to determine the correctness of particular software functions from the simulation data. These problems were generally overcome by some selective explicit testing of software functions, particularly when the function was first programmed.

ACTIVITY: Item 12, Program Demonstration

DESCRIPTION:

All operational software (ground or flight) and software designed for test activities at VAFB, except R&D flight tests, are controlled as "hardware" end items and require a sequence of three formal demonstrations as a prerequisite to customer acceptance. These demonstrations are witnessed by Autonetics Quality Assurance (ANQA), Air Force Quality Assurance (AFQA), and a customer team (SAMSO, TRW, SAC, etc.), respectively. Prior to the ANQA demonstration, an engineering demonstration is conducted to insure that the software and procedures are ready for sell-off. The test procedures to be

included in the demonstration are published in a Demonstration Test Plan which is approved by SAMSO/TRW prior to the demonstrations. These tests procedures are, in most cases, a duplication of tests run previously during the verification process and, in the case of the ground program, are selected to include the "normal" operating modes/functions of the program but to not exceed 16 hours of demonstration time. The flight program demonstration consists primarily of several flights simulated in the SSL using TRW-generated targeting tapes.

The "Team" demonstration is somewhat flexible in content at the discretion of SAMSO. For ground programs, a special team demonstration is normally jointly agreed upon between SAMSO, TRW, and Autonetics. This demonstration is usually shorter than the previous demonstrations and contains tests of the specific functions which were changed or added to the particular program being delivered as well as tests of interface compatibility requirements. Part of the formal acceptance of the software subsequent to team demonstration is a review of all applicable documentation including data from the software verification activity. In terms of witnessing proper operation of the software, this review is fairly significant since the amount of testing included in the demonstrations is obviously limited.

PRODUCTS:

Formal technical acceptance of software product.

REQUIREMENTS:

No direct relationship.

TESTING:

As indicated, the purpose of the demonstration process is to exhibit operation of the software rather than to provide additional verification of its performance. However, it has not been uncommon to discover software problems during demonstrations.

INTERFACE:

ANQA, AFQA, site personnel, programmers, system engineers, customer representatives.

COMMENTS:

a. It would appear that the benefit of demonstration in terms of software testing is that it forces a formal, "externally" controlled procedure to be run and checked on a step-by-step basis. This formality seems to partially overcome carelessness induced by schedule pressures, human nature, etc. On the other hand, the demonstration by itself is obviously no guarantee of software correctness and examination of the verification process and the data from that process would appear to be the only reasonable technique for assessing the quality of the software product being delivered.

b. By contract, all equipment used for demonstration of a software end item must be "certified" unless deviation has been arranged. This certification is intended to establish and control the configuration of the demonstration facility and to insure that it is appropriate for the item being demonstrated. The benefit of this certification procedure, beyond the configuration control aspects, has been somewhat questionable.

ACTIVITY: Item 13, Field Data Review and Test Support

DESCRIPTION:

Data from system integration, test, training, and operational activities was available from several sources at various stages of the Minuteman III software development. These sources are summarized below:

a. Eastern Test Range (ETR) - An R&D test program was conducted at ETR to integrate and flight test the airborne hardware and software. This test program was conducted by Autonetics test engineering personnel. During ground integration and test and preparation for flight test, communication between the test engineers and the programmers/system engineers was established on a daily basis (by telephone) to resolve problems and provide information.

b. Seattle Test Program, Part III (STP III) - The Boeing Company conducts a test program in Seattle as a part of their role as integration and communication contractor for the Minuteman system.* This activity involves integration and testing of the Force Mod Minuteman ground system, particularly the communications network between Launch Control Facilities (LCF) and Launch Facilities (LF).

   Autonetics' test engineers participate in the STP III activities on-site. Anomalies and/or potential software problems are normally communicated by telephone with the programmers/software system engineers, but more formal reports of STP III activities are also issued by Boeing.

c. Ground Integration Test Program (GITP) - Sylvania conducts a test program at Waltham, Massachusetts for the Wing VI configuration of the Minuteman III system. The primary purpose of this activity is the integration and test of communications network between the LCF and LF.

   This activity is similar to STP III and is similarly supported by on-site Autonetics test engineers.

*The Boeing Company is the integration contractor for the Minuteman system and also developed the communications equipment for the Force Mod configuration. Sylvania developed the communications equipment for the Wing VI configuration.

d. Vandenberg Air Force Base (VAFB) - Various integration and system level test activities were conducted on the Minuteman III system at VAFB (also called Western Test Range, WTR). These activities represent the bulk of the system testing and include R&D flight tests, Cat I/II flight tests, operational training launches, and Assembly & Checkout (A&CO) activities.

   VAFB activities are supported on-site by Autonetics test engineers, and the process of identifying and resolving software questions or problems is much the same as described previously. However, the VAFB test activities generally occurred subsequent to ETR and STP III/GITP and, except for the R&D flights, represent more formal testing of "operationally configured" equipment (hardware and software). Therefore, fewer problems are uncovered, but the problems that are discovered have a more significant impact on the development process.

e. Operational Minuteman Wings - Once deployment of the Minuteman III system in operational SAC wings was begun, data on actual field performance became available. Analysis of systems removed from operation for maintenance is conducted by Autonetics Quality Assurance personnel and/or Reliability Engineering personnel.

   Anomalies or problems in system operation in the field are generally reported through the Autonetics field engineers (part of the Logistics organization) who are stationed at the various operational wings. However, depending on the significance or urgency of the situation, reports may come from SAMSO, SAC, or TRW personnel also. The field engineers submit regular reports which may suggest changes to the software or hardware, or request investigation of some aspect of the system operation.

   As the "user" of the system, SAC personnel suggest/request modifications or additions to system capabilities or procedures. These comments, suggestions, or requests may be reported by the field engineers, forwarded to SAMSO/TRW, or introduced at subsequent design reviews.

PRODUCTS:

Test Data, problem reports, suggested changes, etc.

REQUIREMENTS:

a. Some unique requirements are imposed on the software to support system test activities. For example, "simulated flight" sequences may be included in the ground program to exercise interface signals used in flight and special perturbations may be added to the R&D flight programs to evaluate system performance during non-nominal operation.

b. Problems or shortcomings in requirements/mechanizations are sometimes discovered during system level testing and/or field operation.

c. The additional "exposure" of the system in terms of additional people/agencies using it tends to generate suggested modifications, corrections, or improvements to the software mechanizations.

TESTING:

The activities described are all forms of system level testing. This testing is not generally designed to specifically test the software as such, but to verify system (including software) operation.

INTERFACE:

Test engineers, programmers, software system engineers, project engineers, etc.

COMMENTS:

Extensive testing is performed to integrate, verify, and deploy the Minuteman Weapon System. It is outside the scope of this study to extensively analyze the activities in terms of their effectiveness in software testing. It should be noted, that regardless of this effectiveness, the cost in terms of testing resources and re-development expense, of discovering software errors at this level of testing is considerable.

ACTIVITY: Item 14, Technical Interchange Meetings

DESCRIPTION:

TI Meetings were held on a frequent, though not necessarily regularly scheduled basis. The meetings vary in formality, attendance, documentation, etc., but the primary attendees were Autonetics, TRW, and SAMSO. Other associate contractors would attend when specific interfaces or activities were to be discussed. These meetings were used to review activities, discuss problems, request data/action/direction, perform technical engineering, review/develop documentation, etc.

PRODUCTS:

Meeting minutes, assignment of action items, technical direction from customer.

SOFTWARE REQUIREMENTS:

These meetings were very significant in terms of resolving mechanization alternatives, identifying customer (SAMSO, SAC, etc.) requirements, concerns, desires, and opinions, and organizing activities necessary to develop requirements and resolve conflicts.

SOFTWARE TESTING:

a. Considerable discussion between Autonetics and TRW was involved in resolving disagreement about early flight simulation results. These resulted from disagreement over error sources and models between TRW targeting and flight simulation and Autonetics flight simulation. An attempt was made to organize a thorough review/comparison of the simulation models between TRW and Autonetics. Though this was never completely successful, most of the major disagreements were subsequently resolved.

b. Generally TI meetings were held to discuss plans/procedures for acceptance testing (Team Demonstration) of a software product. TRW and SAMSO participate directly in selecting the tests to be conducted during this final demonstration.

INTERFACE:

Everyone.

COMMENTS:

The importance of this activity in the day-to-day activities during the software development process cannot be overemphasized, particularly in terms of requirements development. Though the organization of the Minuteman system development appears to be highly formal from the standpoint of contracting, agencies involved and documentation requirements, resolution of technical software problems between the contractor (AN) and customer (SAMSO/TRW) was surprisingly informal.

ACTIVITY: Item 15, Tape Checkout (TXO) Site Development and Use

DESCRIPTION:

Engineering test facilities consisting of "operationally" configured missile hardware and ground equipment were used for a large portion of the Minuteman III ground program checkout, verification, and demonstration. These facilities, most recently called TXO sites, grew out of hardware (and software) integration facilities originally established on Minuteman I to integrate the Autonetics built quidance and control system, ground equipment, and software. As the Minuteman system grew, in terms of different configurations and amount of total software development, facilities were developed specifically to support software development. These TXO sites, however also support various integration, troubleshooting, and special test activities.

The philosophy of these sites is to use actual hardware wherever feasible. This hardware includes the airborne computer, IMU, flight control electronics, missile downstage control hardware (actuators), and operational launch facility ground equipment. Some provisions in the form of special test equipment have been added to aid in software testing and troubleshooting - the most obvious of these is a control console for the airborne computer. (See Comments.)

PRODUCTS:

Site equipment lists, certification letters, log books.

REQUIREMENTS:

No direct relationship.

TESTING:

a. This was the primary tool for ground program checkout and verification prior to Wing VI -11 program.

b. These sites provide a means for testing potential requirements/mechanizations in a "realistic" environment. They are used to troubleshoot systems returned from the field because of unusual or unexplained behavior which could not be diagnosed during field level maintenance and to diagnose reported anomalies or problems, since it provides a means of reproducing or attempting to reproduce reported problems in an environment where the problem can be studied.

c. The sites are also used for verification of interfaces to Autonetics equipment (hardware/software) - physical and electrical as well as functional.

d. In order to be used to conduct program demonstrations, the TXO sites must be "certified". Certification involves establishing the configuration of all equipment (hardware and software) used during the demonstration and providing assurance to quality engineering personnel that the equipment is appropriate for demonstration of the particular software end item.

   In the case of the TXO site, most of the equipment is operational hardware obtained as GFP. The remainder of the hardware is STE, which is individually certified through demonstration of "functional equivalence" using Interface Control Documentation as a primary reference. The decision as to functional equivalence is obviously somewhat subjective and becomes particularly troublesome when software simulations are used (see System Simulation Laboratory). The certification activity is the primary validation of the TXO site as a software development tool. Due to the philosophy of using primarily GFP hardware (instead of simulators), this activity provides a rather high degree of assurance that the TXO site is representative of an actual Launch Facility configuration.

INTERFACE:

Programmers (users), quality engineering.

COMMENTS:

a. The primary evolution of the TXO site has involved addition of specific capability for software checkout or improved turnaround in conducting software tests. Most changes/additions have resulted from suggestions by site personnel of ways to make checkout/verification procedures faster and easier.

b. As the capability of the SSL has improved, it has been used more and more in place of the TXO site for software checkout and verification.

ACTIVITY: Item 18, System Simulation Laboratory (SSL) Development and Use

DESCRIPTION:

The System Simulation Laboratory (SSL) as currently configured, consists of a D37D computer, two general purpose computers (XDS920/930), an EAI 231R analog computer, various specially designed equipment and software to provide:

a. Real-time simulation of missile powered flight. The simulation does not include free fall, re-entry or impact.

b. Real-time simulation of the Launch Facility ground system for both the Wing VI and Force Mod configurations. This simulation includes only those aspects of the ground system which affect, or are affected by, the on-board software/computer.

This simulation lab was originally developed for Minuteman I as strictly a powered flight simulator to be used for flight program checkout. During the Minuteman II development, the ground system simulation was proposed and implemented to a limited extent as a back-up for the TXO sites which by that time were becoming heavily loaded due to the number of software configurations and changes being developed and maintained. The ground system simulation was not used extensively for ground program development until fairly late in the Minuteman III system development, but by the time of the -11 revision of the Wing VI ground program, it was preferred as a checkout tool when it was available and the majority of the -11 verification was conducted on the SSL due to the reduced test time and better control over the "System" in terms of the ability to establish specific test conditions.

REQUIREMENTS:

No direct relationship.

TESTING:

The SSL was the primary tool for flight program checkout. It was primarily used to establish program "continuity," execution of flight events (staging, thrust segments, R/V deployment, etc). It does not provide the capability to verify overall flight accuracy.

It became the preferred tool for ground program checkout and verification about the time of the -11 Wing VI ground program.

Certification of the SSL for use in demonstrations was accomplished by establishing a procedure with the Quality Assurance personnel which would be used to "demonstate" the SSL. Subsequent to performing the site certification procedure, the SSL software was impounded by QA and held for use during flight program demonstrations. About the one function which this certification served was to establish configuration control on the simulation software. The certification procedure in no way verified performance or suitability of the simulation.

COMMENTS

The initial development of the SSL for Minuteman I was a natural outgrowth of the general philosophy of using as much actual hardware as possible in software testing, the use of the airborne computer, flight control electronics, and downstage hardware. Due to the constraint of operating in real-time, in order to satisfy the airborne computer interface, the modeling of missile dynamics was severely limited.

Partly because of SSL limitations and partly because of organizational parochialism (responsibility for the SSL was in a different company division than the software development), the MFS (then called CLAMPS) simulator was developed to supplement the SSL for Minuteman II. At the time that CLAMPS was developed there was apparently no consideration given to extending the capability of the SSL rather than developing a second simulation. (It is not clear whether this would have been feasible.) In any case, CLAMPS/MFS has evolved to be the primary tool for flight program verification and the SSL strictly supports early flight program checkout and and provides testing of "compatibility" with the airborne computer hardware.

ACTIVITY: Item 17, Minuteman Flight Simulator (MFS) - Development and Use

DESCRIPTION:

The Minuteman Flight Simulator (MFS) is a software program which executes on the IBM S/360. This program simulates operation of the Minuteman III on-board computer (D37D) and the dynamic behavior of the Minuteman III missile during flight. The actual flight program and targeting data are loaded into MFS and "executed" during the simulated flight. MFS was the primary tool used for Minuteman III flight program verification.

A similar program, called CLAMPS, was used for Minuteman II flight program development, and MFS was a continuation of the same testing philosophy for Minuteman III. Originally (Minuteman I), flight program checkout and verification was conducted on the SSL (see separate description). Shortcomings of the SSL during Minuteman I and the fact that responsibility for the SSL was in a different division of the company than the software development, led to the development of CLAMPS.

The MFS Program uses the D37D computer interpretive simulation program, DSIM. The mathematical models necessary to simulate the missile dynamics came from a wide variety of sources generally chosen at the discretion of the MFS programmer. Some of the models/mechanizations used in MFS were taken from work in the SSL; others differ considerably between MFS and SSL. Some modeling information was derived from work done by TRW on the targeting program, but again there was no universal commonality between the two programs, though the method of flight program verification required "compatibility" between the MFS and targeting program (see Comments and Description of Flight Program Verification).

PRODUCTS:

MFS Program and documentation.

REQUIREMENTS:

Flight Program requirements affect "requirements" for MFS. No formal requirements were established for MFS and, to a great extent, MFS was designed to accommodate the flight program and TRW's targeting program. In fact, the three programs were initially checked out simultaneously and MFS problems were generally resolved such that the system "worked".

TESTING:

a. MFS was the primary tool for Minuteman III flight program verification. Most of the checkout activity is done in the SSL because of turnaround and cost considerations.

   MFS and the SSL are also used to verify the targeting data for each R&D flight test missile and for new targeting configurations.

   Testing of MFS itself was not conducted as an independent effort but as a part of early flight program (and targeting program) checkout.

b. There was no formal verification/validation of the MFS program itself. Testing of MFS was done as an integral part of early flight program verifications. Various analysis and comparison of modeling were performed on a selective basis as a result of debugging impact point discrepancies during simulation of early R&D flights, but this was generally not done on an organized basis but as required to resolve flight program simulation problems/discrepancies.

c. MFS was "certified" prior to the verification and delivery of the operational (Block IV) flight program. This certification consisted of achieving an agreement between the software organization and the Quality Assurance organization as to a procedure which would demonstrate that MFS was acceptable for formal demonstration of the flight software. Certification is intended to provide assurance that the demonstration facility is functionally equivalent to, or at least representative of, the actual environment in which the software end item is to operate. This is a very difficult task when the facility is a software simulation. Certification also establishes configuration control; in the case of MFS, this involved QA's impounding a copy of the program.

INTERFACE:

TRW (Targeting program developers); Flight programmers; SSL programmers.

COMMENTS:

a. The MFS program development is best described as evolutionary. Firm design objectives/program requirements were not established. Rather, the program was designed and evolved primarily from the standpoint of accommodating the flight program and targeting data such that successful flights; i.e., impact points within CEP, could be simulated. Continuous schedule pressure arising from the need to support R&D flight test schedules also contributed to the evolutionary approach, since major program re-designs were generally not feasible from a schedule and manpower standpoint.

b. In early use (Minuteman III R&D flights) data from MFS was somewhat unmanageable, consisting of a printed listing composed of periodic dumps of simulation/program variables. The output format for MFS was carried over from Minuteman II, which required a much shorter, less involved flight mission. Flight program errors in the FTM 201 (initial ETR test flight) tape which were discovered during post-flight analysis could have been seen in the MFS runs if they had been scrutinized carefully. This problem led to additional graphical outputs and tabulations being added to MFS to improve data presentations.

c. Simulations of the early R&D flight programs produced some unexplained deviations/errors in expected impact points which were a source of considerable disagreement between Autonetics and TRW as to whether the flight program, TRW's targeting program, or MFS was the contributor. A committee was organized between Autonetics and TRW to review and compare the modeling between MFS and the targeting program. This committee operated for a short time, and changes to both programs resulted as well as explanations of most of the primary discrepancies.

ACTIVITY: Item 18, Interpretive Computer Simulator (DSIM) - Development and Use

DESCRIPTION:

An interpretive simulator was developed to simulate functional, instruction-by-instruction operation of the Minuteman III guidance and control computer (D37D). This simulator, called DSIM, is the IBM S/360 program which is combined with simulation of missile dynamics in MFS (see separate description), or used as a "stand-along" batch-oriented computer simulator.

This program was developed as a general support software tool during initial design and development of the D37D computer. It is similar to most interpretive, bit-by-bit computer simulators except for the added complexity imposed by the program timing characteristics of the D37D.

PRODUCTS:

Computer Program and User Documentation.

REQUIREMENTS:

No direct relationship.

TESTING:

a. DSIM is not a primary software checkout or verification tool except as part of MFS. Programmers generally prefer using SSL or a TXO site for checkout because the "hands-on" situation allows immediate debugging of problems, and does not require as much "set-up". Additionally, the turnaround on batch processing is a continual problem.

   DSIM is used for verification of detailed timing and/or arithmetic computations in portions of Missile Test, Terminal Countdown, and some IMU functions.

b. No specific validation procedure was performed on DSIM. It was initially debugged/verified using computer functional test programs and subsequently evolved through debugging of problems encountered in using the program or as a result of additions/modifications suggested by users of the program.

INTERFACES:

D37D computer designers; D37D programmers.

COMMENTS:

a. Computer operating characteristics which, in the case of the D37D, are not straightforward are a type of programming constraint, hence a form of program requirement. DSIM provides a potential of detecting programming idiosyncrasies of the computer which might not be readily observed on actual computer hardware. This potential has not been realized to any great extent since:

   (1) Tests for machine idiosyncrasies were generally not added until a programming problem brought them to attention, and
   (2) DSIM is not used extensively except for flight program verification.

b. As previously indicated, TXO sites or SSL are preferred to DSIM by programmers, primarily because of turnaround considerations and the interactive aspects of the lab sites.

ACTIVITY: Item 19, T.O. Validation

DESCRIPTION:

A Technical Order (T.O.) was prepared by the Logistics organization in Autonetics. This document describes the step-by-step procedures for operating the guidance system in the Minuteman silo, including initial loading of the airborne computer memory, IMU alignment, status responses, etc. Once validated, this document is published by the Government Printing Office and is the manual used by SAC personnel during field operation.

The majority of the procedures in this T.O. are directly or indirectly related to the on-board software. Validation of the T.O. involves "dry running" the procedures on an engineering TXO site, using a copy of the ground program. This activity is normally conducted subsequent to ANQA demonstration of the software (see Demonstration), and prior to program sell-off.

PRODUCTS:

Updated T.O. Manual.

REQUIREMENTS:

No direct relationship. Desired system operating procedures affect software requirements but these procedures are determined prior to T.O. development.

TESTING:

This activity normally results in corrections to the T.O. procedures. Since the activity involves exercising the software, it is possible that software problems might be uncovered though there is no evidence that this has ever occurred.

INTERFACE:

TXO site personnel (site operation/scheduling); software project engineering/ programmers (problem resolution).

ACTIVITY: Item 20, Test Requirements Cross Reference

DESCRIPTION:

The Minuteman III contract required that a Test Requirements Cross Reference Index (TRCRI) document be prepared for each end item software product. Preparation of this document involves examining the Part I software specification (Figure A, Part I), the Program Description Document, the Program Requirements Document (in the case of the ground program), and the Demonstration Test Plan (DTP) and preparing a cross reference matrix which relates the Part I requirements to: (1) the paragraph in the PDD which describes the implementation of the requirement, (2) the paragraphs in the PRD which describe the detailed requirements (ground program only), and (3) the test numbers in the DTP which verify the requirement.

PRODUCT:

TRCRI document.

SOFTWARE REQUIREMENTS:

See COMMENTS.

SOFTWARE TESTING:

As a result of this activity, the Verification Test Plan (DTP) was modified to change or add tests to verify all specified requirements.

INTERFACES:

Programmers (PDD and program implementation), Test Planner (Test Objectives), Quality Assurance (coordination).

COMMENTS:

One difficulty encountered in satisfying Quality Assurance of the validity of the TRCRI document was in the wording used in the requirements document versus that used in the program description documents. In some cases the wording of the description document had to be changed to be more consistent with the requirements document wording. The possibility for turning this sort of an activity into a documentation exercise is very real.

ACTIVITY: Item 21, Guidance Mechanization Development

DESCRIPTION:

The basic Minuteman III flight guidance mechanization was developed by TRW. This information ultimately was published in the design criteria document prior to the Operational Flight Program delivery. However, this publication was after the fact; the requirements information necessary to develop the Block II/III flight programs came from relatively informal documentation and technical interchange between Autonetics and TRW. The requirements for these R&D flight programs were subsequently documented in Program Description Documents and Program Requirements Documents. The versions of the flight program developed prior to the operational configuration (Block IV) were not controlled as end items and had no formal Part I specification associated with them.

Since Autonetics was responsible for flight accuracy, error analyses were conducted at Autonetics on TRW's guidance mechanization. Hence, there were analysts at Autonetics who could assist the programmers in resolving problems, but the process was not well-structured. As the flight program evolved through the R&D flight test program toward the operational requirements configuration, a flight software requirements (system engineering) function/organization developed which became the central point for accumulating and documenting flight software requirements, but most of the actual mechanization development had been completed prior to this time.

PRODUCTS:

Flight equations, mission configurations, mechanizations (Flight Program Requirements).

SOFTWARE REQUIREMENTS:

As indicated above, the development of the early flight software requirements was a rather informal process even though an outside agency was responsible for the guidance mechanization.

SOFTWARE TESTING:

a. Considerable difficulty was experienced during verification of the early flight programs in separating guidance problems from targeting problems. TRW was responsible for targeting and their targeting programs were being developed and verified at the same time.

   This situation was aggravated because the technique used to verify the flight programs consisted of simulated flights using the MFS simulator and TRW-generated targeting data (see separate descriptions of Flight Program Verification and Minuteman Flight Simulator). This technique precluded separation of targeting and flight program problems. There was in fact a third variable involved - the MFS simulator itself which was also being developed/verified at the same time. The design/development of MFS relied heavily on "satisfying" the flight and targeting programs; hence, independent verification of it was essentially impossible.

b. The development and evolution of Minuteman III flight program mechanizations involved considerable "testing" of potential requirements/mechanizations with regard to feasibility of implementations, programming penalties, etc. These activities were conducted in a relatively informal manner jointly, redundantly, and/or independently by Autonetics, TRW, General Electric (RV Contractor) and SAMSO. Various simulation and analysis tools were employed at the discretion of the principals involved.

INTERFACES:

TRW (guidance analyses and targeting), Autonetics (error analysis, programming, flight control analysis, hardware design/interface).

COMMENTS:

During the early development, a major conflict arose between Autonetics and TRW over what guidance mechanization to use and who should be responsible for guidance analysis. For a period of time the programming area had two sets of guidance requirements to implement.

ACTIVITY: Item 22, Memory Optimization Study

DESCRIPTION:

A study to identify and select candidates functions/recommendations to be "scrubbed" from the then-current set of requirements for the Block IV (operational) configuration. A definite procedure was established jointly between Autonetics and TRW for identifying candidates and analyzing the impact of changing/deleting the requirement, and quantifying the effect in terms of numerical weighting factors. SAC made the final selection. This exercise was conducted separately for both the ground and flight programs. This activity resulted in independent study by Autonetics and TRW with a series of T.I. meetings to review candidates and analysis.

PRODUCTS:

Minutes of T.I. Meetings, final report giving candidates and assessment.

SOFTWARE REQUIREMENTS:

Direct relationship, the final selection from this study was reflected directly into software requirements.

SOFTWARE TESTING:

No direct relationship.

INTERFACES:

Joint activity between Autonetics and TRW technical people involving system engineering and programming people.

COMMENTS:

In making the final selections of items to be "scrubbed", SAC indicated concern with the necessity to "delete" any functions which would "degrade the capabilities of the weapon system". It is interesting to note that there appears to be an unlimited list of functions whose addition would in some sense increase the capability of the weapon system. During the early development of the Minuteman III software, the list of potential additives grew rapidly. When the time came to "baseline" the operational configurations, the total list could not be accommodated within the airborne memory constraints. The memory optimization study was initiated as an orderly approach to selecting the most optimum configuration. It appears somewhat misleading to view this activity as potentially degrading system capabilities.

It has been suggested that the type of quantitative rating system used during the optimization study be applied to all program functions from the early development phases up until program delivery as a normal procedure, but this suggestion was not adopted.

ACTIVITY: Item 23, Launch Actions Study

DESCRIPTION:

This activity was a special study performed by each contractor on the Minuteman system relative to hardware/software which he produces. The purpose of the study is to identify any "launch actions", - deliberate acts directed by a single individual against an item of equipment which could contribute to or cause an unauthorized launch. The software being, in some sense, in control of the equipment in the silo is a prime candidate for launch actions. This study consisted of analyzing the actions necessary to cause a launch and attempting to devise ways by which the system interlocks and security could be overcome or circumvented by specific actions.

The results of all the Launch Action Studies are combined into a single report by TRW.

PRODUCTS: Launch Action Study Report

SOFTWARE REQUIREMENTS:

The results of the study may result in software or hardware/software requirements changes to avoid threats of possible launch actions.

SOFTWARE TESTING:

This activity could be considered part of the software testing process.

COMMENTS:

The original Launch Actions Study conducted for the Minuteman II system resulted in a minor change to ground program requirements. This new requirement was carried over to Minuteman III but no additional requirements changes were identified as a result of the Minuteman III study.

ACTIVITY: Item 24, Safety Analysis

DESCRIPTION:

Autonetics provides support to The Boeing Company, who has responsibility for the safety of the overall Minuteman weapon system. Safety is primarily concerned with protection against inadvertent missile launches, catastrophic events in the silo, and Class A incidents (inaccurate flight with an armed warhead). This activity involves analysis of the various interlocks provided in the system to protect against such occurrences.

PRODUCTS:

Reports and verbal concurrence.

SOFTWARE REQUIREMENTS:

The Part I software spec contains several extremely general requirements with regard to safety. These are sufficiently general that they could not be verified. The "detailed" requirements/mechanization which implement the system safety provisions are buried in other requirements or implementation.

SOFTWARE TESTING:

As a part of the Safety Analysis, the flow charts of the software are reviewed to insure that agreed upon interlocks are included in the program design.

INTERFACES:

Programmers and software system engineers.

ACTIVITY: Item 25, High Explosive Simulation Test (HEST)

DESCRIPTION:

The HEST Test was a special test conducted on the Minuteman II Weapon System to assess in-silo structural survivability. This test was planned and conducted by The Boeing Company and consisted of detonating explosives above an operationally configured silo. A "live" and operating guidance system was not required to meet the HEST objectives but it was decided to use one and instrument it to record IMU data during the test.

PRODUCTS:

Test data.

SOFTWARE REQUIREMENTS:

As a result of analysis of the data recorded during the HEST testing, the Minuteman III criteria and software requirements for detection of a seismic event were modified and the gyrocompassing mechanization was improved resulting in other ground program requirement changes.

SOFTWARE TESTING:

Though this test was not originally intended to include G&C performance or software testing, it in effect provided such a test.

INTERFACES:

Test data from Boeing.

COMMENTS:

Prior to the HEST data the criteria for detection of a seismic event consisted of a very general statement about unusual platform motion. The original Minuteman III software mechanization was designed as a relatively simple means of detecting some classes of "unusual motion". This mechanization was jointly agreed upon and therefore became the "requirement".

ACTIVITY: Item 26, Flight Test

DESCRIPTION:

The Minuteman III flight test program consisted of an R&D test program at ETR and VAFB and Cat I/II and OT (Operational Test) program at VAFB. The planning (schedules and objectives) for this program was performed by TRW. Autonetics reviewed and could request changes/additions to the tests/test objectives on a flight-by-flight basis, but the flight schedule was essentially "firm," as received by Autonetics. Generally, separate flight programs were prepared for each ETR flight; but later in the flight test program, the "operational" flight program was released and used for all subsequent tests.

PRODUCTS:

Telemetry Data from flights.

SOFTWARE REQUIREMENTS:

The flight test objectives appeared to be the "forcing" function for the requirements imposed on the flight program at each block change point and in some cases, on the flight programs for each flight test.

In some cases, the specific flight test objectives imposed unique requirements, such as the programming of "perturbations" in the flight profile in order to evaluate system responses.

SOFTWARE TESTING:

a. The primary objective in the verification of programs delivered for a specific flight test was to ensure that the objectives of that flight would be achieved. Since there was considerable schedule pressure during this period, the verification process relied mainly on simulated flights using the specific targeting data for the particular flight test.

b. Comparison of flight test data (telemetry) with simulation data allowed refinement/increased confidence in simulation models, but this did not occur until later in the Minuteman III development and was not done on a regular, planned basis.

INTERFACES:

TRW publishes flight test schedules, coordinates test objectives for each flight, and generates targeting information.

ACTIVITY: Item 27, Post-Flight Analysis

DESCRIPTION:

The telemetry data from the R&D test flights at ETR and VAFB is reduced/analyzed using several analysis programs to evaluate the functional performance and flight accuracy of the system. Included in this analysis are guidance system performance/accuracy, control systems performance, and discrete event conformance to design criteria. Where discrepancies or anomalies are noted, they are investigated to determine the cause. The results of this analysis is published for each test flight.

PRODUCTS:

R&D Flight Test Summary Report(s)

SOFTWARE REQUIREMENTS:

Problems noted as a result of this analysis may result in changes to flight program requirements. (See GBI sampling change for example.)

SOFTWARE TEST:

This analysis provides valuable test data on the flight programs. Several flight program errors were discovered during the flight test program as a result of this analysis. These errors were all relatively subtle in terms of effect on system performance.

INTERFACES:

Telemetry Data from ETR/VAFB instrumentation; resolution of anomalies may involve programmers, system engineering guidance/control analysts, etc.

COMMENTS:

An anomaly (a small discontinuity in the guidance trajectory) was observed in the post-flight analysis of the 1st test flight. This was discovered to be an error in the targeting data, but as a result of re-examining the flight program and looking more carefully at the test data, several program errors were discovered. These errors were all of such a nature as to represent relatively minor problems in terms of performance - an incorrect sign in a low order term of a control equation, and a longer than desired lag in the roll control mechanization are two examples.

ACTIVITY: Item 28, Design Reviews

DESCRIPTION:

Three types of formal design reviews were used on Minuteman III software: System Design Review (SDR), Preliminary Design Review (PDR), and Critical Design Review (CDR). These generally represent a one or two day meeting during which Autonetics reviews the status, requirements, problems, etc., with

a particular software "tape" at different stages of the development. Various SDR's, PDR's and CDR's were held for different versions (block changes) of the flight and ground programs. These reviews resulted in action items to resolve problems, conduct special studies, or change the software requirements or design. Responses to action items are subsequently supplied to "close" the review.

PRODUCTS:

Meeting minutes with action item assignments; response to action items.

SOFTWARE REQUIREMENTS:

Additions/modifications to program requirements often result from the reviews. For a program revision, a "shopping list" of potential requirements changes or additions may be presented and approved, deleted, modified, or carried on for further study as a result of the review.

The reviews provide a vehicle for publishing baselines of the various requirements documentation (SRA data package).

SOFTWARE TESTING:

No direct relationship. General plans for verification and demonstration may be presented, but no significant action is taken.

INTERFACES:

This activity involves all concerned associate contractors and a variety of SAMSO/ Air Force personnel. In particular, customer offices may be present who are not involved in the day-to-day software development activities and whose only contact with the software development is the design review.

COMMENTS:

In the "plan" of the software development process, design reviews are intended to occur at specific times relative to the state of the software development, i.e., PDR is intended to "finalize" Part I software requirements, CDR finalizes Part II requirements, etc. This plan is apparently not fulfilled to any great extent since the point in time at which the design review occurred relative to the state of the software development varied considerably. In fact, the reviews appear to be best viewed as a more formal T.I. meeting with larger attendance.

4.3 REQUIREMENTS DEVELOPMENT

4.3.1 Discussion

The process of developing, identifying, and establishing software requirements as depicted by the Minuteman III development can be best described by its three key characteristics: (1) continuity, (2) iteration, and (3) evolution.

4.3.1.1 Continuity. To understand the process of developing and identifying software "requirements," it is important to recognize that the final result or end product of this process is not a Part I or Part II software requirements specification. Rather, it is the total set of information, data and decisions which the programmers apply in generating the computer instructions which comprise the program. The process of developing these "programming-level requirements" begins with the earliest "concept-level" design of the system of which the software is a part. This is a somewhat broader application of the term software "requirements" than is found in general usage. However, general usage of the term is often misleading because it tends to imply that the information used to program a given software product consists, at least primarily, of functional performance requirements; i.e., descriptions of functions which the computer subsystem (including software) must perform. This leads to the erroneous conclusion that this information in total is determined (and potentially documented) as a more or less distinct activity prior to the Program Development activities.

A more accurate understanding of the actual process typified by the Minuteman III software development is gained by visualizing the development of software requirements as a continuous, or at least continuing, process starting during initial conceptual design of the overall system, continuing through design of each of the related subsystems, culminating in the development of a specific software product, and then restarting at some point in the process (maybe as far back as the concept-level requirements activity) in order to assess, implement, or originate changes to the software/system.

In the early "concept" stages of system/software development, the software requirements activity tends to be buried in the overall system requirements activity. As the various system elements are identified/designed, the software requirements activity becomes more specific (that is, more specifically directed toward implementing software) and the activity becomes subdivided around major hardware subsystems or major system functions. In the final stages of software development, the requirements activity becomes very specifically related to the detailed "programming-level" information necessary to program a specific software end item or version of a software end item. Subsequent to delivery of a given software product, the requirements activity may be restarted at one or more levels in response to problems discovered during field testing or use of the software, or as a result of suggested improvements to, or extensions of system performance. This understanding of the requirements process leads to the concept of levels of requirements/requirements activity. An almost infinite number of levels could be identified where each level represents an activity which takes some subset of the total "requirements" as identified at the higher levels, and attempts to "solve" or "implement" this subset of requirements, thereby establishing requirements (generally more detailed and specific) at another level.

It seems most meaningful to identify a minimum of three levels: (1) concept level, (2) system level, and (3) programming level. A definition of these levels is presented below:

1. Concept Level

   Objectives, data, desires, problems, and environment which produced the basic concepts/design goals from which the system* was formulated.

2. System Level(s)

   The functional, physical, electrical, and procedural organization of the system* including black box performance requirements, interface definitions, functional allocations, etc. These requirements can be identified at various levels between the system* level and the programming level, based on selection of meaningful "subsystem" definitions. The airborne and ground subsystems are one natural subdivision for Minuteman. Likewise, the IMU subsystem and the Re-entry subsystem are also logical subdivisions.

3. Programming Level

   The total information from which the computer program instructions are coded. This includes what is generally called program design or solution information (Part II specification) as well as any additional knowledge/data required to produce the program.

The significance of characterizing software requirements in this manner is as follows:

1. It focuses attention on the major activities/personnel/disciplines involved in different aspects (and possibly at different times) during the requirements development process.

2. It provides a basis for distinguishing different types/sources of requirements. This idea will be developed further in subsequent sections of the report.

3. It aids in recognizing the problems associated with documenting requirements and testing software against requirements; i.e., which requirements? or what form (level) of the requirements?

The concept of different levels of requirements is developed further in Section 4.3.4. References to requirements levels will be made throughout the remainder of the report.

*The term "system" is used here to mean the largest self-contained complex of hardware and software of which the particular software item is a part; i.e., the Minuteman Weapon System.

4.3.1.2 Iteration. The second key factor necessary in understanding the software requirements development process is to recognize that while the process is continuous from initial system conception, the flow is not unidirectional or single path. At each level and within levels multiple loops are established in which subsets of requirements are proposed, evaluated, modified, expanded, etc. Generally, there are many alternate "solutions" at each level to requirements at higher levels. The process of selecting alternatives and discovering the impact of the selection on other levels or subdivisions of levels is highly iterative. The common conception that requirements are established in some independent manner as one step and then "solved" or implemented as a second step is erroneous. In fact, "requirements" at one level may be determined solely by the capability of the system at a lower level rather than on some independent assessment of what is necessary to meet overall system goals. In general, requirements at all levels are a combination (and compromise, because it requires interaction at different levels, is by nature iterative.

4.3.1.3 Evolution. The evolutionary characteristics of the requirements process is similar and related to its iterative nature. The lack of absolute requirements for all but a minute amount of system functions means that there is a great deal within the system. Moreover, due to the size, scope, and duration of a system like Minuteman III, considerable changes occur in conditions, motivation, objectives, and understanding in the space of time required to develop the software. It is almost axiomatic that experience and exposure to a system problem in an engineering environment breeds suggestions of changes.

4.3.2 Requirements Documentation

Considerable variation has been and is evidenced in the content, format, and level of detail/requirements presented in the primary documentation of software requirements, in particular the Figure A, Part I and the ICDs. Some observations relative to these documents seems pertinent. The subject of requirements documentation will be discussed in more detail in Section 5.

There is no consistent format or method of describing requirements between the two Figures A (ground and flight) and the three most important ICDs (FM communications, Wing VI communications, Re-entry system). These documents have tended to evolve separately with few definite ground rules as to their content. For example, there are two ICDs which describe the interface between the ground program and the communications hardware/software for the Wing VI and Force Mod configurations, respectively. The two ICDs correspond to different ground equipment designed by different contractors, Boeing (FM) and Sylvania (Wing VI). These ICDs differ considerably in content even though the apparent function/purpose of these documents is identical. The FM ICD contains much more detail than the Wing VI and considerable information which is not directly constrained by the interface being defined. This difference was apparently due to different motivation on the part of the non-Autonetics side of the interface in terms of amount of interface "control" desired.

The content and intent of the Figures A (Part I) underwent some amount of discussion and change during the Minuteman III development. This discussion centered around how much and what level of detail should be contained in these documents. During the initial Minuteman III development, these documents lagged behind the software development. Since a different division of Autonetics was responsible

for these documents during early Minuteman III (the activity was subsequently reorganized), there was some motivation to make these documents detailed for control purposes. Therefore, as Minuteman III software began to stabilize, detailed "requirements," taken from the existing software, were included in the Figures A. Hence, by the start of Block III development, the ground program Figure A was quite detailed though not necessarily complete. One of the stated goals in modifying the Figure A for Block III was to generalize the Figure A and remove "mechanization or implementation" details. The main motivation behind this was apparently the classic concept of "requirements" in the Part I and "solution" or "mechanization" in Part II (the Program Description Document). Further, the formality and contractual aspects of the Figure A, Part I, make review and negotiation of detailed requirements somewhat time-consuming. The more formalized verification procedure used first on a Minuteman II program and later on the Wing VI (-11) ground program, caused the generation of a separate Program Requirements Document (PRD) when it was decided that the Figure A was not (and should not be) complete and detailed enough to use as a reference for testing.

The general trend which evolved during the Minuteman III development was to keep the Figure A, Part I relatively general to ease the contractual and formal review problem and document the detailed requirements in a separate PRD which would be used for verification.

Interestingly enough, this trend has apparently been reversed on the most recent Minuteman III software development (subsequent to the software being studied). On this recent development, the Figure A, Part I is being used for detailed requirements.

4.3.3 Minuteman III Requirements Activities

Figure 11 illustrates the primary activities involved in the Minuteman III software requirements development. This section summarizes the major requirement changes which occurred during the Minuteman III software development.

4.3.3.1 Ground Program

4.3.3.1.1 Block II. The initial Minuteman III ground program was developed to support system integration and VAFB R&D flight test with the Block II, Force Mod hardware configuration. (The Block I hardware was only used for in-house testing.) The requirements for this program were baselined from the Minuteman II Operational Ground Program, though the entire program had to be developed from scratch for the new Minuteman III Computer (D37D). The major changes from the Minuteman II baseline are discussed below.

1. Self-Alignment Technique (SAT)

The Minuteman II System used a silo-implaced autocollimator to provide the primary azimuth (East-West) reference for IMU alignment. The IMU contained a Gyrocompass (GCA) which was used strictly as a secondary azimuth reference in the event that the autocollimator function was lost. A significant result of this mechanization was the requirement for highly accurate implacement of the autocollimator at each silo. Prior to Minuteman III, a gyrocompassing/gyrocompass calibration mechanization called SAT was developed at Autonetics. This mechanization provided a sufficiently accurate azimuth reference

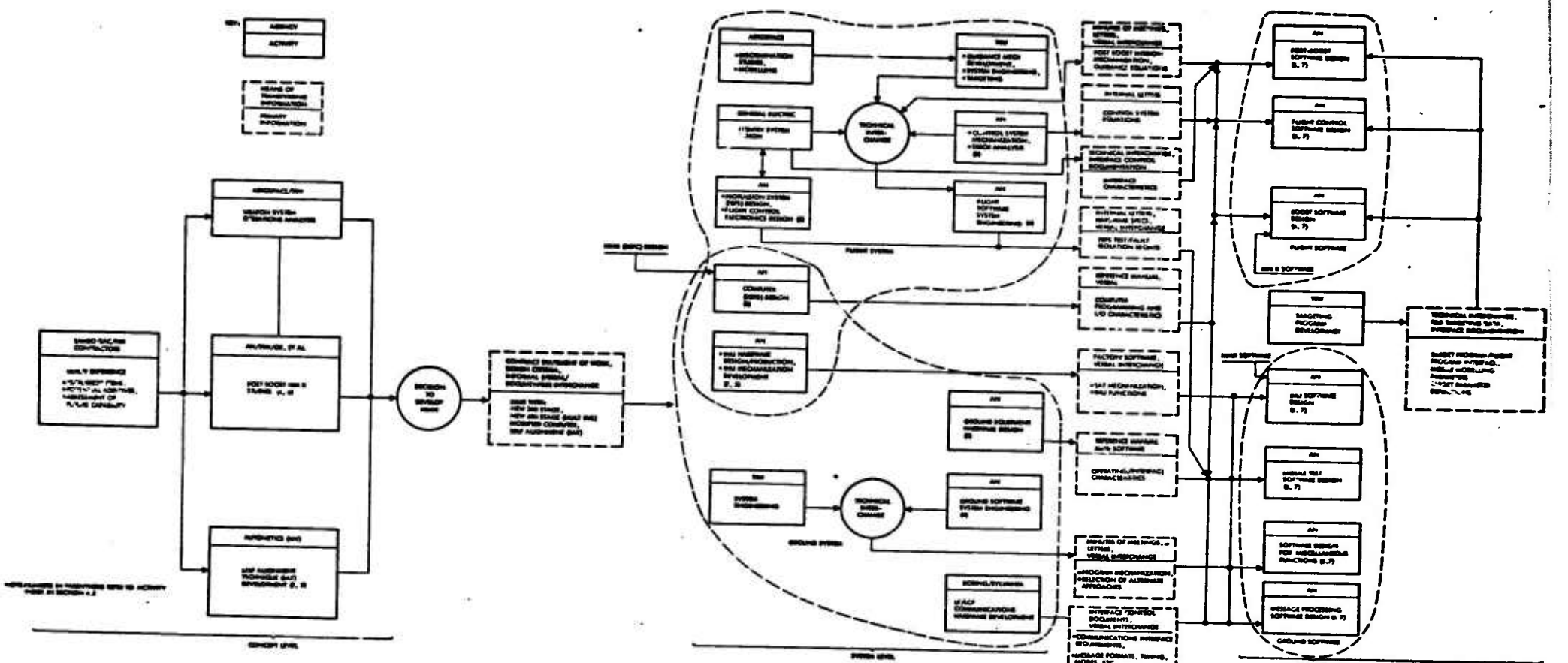


Figure 11. Minuteman III Requirement Activities

73/74

to potentially eliminate the need for an autocollimator. A feasibility study was funded, a Minuteman II IMU was modified, and software developed to test the SAT technique.

With the advent of Minuteman III, this technique was included in the baseline system requirements; however, the mechanization still used the autocollimator as the primary reference due to lack of confidence/data on the SAT technique and disagreement as to the accuracy of the autocollimator implacement process. The SAT mechanization required an additional bubble level on the IMU to allow for leveling the platform in an "upside down" orientation.

2. PBPS Test

The missile test portion of the program had to be expanded to test the new fourth stage hardware, particularly the control hardware on the Post-Boost Propulsion System (PBPS). Addition of this subsystem also required another level of fault isolation since the system maintenance concept required separate removal of the Re-entry System (R/S), PBPS, and G&C System. The Block II PBPS test mechanization was an extension of the boost stage testing philosophy being used on Minuteman II. Since the PBPS was a new design for Minuteman III, there was a shortage of data on its operation and hence test requirements were somewhat lacking.

3. R/S Fuzing

The new Minuteman III re-entry system had unique requirements in terms of fuzing the re-entry vehicles. This fuzing was accomplished by a serial transmission from the D37D to the R/S and was performed as a result of receiving target selection messages.

There was some concern over the reaction time requirements for target change due to the additional time required to fuze the RV's. As a result, several "high-speed" fuzing mechanizations were investigated but ultimately dropped.

4.3.3.1.2 Block III. The Block III change point was identified to incorporate several hardware changes:

1. Redesigned GCA Servos

The GCA servo system was redesigned as a result of a thorough reliability analysis conducted as a part of the Minuteman II reliability "recovery" program.

2. New Roll Torquer Motor

Analysis of attitude maneuvers with large yaw angles required by Block III flight missions/mechanization uncovered marginal performance of the platform roll axis torquer motor.

3. Repositioned Roll Gimbal Stop

The roll axis platform gimbal stop was reoriented 90° to allow additional rotational freedom in one direction. It is not completely clear why the original orientation was chosen but analysis indicated that certain post-boost maneuvers occurring with a particular launch/alignment orientation required this additional rotational freedom.

These hardware changes necessitated direct changes to the ground program requirements. In addition, the following change resulted indirectly from the GCA servo redesign.

Coarse Zeta Check

A completely new technique for initially indexing the GCA position to platform position. The previous (Block II) mechanization used the GCA position stop as a reference, but this information was not reliable with the new servo system.

In addition to changes resulting directly or indirectly from the Block III hardware changes, the following major requirement changes were also incorporated:

1. In-Silo Circumvention

The Minuteman II system was designed to survive a hostile radiation environment during flight. This was accomplished by a hardware/software circumvention technique. Prior to Minuteman III, a radiation environment in the silo was considered to be a potential problem and studies were initiated to assess the problem; i.e., determine potential radiation levels and investigate circumvention schemes. These studies resulted in Minuteman II ground program circumvention requirements which were subsequently applied to Minuteman III at the Block III change point. The need for in-silo circumvention was apparently known during initial Block II development, but the requirements was not included in Block II due to schedule pressure and the fact that Block II was strictly an R&D program.

2. Seismic Detection

Concern over ability of the in-silo system to withstand seismic shocks motivated the software requirements to detect and compensate for seismic events. However, there was no data available from which to determine how to characterize such events in terms of IMU-sensed motion. A software mechanization was developed based on ease of program implementation and a guess as to the effects of a seismic event on the IMU. This mechanization was discussed with SAMSO/TRW and subsequently became the "requirements."

3. PBPS Test Changes

Several changes were made to the PBPS portion of the missile test requirements at the Block III change point. These resulted primarily from a review of the requirements for PBPS testing caused by a change in personnel working on missile test requirements and some additional experience/data on PBPS characteristics.

4. Gyrocompassing Changes

Several changes were made to the GCA mechanization in addition to those resulting from the previously mentioned hardware changes. The most significant of these was a more sophisticated mechanization for detecting and distinguishing shifts in instrument biases and shifts in autocollimator data. The new mechanization resulted from the more or less continuous analysis/design process which occurred in the area of GCA mechanizations during the Minuteman III development. The more sophisticated mechanization was based on theoretical analysis and did not reflect actual GCA operating data; as it turned out, it was not effective and was subsequently removed.

Mechanization changes were also made in the area of "forgiveness" checks; i.e., means of overcoming apparent anomalies in calculated GCA data. Sophistications of this type were highly evolutionary throughout the software development.

4.3.3.1.3 Block IV. This configuration of the ground program was to be the "operational" configuration; i.e., it was scheduled to support Category II system testing and operational deployment of the weapon system. This was the baseline configuration for the Wing VI ground program.

The Block IV development was preceded by a six month "Memory Optimization Study" which was initiated to reduce the then-current list of potential requirements for both Ground and Flight programs to something which could be accommodated in the on-board computer memory. This resulted in the elimination/modification of many functions though most of the changes were relatively small and test-type functions seemed to be primary candidates.

A significant hardware change occurred at Block IV also:

AAU Removal

A "value" change was proposed by Autonetics as a result of an internal study which developed an alternate flight control mechanization which did not require data from Stage I and II body mounted accelerometers (AAU's).

This change required deletion of the AAU test portion of missile test.

The following paragraphs describe the major requirements changes which occurred at the Block IV change point.

1. Cancel Launch-in-Process (CLIP)

This function was an extension of the basic system capability and provided for delaying or cancelling a previously commanded launch. This requirement apparently originated in SAMSO/TRW and was primarily designed to be able to avoid launching missiles through a hostile environment.

2. High Altitude Fuzing (HAF)

The capability for detonating a warhead at high altitude was added to the weapon system at Block IV. This change to the Re-entry System necessitated ground program changes to fuze the RV's appropriately.

3. Data Transrecorder (DTR)

The DTR was a magnetic recorder/printer which was developed for Minuteman II by direction of the OOAMA as a means of obtaining additional maintenance data in the silo. Having been already developed, it was added to Minuteman III at Block IV. However, it was only added to the Force Mod configuration due to more available computer memory to accommodate the additional software requirements.

4. New Seismic Criteria

The criteria for identifying a seismic event was modified at Block IV. This change resulted from assessment of data gathered during a special test conducted by The Boeing Company to assess in-silo structural survivability. Though this test was not intended to evaluate G&C system performance during shock, it was decided to use a "live" and operating guidance system, and instrument it to record IMU data during the test. This data provided the first source of information about characteristics of seismic events. A mechanization was then designed to detect the conditions exhibited by the test data and this mechanization became the new "requirements."

5. Gyrocompassing Changes

A significant amount of changes were made to the GCA mechanizations at Block IV. Some of these were related to the amount of time required to complete SAT and conditions of acceptance of a SAT command. These changes were apparently triggered by TRW becoming more knowledgeable of the GCA mechanizations and questioning potential times required to complete SAT after initial system start-up. The Block III mechanizations allowed this operation to continue beyond the design criteria limit of 4.4 hours under some conditions. (It is interesting to note that the 4.4 hour criteria was apparently originally established by estimating how much time the mechanization actually required.)

A completely new GCA slew sequence called Optimum Slew was also included. The new mechanization was designed to overcome instrument bias variations during the initial 4 hours of instrument operation after start-up. This phenomenon was observed in the factor and the new mechanization was motivated primarily to reduce factory time lines.

6. SAO Mode/SA3 Command

Changes were made to the procedures for using GCA data by customer (SAC) direction. These changes prevented the use of GCA data under certain conditions and were apparently motivated by lack of confidence in the SAT capability and disagreement as to implacement accuracy of the autocollimators.

At this time, the autocollimator was still being used, in conjunction with gyrocompassing, as the primary azimuth reference and autocollimator implacement procedures had not been relaxed. The result of subsequent examination of implacement accuracies and growing confidence in GCA accuracies ultimately caused relaxation of implacement procedures and reliance on GCA data. The SAO Mode requirements were subsequently deleted (see -11 revision).

7. Improved Fault Isolation in Missile Test

The fault isolation requirements (stated in the design criteria as 95 percent probability with 95 percent confidence) were not fully met in earlier versions of the missile test program primarily in the area of the PBPS. This was due to lack of data on PBPS performance and lack of pressure to include these requirements in R&D programs.

The PBPS fault isolation requirements were patterned after the Minuteman II/III mechanization for downstage (booster) fault isolation.

8. PIGA Warm-up

This change involves a software controlled delay before closing accelerometer servo loops after initial start-up. This was a temporary measure designed to reduce instrument wheel-bearing degradation due to "cold" starts. A hardware design change was initiated to allow delaying wheel power turn-on also, but this change could not be scheduled in at Block IV (see -11 revision).

4.3.3.1.4 Wing VI -11 Revision. Subsequent to initial deployment of Minuteman III missiles in Wing III and Wing VI, a revision was scheduled to the Wing VI ground program. This revision was motivated primarily by desire to make improvements to the GCA mechanizations based on long-term field data derived from a "burn-in" exercise at Wing III. Several more or less minor program errors had also been discovered. The major changes incorporated at the (-11) revision are summarized below:

1. Gyrocompassing Changes

Review of long-term data gathered at Minot AFB (Wing III) indicated a high frequency of GCA alarms. The original GCA mechanizations were based on relatively short-term factory data. When the long-term data were analyzed, mechanization changes were indicated. Also, the additional study produced more "goodies."

2. Correction of Problems

Several program errors which had been identified subsequent to delivery of the program were corrected. These errors were relatively minor and non-catastrophic in terms of system operation.

3. Data Transrecorder

At the last minute, it was decided to incorporate the DTR capability into Wing VI at this change point. A simpler mechanization than the one implemented for the Force Mod program was established.

4. PIGA Hot Start

The IMU accelerometers were modified to delay PIGA wheel start-up until proper operating temperature was attained. This change resulted from analysis of PIGA failure history and a special test program which indicated wheel bearing degradation on "cold" starts.

Software changes were also required to implement the modified start-up procedure. These changes were included along with DTR at the last minute and caused a slip in delivery of the -11 program.

4.3.3.2 Flight Program

4.3.3.2.1 Block II. The initial Minuteman III flight program was developed to support R&D flight test at ETR and WTR with a Block II hardware configuration. This program was developed from scratch for the new Minuteman III computer though the boost phase was essentially copied from Minuteman II.

The post-boost re-entry system and the software were completely new for Minuteman III. Considerable study and evolution preceded the establishment of the mechanization in the Block II flight program. The major additions/changes which occurred during this evolution are summarized below.

1. Decoy/Chaff Deployment

The initial Minuteman III requirements included only multiple Re-entry Vehicles (RVs). The addition of requirements for deployment of decoys and chaff greatly complicated the overall design, both hardware and software. In fact, the requirement for decoys was ultimately deleted during Block IV, due at least partially to R/S design problems. The decoy/chaff requirements directly or indirectly created the next three requirements also.

2. Plume Avoidance

Engine plume impingement effects on decoy/RV/chaff deployment were recognized as a potential problem early in Minuteman III development. The major problem was associated with chaff deployment. Various chaff dispensing designs were investigated but the design finally chosen required software mechanizations to avoid plume impingement effects. The Block II mechanization attempted to inhibit the attitude control engines during chaff dispensing, and delayed ignition of the main axial engine subsequent to chaff dispensing to allow time for the chaff cloud to drift out of the plume cone.

3. Universal Flight Tape

Initially, three separate flight programs were planned to account for three basic configurations of post-boost mission. Prior to Block II, it was decided to develop a single, "universal" program which would have to be capable of handling all possible missions.

4. Decoy Anticipation Torque

Analysis indicated that a disturbance torque would be imposed on the Post-Boost Vehicle (PBV) as a result of decoy ejection. This would adversely affect deployment accuracy. A mechanization was designed to compensate for this disturbance torque by establishing a bias torque in the opposite direction prior to decoy ejection.

5. Deployment Attitude Guidance (DAG)

Mission requirements for deployment of various objects led to the requirement to perform small translational maneuvers using the attitude control engines rather than the main axial engine. The positioning of the various attitude control engines on the PBV did not allow vehicle translational motion without cross-coupling on the attitude control axes, hence there was considerable mechanization logic required to select the necessary sequence of maneuvers to accomplish the desired translation.

4.3.3.2.2 Block III. The major changes incorporated in the flight program at the Block III change point are summarized below.

1. Simultaneous Attitude Maneuvers

The Block II mechanization used a sequential series of single axis rotations to accomplish necessary attitude maneuvers. This mechanization was used because it is relatively straight-forward and the data on cross-coupling on the control axis was not available. As the Minuteman III system evolved, there was considerable concern over increased mission time estimates due primarily to additional maneuvers required by decoys and various delays required for plume avoidance. This concern prompted a suggestion that the attitude maneuvers be performed simultaneously on all three axes. Studies were performed to determine potential mission time savings and determine feasibility, and the new mechanization was added at Block III.

2. Control System Changes

Two additional changes were made to the control system mechanization as a result of the SAM implementation. Additional control system gain change points had to be included to deal with the increased "overshoot" and cross coupling of the SAM mechanization. Also, the control equations were modified to yield a fixed 25 deg/sec rate rather than the previously variable (12 to 25 deg/sec) to further reduce the mission time requirements for the attitude maneuvers.

3. Plume Avoidance Changes

Several additional requirements were added to avoid plume impingement affects on chaff deployment. These generally reflected additional analysis, better modeling, and additional data concerning chaff deployment and plume affects.

Additional delays were added and a special Plume Avoidance Thrust (PAT) mechanization was added to move the PBV away from the chaff prior to an attitude maneuver which would trun the axial engine into the chaff cloud. Also, the previous mechanization for inhibiting the pitch attitude engines during chaff dispensing had not been completely effective.

4. New Gravity Model

The Minuteman II gravity model which was carried over to Minuteman III was a relatively simple mechanization. Errors induced in the flight guidance computations due to this simplified model were compensated for by "biasing" the targeting computations. With the increased targeting problems on Minuteman III (multiple RVs), it was suggested that the flight program gravity model be upgraded to simplify the targeting (this would also produce somewhat improved accuracy on "off-nominal" trajectories, since the targeting biases were based on the nominal). This change was originally scheduled for Block III, but programming the new model turned out to require a major program change to "fit it in" to the timing structure; the implementation was therefore rescheduled for Block IV. It was subsequently "scrubbed" as a result of the memory optimization studies and, though it was coded and debugged, it never went into the operational program.

4.3.3.2.3 Block IV. The primary effort on the Block IV flight program was related to reducing computer memory requirements. A significant number of changes/deletions were identified as a result of the Memory Optimization Study previously discussed. Among these changes were some simplification of the plume avoidance requirements. Other significant program changes are described below.

1. Control System Mechanization

The Stage I and II attitude control equations were modified to derive acceleration from gimbal angle differences rather than using the body-mounted accelerometers (AAUs). This change was proposed to allow deletion of the AAU hardware.

2. High Altitude Fuzing (HAF)

Additions were required to accommodate the new HAF capabilities/requirements of the Re-entry System.

4.3.3.3 Examples. Figure 12 is an illustration of the requirements development process exhibited by one of the program examples (see Appendix). The figure attempts to illustrate the process involved relative to the different requirement levels and the rough chronological sequence. The loops drawn with dashed lines identify primary iteration loops in the requirements process. The flow of time is along both axes on the drawing but should generally be read like a book; i.e., left to right, then top to bottom. Hence, the earliest point in time is at the upper left and the latest, the lower right.

4.3.4 Generalized Requirements Characteristics

This section presents a generalized definition of the characteristics of the software requirements development process in terms of types and sources of requirements at three levels and primary ingredients in the requirements change process.

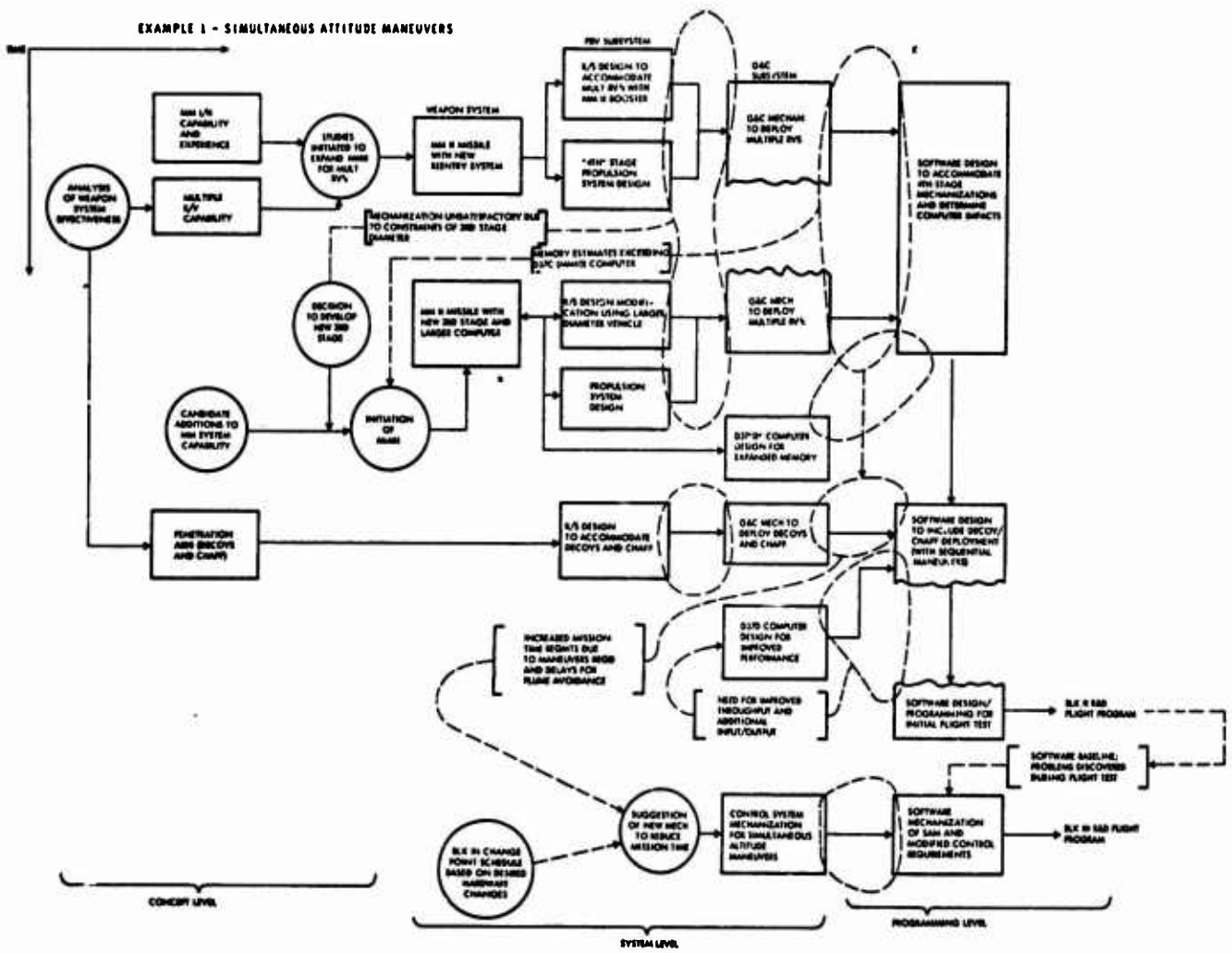


Figure 12. Example 1. Simultaneous Attitude Maneuvers

The motivation for presenting these definitions is simply to illustrate the range of software requirements in terms of method of derivation, testability, disciplines involved, etc. Study resources did not permit developing this characterization fully, but though the definitions are somewhat arbitrary, some examples will illustrate the utility of the classifications.

1. "State-of-the-Art" concept level requirements can obviously not be tested or programmed as such. This type of requirement must be translated into a more definitive system or programming level requirement(s). In documenting these lower-level requirements, the primary source tends to become obscured. For instance, a significant number of response time "requirements" in Minuteman III originated as this type of requirement, but are documented only as "Timing" type programming level requirements. When attempting to modify mechanizations in these areas, the impact is difficult to assess because the requirements source has been lost.

2. "Program Organization" programming level requirements can most effectively be tested at a detailed program structure level. While system level tests may potentially verify this type of requirement implicitly, testing at the higher level is less effective and efficient.

3. "Interface" requirements at both system and programming level represent a more severe documentation problem than some other types. Generally speaking, interfaces are established/defined in order to allow more or less independent development of the interfacing elements. As a result, documentation of this type of requirement is relied on heavily since detailed knowledge/understanding of the opposite side of the interface is usually limited.

4.3.4.1 Types and Sources. This section summarizes the general characteristics of the software requirements process in terms of types and sources of requirements at each of the three levels.

4.3.4.1.1 Concept Level

Types

Code	Type	Description
O	Mission Objectives	Motivating goals and objectives.
P_P	Performance (Primary)	Major system performance goals (Minuteman III - accuracy, payload)
P_S	Performance (Secondary)	Other performance considerations (Minuteman III - maintenance, size, timelines, safety, security)
SA	State-of-the-Art	Stated or implied criteria to do "as well as possible" within other system constraints.
GR	Ground Rules	Framework in which system must operate.

Code	Type	Description
		Sources
ES	Existing Systems	Assessment of existing capability - gaps in performance, problems, comparative capability, etc.
CS	Conceptual Studies	Specific consideration of future systems, alternatives, methods, etc.
SA	State-of-the-Art	Assessment of current technology, recent developments, projections, theories, data, etc.
OA	Operations Analysis	Analysis to determine system parameters (war games/strategy, logistics, effectiveness studies, etc).
B	Budget	Anticipated dollars, dollars vs time, dollars vs cost projections.

4.3.4.1.2 System Level

Types

Code	Type	Description
HC	Hardware Characteristics	Performance, operating characteristics, and limitations of hardware black boxes.
AS	Analytical Solutions	Physical/mathematical representation of a system function.
FS	Functional Solutions	Approach (Mechanization) for the implementation of a system function.
P	Procedures	Definition/agreement as to mechanics of system operation.
I	Interface	Definition/agreement for interfaces to elements external to the "system" or between elements within the system.
T	Test	Test/maintenance philosophy; fault isolation levels; testing provisions; test sequences, etc.

Code	Type	Description
		Sources
HD	Hardware Design	Black box hardware development.
TD	Test/Field Data	Assessment of Data on performance of system elements.
AN	Specialized Analysis	Use of tools/disciplines for specialized analysis (phase plane analysis, circuit analysis, reliability analysis, etc).
S	Modeling/Simulation	Application of simulation techniques.
TR	Trade Off's	Hardware/software; cost/performance; etc.
E (H, S, SY)	Engineering (Hardware, Software, System)	System problem solving/designing.
SI	Suggested Improvements	Creative thought based on system experience and exposure.
SS	Selective Study	Application of results of related study activity (direct or indirect).

Note: Concept-level requirements are also a direct source of system level requirements.

4.3.4.1.3 Programming Level

Code	Type	Description
		Types
M	Mechanization	Function descriptions including equations, logic, method, etc.
SE	Sequence	Mode/decision logic, priorities, command/response, coexistence of functions.
T	Timing	Computational frequencies, response times, delays, etc.
A	Accuracy	Scaling, arithmetic precision, function approximations.
C	Communication	Status reporting, telemetry, man/machine.
H	Hardware Operation	Operating characteristics of hardware having a direct software interface (Particularly the computer itself).

Code	Type	Description
CN	Constraints	Ground rules, agreements as to programming standards, restrictions, methods, philosophy, etc.
P	Parameters	Program variables whose values are not fixed/computed by the program.
I	Interface	Direct software/software or software/hardware interface definition.
O	Program Organization	Executive structure, subprogram, interfaces, memory/time partitioning, etc.
	Sources	
S	Modeling/Simulation	Use of simulation techniques to determine mechanization, sequence, etc.
MA	Mathematical Analysis	Use of mathematical techniques, particularly numerical methods, to derive mechanizations.
E (S, SY)	Engineering (Software, System)	Problem solving at a software design level.
TR	Trade Off's	Memory/speed/performance/response/etc; function A/function B.
PD	Program Design	Software engineering, determination of program structure, methods, etc.
W	Work Around	Expedient for "temporarily" overcoming non-software-created problems.
TD	Test Data	Assessment of data on mechanizations/software performance.
SC	Software Capability	Assessment/determination/discovery of capabilities/limitations of software.

Note: Concept and system level requirements are also a direct source of programming level requirements.

4.3.4.2 Change Process. Software requirements can and are changed at any of the three levels or any combination of levels. There are two ingredients in this process: (1) motivation to make the change, and (2) a catalyst for initiating a change. These two characteristics are categorized below.

4.3.4.2.1 Motivation

Code	Type	Description
PC	Problem Correction	Overcome an identified problem due to a programming error, a mechanization error, or a hardware design error.
ID	Implementation Difficulty	Reconcile unexpected, excessive, or prohibitive difficulty, resources, or time to satisfy current requirements.
SI	System Improvement	Improve, in some sense, the system's ability to perform its current role.
SE	System Extension	Extend the capabilities of the system.
SC	System Change	Coordinate software operation with a new or modified system element.

4.3.4.2.2 Catalyst

Code	Type	Description
MW	Make System Work	Change is necessary in order for system to function properly.
SR	Scheduled Revision	Fit changes into previously identified change point.
TO	Trade Off's	Benefit vs cost vs alternatives.
P	Personalities	Interaction of the principals involved in the decision making process.

4.3.4.3 Examples. Figure 13 illustrates the application of the requirements characterized and change process to some examples selected from the Appendix. These are presented simply to further illustrate this method of requirements characterization.

4.4 SOFTWARE TESTING

This section summarizes the software testing process exhibited by the Minuteman III development. Emphasis is placed on the Program Testing activity (see definition below) since it is the most direct form of software validation. In general, specific examples are not cited in the discussion in this section. The reader should refer to the detailed data in the Appendix for examples and substantiating information.

4.4.1 Definitions

For purposes of this study, it was desired to consider the software testing activity in its broadest sense. The following paragraphs define the terms used to identify various phases of this activity. Figure 14 is a diagram of the categories of software testing.

Program Example (See Appendix)	Concept Level		System Level		Programming Level		Motivation/Catalyst
	Type	Source	Type	Source	Type	Source	
1. Data Transrecorder (DTR)	P_s - Provide additional data to aid maintenance/ troubleshooting.	ES - Maintenance process could be improved with additional system data.	HC - DTR hardware operation. I - DTR/D37/AGE interfaces.	HD-DTR design	SE - Conditions for commanded/automatic dump H - DTR operating characteristics	SC - "simple" dump of all scratch - inhibit local cummunication during dump - octal format PD - "fit" DTR dump into existing program structure.	SE/P - OOAMA developed DTR for MM II in order to enhance maintenance. - added to MM III with some hesitation, but hardware was already developed.
2. Simultaneous Attitude Maneuvers (SAM)	O - Deployment of multiple R/V's and penetration aids. P_D - Range/deployment capability.	OA - Analysis to determine weapon system effectiveness of multi R/V's/Pen Aids.	HC, I - Physical, functional, electrical char. of R/S. AS - Mechanization for deployment of R/V's, decoys, chaff. - control system mechanization. FS - Use of simultaneous rather than sequential maneuver.	HD - R/S design. S - Control system analysis/evaluation. - mission time determination SI - idea to use simultaneous instead of sequential mech. AN - Control system design.	M - Control equations. - logic for initiating and terminating a maneuver P - Control system gains and limits. T - Sampling frequency for terminating maneuver.	S - Control System simulation.	SI/TO - Improvement in mission time requirements provides significant benefits in terms of overall system performance.
3. SAO Mode	P_o - Weapon system accuracy directly related to IMU alignment accuracy.	SA - Though numerical accuracy requirements are specified, general requirement is "as good as possible."	P - Procedures associated with use of GCA data.	E(SY) - Define new system submode to avoid use of GCA data prior to calibration.	SE - Definition/ logic for selection of program modes.	E(S) - Modify program logic to implement functions of new modes.	SI/SR - SAC requested this new mode due to lack of confidence in GCA/SAT. The rationale was apparently questionable.
4. Improved Missile (PBPS Fault Isolation)	P_s - Launch reliability - Maintenance time. GR - Maintenance philosophy requires isolation to R/S, PBPS, G&C, downstage.	ES - Test/maintenance philosophy established for MM I/II.	T - Hardware test/ monitoring features - maintenance philosophy. HC - Operating characteristics of control electronics and actuators.	TD - Field data used to refine fault isolation procedures. AN - analysis of circuits to determine test characteristics.	C - Status reporting (results of test). M, T - Detailed test sequences.	TR - Amount of fault isolation and completeness of testing versus required memory.	SI/TO - Amount of testing and degree of fault isolation to be included was generally a trade-off of assessed benefits versus required memory (i.e., versus other uses of available memory).

Figure 13. Requirements Characterization

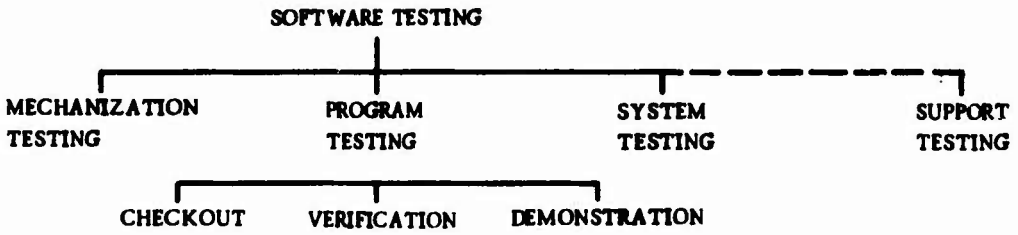


Figure 14. Software Testing Categories

1. Software Testing

Any and all activities which provide (or are designed to provide) evaluation, assessment, or assurance of the feasibility or performance of a software product or a product which directly supports development/testing of a software product.

Software testing is subdivided into four main categories: Mechanization Testing, Program Testing, System Testing, and Support Testing.

2. Mechanization Testing

Any activity designed to assess the feasibility or determine the impact(s) of implementing specific mechanizations in a software product. This does not include testing whose purpose is to establish or derive the mechanization or to verify its analytical feasibility.

3. Program Testing

Any activity whose primary purpose is to evaluate, verify, or demonstrate the performance of a specific software product.

4. System Testing

Any activity whose primary purpose is to evaluate, verify or demonstrate the performance of a "system" of which the software product is a part. A system test activity normally implies that the software product has undergone some prior program testing.

5. Support Testing

Any activity designed to evaluate, demonstrate, or validate a product which is used in direct support to the development/testing of a software product. This is limited to activities which are intended to test the applicability, suitability, acceptability, or performance of the support product within the software development process; i.e., tests which are not directly related to this product's use as a support tool and not included. The Program Testing category is further divided into three subcategories:

a. Checkout

The portion of the Program Testing activity whose purpose is to provide assurance of program continuity and gross performance. This activity is often referred to as program "debugging."

b. Verification

Any program testing designed specifically to assure that the software product satisfies its performance requirements. The distinction between this activity and checkout is in the intended purpose of the testing; i.e., checkout is intended to provide confidence in general program operation whereas verification is intended to provide specific assurance of proper program performance.

c. Demonstration

Any Program Testing whose primary objective is to exhibit (rather than verify or evaluate) performance or operation of the software product. This testing is normally conducted as a condition for delivery or acceptance of the software product.

4.4.2 Testing Activities

Figure 15 is a diagram of the software testing activities for the Minuteman III software development. Gross time relationships are pictured on the diagram and the activity generally proceeds from left to right and from top to bottom. The support testing activities are not shown but are described in Section 4.2 together with the products which they support.

In order to put the Minuteman III testing activities in proper perspective, the history of the overall Minuteman weapon system development must be noted. Since Minuteman III was a modification, or rather an extension of the Minuteman II system, many of the detailed software mechanizations for Minuteman III were directly carried over from Minuteman II software. Moreover, the development process which resulted in the Minuteman III Operational programs involved development and testing of many other versions of the software routines which were ultimately delivered in those programs. (The block changes were the major versions, but various "engineering" versions of these programs were also developed.) Though each version of Minuteman III programs which was "delivered" outside of Autonetics was tested as a separate software product, some reliance on previous testing of the program functions as

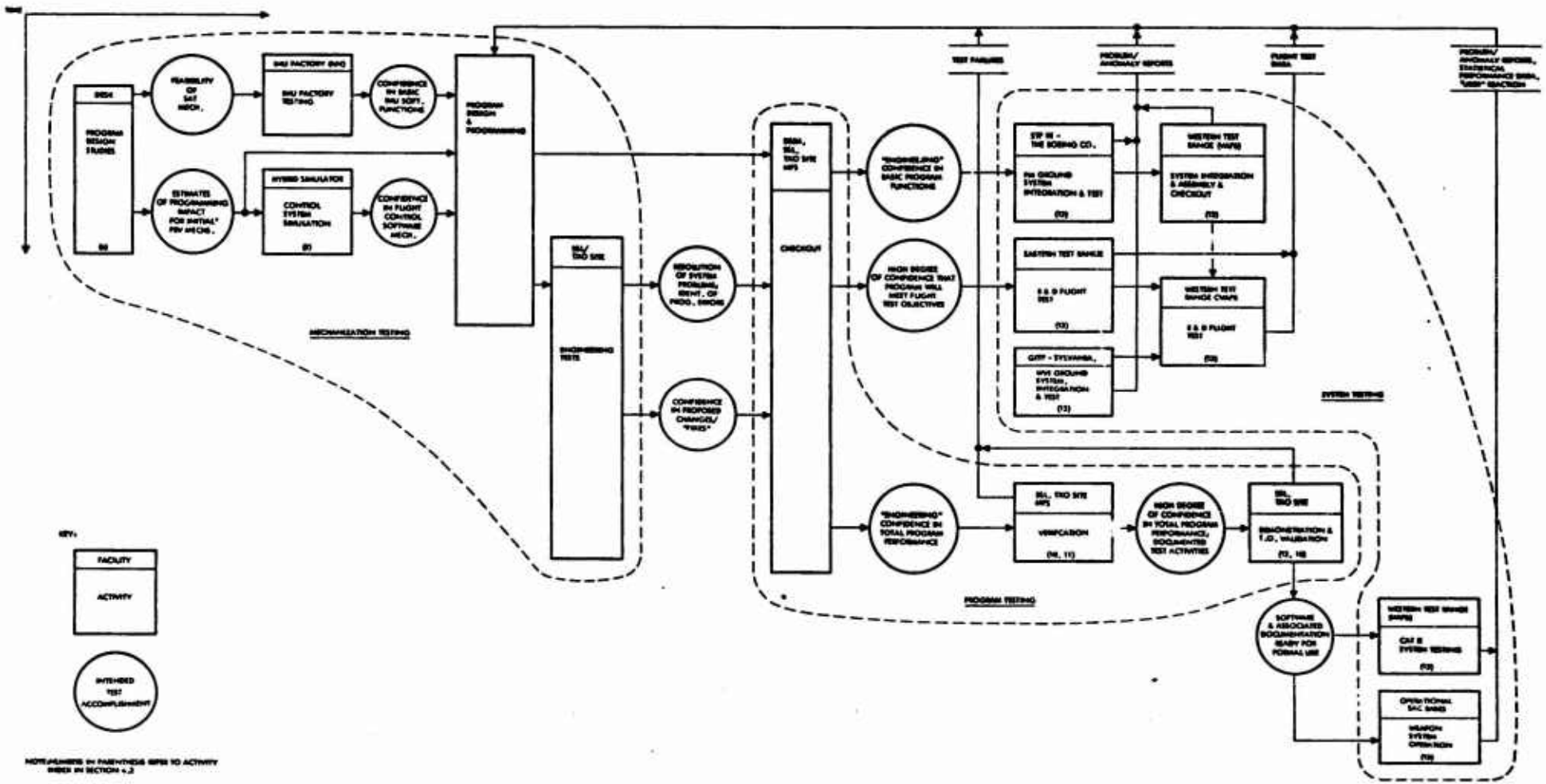


Figure 15. Minuteman III Testing Activities

contained in prior versions of the Minuteman III or Minuteman II programs is at least implicit in the testing process. For example, program functions which were new or significantly modified for a given version of the program were tested more extensively at all levels than functions which were carried over directly from previous programs.

This evolution and multiple program development tends to make the software testing process somewhat confusing to describe and evaluate in a concise, step-by-step manner. For instance, the duration of the development itself introduces confusion due to process evolution/changes, personnel turnover, organizational changes, etc.

Another factor which strongly influenced the software testing process is the previously discussed "system" involvement integral to the software development. Since Autonetics is the guidance and control contractor as well as the software contractor, there is a strong system flavor to the software testing, and the people/organizations involved generally relate to the total system performance rather than just the software. Likewise, isolation of software and system responsibilities and duties, particularly when problems are discovered, is not a major concern from a contractual standpoint as it might be with a separate software contractor.

The following description of the software testing process and activities will generally be organized as if a single software product were being developed since it is this process which is of primary interest in the study. Significant changes which occurred in the testing process during the Minuteman III development will be noted, but the process being described is the one in effect at the time of delivery of the Minuteman III Wing VI Operational Ground Program (-11 Revision), March 1972.

4.4.3 Program Testing

As previously indicated, the term "Program Testing" refers to all direct evaluation of the software product. The program testing activities are of primary concern in understanding and assessing the testing portion of the software development process. While the other phases of testing (particularly System Testing) are vital to the overall development process, the program testing phase, by definition, bears primary responsibility for performance of the software product.

This section describes the Checkout, Verification and Demonstration activities which comprised the program testing process on Minuteman III.

4.4.3.1 Background. Program testing activities underwent considerable growth, change and evolution during the Minuteman III development, and in fact, during the entire history of the Minuteman system. Two significant trends can be seen in this evolution:

1. The relative amount of effort, resources, and concern devoted to the activity has increased significantly.

2. The activity has become increasingly more formalized. (This characteristic is generally true of the entire software development process.)

The most significant change occurred in the Verification activity related to the Ground Program.

During the initial Minuteman III development, this activity was planned and directed by the programming personnel who for all intents and purposes were solely responsible for the amount and type of tests performed. The tests were documented but no specific assessment of the adequacy or completeness of the process was made, other than the assessment implicit in initially designing the tests. (The documentation was reviewed by various personnel but with no significant results.) Verification consisted of site personnel performing the procedures specified in a test plan on the TXO site(s), with the programmers evaluating any unusual results. The sites were generally operated on a three-shift basis and tests were run more or less continuously; i.e., in "slack" times when a program was being debugged, tests of other program functions would be rerun by the site personnel. A confidence level was generally associated with the amount of total calendar time that a given program had been "on the site." Since schedules were never leisurely (overtime operation was more or less standard), as soon as the entire test procedure was successfully completed once, the program was considered ready for demonstration (or delivery in the case of an R&D Program).

The test plan itself was an evolving document which for Minuteman III was derived from the then-current Minuteman II Test Plan(s). The programs being tested were also evolving; i.e., by the time the Operational (Block IV) Program was "verified," large portions of the program had gone through two previous "verifications" (Blocks II and III) in addition to the system testing activity. A new verification process became established during later Minuteman III development, subsequent to the initial delivery of the operational programs. This process evolved from a general trend toward using personnel other than the programmers to perform the verification activity. This trend was motivated by schedule/manpower considerations (the verification planning could be done in parallel with the programming, and the testing in parallel with the program documentation activity) and some feeling that "independent" verification would be beneficial. At one point, a separate unit was established within the software development organization with the charter for verification and development of software aids/tools. However, the personnel were relatively inexperienced and hence relied heavily on the programmers. This unit was disbanded in mid 1970 due to an overall reduction in work-load and funding subsequent to verification of the initial Wing VI Minuteman III operational program. The next program to be delivered was the initial Minuteman II program, called the -211, contracted by OOAMA (previous programs were contracted by SAMSO). Due to changes in contracting policies, it was decided to create a separate Program Requirements Document (PRD) for this program which would contain detailed requirements and thus allow the contractual requirements document, the Figure A, to be kept relatively general. In looking for personnel to assign to the verification on this program, "software system engineers," where available, were a logical choice since they were familiar with the system and the program functions. Likewise, the newly created PRD was a logical choice as a test reference.

The -11 revision to the Wing VI Minuteman III ground program was identified shortly after delivery of the Minuteman II -211 program and used a more formal version of the "new" verification process. In this process, the testing activity was planned, directed, and evaluated solely by "system engineering" personnel within the software development organization. The same personnel who were responsible for developing/documenting software requirements (Part I specification and Program Requirements Document). The tests were specifically designed to verify the program requirements specified in the Program Requirements Document (see Section 4.3.2, Requirements Documentation), and a cross reference index (Activity No. 20, Section 4.2) was prepared which correlates requirements, program description, and specific tests in the test plan document.

After checkout by the programming personnel, the software product was turned over to a completely different set of people for verification. The actual testing was performed by personnel in the System Simulation Laboratory (SSL) and TXO site(s) using the test plan document. Data from the tests was evaluated by the test planning personnel and programmers were consulted for problem analysis and correction.

Discussions of the verification process in subsequent sections of this report will refer to the procedure established with the (-11) activity.

4.4.3.2 Checkout. The primary checkout activity is described in Section 4.2 (Activity No. 9).

Since this activity was not documented to any extent on Minuteman III, it is somewhat difficult to assess the details of the process in any empirical sense. However, the main characteristics of the process can be readily identified.

1. Documentation

The Checkout activities, with only minor exceptions, are not documented in any sense; formally or informally. The lack of documentation includes both the "plan" and the results of the activity. The only significant exception is the documentation by internal letter of the testing of several routines (primarily solution of mathematical equations) using pre-computed test data.* This particular testing was apparently documented primarily because it represented a different approach to testing the particular functions.

2. Structure

The Checkout activity as a whole, is relatively unstructured. There is no stated set of objectives to be accomplished and to a large extent, no real plan established as to what will be done. As the programs/system/programmers have evolved, the general approach taken for checkout of the major program functions achieved a certain amount of stability, but this represents experience and stability of personnel/organizations rather than a structuring of the process.

3. Variation

Due to the lack of overall structure to the Checkout activity, there is considerable variation between programmers and program functions in terms of techniques used and extent of testing. A given programmer may be highly disciplined in conducting checkout of a particular function, whereas another programmer (possibly less experienced) may tend to be rather haphazard. Likewise, one program function may lend itself more readily to highly structured checkout than another; the extent to which the function can be isolated from the rest of the program seems to be of particular significance.

*A summary description of the testing of Missile Test and Terminal Countdown routines was published (Internal Letter). However, this testing is more accurately considered part of Verification since it represents the only specific performance testing of those functions.

4. Relationship to Verification

As defined in the overall process identification, the checkout activity is intended to provide general assurance of program continuity and gross performance, whereas verification is intended to specifically verify software performance. In actual practice, the distinction between the two activities and their relationship in terms of software testing is somewhat vague due to several considerations:

a. The objectives of the checkout were not identified explicitly or implicitly.

b. The process being studied was only applied to a program revision, not a new development.

c. The particular programmers who did the programming and checkout on the -11 revision were relatively experienced and apparently treated the checkout activity as a more or less complete validation (at least of the changed portions of the program).

d. The same tools (TXO sites and SSL) were used for checkout as for verification, and in general the same test methods were used (due apparently to similar background/orientation of the personnel and use of the same tools).

As a result, there was considerable overlap and apparent redundancy in the checkout and verification activities. The appendix contains a summary of the problems discovered and corrected during verification which is of some interest in assessing the checkout/verification relationship. However, since only a program revision was involved, the sample is relatively small.

4.4.3.3 Verification. The primary ingredients of this activity are described in Section 4.2 (Activities No. 10 and 11). This activity was conducted independently for the ground and flight programs and the process differed for the two activities. Therefore the following section will describe these two processes separately.

4.4.3.3.1 Discussion

1. Ground Program

The stated philosophy of the verification testing on the Ground Program was to explicitly verify "conformance to program requirements." While the objective seems relatively simple and concise, the apparent simplicity rapidly disappears when attempting to implement the philosophy.

The first problem was to determine what was to be the reference or standard for identifying "program requirements." The discussion in Section 4.3 shows the difficulty in identifying and particularly in documenting a single, homogeneous set of program "requirements." As previously indicated, the contents of a requirements document(s) varies considerably depending on its intended function. The natural choice of reference documents for the intended verification testing would be the Part I Software Specification (in Minuteman the Figure A, Part I). However, at the time that the new verification process was initiated,

the Figure A was already in existence and being used (and hence designed) for purposes other than verification - primarily as a contracting medium. Likewise, Interface Control Documents which of course also describe/impose/document software requirements were also in existence and again designed for other purposes - a contractual medium and means of establishing communications between contractors.

It was decided that the Figure A could not be used for several reasons:

a. The document did not contain all the requirements which the program had been designed to satisfy.

b. The description of requirements varied in the amount of detail presented and level of requirement described but generally it was felt that insufficient detail (too "high" a requirements level) was included.

c. Modifying the Figure A to suit the purposes of the verification would create considerable difficulty because of the formality and contractual implications of that document.

It was therefore decided to create a separate document called a Program Requirements Document (PRD) which would be a single source of requirements information (including interface requirements) for purposes of verification. This document was formally published but did not require the same approval/coordination as the Figure A.

Since the approach was to explicitly verify documented requirements, the PRD becomes a very significant document in terms of testing. This is generally true and in particular, the level (i.e., requirements levels as described in Section 4.3) at which the requirements are documented can greatly affect the type and quality of testing which results. If requirements are documented at the programming level, the testing tends to evaluate the program structure and implementation whereas requirements documented at the system or concept level tend to result in evaluation of total software/system performance at the expense of detailed program mechanization.

The PRD for the Minuteman III Wing VI Ground Program (-11), was patterned after the document developed for the most recent Minuteman II Wing VI Program (the -211) which was verified with a similar process. The Minuteman III document was of course written after the Minuteman III program was developed. The document is therefore a combination of "requirements" (concept/system level) and "mechanization" (programming level requirements) and is generally organized in a manner consistent with the Program Description Document (PDD) which simplifies the cross reference problem and the task of test design. The test requirements Cross Reference (TRCRI, Activity No. 20, Section 4.2) provides an explicit correlation between the PRD, Figure A and Test Plan.

2. Flight Program

The philosophy of flight program verification did not undergo the same major change as the ground program during the Minuteman III development. Because of the inherent nature of the flight mission, and the reliance on "system-

level" simulation as the primary verification tool, the flight program did not lend itself to explicit testing of individual requirements. Moreover, the flight mission is much more of a straight line, single function flow than the ground and therefore more adaptable to "mission" simulation. As previously indicated, the system involvement created by Autonetics role as G&C Contractor tends to favor this system-level verification. Hence, the flight program verification process relied on analysis of the data from simulation of multiple (5) flight missions. Three significant extensions of the verification process were added for verification of the Operational Flight Program.

a. The flight missions (trajectories and mission events) were specifically designed/selected for verification purposes. Previous R&D programs had used actual flight test mission profiles for verification.

b. A cross reference (TRCRI, Activity No. 20) was prepared which correlated Figure A, Part I requirements with test data.

c. Simulations were performed using the control system hybrid simulator (see Activity No. 3, Section 4.2) to verify flight control functions for specific portions of the test mission.

In preparing the TRCRI document, it was necessary to design some explicit tests. This was generally done by modifying or extending control parameters in the simulation to output the specific data needed to verify a particular requirement. In general, the cross reference consisted of identifying the particular portion of the simulation data which exhibited the operation of the required function.

Overall then, the philosophy of verification differs between flight and ground programs in that one relies on verification of the primary system function of the program whereas the other attempts to explicitly test individual program functions. In actual practice, the two processes do not differ as much as might be expected for the difference in philosophy. Two factors tend to reduce the differences:

a. In some cases, the design/performance of ground program verification tests uses "system level" functions/interfaces to create the specific test conditions desired. For example, most of the testing of message processing program functions is conducted by simulated transmission of actual LCF/LF messages and subsequent "receipt" of LF replies (as opposed to, for instance, an analysis/test of internal program flow).

b. Flight program verification does contain some exhibit testing of individual requirements. In particular, some equations are verified by comparison of hand calculations of a specific test solution and timing requirements are sometimes verified by introducing explicit simulation outputs triggered by specific program events.

Nevertheless the two processes represent some basic differences in approach and technique.

4.4.3.3.2 Test Design

1. Ground Program

As previously indicated, specific ground program verification tests were designed, and the results analyzed by software system engineering personnel. In many cases, the same people who designed the verification tests also wrote the Program Requirements Document. The program was subdivided by major function (IMU, communications, etc) and one or more people assigned to each function. In no case did the programmers actually write the tests and generally speaking, the test designers were not aware of the details of the previous checkout activities.

No specific ground rules were established for the test design process. Since the tools to be used for the testing were already in existence and could not be modified significantly for the verification testing, some design constraints were implicitly imposed. However, the specific methods used to test the individual program functions were at the discretion of the particular test designer within the confines of the available test tools. Note that these tools were the same as those used for checkout. The test designers had access to, and made extensive use of detailed programming level information, particularly the Level III flow charts (lowest level detailed program flow and decisions) and program listings. They also consulted the programmers concerning program details. For example, the tests of IMU alignment sequences utilizes an internal program "flag" as the primary indication of completion of various steps in the sequence (Coarse Zeta example in Appendix) and in testing a limit check in the precision time calculation, a program branch is "forced" by modifying the limit value stored in the program (Tau example in appendix). However, the test techniques varied considerably. For example, in one case a logical branch (decision point) in the program might be verified by establishing the external (system) conditions necessary to cause the program to execute the branch, whereas in other cases the program might be modified (with "Key-ins") to directly "force" the branch. Likewise, one equation might be tested by recording specific conditions and analyzing the results of the computation and another equation might be verified more or less "implicitly" by observing a large function of which the equation is a part.

Aside from the cross checking related to the TRCRI, there were no specific measures of test completeness or test quality established for the verification process. The extent of the testing varied from one function to another and, though not explicitly stated, previous software/system testing was apparently implicitly considered in determining the extent of testing necessary. Schedule pressure, both in terms of test design and required test time was undoubtedly also a factor.

2. Flight Program

The test design process for flight program verification was largely inherent in the selection of the basic approach; i.e., mission simulation, and the choice/construction of the specific mission trajectories/configurations to be used.

Design of the test missions was a joint effort between Autonetics software and software systems engineering, and TRW targeting and system engineering personnel. Though consideration of detailed flight program implementation was implicitly given, the primary approach was to attempt to cover the "envelope" of possible operational flight missions with "off-nominal" conditions introduced to simulate variable vehicle performance characteristics.

Once the missions were established, TRW used their targeting program to generate the necessary mission profile information which was then used in the MFS simulation. Actual performance of the testing consisted of analysis by the programming organization of the simulation data, in particular, deployment accuracies and event sequence and timing. The cross reference activity served as a check on the completeness of the simulation and a reference document for the data.

SECTION V

ANALYSIS

The primary purpose of the Software Validation Study was to identify the specific process used in the development of a current (and hopefully representative) space-borne software product. The process used for Minuteman III Operational Ground and Flight software has been described in the preceding sections of this report. This section attempts to summarize and evaluate this process in terms of overall quality, effectiveness, and potential for improvement.

5.1 SUMMARY

In assessing the overall software development process, two characteristics of the Minuteman system development tend to cloud the analysis.

1. It is difficult to isolate the software development process from the system development, particularly during initial system R&D.

2. It is difficult to isolate the software development process in time. The Minuteman III development encompasses more than five years and the software was to some extent, "carried over" from Minuteman II.

These system development characteristics seem to represent both causes and symptoms of the primary software development characteristics. The Minuteman software development process has evolved in response to "system" organization, procedures, and needs rather than from considerations of software as an independent product. The most meaningful overall assessment of software quality would appear to be that the software has generally met the needs of the weapon system during development and deployment. Though software problems have been uncovered on a not infrequent basis, no Minuteman III test flights have been aborted due to software errors and software capability has adequately supported system operation. Though this software development process is supporting the current needs of the Minuteman system and the rate of discovery of errors in operational software is currently very low (but not zero), concern over software quality has been and is still in evidence. Concern over software quality/cost is certainly warranted when one considers the total amount of resources expended in testing the Minuteman III software and attempts to extrapolate to the software development process for future airborne systems of considerably greater size and complexity.

In order to put the following discussion in proper perspective, it must be remembered that the process being described is currently applied to a very mature system. Though the process has evolved, there was rarely the opportunity to "start from scratch" in any aspect of the process; hence, there is a strong implicit emphasis on use/improvement of past experience and methods.

Many specific problems can be identified in the Minuteman software development process such as the continual pressure to meet "unrealistic" schedules or the inefficiency in design and operation of the testing tools, but these represent mainly difficulties in implementing the mechanics of the current process. Some of these

specifics will be discussed briefly in subsequent sections, but the primary emphasis is placed on an analysis of the inherent characteristics of the process and the implications of those characteristics.

5.2 REQUIREMENTS DEVELOPMENT

As described in previous sections, the requirements development process typified by Minuteman III should not be visualized as an isolated step in a sequential process, but as a continuous, iterative, evolutionary process dealing with broad concept level requirements on one hand and detailed programming level requirements on the other.

The process of identifying/deriving/designing software requirements involves many different engineering/scientific disciplines and techniques and is quite creative in many respects. As can be seen from the discussions in Section 4.3 and the examples described in the appendix, the process involves iteration and tradeoffs in order to achieve a compromise between system performance desires/objectives, subsystem characteristics/performance, programming impact, analytical techniques, etc. Because of the range of disciplines involved and the tradeoffs necessary to achieve an "acceptable" end product, communication is the major factor which determines the effectiveness of the process. The reliance on TI meetings and informal "engineering" as the primary means of establishing requirements appear to be indicative of the nature of the process involved. Likewise, the reorganization of the software system engineering activity (see Activity No. 4, Section 4.2) which shortened communication lines and reduced the organizational formality of the "requirements"/programming interface is generally considered to be an improvement of the overall software requirements development process.

While formal techniques such as the System Requirements Analysis activity (Activity No. 2) were helpful in initially organizing the elements and functions of the Minuteman System, there is no evidence to indicate that the software requirements development process is improved generally by formalized procedures and control. In fact, in so far as the formalization inhibits or burdens the necessary communication, it may be detrimental to the early requirements development phase. However, some formal control is obviously necessary in order to manage the requirements process and maintain the software. Determining the level and type of control/documentation appropriate to the requirements process necessitates an understanding of the characteristics of the process and the specific functions to be performed. The variation in requirements documentation (and implicitly in control over the process) discussed in Section 4.3.2 is apparently indicative of a lack or variation, of definition/understanding of the specific functions which requirements identification/documentation serve within the overall development process. In particular, five relatively distinct functions are evident in the Minuteman process. These functions, (1) programming support, (2) contracting/organization, (3) baseline, (4) test support, and (5) operation/maintenance support are discussed individually in the following sections.

5.2.1 Programming Support

Obviously, a set of software requirements must be identified to the programmer in order for the software product to be programmed. The classic vehicle for this function is the Part I software specification (in Minuteman, the Figure A, Part I).

In examining the Minuteman process, it becomes clear that with only minor exceptions, this document has never been the means by which the programmers determined "requirements. This situation can be readily understood by reviewing the nature of the requirements process described previously. The iterative process involved in originating requirements includes the programming level, hence, programmers. Therefore, prior to the existence of the necessary information for the Part I specification, the programmers are already knowledgeable of the requirements, having been involved in the process of deriving the information. In fact, due to the priority attached to maintaining a tight software development schedule, the requirements were normally programmed prior to the Part I specification being published.

This situation is not simply the result of lack of planning/control over the software development as might be concluded by comparing this process with the classic description of the software process. Regardless of when and where the Part I (and Part II) software requirements are published, this documentation activity is not the source of the information, and unless a very formal contracting interface is involved (which was not the case on Minuteman) between requirements development and software development, this document is not the vehicle for transmitting requirements to programmers.

The actual sources of requirements information to support the programming of a given Minuteman software product were:

1. Previous Programs; i.e., the program listings, detailed flow charts and the actual program code from previous versions of the Minuteman III or Minuteman II programs.

2. A generally informal decision-making process involving technical interchange between programmers, system engineers, analysts, etc, from Autonetics and TRW. Sketchy documentation was published in the form of meeting minutes and letters.

3. Results of specific analysis such as the flight control system equations, PBPS Test Sequences, and various IMU mechanizations. This information was generally documented by informal letters.

4. Reference documentation; i.e., published descriptions of hardware operating characteristics and procedures, interface agreements, etc. Note that Interface Control Documentation (ICD) tended to be somewhat more significant as an actual source of requirements than the Part I specifications since they represented agreements with other contractors. The primary type of information for which the ICD's served as a source of programming requirements were detailed operating characteristics (particularly timing and data formats) of the non-Autonetics hardware and constraints imposed by the non-Autonetics side of the interface. Much of the information published in the ICD's was already known by the programmers as a result of informal contact between the involved engineers. Likewise, when questions arose, the programmers would typically talk directly with the technical people who had the information rather than attempt to get the information clarified in the ICD's.

5.2.2 Contracting Organization

In organizing the overall system development, it is necessary to partition responsibilities both between and within contractors. In terms of software requirements this results in:

1. Necessity to establish and control specific "interface requirements" so as to limit the interaction of two independently contracted elements. This includes system contracting subdivisions such as the re-entry system and the ground communications systems as well as hardware/software interfaces such as the computer, IMU, and silo ground equipment. This need is satisfied by freezing/restricting hardware design changes and by establishing formal agreements as to interface characteristics. This information is documented in hardware reference manuals and Interface Control documentation.

2. Need to establish a vehicle for customer identification, control, and acceptance of the software product. This function is normally served by the software end item specification.

Both of these functions require a more or less arbitrary definition of a requirements/solution relationship for some set of software requirements. This particular function of requirements identification is the only one where such a definition is meaningful and necessary.

In the case of interface definition, the resultant software requirements are generally in the form of constraints or ground rules which are "enforced" on the software in order to avoid or reduce subsequent hardware changes. These become part of the framework in which the software must operate as opposed to functions it must perform. In fact, they often are not thought of as software requirements since they tend to get absorbed in the overall framework in which the software is developed. However, if one examines the early software requirements activity related to the re-entry system (see SAM and Plume examples in Appendix) or even the early design of the Minuteman III computer (particularly the I/O), it is clear that this type of requirement differs only in its relative inflexibility and, once established, its relative stability.

The identification of software requirements for contracting purposes, represents a less clearly defined function. The point in the requirements development process (relative to the various levels of requirements) where the requirements/solution line is drawn has varied considerably based on the sophistication and amount of customer software interest/involvement, the formality and general status of the customer/contractor relationship, and the maturity of the system. On Minuteman III, customer control over software requirements was exercised primarily though actual participation in, and review of, the technical system engineering activities in the requirements development process rather than through careful control of the Figure A documentation. *

*TRW, as the customer's (SAMSO) technical advisor, was directly involved in the software requirements process.

5.2.3 Baseline

At various stages during the software development process, particularly R&D block change points, it was necessary to establish a requirements baseline in order to support software development schedules. This is primarily a matter of making choices as to alternate software mechanizations and evaluating overall hardware and software development schedules for potential additives. This process was very informal during Minuteman III development, being handled at TI meetings and documented primarily in meeting minutes. Generally, definition of a requirements baseline after the initial development consists of identifying changes to the previous baseline. After the first end item version of the program is delivered, the ECS/ECP (contract change documentation) becomes the vehicle for establishing a new requirements baseline (as well as the contracting vehicle).

5 2.4 Test Support

All testing requires a reference or standard of performance against which to evaluate the test results and/or design the tests. This test standard is not always explicitly stated (as for instance, in the Checkout activity), but explicit or implicit, a standard must exist. The need for test standards creates another function for requirements identification since software performance standards are a subset of software requirements. The level and type of requirements selected as the test standard varies with the particular test objectives, test methods, and general test philosophy. There is a two-way relationship between the selection of requirements and the test design process. The testability of requirements and available test methods/tools affects the selection of requirements (level and type). Likewise, the selection of requirements affects test methods and design of tools. If requirements are specified at a detailed programming level in the test standard, resultant testing tends to verify detailed program structure. Whereas, if the requirements are specified at a system or subsystem level, the testing tends to be oriented toward verifying overall system or subsystem functions. The Minuteman III (-11) PRD, though not consistent, generally describes requirements at a LF subsystem or IMU subsystem level. For example, a portion of the requirements for Coarse Zeta (see example in appendix) reads, "After slewing the GCA through $\emptyset_G$ (nominal), LD No. 2 [level detector No. 2] shall be selected and the platform shall be torqued at maximum rate about Z_p from LD No. 4 to LD No. 2." The testing of these requirements consists of running a GCA alignment sequence on the SSL and verifying the occurrence and timing of the events; slew through $\emptyset_G$, select LD No. 2, torque platform to LD No. 2. The requirements could have been specified (and the testing performed) at either a higher (system) level or lower (programming) level. At a slightly higher level, the coarse zeta requirements would be buried in an overall requirement to align the platform and GCA to a specific accuracy within a specified amount of time. At a lower level, the requirements would describe the detailed program flow (program "flags," branch points, etc) used to implement the Coarse Zeta function.

The implications of testing to the requirements specified at higher or lower levels if fairly obvious, but they can be summarized by saying that testing at higher levels provides emphasis on verification of overall system or subsystem functions at a sacrifice of detailed software implementation. The higher levels of testing tend to be effective at detecting errors in the analytical or functional mechanization (within the confines of the testing tools), but are inefficient/ineffective at detecting subtle programming errors (see Tau Problem and PLC/Fuzing Race Problem in appendix).

Likewise, testing at a lower level, emphasizes verification of the program structure and detailed programming at the expense of system/subsystem operation.

5.2.5 Operation/Maintenance Support

Since the Minuteman software development supports a continuing system operation (as opposed to a fixed number of missions), there is a significant need for requirements identification/documentation for purposes of operational/system test usage of the system and subsequent maintenance (revisions) to the software.

This function often demands a combination of requirements levels and types, but generally system level rather than programming level requirements are desired for two reasons:

1. For purposes of support to system operation, the details of the software are only important insofar as they affect or determine overall system operating characteristics. Generally, "users" of the system are hindered rather than helped by detailed programming level requirements.

2. For maintenance purposes, the program listing is an adequate source of programming level requirements. The system level requirements which bound the flexibility of the programming level requirements are often the most difficult to re-identify; i.e., the detailed requirements for a given function are known but the "background" (higher level) requirements and constraints are forgotten, and hence the framework in which to change the detailed requirements must be reconstructed.

5.2.6 Evaluation

The preceding paragraphs have identified and discussed the five primary purposes/functions which the requirements development process supports. In attempting to assess how well the Minuteman software development process supports these functions, several observations appear pertinent.

The process of determining requirements in support of programming, once organized to maximize communications; i.e., dedicated to software (rather than system) requirements and organized in the same area as the programming, was relatively effective (though somewhat sensitive to the personalities and abilities of the personnel involved).

Generally, documentation of Minuteman software requirements has been somewhat inconsistent between different software products, at different times in the overall system development, and between different functional areas of the software. The inconsistency results primarily from the personalities involved and the most pressing concerns at the time, and generally reflects a lack of understanding/definition of the specific functions which the documentation is intended to serve. This can readily be demonstrated by the lack of specific groundrules and guidelines as to the content, level of requirements, type of requirements, etc, applied to the ICD's, Figure A, and PRD. The primary cause/effect of this inconsistency is related to problems with the testing process. These problems will be discussed in the next section.

5.3 SOFTWARE TESTING

5.3.1 Summary

There are several pertinent questions to be answered in evaluating the program testing process exhibited in the Minuteman software development. *

1. How good is the current process?

In particular, what is the overall quality of the software end product? As previously indicated, the only available answer to this question is that the quality of the software product has been adequate to support the needs of the Minuteman Weapon System. In answering this question, one is immediately confronted with the problem of assessing the quality of the software. This raises the second question.

2. How well can the quality of the product be measured?

In particular, how well does the process lend itself to measurement? These questions must be answered with a "very poorly." About the only real means of measuring software quality with the current process is to evaluate the number and type of test flights flown and the amount of total system usage. One could also attempt to extrapolate from a statistical analysis of the frequency/type of software problems discovered over the Minuteman system development but correlating this information with process variations would be difficult. Assurance of quality is apparently only gained by actual use of the product which raises the third question.

3. What is the desired/planned level to which the product is to be tested?

Since almost infinite resources can be spent in testing the software, a decision must be made as to what objective(s) the testing is to satisfy. On Minuteman, the implicit objective was to provide as much confidence as possible that the combination of software and Autonetics built hardware meets its performance objectives within the system and conforms to all interface constraints defined in applicable software interface control documentation. It is not clear whether this situation was the result of definite decision or simply of system evolution.

4. What is the "return on investment" for the resources devoted to testing?

That is, in attempting to achieve the desired level of testing, how efficient is the process in terms of resources expended versus software quality? Again, only a more or less subjective answer is possible, but it appears that a rather unfavorable conclusion can be supported. There are two significant Minuteman system characteristics which strongly influence this situation. The severe computer memory limitations and the non-modular computer/software organization force considerable "juggling" of functions/routines and hence, considerable retesting. However, the testing process itself is distinctly not designed to maximize the return on investment in testing.

*The following discussion is centered around the program testing process conducted by Autonetics on the Minuteman software. It is not an assessment of the overall Minuteman weapon system testing.

5. Is the current process adequate for future needs?

The answer to this question is a highly probable NO! A gross extrapolation of the resources (time and money) required to test a software product of several times the size and complexity in a system less amenable to extensive total "mission" testing (such as future space projects), at least results in concern.

Unlike the requirements development process, the testing process lends itself reasonably well to formalization and structuring. The only aspect of testing for which this statement is untrue is true "debugging;" i.e., problem analysis. The fact that the Minuteman software development process does not exhibit this formalization is the major shortcoming in the testing process.

5.3.2 Evaluation

There are two characteristics of the Minuteman software testing process which primarily determine and limit its effectiveness.

1. System Orientation

The previously discussed system involvement/orientation of the personnel and activities is evident in all phases of software testing. Hence, the vast majority of the testing is in fact system (or subsystem) level; i.e., testing of system level requirements, regardless of whether it is conducted by programmers or system engineers and regardless of its implicit or explicit test objectives.

2. Lack of Formalization

The software testing process can be categorized as generally unstructured and undisciplined. This is evidenced immediately by the lack of explicitly stated rules, guidelines, etc, as to specific goals, methods, extent of testing, types/levels of requirements, etc, for each major testing activity.* Likewise, the inconsistency in testing between the people and program functions further exhibits the lack of overall structure. Moreover, very little explicit reliance is made on previous testing when planning a given test activity. This is particularly true between Checkout and Verification.

The result of these two characteristics is mainly inefficiency in performing a given degree of testing. Primary G&C system functions are tested extensively at a system level - much of the testing being redundant since it is merely a repetition of the same tests. However, no specific assurance is provided that all program branches have been executed or that the accuracy of all equation mechanizations has been verified. Testing of program structure; i.e., programming level requirements, appears to be the area where the greatest inefficiency and inadequacy exists. Though any programming level requirements can potentially be tested at a system-level, the amount of test resources required and the low probability of test completeness make this a very inefficient approach. This is particularly true of Constraint and Program Organization type requirements. The "Tau Problem" and the "PLC/Fuzing Race Problem" (see examples in Appendix) are two examples of software errors which survived considerable testing and which can only be reasonably tested at a program structure level.

*While some guidelines may be generally accepted, no specific procedures are consistently applied or enforced.

5.3.3 Testing Tools

The area of software testing tools, particularly various simulation facilities, has received considerable attention over the years. While these tools are quite significant in terms of impact on test efficiency and quality, their primary shortcomings, as evident in the current Minuteman process are the same as those previously identified for software testing generally and in fact, reflect the same basic problem. That is, their design, development, and use has been relatively unstructured and undisciplined. Beyond the basic design objectives, the simulation tools tend to evolve primarily from within; i.e., based on desires/ideas of simulation developers rather than identification of specific objectives in the software testing process. This is obviously related to the lack of structure and formalization in the testing process itself; i.e., it is fairly difficult to design tools to satisfy test objectives/techniques which are not explicitly identified. Also, simulation facilities involving "hands-on" hardware have had a definite tendency toward undisciplined use due apparently to the informal control, ready access by engineers, and the "fun" of using them.

5.4 RECOMMENDATIONS

There has been considerable effort expended in the software community in recent years toward improving the software development process. Much of this effort has been directed toward, or at least resulted in, formalizing the documentation rather than the development process itself; for instance, how testing is to be documented and cross referenced rather than how it is to be designed and conducted. Further, the effort has been oriented toward the process as seen by a software customer rather than a software developer; hence, emphasis has been placed on controls and management aspects of software development. While this work was necessary and valuable, analysis of the Minuteman software development process indicates that a shift in emphasis may be appropriate in the future.

The current emphasis is partly the result of attempting to extend procedures/processes designed for development of hardware to the software development problem. An analogy between hardware and software development must account for one distinct difference. Almost the entire software development process involves design, analogous to building a hardware breadboard. The activity (and particularly controls and documentation) related to the production phase of hardware development does not really apply to software. (Software "production" is simply reproducing the card deck or punched tape version of the program.)

5.4.1 Requirements

The recent trend in the Minuteman system and the software community generally appears to be toward formalizing the content and control of the software end item specification, apparently in an attempt to eliminate or at least reduce the characteristic "flailing around" which the early requirements development process exhibits. However, it appears that to a great extent this characteristic is inherent to the nature of the process and attempting to eliminate it by control and formalization serves primarily to further burden the process with little benefit to either the quality or control of the process. It appears that real control over this process during early development can only be achieved by in-depth participation, which may not be practical at a contracting level for a large system.

It seems appropriate to emphasize organizing the early requirements development process for maximum communication rather than maximum control. The control should be applied to the software product itself since it is really the best indication of the total requirements.

It appears that the activity associated with Part I requirements should be directed toward identifying/documenting only "constraining" requirements rather than attempting to document the total range of requirements in the form of specific constraints. In particular, interface requirements and primary performance goals (requirements) are the most significant. This subset of the total requirements would provide the basis/framework for the initial software development as well as for making subsequent changes to the system/software.

5.4.2 Testing

The primary problem with the software testing process is fairly obvious; namely, the lack of formalization in terms of test design, division of test objectives, and test standards. Without such formalization, it is impossible to measure the effectiveness and to reduce the amount of redundancy and inconsistency of the testing process.

Starting from definitions of requirements types, levels and sources such as presented in Section 4.3, and using specific examples from a current software development such as Minuteman III, one could construct a specific correlation between requirements types, levels, and sources and appropriate testing levels, testing methods, and test standards which would attempt to optimize the ratio of test effectiveness to testing resources. This formalized structure would then provide a basis for establishing specific test objectives, designing/evaluating testing tools, and measuring/refining the process.

SECTION VI

DEFINITION OF TERMS

AAU	Angular Accelerometer Unit
A&CO	Assembly and Checkout
AFWL	Air Force Weapons Laboratory, New Mexico
AFQA	Air Force Quality Assurance Organization
AGE	Aerospace Ground Equipment
AN	Autonetics, Division of North American Rockwell Corporation
ANQA	Autonetics Quality Assurance Organization
CDR	Critical Design Review
CEP	Circular Error of Probability
CLAMPS	Closed Loop Advanced Minuteman Power System Simulation
CLIP	Cancel Launch-in-Process
C18, C19	Designator for R&D Ground Equipment for System Level Test
C53	Designator for the FM Operational Ground Equipment
C163	Designator for the Wing VI Operational Ground Equipment
DAG	Deployment Attitude Guidance
DCU	Digital Computer Unit
DEMO	Demonstration
DSIM	D37D Computer Simulator
DTP	Detailed Test Plan
DTR	Data Transrecorder
D37C	Minuteman II Digital Airborne Computer
D37D	Minuteman III Digital Airborne Computer
ECS	Engineering Change Summary
ETR	Eastern Test Range, Cape Kennedy, Fl.
FACI	First Article Configuration Inspection
Figure A	Basic Requirement Document for Hardware Contract End Item
FM	Force Modernization
FTM	Flight Test Missile
G&C	Guidance and Control
GCA	Gyrocompass Assembly
GFP	Government Furnished Property
GITP	Ground Integration Test Program
GND	Ground
HAF	High Altitude Fuzing
ICD	Interface Control Document
ICBM	Intercontinental Ballistic Missile
IL	Inadvertent Launch
IMU	Inertial Measurement Unit
IOC	Initial Operational Capability

LCF	Launch Control Facility
LF	Launch Facility
MFS	Missile Flight Simulator
MIRV	Multiple Independent Re-Entry Vehicle
MGS	Missile Guidance Set
MM	Minuteman
MOTP	Minuteman Operational Targeting Program
NSA	National Security Agency
NS10	Navigator for the Minuteman I Missile Guidance Set
NS17	Navigator for the Minuteman II Missile Guidance Set
NS20	Navigator for the Minuteman III Missile Guidance Set
OOAMA	Ogden Office Air Force Material Agency
OSR	Operational Status Reply
PAT	Plume Avoidance Thrust
PBCS	Post-Boost Control System
PBPS	Post-Boost Propulsion System
PBV	Post-Boost Vehicle
PDD	Program Description Document
PDR	Preliminary Design Review
PENAIDS	Penetration Aid Subsystem
PIGA	Pendulous Integrating Gyroscope Accelerometer
PLD	Program Listing Document
PRD	Program Requirements Document
P92	Designator for the Amplifier Assembly for Downstage Flight Control Signal
R/S	Re-entry System
R/V	Re-entry Vehicle
SAC	Strategic Air Command
SAM	Simultaneous Attitude Maneuver
SAMSO	Space and Missile Systems Organization
SAT	Self Alignment Technique
SDR	System Design Review
SETD	System Engineering Technical Direction Div., TRW, Norton
SRA	System Requirements Analysis
SSL	Systems Simulation Laboratory
STE	Special Test Equipment
STP III	Seattle Test Program, Part III
T.I.	Technical Interchange
T.O.	Technical Order
TRCRI	Test Requirements Cross Reference Index
TRW	Thompson-RAMO Wooldridge, Inc.
TXO	Tape Checkout
UL	Unauthorized Launch

VAFB	Vandenberg Air Force Base, California (Also WTR)
V&V	Validation and Verification
WS	Weapon System
WTR	Western Test Range

APPENDIX

ANALYSIS OF PROGRAM EXAMPLES

APPENDIX

ANALYSIS OF PROGRAM EXAMPLES

1.0 SUMMARY OF EXAMPLES

This section provides a summary description of 38 examples of requirements, hardware, or program changes which occurred during the Minuteman III development. For each example a brief description is provided together with an indication of the type of change and the functional area of software involved. The abbreviations used in the summary are defined below.

FUNCTIONAL AREA		
1.	IMU	I
2.	MSG. PROCESSING	MP
3.	MISSILE TEST	MT
4.	FLIGHT GUIDANCE	FG
5.	FLIGHT CONTROL	FC
6.	GROUND - OTHER	GO
7.	FLIGHT - OTHER	FO

TYPE OF CHANGE		
1.	PROGRAMMING ERROR	PE
2.	NEW REQUIREMENT (ADDITIONAL CAPABILITY)	NR
3.	HARDWARE CHANGE	HC
4.	MECHANIZATION ERROR	ME
5.	MECHANIZATION IMPROVEMENT	MI
6.	MODIFIED REQUIREMENT	MR
7.	MEMORY OPTIMIZATION (SCRUBBING)	MO
8.	NEW DATA UNCOVERED	ND
9.	WORKAROUND	W

The examples marked with an asterisk (*) are ones that were selected for more detailed analysis. Ones marked with a dollar sign ($) were selected as alternates.

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
*Simultaneous Attitude Maneuvers (SAM)	The initial mechanization for post-boost attitude maneuvers used sequential, single-axis rotations; this was a relatively straightforward approach, but there was considerable concern over required mission times due to considerations of battery life, heat, propellant, etc. A study was initiated to determine the potential mission time savings from a simultaneous 3-axis mechanization. The SAM mechanization was scheduled for Block III. Among other things, the new mechanization caused an increase in the number of control gain change points primarily to deal with increased "overshoot" problems, and indirectly resulted in a hardware change (see Roll Torquer motor).	FC,FG	MI
"C" Mission	The requirement for a particular mission capability was deleted from the weapon system criteria. Considerable effort had been expended, and many problems encountered in attempting to implement this complex mission. Apparently, the reasons behind the deletion were a combination of R/S design problems, schedule pressures and lack of a really compelling need from an operations analysis standpoint.	FG,FC	MR
GBI Sampling	Analysis of post-flight data from R&D flight test uncovered "erroneous" platform bottoming indications (GBI's) during large pitch angle maneuvers caused by physical resistence to platform motion by wire bundles being erroneously sensed as a platform stop. The program uses the GBI indication as one of the flight safety checks which are conditions for arming the warhead. As a result of this problem, the mechanization was changed to require 3 successive GBI's instead of just one before the flight is aborted.	FO	ND,MI
GCA Servo Change	A thorough reliability analysis was conducted on MM II hardware as a result of the "recovery" program instituted to improve system reliability. As a result of this analysis, the GCA servo system was redesigned and scheduled into MM III at the Block III change point. Two program changes resulted from this hardware change (See GCA Slew Changes and Coarse Zeta Check).	I	HC

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
Deployment Attitude Guidance (DAG)	The requirement for small translational maneuvers prior to deployment of R/V's, chaff and decoys was added early in the MM III development (part of Block II requirements). There was considerable activity and lots of iterations involved in selecting and refining the mechanizations necessary to accomplish this. The engine configuration on the Post-Boost Vehicle did not allow a translation of the vehicle position without attitude control axis cross-coupling, hence there was considerable program logic required for selection of the sequence of maneuvers necessary to accomplish the desired translation.	FG,FC	NR
*Plume/Chaff	Engine plume impingement effects on decoy/RV/chaff deployment was recognized as a potential problem early in the MM III development. There was some disagreement as to who had responsibility to work the problem --Autonetics, R/S contractor, or TRW. Studies were conducted at various places. Chaff dispensing became recognized as the major problem. Many relatively minor program changes/additions were made in this area during the flight tape development. This area exhibits extensive iteration of requirements/mechanizations.	FG,FC	MR
*New Gravity Model	The MM II gravity model which was carried over to MM III was a relatively simple mechanization. Errors induced in the flight guidance computations due to this simplified model were compensated for by "biasing" the targeting computations. With the increased targeting problems on MM III (multiple RV's), it was suggested that the flight program gravity model be upgraded to simplify the targeting (this would also produce somewhat improved accuracy on "off-nominal" trajectories, since the targeting biases were based on the nominal). This change was originally scheduled for Block III, but programming the new model turned out to require a major program change to "fit it in" to the timing structure; the implementation was therefore rescheduled for Block IV. It was subsequently "scrubbed" as a result of the memory optimization studies and, though it was coded and debugged, it did not go into the operational program.	FG	MR,MI
Roll Torquer Motor	Simulation of the Block III flight program during development in the SSL uncovered questionable performance during Simultaneous Attitude Maneuver (see separate description). IMU system engineering organization did an analysis of platform performance in this environment and	F	HC,W

Implementation of the SAM mechanization (see separate description), triggered concern over potentially more severe requirements on platform servo loops.

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
Roll Torquer Motor (continued)	discovered a potential problem with roll torquer motor saturation. As it turned out, the potential problem was not related to the SAM mechanization as such, but to the larger angle maneuvers required of Block III missions. A hardware change was prepared but could not be implemented in time to support early Block III flight tests (the SAM requirement was added at Block III). As a result, a special version of the flight program (called Block IIA) was developed to support early "Block III" flight test objectives without the torquer motor change.		
Roll Gimbal Stop	The roll axis platform gimbal stop was reoriented 90° to center the roll axis rotational freedom about the +Y missile axis rather than the +X axis. It is not completely clear why the original orientation was chosen, but an analysis of certain post-boost maneuvers occurring on a due-west alignment and/or launch required additional rotational freedom in one direction.	-	HC
*TAU Problem	This is a current problem discovered recently in the Wing VI Program. This problem resulted from a programming error in the computation of tau, the ratio between the precision time source and the airborne computer timing. The error was such that it induced a bias in the calculation which was very small (or zero) for "nominal" systems but increased non-linearly with system deviation from nomimal. The discrepancy was noted (more of less accidentally) in analysis of data from VAFB which was taken on an unusally "off-nominal" system. The error was isolated by attempting to reproduce the data on the TXO site. The programming error was very subtle, related to a tricky timing relationship in the computer which was inadequately described in the programming reference document. The impact of the error on system performance is relatively small, hence the problem is scheduled for correction at the next program change point.	GO	PE
*Improved Fault Isolation in Missile Test	The in-silo maintenance philosophy for MM involves replacement of either the MGS (guidance system), the downstage (booster), post-boost propulsion system, or the R/S. Therefore, isolation of failures to one of these three items is a requirement on the Missile Test portion of the ground program. For MM III, the design "objective" for this fault	MT	IM,ND

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
Improved Fault Isolation in Missile Test (continued)	isolation capability is 95% isolation, with 95% confidence. This capability was only partially existent in early ground tapes (Block II and III) due to the time required to characterize and analyze hardware fault behavior, particularly in the PBPS which was a completely new end item. By the time of the Block IV change point, sufficient data and study results were available to implement more complete fault isolation capability in the Missile Test routine.		
Rapid Fuzing	The basic problem here was that fuzing of the MM III reentry system with its multiple R/V's, required a relatively long time to accomplish and, under certain conditions, could affect system reaction time. Various preliminary studies were conducted in 1967 in response to vacillating customer direction . These studies considered methods for fast fuzing which involved hardware changes, software changes, and combinations thereof. Results indicated that there were significant disadvantages for all methods studied and that further intensive studies were required. At that time (early 1968) funding for further work was not available, and since there were no clear-cut overall benefits to be had, SAMSO directed Autonetics to stop any work on this subject.	GO	MR
$Asynchronous Trigger Problem	This problem was first observed as a result of a G&C shutdown on FTM 202 at ETR in September 1968. Investigation of the problem showed that a basic design problem existed in airborne computer logic circuits At that time, two things were done: First, a simple programming workaround was made; and second, Autonetics proposed a computer hardware change. The latter proposal was disapproved by SAMSO who directed a software-only workaround. On FTM 204 (which had the programming workaround) the problem occurred again. Further investigation revealed hardware performance characteristics which were not known previously and which were not compensated for by the programming workaround. An improved programming workaround was then developed which reduced the probability of occurrence to a negligible number. No further failures have occurred.	GO	W

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
*P-Plug sampling requirements	Problem encountered during field installation procedure on MM II was traced to marginal design/operating conditions in a particular piece of interface hardware. The problem caused incorrect sampling of a code plug used for launch code validation. A hardware change was initiated, but a programming change was required to "work around" the problem. In addition, in analyzing the problem, it was discovered that the program sequence for this sampling operation differed in two programs (the ground program used the code, and another program tested the code.) It was decided that these two programs should be consistent. Therefore, a specific computer inst. and timing sequence became a program "requirement". MM II hardware solved this problem thus removing the specific timing requirement, however, a change to the computer interface design resulted in the requirement to make two successive samples and discard the first in order to ensure accurate sample.	GO	NR,W
	Security consideration, as interpreted by NSA and TRW, dictated two other requirements: 1) the two halves of the code cannot be in computer memory simultaneously, 2) the sampling should be done "as little as possible" to avoid possible emanation of the code information. The second requirement was inadvertently violated in an early version of the MM III Program, but this was noticed by a system engineer who initiated a correction in later versions.		
Phi Calibration	This requirement involves field calibration of the physical misalignment of the platform level detectors. The requirement was "scrubbed" from MM II, based on the apparent stability of this measurement (from field and factory data). It was added back in to initial MM III because of increased accuracy requirements of MM III and "available" memory. Subsequently, it was again "scrubbed" when memory budgets became tight (operational); the same rationale was used for deletion.	I	S,MI
Optimum Slew	This is a mechanization which was developed to overcome "anomalies" observed in GCA bias data during initial 4 hours of operation (phenomenon was observed in factory data). Delaying until the data is stable is unacceptable from a timeline standpoint in field operation, so this mechanization allows accurate azimuth estimates to be made during this period.	I	MI,ND

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
*Phase Register Monitor Test	The problem in this case was that the computer phase register flip-flops, which perform a critical function during flight, were not being tested by the ground program. This was discovered by an engineer just "thinking about it." It was also found that the phase register was not tested in Minuteman II ground tapes (which was probably one reason it had been overlooked for Minuteman III). Studies indicated that the phase register failure rate was rather low, but since computer memory was available, a test was devised and put into the ground tape.	MT	NR,MI
High Altitude Fuzing	This change occurred as a result of a new Air Force directed requirement for high altitude burst capabilities for the reentry system. As a result, the ground program had to be changed at the Block IV change point. Detailed requirements were developed jointly between Autonetics and G.E., with TRW and SAMSO technical concurrence.	GO	NR
GCA Skating Problem	This problem was caused by a noisy GCA servo signal which, under some conditions, could cause a loss of phase-lock during a GCA slew, resulting in an uncontrolled slew. The problem was first observed in factory-level testing. In order to reduce the probability of occurrence, both a hardware change and a software change were made.	I	HC,W,MI

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
GCA Slew changes	Modification to slewing mechanization, resulting from new GCA servos which were added at Block III change	I	HC
*Coarse Zeta Check	Completely new technique for initially indexing GCA position to platform position. The previous mechanization used the stops on the GCA as a position reference, but this information was not reliable with the new Block III GCA servos.	I	HC
*SAO Mode	This was an additional mode added by customer direction. The mode provided for use of the Autocollimator data until certain GCA errors had been calibrated, under the assumption that the A/C would provide more accurate information even though GCA data would still be within accuracy criteria. It was argued that the accuracy of A/C placement was not as reliable and subsequent to a large placement error in one system; this mode is scheduled to be deleted. The requirement appeared to be motivated by lack of confidence in GCA.	I	NR
PIGA Leveling	This is a new mechanization currently being implemented in MM III. It is used for seismic avoidance and involves use of sensed gravity for platform leveling during a seismic event instead of maintaining a "free" inertial mode using previously computed biases.	I	MI
SA3 Command	This was an additional system command added by customer direction. The command provides for ignoring the GCA during a system restart and would be used in the event that a GCA failure had been previously indicated. Rationale is somewhat fuzzy; this was related to the SAO Mode addition and is also scheduled for deletion.	I(GCA)	NR

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
IMU Power Shutdown during Standby No-Go Mode	This change was suggested and implemented as a memory saving on a MM II program. It had no disadvantages over the previous mechanization.	I	MI,S
Remote Reset of GCA Alarm	Field use uncovered necessity for excessive maintenance team activity to reset alarm conditions on systems which would not have required maintenance otherwise. The change provided for resetting these alarms remotely.	MP	MI
*Target Numbers Check	Programming error-program did not implement a specific Part I requirement. This error was discovered by the programmer subsequent to delivery when he was looking at the program to assess another potential change.	MP	PE
*"Race" condition between GCA Slew and Fuzing routine	A problem was reported from VAFB. Analysis indicated a program error caused by inadvertent interaction of two routines under a specific set of conditions.	GO	PE
Change of Stage III LITVC Pressure Monitor Failure from No-Go to Alarm	Error in original requirements documentation or analysis. A field failure brought the condition to the attention of SAC/TRW who requested the change.	MT	ME
Modification of DSCS Arming Sequence	A procedural problem in the field caused a problem discovered during MM II operations. This program change was a "workaround" to allow operation until the problem could be remedied.	GO	W
Complementary Enable Test Addition	Field experience showed marginal operation or failure of code devices which successfully passed the programmed test sequence. An improved test was devised.	MT	MI
*R/S Ground Power Fault during Fuzing	TRW discovered a potential safety problem which could arise if a failure occurred during the R/V fuzing operation. This problem arose as a result of adding the high altitude fuzing (HAF) requirement (see description). The potential problem was corrected in the -11 program but has never been changed in the Force Mod program for reasons which are not completely clear.	GO	ME
Critical No-Go Power	Shutdown procedures after a Critical No-Go under certain conditions results in excessive maintenance time to restart the system. Situation discovered during operational usage on MM II.	GO	MI

TITLE	DESCRIPTION	FUNCT. AREA	TYPE OF CHANGE
*Data Trans-Recorder	Addition of capability to record maintenance data. This change, which involves additional hardware in the silo, was promoted by the logistics command and initially fought by SAC and others. It was previously added to MM II and was added to the -11 revision at the last minute.	GO	NR,HC
PIGA Hot Start	Modification to hardware and software procedures to delay PIGA wheel start-up until proper operating temperature is attained. This change resulted from analysis of PIGA failure history and a special test program which indicated wheel bearing degradation on "cold" starts.	I	HC,MI
*Gyrocompassing Improvements/ Corrections	Review of long-term data gathered at Minot AFB during a burn-in exercise indicated high frequency of GCA alarms. The original GCA mechanization was based on relatively short-term factory data. When the long-term data was analyzed, mechanization changes were indicated. Also, the additional study resulted in several "goodies" being suggested.	I	MI,ND
RS #6 Current Monitor Test	Analysis of an anomaly reported from ETR uncovered a shortcoming in the circuitry provided for testing a particular reentry system discrete. A hardware/software change was proposed to add a current monitor to this circuitry. The change was initially disapproved and later approved.	MT	HC
$ AAU Removal	A "value" change was proposed by AN as a result of an internal study which developed an alternate flight control mechanization which did not require data from Stage I & II body mounted accelerometer (AAU's). This change allowed the hardware to be removed from the missile.	FC	HC,MI
EMP Detectors	Field sensor probes were installed by Air Force direction to detect random currents in the silos. Inputs from these detectors were used to trigger circumvention in the computer.	GO	NR,HC

2.0 VERIFICATION OF (-11) PROGRAM

This section summarizes the changes made to the Minuteman III Wing VI Ground Program, -11 revision as a result of the Verification activity.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #2

Dated 11/22/71

TITLE OF CHANGE

- Zeta Calculation Correction

REASON FOR CHANGE

- Key-punch error in Scaling of constant.

DESCRIPTION OF CHANGE

- Change Scaling of Constant from B + 23 to B + 25

PROGRAMMING IMPACT

- Routines Affected - Zeta Calibrate (C7)
- Number of Available Words Used - Zero
- Tests for Engineering Checkout - DSL Calculation of known and controlled Zeta angle.
- Documentation - Not Affected

RECOMMENDED TESTS FOR VERIFICATION

- DSL Calculation of known and controlled Zeta angle.

SUMMARY

- During Engineering Checkout, Zeta Cal was run and checked out; however, the angle experienced was small and the scaling error did not show up in the results. This scaling error is only apparent when Zeta angle exceeds ½ OER count.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #2

Dated 11/22/71

TITLE OF CHANGE

- GIT-1B Rotor Test Failure Response (ECS #24)

REASON FOR CHANGE

- Eliminate ambiguity between Rotor Test Fail and GCA Slew Timer Expiration MSR.

DESCRIPTION OF CHANGE

- Rotor Test Failure will set MSR's 39 and 44.
- GCA Slew Timer Expiration will set MSR's 33 and 44.

PROGRAMMING IMPACT

- Routines Affected - Logic changes in: GCA Slew (A5); SAT Calibrate and Checks (GC4). Juggles in: GCA Slew Initialization (A4); GCA Rotor Turn-On (A7); PIGA Bias Checks (B8); GCA Activation (GC1); GCA Calibrate, Filter, Data Rejection.
- Number of Available Words Used - Four

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #2 (Cont'd)

Dated 11/22/71

PROGRAMMING IMPACT (Cont'd)

- Tests for Engineering Checkout - Satisfactory Completion of: Retargeting; Simulate GCA Slew Timer Runout; Rotor Tests and Failure; Normal Biasing and Linkage; Desk Analysis.
- Documentation - FDD Text and Flows

RECOMMENDED TESTS FOR VERIFICATION

- Verification Tests as planned.

SUMMARY

- This was a change in Requirements (ECS #24)

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #3

Dated 11/30/71

TITLE OF CHANGE

- Lost Disk during IMU Calibrate

REASON FOR CHANGE

- Coding error discovered in DSL 11/23/71. Lost disk present upon exit of 12-minute setting delay before starting Check Sum previous to bias collection during Cal.

DESCRIPTION OF CHANGE

- Change the entry point to not lose a disk.

PROGRAMMING IMPACT

- Routines Affected - Juggle in Bias Delay (C3).
- Number of Available Words Used - Zero
- Tests for Engineering Checkout - IMU Calibrate Linkage
- Documentation - Not Affected

RECOMMENDED TESTS FOR VERIFICATION

- Run IMU Calibrate with No Lost Disks

SUMMARY

- Tests were requested to be run in this mode; there were no errors reported.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #3

Dated 11/30/71

TITLE OF CHANGE

- Correct the KΔPn Limit used during Optimum Slew

REASON FOR CHANGE

- Error in the Logic design of Routine GC3.

DESCRIPTION OF CHANGE

- Change the linkage internal to Routine GC3.

PROGRAMMING IMPACT

- Routines Affected - Juggle in GCA Calibrate, Filter, Data Rejection (GC3)
- Number of Available Words Used - Zero
- Tests for Engineering Checkout - Record KΔPn Limit in both optimum Slew and Steady State gyrocompassing for checks.
- Documentation - Not Affected

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #3

Dated 11/30/71

RECOMMENDED TESTS FOR VERIFICATION

- No need for further tests beyond those already planned.

SUMMARY

- Limits were checked by simulation on a single pass basis. Results were satisfactory. Error results from a dynamic mode in which a supposed benign linkage used only for time delay actually affected the result.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #4

Dated 12/16/71

TITLE OF CHANGE

- ELC Exit from SAT Calibrate

REASON FOR CHANGE

- Correct Program Error

DESCRIPTION OF CHANGE

- Provide Exit Linkage

PROGRAMMING IMPACT

- Routines Affected - Coding change in GIT-1B Torquing & Pulse Accumulation (GC2) level check GC Bias Relay for each Position C2, C3.
- Number of Available Words Used - 1
- Tests for Engineering Checkout - ELC during SAT Cal
- Documentation - None

RECOMMENDED TESTS FOR VERIFICATION

- Repeat existing tests of ELC during SAT Cal

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #4

Dated 12/16/71

SUMMARY

- GC4 changes removed linkage for ELC received during SAT Cal in GC2. No new linkage was provided. Requested tests were not run due to higher priority programs - A&CO, Flight, Wing V Model, etc.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #5

Dated 1/17/72

TITLE OF CHANGE

- PIGA Hot Start (ECS #28)

REASON FOR CHANGE

- Allow the PIGA's time to warm up before enabling power to the PIGA Wheel and Servo.

DESCRIPTION OF CHANGE

- IMU Power Turn-On sequence will be modified to allow 30 minutes for PIGA warm-up before application of power to the PIGA wheel; followed by a minimum of six seconds delay before enabling PIGA Servos. A Spare "C" Discrete is used to control application of PIGA wheel power.

PROGRAMMING IMPACT

- Routines Affected - Juggles in Ground Initialization (G.I.).
- Number of Available Words Used - Six
- Tests for Engineering Checkout - Time the issuance of Discretes for PIGA warm-up.
- Documentation - Minor Change to Flow

RECOMMENDED TESTS FOR VERIFICATION

- Same as for Engineering Checkout

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #5

Dated 1/17/72

TITLE OF CHANGE

- Addition of DTR Capability (ECS #27)

REASON FOR CHANGE

- Add Capability of obtaining Data (Hot Channel Dump) from DCU Memory

DESCRIPTION OF CHANGE

- Channels 40, 42, 44, 46 will be "dumped" on Local Command or on Startup (Non-Tape Fill)

PROGRAMMING IMPACT

- Routines Affected - Logic changes in Ground Initialization (G.I.) and Local (L). Juggles in: Minor Cycle msg. proc. (M1 & M2); Message Processing (M3); Fuzing (M5); Checksum (CK); CSD Control (W4); PIGA Limit Decay and Bias (B7); Terminal Countdown (TCD).

- Number of Available Words Used - 95 words

- Tests for Engineering Checkout - Command DTR during Align, Cal, Strat Alert, Computer Standby, and Standby NO-GO while in Local Mode. While in Local with IMU Power off, command Master Reset. While in DTR, switch from Local to Remote.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #5

Dated 1/17/72

- Documentation - PD Flows and Text Ground Initialization, Local.

RECOMMENDED TESTS FOR VERIFICATION

- Similar to tests used for Engineering Checkout.

SUMMARY

- Customer requested DTR capability for field maintenance and system diagnostic purposes.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #6

Dated 1/21/72

TITLE OF CHANGE

- Common Entry for Local

REASON FOR CHANGE

- To make the Minuteman III Wing VI -11 Operational Ground Program compatible with the A&CO Tapes.

DESCRIPTION OF CHANGE

- Change Instruction so checksum will pass.

PROGRAMMING IMPACT

- Routines Affected - Local (L) change in constant location, and setting of Bank flag.
- Number of Available Words Used - 0
- Tests for Engineering Checkout - Load all operational tapes and A&CO pass checksum. Then run through Local communications and minor cycle msg processing.
- Documentation - None

RECOMMENDED TESTS FOR VERIFICATION

- Same as for Engineering Checkout.

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #7

Dated 2/22/72

TITLE OF CHANGE

- Reset of GCA MSR.

REASON FOR CHANGE

- Change of Requirements for Resetting GCA MSR's to allow restart capability for GCA faults. (IL 72-243-036-SDU-16).

DESCRIPTION OF CHANGE

- For a Master Reset, A/C Return, or Commanded Restart:

  1. MSR's 25, 33, 39, 41, 43, 44, and 46 shall be reset in addition to the MSR's already specified in PRD 20151-309-11.

  2. GCA status shall be set good.

  3. GCA Filter Failure Counter Reset to Zero.

  4. Change in Strat Alert First Pass Flag.

  5. No longer reset Sat Cal Timer in GI. (not required)

PROGRAMMING IMPACT

- Routines Affected - GI: Necessary Changes
  AI: Juggles to acquire necessary word times.

- Number of Available Words Used - 8

  Total Changes for Update - 26 cards

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #7

Dated 2/22/72

PROGRAMMING IMPACT (Cont'd)

- Tests for Engineering Checkout

  1. Satisfactory completion of platform alignment and retargeting.

  2. Make sure MSR 25, 33, 39, 41, 43, 44, and 46 are reset in GI for Master Reset, A/C Return, or Commanded Restart.

  3. Monitor for correct states of GCA Status, Filter Failure Flag, and S/A First Pass Flag.

- Documentation - Flows & P.D.

RECOMMENDED TESTS FOR VERIFICATION

- Reach SA 3 (Hold Z11 state during cage checks after reaching SA1) - go to SBNG - commanded restart and return to SA1.

SUMMARY

- This Cut was determined to be unacceptable as the overall system impact had not been correctly evaluated. All verification tests performed using Cut V #7 were retested.

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #8

Dated 2/26/72

TITLE OF CHANGE

- ECS change of ECS Items 9 and 24.

REASON FOR CHANGE

- Change of requirements for resetting GCA MSR's to allow restart capability for GCA faults.

DESCRIPTION OF CHANGE

- For a Master Reset, A/C Return, or Commanded Restart:

  1. MSR's 33, 39, 41, 43, 44, and 46 shall be reset in addition to the MSR's already specified in PRD 20157-309-11.

  2. GCA status shall be set good.

  3. GCA Filter Failure Counter reset to zero.

  4. Change in Strat Alert First Pass Flag.

  5. No longer reset SAT Cal Timer in GI. (not required)

  6. Azimuth Reference = Autocollimator.

PROGRAMMING IMPACT

- Routines Affected - GI: Necessary changes
  A1: Juggles to acquire necessary word times.
  A7: Necessary Logic changes.
  W2: Juggles for word times.
  GT: Juggle for word times.

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #8

Dated 2/26/72

PROGRAMMING IMPACT (Cont'd)

- Number of Available Words Used - 4
- Total Changes for Update - 65 cards
- Tests for Engineering Checkout
    1. Satisfactory completion of platform alignment and retargeting.
    2. Make sure MSR 33, 39, 41, 43, 44, and 46 are reset in GI for Master Reset, A/C Return, or Commanded Restart.
    3. Monitor for correct states of GCA Status, Filter Failure Flag, S/A First Pass Flag and AZ Ref. Flag.
- Documentation - Fifteen flows and words in the PD.

RECOMMENDED TESTS FOR VERIFICATION

- Reach SA 3 (hold Z11 state during cage checks after reaching SA1) - go to SBNG - commanded restart and return to SA1.

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #9

Dated 3/2/72

TITLE OF CHANGE

- ECS change of ECS Items 9 and 24
- Logic correction: W3 and GC4.10
- Correct lost disc in C1

REASON FOR CHANGE

- ECS change
- Correction of logic error
- Lost disc

DESCRIPTION OF CHANGE

- Necessary logic changes
- Creation of new CP entry
- Correct timing

PROGRAMMING IMPACT

- Routines Affected - GI logic changes and juggles

  W2: Correct card reader error
  W3: One card change where new CP entry is acquired
  C1: Two card change to correct lost disc (timing), no logic change.
  GC4: Create new CP entry for loss of AC during SAT.

- Number of Available Words Used - 8

  Total Changes for Update - 23 Cards

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #9

Dated 3/2/72

PROGRAMMING IMPACT (Cont'd)

- Tests for Engineering Checkout - No lost disk on SAT Cal command, Immediate exit on loss of A/C in SAT.
  Check out various GI paths.

- Documentation - Add GC4.10 in GC4 and W3.

RECOMMENDED TESTS FOR VERIFICATION

- Re-do GI, SAT Command, and loss of A/C in SAT.

SUMMARY

MINUTEMAN III WING VI 20157-309-11

VERIFICATION CUT #9

Dated 3/02/72

TITLE OF CHANGE

- DTR Timing

REASON FOR CHANGE

- Incorrect timing for DTR output.

DESCRIPTION OF CHANGE

- The Data Storage/retrieval Software Mechanization will be changed to maintain timing through the complete Hot Channel Dump.

PROGRAMMING IMPACT

- Routines Affected - Data Trans-Recorder (DTR), Ground Initialization (G.I.) Terminal Countdown (TCD).
- Number of Available Words Used - None
- Total Changes for Update - 24 Cards
- Tests for Engineering Checkout - 1. Run DTR through all entries and exits.
  2. Run TCD for timing.
- Documentation - None

RECOMMENDED TESTS FOR VERIFICATION

- Same as Engineering

SUMMARY

- The timing error was not detected previously because of available DSL time and overall schedule impact.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #9

Dated 3/02/72

TITLE OF CHANGE

- Local Output

REASON FOR CHANGE

- Coding error resulting in local DRO being inhibited.

DESCRIPTION OF CHANGE

- Change next instruction sector.

PROGRAMMING IMPACT

- Routines Affected - Local only.
- Number of Available Words Used - None
- Total Changes for Update - 1 card.
- Tests for Engineering Checkout - Command DRO
- Documentation - None

RECOMMENDED TESTS FOR VERIFICATION

- Command DRO

SUMMARY

- Area was checked out and subsequently was inadvertently changed.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #10

Dated 3/15/72

TITLE OF CHANGE

- Eliminate Cage Check between E0 and E1 Bias Cycle.

REASON FOR CHANGE

- Reduce timeline on retargeting alignment.

DESCRIPTION OF CHANGE

- Change routing on Restart of Optimal Slew sequence.

PROGRAMMING IMPACT

- Routines Affected - Gyrocompass Activation (GC1). Change instruction.
- Number of Available Words Used - None
- Tests for Engineering Checkout - Establish timelines to S/A from various operating modes.
- Documentation - Flow change

RECOMMENDED TESTS FOR VERIFICATION

- Same as Engineering Checkout

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #11

Dated 3/28/72

TITLE OF CHANGE

- MM III Wing VI Operational Ground Program/A&CO Interface

REASON FOR CHANGE

- Make the -11 tape compatible with A&CO Tape.

DESCRIPTION OF CHANGE

- Change location of instruction so checksum will pass.

PROGRAMMING IMPACT

- Routines Affected - Ground Initialization (GI) Juggles
- Number of Available Words Used - 0
- Total Change for Update - 5 cards
- Tests for Engineering Checkout - (1) Load Operational Ground Program/A&CO Tapes; pass checksum; do a data shuffle.
  (2) Load Operational Tapes and run system through G.I. Make sure MSR's 33, 39, 41, 43, 44, 46 are unlatched.
- Documentation - No impact

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #11

Dated 3/28/72

RECOMMENDED TESTS FOR VERIFICATION

- Same as Engineering.

SUMMARY

- This error occurred when the major revision (Cut V8) was made. The interface with A&CO was not checked as part of Engineering Checkout.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #11

Dated 3/28/72

TITLE OF CHANGE

- SAT Calibrate Inhibit

REASON FOR CHANGE

- Allow Forgiveness for GCA Failure due to Filter Failure Runout.

DESCRIPTION OF CHANGE

- Delete check for SAT Cal Inhibit in Msg. Processing.

PROGRAMMING IMPACT

- Routines Affected - Major cycle Msg. Processing (M3)
- Number of Available Words Used - One made available
- Tests for Engineering Checkout - Set a combination of 5 tests of GCA GOOD/BAD; ZETA GOOD/FAILED; and Failure Counter Good/Runout.
- Documentation - No impact

RECOMMENDED TESTS FOR VERIFICATION

- Same as for Engineering Checkout

SUMMARY

- These events necessary for this error to manifest itself when not duplicated in engineering checkout.

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #12

Dated 4/13/72

TITLE OF CHANGE

- SAT Timer Runout MSR's

REASON FOR CHANGE

- Coding error

DESCRIPTION OF CHANGE

- Constant change

PROGRAMMING IMPACT

- Routines Affected - GC4
- Number of Available Words Used - 1 word released
- Tests for Engineering Checkout - MSR Verification
- Documentation - None

RECOMMENDED TESTS FOR VERIFICATION

- SAT Timer Runout shall set MSR 25, 39, and 41

MINUTEMAN III WING VI 20151-309-11

VERIFICATION CUT #12

Dated 4/13/72

TITLE OF CHANGE

- Absolute Phi(N) check lockup.

REASON FOR CHANGE

- If $\emptyset_n$ is less than 0.0005 arc sec and greater than -0.0005 arc sec, a program lockup will occur.

DESCRIPTION OF CHANGE

- Change coding to eliminate the cause of lockup.

PROGRAMMING IMPACT

- Routines Affected - GC3
- Number of Available Words Used - 1
- Tests for Engineering Checkout - Show that a value of 00000000_8 B + 14 computed for Phi'(N) will not lock up program.

RECOMMENDED TESTS FOR VERIFICATION

- Show that no lockup occurs when Phi'(N) = 0 at CP = GC 3.9.

SUMMARY

- The chance of this lockup occurring is considered negligible. This problem is an example of what can happen if data is used for program logic decisions.

3.0 DETAILED PROGRAM EXAMPLES

This section presents the data from the analysis of 13 program examples. Fourteen examples were originally selected but the "New Gravity Model" was deleted do to lack of information.

EXAMPLE 1 - SIMULTANEOUS ATTITUDE MANEUVERS

MISSION OBJECTIVES/CONSIDERATIONS

- Earliest MM III criteria (appendix to MM II criteria) did not include deployment of chaff or decoys in the mission requirements. The basic MM III mission consisted of a furth stage (PBV) with multiple R/V's. Only simple roll maneuvers were required to deploy the multiple vehicle.
- Initially, studies were conducted to assess feasibility of implementing the multiple R/V capability with the existing MM II boost vehicle/G&C system. However, the small diameter of the third stage proved too severe a constraint. Likewise, estimates of software requirements led first to need for larger memory in the airborne computer and subsequently to improvements in computational throughout and input/output. This system configuration; i.e., MM II with new third stage, post-boost (4th stage) vehicle with multiple objects, and a modified computer, became identified as MM III.
- By the Block II Software Critical Design Review, the mission requirements had grown to include deployment of chaff clouds and decoys. These additions to system requirements presumably resulted from operations analysis and war strategy considerations, though Autonetics was apparently not involved in these decisions.
- The addition of chaff and decoys to the mission requirements caused severe problems in terms of Reentry System (R/S) design.* As this design proceeded, requirements for various attitude maneuvers for deployment of objects were identified. General Electric was responsible for the R/S development, but the system design was jointly accomplished by GE, TRW, Aerospace and Autonetics; with A/N's involvement being primarily related to software, propulsion system, PBV control system, and overall system accuracy.

*The requirement for decoys was ultimately deleted due at least partially to R/S design problems.

SYSTEM MECHANIZATION

- The original mechanization (Block II) for the required attitude maneuvers was a sequence of the single axis rotations. This mechanization apparently came from TRW and was chosen because of simplicity, concern (and lack of data) on interplane coupling effects of multi-axis maneuvers, and lack of strong motivation to consider a more complex approach.

- As the R/S design evolved, concern grew over increased mission time estimates. The original criteria specified a 360 sec. maximum PBV mission time, which was subsequently increased to 440 secs. Increased mission time has an adverse effect on virtually all fundamental performance parameters (range capability, accuracy, reliability, etc.) and there was concern that excessive mission time growth might force redesign of some hardware such as the battery or result in component temperature problems.

- The major factors which contributed to the growth in mission time estimates include the following:

  1) Introduction of PBV attitude maneuvers for decoy orientation when it was learned that the RS could not, by itself, do the entire job.

  2) Identification of various delays suggested to solve the plume avoidance problem. These included: (1) an axial engine pre-positioning delay which was required in order that the axial engine could be biased prior to a chaff deployment thrust (and the bias correspondingly reoriented) in order that the pitch engine on the side of chaff ejection would not be required for control, and (2) a post ejection plume avoidance delay to allow the chaff to drift sufficiently far from the bus so as not to be adversely impacted by the subsequent axial thrust.

- Mission analysis simulations were performed by Autonetics (and others) which defined the range-reentry angle map coverage capability of the various PBV designs considered. These maps clearly indicated the shrinkage of deployment capability as mission time estimates increased. While they defined no specific mission time requirement as such, they did motivate the search for ways to reduce mission time. Also, as mission time estimates increased, the possibility of not being able to accomplish some of the "desired" missions arose.

- This concern over mission time was reflected in many discussions at T.I. meetings and among technical personnel. At some point, the possibility of replacing the sequential mechanization for attitude maneuvers with a simultaneous three-axis mechanization was suggested (the source of the suggestion was apparently someone in either AN or TRW involved in flight software system engineering or flight control system analysis.

MECHANIZATION TESTING

- Autonetics, by informal agreement with SAMSO/TRW, conducted a study to determine the potential mission time savings which might result from using the simultaneous maneuver mechanization. This study was conducted using a mission analysis simulation program. This program is a 6 degree-of-freedom digital simulation designed to determine such things as propellant utilization, mission event timing, vehicle deployment "Footprints", etc., for the PBV mission. This study

MECHANIZATION TESTING (continued)

indicated a potential saving of 60 secs. for one of the more complex missions.

- A 6 DOF hybrid simulation which uses the D37U computer was used by Autonetics to verify/demonstrate the feasibility of the simultaneous maneuver mechanization and to determine the control system parameters. This simulation was performed for various combinations of attitude changes. Since the attitude maneuvers are not pre-programmed in the flight program but are a function of the targeting data, the simulations were designed to cover the envelope of possible operational missions.

- One result of the hybrid simulation activity was identification of a potential problem. This was the possibility of encountering the yaw platform gimbal stop due to cross coupling/overshoot during a simultaneous yaw to 45° and roll 180° maneuver. Such a maneuver was a definite mission possibility, but the problem could be avoided by using an alternate sequence of maneuvers to accomplish the same mission. This imposes a requirement on TRW's targeting program, i.e., to avoid scheduling the particular maneuver when targeting a mission. As it turned out, this particular maneuver is/was precluded in the targeting program due to consideration related to modeling of plume impingement effects.* It is not clear, however, whether this maneuver is currently considered/documented as a restriction on the targeting

*It is not completely clear whether this was true at the time when SAM was being considered, however it was definitely true by the time the operational program was developed.

MECHANIZATION TESTING (continued)

program; i.e., if the plume modeling were changed for some reason such that this maneuver was acceptable from a plume standpoint, would there still be sufficient "control" to prevent its use.

IMPLEMENTATION

- The decision to change the flight software requirements to replace the sequential attitude maneuver with the SAM mechanizations was made by SAMSO/TRW, apparently based on:

  1) the mission saving estimates from Autonetics studies
  2) programming impact assessment by AN (small impact)
  3) technical agreement on feasibility based on AN and TRW analysis and simulation.

  This change in requirements was imposed/scheduled at technical interchange meetings between AN, TRW, SAMSO, and documented in the minutes of these minutes. The change was scheduled for the Block III change point.

- The decision to implement SAM caused the IMU Systems Organization to conduct an analysis due to concern that the SAM mechanization might represent more severe requirements on IMU stabilization loops. It is not clear exactly what/how the analysis was initiated, but it seems to have been very informal. As a result of this analysis, it was determined that the roll torquer motor performance was marginal. As it turned out, this problem was not really associated with the SAM mechanization but with the large yaw platform angles required for the missions then being implemented.

PROGRAM REQUIREMENTS (FROM PRD AND FIGURE A)

1) Platform Control Commands:

The flight control attitude command computations were changed to achieve constant 25 deg/sec attitude rates during the simultaneous maneuvers in order to further reduce mission time over the previous mechanization in which the attitude rates were variable between 12 degs/sec and 25 degs/sec, depending on the particular maneuver.

Blocks II and IIA - Sequential Maneuvers

$$\delta_p' = \varepsilon_\theta^L + K_{\dot\theta}\dot\theta$$

$$\delta_y' = \varepsilon_\psi^L + K_{\dot\psi}\dot\psi$$

$$\delta_r' = \varepsilon_\phi^L + K_{\dot\phi}\dot\phi$$

Block III - Simultaneous Maneuvers

$$\delta_p' = 2^{-3}\left[(K_{\theta_j}\varepsilon_\theta)^L + \dot\theta_n\right]$$

$$\delta_y' = 2^{-3}\left[(K_{\psi_j}\varepsilon_\psi)^L + \dot\psi_n\right]$$

$$\delta_r' = 2^{-3}\left[(K_{\theta_j}\varepsilon_\theta)^L + \dot\phi_n\right]$$

Block IV

$$\delta_p' = \left[(K_{\theta_j}\varepsilon_\theta)^L + \dot\theta_n\right]$$

$$\delta_y' = \left[(K_{\psi_j}\varepsilon_\psi)^L + \dot\psi_n\right]$$

$$\delta_r' = \left[(K_{\theta_j}\varepsilon_\theta)^L + \dot\phi_n\right]$$

where ε_θ, ε_ψ, ε_θ are the platform attitude errors

$\dot\theta_n$, $\dot\psi_n$, $\dot\phi_n$ are the platform angle rates (= 32 delta gimbal register ÷ .03 sec

PROGRAM REQUIREMENTS (continued)

2) Attitude Maneuver Mechanization

In the Figure A (Part I) documentation, the mechanization to be used in performing an attitude maneuver is as follows:

In Blocks II and IIA, the Sequence of Events defines an attitude maneuver as being performed in a sequential fashion. Typically,

"d_{426}	in sequence	Roll to $\emptyset$
d_{427}	$\lvert\dot{\emptyset}\rvert \leq C_{145}$;	End roll maneuver, yaw to zero
	$\lvert\Delta\emptyset\rvert \leq C_{146}$	
d_{428}	$\lvert\dot{\Psi}\rvert \leq C_{141}$,	End yaw maneuver, and pitch to θ
	$\lvert\Delta\Psi\rvert \leq C_{142}$	
d_{429}	$\lvert\dot{\theta}\rvert \leq C_{143}$,	End pitch maneuver, yaw to Ψ
	$\lvert\Delta\theta\rvert \leq C_{144}$	
d_{430}	$\lvert\dot{\Psi}\rvert \leq C_{141}$;	End yaw maneuver. . . ." [End entire attitude maneuver.]
	$\lvert\Delta\Psi\rvert \leq C_{142}$	

In Block III, The Sequence of Events defined an attitude maneuver as follows:

"d_{425}	in sequence	a. Set next θ, Ψ, and $\emptyset$
		. . .
		c. Commence Simultaneous Attitude Maneuvers
d_{426}	$\lvert\dot{\emptyset}\rvert \leq D_{24804}$,	[Terminate simultaneous attitude maneuver.]"
	$\lvert\emptyset^*_\varepsilon\rvert \leq D_{24904}$,	
	$\lvert\dot{\theta}\rvert \leq D_{21004}$,	
	$\lvert\theta^*_\varepsilon\rvert \leq D_{21104}$,	
	$\lvert\dot{\Psi}\rvert \leq D_{23004}$,	
	$\lvert\Psi^*_\varepsilon\rvert \leq D_{23104}$	

PROGRAM REQUIREMENTS (continued)

2) Attitude Maneuver Mechanization (continued)

Block IV was identical to Block III with one exception: the pitch and yaw error and rate thresholds for ending the maneuver were made identical. Thus, the criteria for terminating a maneuver became:

> "An attitude maneuver shall be terminated when both the attitude error and rate in each of the control axes fall within acceptable ranges. . . . These tests for maneuver termination shall be made within 150 msec after start of the maneuver and every 150 msec thereafter. [P.66]".

Decision		Criteria	Test Frequency
d_{413}	$\lvert\dot{\phi}\rvert \leq C_{IV_{18}}$	$\lvert\phi_\varepsilon^*\rvert \leq C_{IV_{19}}$	0.15 sec
	$\lvert\dot{\theta}\rvert \leq C_{IV_{16}}$	$\lvert\theta_\varepsilon^*\rvert \leq C_{IV_{17}}$	
	$\lvert\dot{\psi}\rvert \leq C_{IV_{16}}$	$\lvert\psi_\varepsilon^*\rvert \leq C_{IV_{17}}$	

The program requirements for SAM, though subsequently documented in the Block III PRD and the Block IV Fig A Part I, were actually transmitted very informally to the programmer when the programming was initially performed. The flight control requirements were documented in an internal letter prior to incorporation in the Block III PRD.

PROGRAM TESTING

1) General Flight Program Checkout

SSL: Event sequencing and timing are roughly compared to the TRW trajectory to obtain program continuity. This phase will reveal any sequencing errors and major timing errors. Vehicle attitude is roughly compared to the trajectory values.

MFS: Event timing and delays are compared with the TRW trajectory and PRD values.

2) Checkout: Simultaneous Attitude Maneuvers
Vehicle attitudes are compared to the reference trajectory values. Terminal maneuver conditions are checked to be within PRD criteria of the commanded attitudes and the maneuver times are compared with the trajectory values.

3) Verification (Block IV)
The only documentation of the flight program verification testing is a data package which presents and summarizes the data from five simulated flights on the MFS simulator. The Test Requirements Cross Reference Index (TRCRI) portion of this documentation provides an explicit cross reference of Figure A requirements to the verification testing. However, since the TRCRI was primarily a cross check/documentation effort, it does not reflect all the testing which was actually conducted. The testing as referenced in the TRCRI is summarized below.

Req. 1 Set next θ, ψ and ϕ

Ver. 1 Apparently verified the same as Req. 2, below; no separate check was made to see if the values calculated for these commands were correct.

Req. 2 Commence simultaneous attitude maneuver.

Ver. 2 Referenced the VIDEO plots of commanded attitudes vs. actual attitudes for one expansion-point thrust segment per simulation.

Req. 3 An attitude maneuver shall be terminated when both the attitude error and rate in each of the control axes fall within acceptable ranges. . . . These tests for maneuver termination shall be made within 150 msec after start of the maneuver and every 150 msec thereafter.

Ver. 3 Referenced to one VIDEO plot of "Total Attitude Error at Maneuver Completion" and one VIDEO plot of "Attitude Rate at Maneuver Completion" per simulation. Also referenced to the MFS Message, "Maneuver Conditions Satisfied." This message is triggered when MFS executes a particular flight program instruction.

Req. 4	Decision	Criteria		Test Frequency
	d_{413}	$\lvert\dot{\phi}\rvert \leq C_{IV_{18}}$	$\lvert\phi_\varepsilon\rvert \leq C_{IV_{19}}$	0.15 sec.
		$\lvert\dot{\theta}\rvert \leq C_{IV_{16}}$	$\lvert\theta_\varepsilon\rvert \leq C_{IV_{17}}$	
		$\lvert\dot{\psi}\rvert \leq C_{IV_{16}}$	$\lvert\psi_\varepsilon\rvert \leq C_{IV_{17}}$	

Ver. 4 See Ver. 3 above. Apparently no explicit verification of test frequency was performed.

Req. 5 Equations for pitch, yaw, and roll attitude commands as a function of platform attitude errors and platform angular rates.

Ver. 5 Values of input variables read from MFS run for one selected minor cycle during a maneuver. Equation hand calculated and compared with values computed by flight program (printed in MFS).

EXAMPLE 2 - PLUME/CHAFF

MISSION OBJECTIVES/CONSIDERATIONS

- See Example 1 - Simultaneous Attitude Maneuvers

- Chaff/decoys presented severe R/S design problems (decoys were ultimately deleted). The chaff deployment system design involved a variety of potential concepts. Three concepts considered were:
  1) Chaff ejected directly from 4th stage bus (body bound system).
  2) Chaff deployed from cannister ejected at high velocity from bus.
  3) Chaff sprayed from a movable arm extended from bus.

  The main advantage of concepts 2 and 3 was reduction/elimination of the plume avoidance problem; this was particularly desirable at the early stage of R/S development before much was known about plume effects and chaff modelling. However, both of these approaches had some severe performance disadvantages in terms of deployment accuracy, system weight, or software requirements. (The software requirements for the extended arm system appeared prohibitive from a timing standpoint.)

- The body-bound chaff deployment system concept was ultimately chosen in the summer of 1967, several months after Block II Software CDR. This choice was based on the comparative performance and the "ability" of the software to implement appropriate plume avoidance functions.

MECHANIZATION:

- The body-bound chaff system design required controlled release of a stream of chaff from the R/S during a thrusting segment of the mission. The chaff was to be ejected at a low relative velocity and analysis indicated that the effectivity of the resultant chaff cloud deployment could be adversely affected by plumes from the PBV engines.
- During the Reentry System/flight software development, analysis and modelling of plume/chaff was conducted on various system/

mission configurations by several agencies. Most of this activity was conducted by General Electric, Aerospace, and TRW and software requirements were established primarily through technical interchange meetings involving primarily AN, TRW, and SAMSO. Autonetics involvement in developing requirements in this area was primarily related to flight control system analysis/simulation and software design activities related to determination/modelling of plume characteristics, R/V/chaff discrimination studies, analysis of types fo chaff, modelling of chaff ejection characteristics, etc., were going on during much of the 3-4 year R&D period. Consequently, the mechanizations for chaff deployment became increasingly sophisticated based on information from these activities and flight test data. Initially, the mechanizations were designed to avoid plume impingement on chaff generally at a sacrifice in mission time. By the time of the Block IV software development, the mechanization had become sufficiently sophisticated that effects of some amount of plume impingement were modelled in the targeting program such that the chaff deployment parameters were biased to account for plume impingement subsequent to ejection.

- The PBV system contains a single, movable-nozzle, fixed thrust engine (the axial engine) mounted on the primary vehicle axis and used to supply translational thrust. In addition, the Attitude Control System (ACS) consists of ten fixed-positions, on-off type engines mounted around the outer periphery of the vehicle. The chaff is ejected perpendicular to the primary vehicle axis from one of two dispensers located at the top and bottom of the PBV. The positioning of the ACS engines was such that the plumes from one pitch engine and two roll engines would impinge on chaff ejected from each dispenser. The duration of the chaff thrust segment was such that the plume from the axial engine would not impinge on the chaff cloud during ejection but the axial engine plume was still a potential problem subsequent to the chaff thrust segment.

- The following paragraphs summarize the history of the plume avoidance software requirements for the Block II, III, and IV (operational) flight programs:

BLOCK II (1ST R&D FLIGHTS, AUG. 1968)

a) ACS Engine Inhibit - The two roll ACS engines on the same side as the chaff dispenser selected were not used for the normal attitude control functions during the chaff thrust segment. Roll axis attitude control was maintained with the other pair of roll engines. This was a straight forward technique for plume avoidance and was used througjout the flight software development.

b) Axial Engine Pre-positioning - Prior to the chaff thrust segment, the axial engine was positioned off the vehicle center of gravity so as to cause a disturbance torque in the pitch during the subsequent thrust. This torque was intended to force the pitch attitude control loop to use the ACS engine on the side opposite the selected chaff dispenser to maintain pitch attitude control and hence indirectly inhibit the use of the pitch engine on the side nearest the chaff dispenser. Note that only two ACS engines are used for pitch attitude control so that a straightforward mechanization analogous to the roll engines inhibit was not possible.

It was subsequently discovered (apparently from flight test data) that this mechanization did not necessarily inhibit the particular engine during the initial portion of the thrust segment. Depending on the pitch attitude at the time of initial thrust, the engine might come on for normal attitude control before effect of the bias torque "tool hold". This discovery lead to the increased pitch deadspace requirement in Block III.

At the end of the chaff thrust segment, the axial engine was repositioned using the arithmetic complement of the pre-positioning command in order to remove the attitude bias torque (see Block III).

The development of this requirement involved simulation (conducted by Autonetics and probably others) to determine increased propellant utilization/thermal affects of the additional pitch control resulting from the mechanization.

c) Axial Engine Plume Delay - At the end of the chaff thrust segment, use of the axial engine was prohibited for a time selectable by the targeting program. This time was computed to allow sufficient time for the chaff to drift out of the effective core of the axial engine plume and represented a delay in the mission which varied depending on the relative direction of the next thrust segment in the mission.

BLOCK III (JULY 1, 1969)

a) Asymmetrical Pitch Deadspace - The deadspace in the pitch control channel is normally small and symmetric about zero. During the chaff thrust segment and the subsequent plume delay, the deadspace is expanded in the direction corresponding to control with the pitch engine on the side of the selected chaff dispenser. This mechanization is designed to indirectly inhibit the appropriate ACS engine during initial thrusting (overcoming the problem discovered in the Block II pre-positioning) and during the plume delay subsequent to the thrust segment.

   Control system analysis/simulation was performed by Autonetics and TRW to determine the feasibility of increasing the pitch deadspace and to determine acceptable limits.

b) ACS Plume Delay - With the Block II mechanization, the ACS engine inhibits were applied only during the chaff thrust segment. It was discovered (apparently as a result of continuing analysis and examination of actual engine activity data from flight tests) that the ACS engines could come on at the end of the thrust segment and the plumes impinge on the end of the chaff cloud. To overcome this problem, the roll engine inhibit and the asymmetrical pitch deadspace were continued after the thrust segment for a period of time selected by the targeting program but within the axial engine plume delay.

c) Roll Engine Inhibit - Same as Block II except that the inhibit is extended till the end of the ACS plume delay.

d) Plume Avoidance Thrust (PAT) - Provisions were added for a thrust segment subsequent to the chaff thrust (and the axial engine plume delay) and prior to a transition maneuver to a new deployment point. This mechanization resulted from a problem discovered at the time of the Block II software development but too late in the development to incorporate the changes in the Block II program. The problem arose when the mission sequence included a transition thrust segment after a chaff thrust which required a large (approaching 90 degrees) attitude change and a translation in a direction away from the chaff cloud. Under these circumstances, the axial plume would be directed toward the chaff cloud and to delay sufficient time to allow the chaff to move out of the plume (the original Block II mechanization) would be prohibitive due to the relative direction of the chaff movement. The PAT segment, performed after the normal 10-15 second plume delay, moved the PBV away from the chaff so that when the transition maneuver was performed the chaff would not be in the axial engine plume.

The transition thrust plume avoidance problem was identified by TRW at a Technical Interchange Meeting. The PAT mechanization was apparently development by Autonetics system engineering personnel.

During the time period of the Block III development, the modelling and understanding of the plume/chaff effects had reached the point where it was felt (by TRW) that the effects of plume impingement on the chaff cloud could be predicted sufficiently well to allow pre-biasing the chaff deployment to account for this effect. The primary motivation for this approach was concern over mission time requirements which were greatly affected by the various plume avoidance delays. These delays could be shortened by using the pre-biasing technique.

e) Axial Plume Delay - Same as Block II except that delays could be shorter due to the addition of PAT and the pre-biasing of the chaff deployment.

f) Axial Engine Pre-Positioning - The Block II pre-positioning technqiue was used except that instead of re-positioning the engine back to its original position as was done on Block II, the engine was re-positioned with a unique command determined by the targeting program. This unique re-positioning was necessary to account for a center of gravity shift when a decoy and chaff were deployed.

Control system analysis was performed by Autonetics which established the need for the center of gravity compensation subsequent to decoy deployment.

A minor change was also made to the pre-positioning sequence to correct a potential problem in the Block II mechanization. This change involved starting the pre-positioning during the previous axial engine plume delay, rather than the end of it, to allow sufficient time for the physical movement of the nozzle to be completed prior to the chaff thrust. This also placed a constraint on the targeting program that the plume delay be of sufficient duration to allow the pre-positioning to be completed. It appears that the mechanization change was based on very conservative estimates of nozzle motion and was apparently unnecessary since even if the nozzle had not reached its final position when the thrust segment began, the expanded pitch deadspace would effectively inhibit the appropriate ACS engine.

BLOCK IV (OPERATIONAL CONFIGURATION -DECEMBER 1970)

By the time of the Block IV development, the list of potential flight program (and ground program) requirements had grown until it significantly exceeded the available computer memory. A joint TRW, SAMSO, Autonetics committee was formed to identify/evaluate candidates for elimination/modification from the Block IV requirements. Three candidates were identified in the area of plume/chaff - related functions.

a) Fixed ACS Plume Delay - In the Block III ACS plume delay mechanization, the duration of the delay is determined by the targeting program and is variable for each chaff thrust segment.

It was suggested that 47 computer memory words could be saved by making the plume delay common for all chaff thrust segments in a given mission while still allowing the delay to be determined by targeting.

Assessment of this candidate indicated that its only drawback was a relatively minor (6 seconds) worst case increase in overall mission time. It had a slight benefit in terms of simplifying the targeting problem.

This candidate was <u>implemented</u> as a Block IV requirement.

b) <u>Fixed Axial Engine Re-Positioning</u> - The Block IV mechanization uses unique commands (determined by targeting) to re-position the axial engine after a chaff thrust segment. It was suggested that 50 memory words could be saved by re-positioning the engine with the complement of the pre-positioning command (the Block II mechanization).

Since the proposed mechanization did not compensate for center of gravity shift due to decoy/chaff deployment, stability problems could result during the initial portion of the post-chaff thrust segment (Block III included a closed-loop axial engine) postioning mechanization during thrust which would ultimately remove the engine misalignment). Under worst case conditions; i.e., several successive, short decoy and chaff segments, a stability problem could result in mission failure. Therefore, this candidate was <u>not</u> implemented.

Late in the Block IV development (early 1970) and subsequent to the requirements "scrubbing" activity, the mission requirements for decoys were deleted. This removed the drawback to this mechanization, but this was apparently too late to change the program.

c) <u>Chaff Thrust On Time</u> - The Block II/III mechanization used a delta-velocity criteria for determing the duration of a chaff thrust segment in flight. It was suggested that a potential memory saving could result by changing this criteria from velocity to time.

This proposed mechanization change had some benefits in terms of mission performance related to the ability to allow shorter chaff thrusting segments. Its only drawback was a 1.2 second increase in mission time for one particular mission configuraiton. However, it turned out that instead of saving memory words it required an additional 6 words. The candidate was nevertheless implemented.

PROGRAMMING REQUIREMENTS

The Block II and III software was R&D programs developed in support of the MM III R&D flight test program. In general, a separate program was delivered for each flight (or possibly 2 or 3 flights) and there often were slight differences in the software between flights of the same Block. These differences account for correction of problems or special perturbations unique to the particular test flight.

The requirements generation/documentation process was relatively informal for the Block II and III software since these programs were designated as "engineering" programs rather than "end items". Program Requirements Documents (PRD) were published for both Block II and Block III (also for Block IIA, an interim version of Block III), but these documents were after-the-fact and the actual source of programming requirements was primarily verbal interchange between the programmers, the software system engineers and TRW technical personnel. These requirements were documented in minutes of meetings abd internal letters if at all.

The Block IV program was designated as an end item product and hence had formal requirements documentation. However, the Block IV program was developed as a revision to the Block III program, so in many cases, this documentation was also after-the-fact in terms of programming the software. The Block IV Part I requirements related to plume/chaff functions are summarized below.

Roll Engine Inhibit for Chaff/Thrust

$t - t_{431} \geq K_{d_{434ex}}$ Inhibit Centerline Control and Proper Roll Engines (Sequence of Events, P. 24.).

During chaff dispensing the two roll engines nearest the chaff dispenser

in use, R_1 and R_4, or R_2 and R_3, shall be inhibited, depending on whether the chaff dispenser at 0 deg, respectively, is in use. [P. 73.]

Inhibit Centerline Control and 2 roll engines. [Flow, P. 27.]

Asymmetric Pitch Deadspace During Chaff/Thrust

$t - t_{431} \geq K_{d_{434ex}}$ Set C_{IV_6} for Limited Pitch Control Inhibit (Sequence of Events, P. 24)

$K_{DS_\theta} = K_{\theta_j} C_{IV_5} + |C_{IV_6}|$ . . . The constant, C_{IV_6}, shall be zero except during chaff dispensing, when its sign shall depend on which chaff dispenser us being used. [P. 57-58.]

Set biased control deadspace constant and resume normal attitude control. [Flow, P. 27.]

Plume Avoidance Thrust and Delays

PAT: Begin axial plume delay and ACS Plume Delay; check to see if a plume avoidance maneuver is next; if yes, wait for the end of the ACS Plume Delay, resume normal attitude control, initialize V_{g_Z}, wait for the end of the axial engine plume delay, turn on the axial engine. enter FCD, begin open-loop steering, and terminate axial thrust/halt FCD when velocity criteria is satisfied. [PBCS Level 1 Flow. P. 27.]

$t - t_d \geq K_{d_{411}}$

(ACS Plume Delay)

a. Resume Normal Attitude Control
b. If last object has been deployed, go to d_{444}.

---and--

a. (1) Depending on value of $K_{d_{412ex}}$, initialize guidance equations for critical or non-critical steering.
   (2) Set up Chaff Dispensers.
b. Initialize V'_{g_Z} if open-loop steering thrust segment (PAT) prior to transition thrust segment, go to d_{414}; if not, continue.
c. Set next θ, ψ and ϕ.
d. Commence Simultaneous Attitude Maneuver. [Sequence of Events, P. 20]

---

d_{414}: $t - t_d - K_{d_{414ex}}$ (Axial Engine Plume Delay)

If Open-Loop Steering, turn on Axial Engine, begin Open-Loop Steering, enter FCD. [Sequence of Events, P. 21]

The FCD hardware shall be used for termination of PBV axial thrust (open-loop steering) prior to transition PBV axial thrust. (The FCD register is initialized by the value of the FCD velocity, V_{fc}, of the major cycle in which FCD is entered.) $V_{fc} = V'_{g_Z}$ for a PAT ...

When $V_{fc} = V_{g_X}$, V'_{g_X} or $V^*_{g_X}$ the FCD register shall be updated each major cycle during FCD by the computed value of equation (34), (37) and (47). No such update is required when V'_{g_Z} is the FCD variable. [P. 13]

Axial Engine Pre-Positioning

Adjust axial engine (offset axial thrust vector from c.g.). [P. 24.]

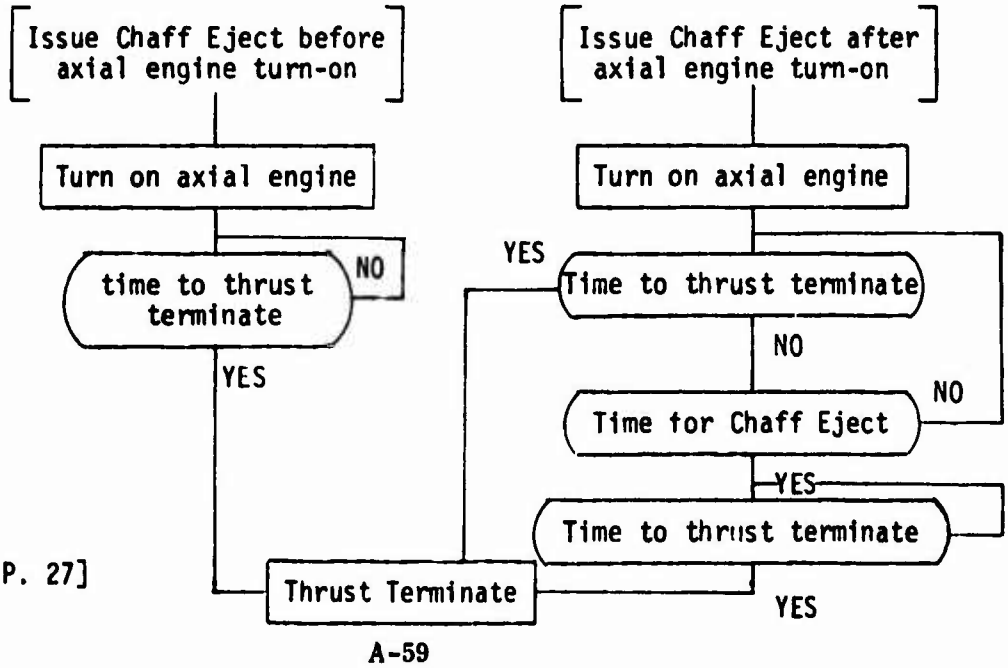
Pre-position axial engine. [Flow, P. 27.]

Terminate Chaff/Thrust On Time

Decision No.	Criteria	Action Performed	Test Frequency
d_{440}	$t - t_{437} = K_{d_{440ex}}$ or $t - t_{438} = K_{d_{440ex}}$	a. Terminate axial thrust based on minor cycle timing. (Sequence of Events, P. 24)	0.03 sec

---

[Flow, P. 27]



The W20 Test for Thrust Termination as shown in Figure 8 shall be provided to sense the occurrence of thrust termination. Continuation of the mission shall be precluded until satisfaction of this test is achieved. This test is not required for chaff thrust segments since thrust termination is under direct program control (based on minor cycle timing). [P. 82]

PROGRAM TESTING

a) General Flight Checkout

SSL: Events sequencing and timing are roughly compared to the TRW trajectory to obtain program continuity. This phase will reveal any sequencing errors and major timing errors. Vehicle attitude is roughly compared to the trajectory values.

MFS: Event timing and delays are compared with the TRW trajectory and PRD values.

b) Checkout: Plume Avoidance Requirements

Pre-positioning: Axial engine pre-positioning eas checked via center-of-gravity offset plots (VIDEO) and printed axial engine orientation as given in MFS .

Roll Engine Inhibit and Pitch Deadband Increase: These two areas were checked via the ACS Thruster Status plots given by VIDEO.

c) Block IV Verification

The testing as referenced in the TRCRI is summarized below. As previously mentioned (under SAM), this does not necessarily reflect the total verification testing in this area.

	DECISION NO.	CRITERIA	ACTION PERFORMED	TEST FREQUENCY
REQ.	d_{440}	$t - t_{437} = K_{d_{440ex}}$ or $t - t_{438} = K_{d_{440ex}}$	a. Terminate axial thrust based on minor cycle timing, b. ...	0.03 sec

VER. Examination of the MFS trace for several chaff thrust segments. The

message "BEGIN AXIAL THRUST FOR CHAFF" is triggered in the simulation by execution of the program instruction which initiates the axial engine for a chaff thrust. Issuance of the D16A (Thrust Termination discrete) also produces an MFS message.

REQ. (PAT): Begin axial plume delay and ACS Plume Delay; check to see if a plume avoidance maneuver is next; if yes, wait for the end of the ACS Plume Delay, normal attitude control, initialize V_{g_Z}, wait for the end of the axial engine plume delay, turn on the axial engine, enter TCD, begin open-loop steering, and terminate axial thrust/ halt FCD when velocity criteria is satisfied. [PBCS Level 1 Flow, P. 27]

VER. None.

COMMENT No attempt was made in the TRCRI to trace requirements dictated by the overall functional flow chart in the Fig. A Part I.

REQ. $t - t_d \geq K_{d_{411}}$

VER. Traced via TRCRI to d_{418}, d_{419} and d_{420} in the PD; referenced to MFS Message of "ACS Plume Delay satisfied." (This message triggered by execution of a specified program instruction.)

REQ. Resume Normal Attitude Control

VER. No specific verification.

REQ. If last object has been deployed, go to d_{444}.

VER. No specific verification.

REQ. Depending on value of $K_{d_{412ex}}$, initialize guidance equations for critical or non-critical steering.

VER. No specific verification.

REQ. Set up Chaff Dispensers (see Section I.B.3.b.(1) for Timing Constraints).

VER. Traced the requirement to the MFS printouts of the time at which the two chaff set-up discretes were issued for one particular chaff segment.

REQ. $t - t_d \geq K_{d_{414ex}}$ (Axial Engine Plume Delay)

VER. Traced only to Program Description Document (PD).

REQ. If Open-Loop Steering, turn on Axial Engine, ...

VER. Referenced MFS message printout of "Begin Plume Avoidance Thrust".

REQ. ... begin Open-Loop Steering ...

VER. None

REQ. ... enter FCD ...

VER. Referenced to MFS printout of the EFC (Enable Fine-Countdown) instruction.

REQ. The FCD hardware shall be used for ... termiantion of PBV axial thrust (open loop steering) prior to transition PBV axial thrust.

VER. Traced to PD. The only verification performed was to reference a VIDEO plot of V'_{g_Z} for one expansion-point thrust for two simulations and to two VIDEO plots of "V'_{g_Z} Error" for one Plume-Avoidance Thrust segment for two simulations.

REQ. $V_{fc} = V'_{g_Z}$ for a PAT

VER. Same as above.

REQ. When $V_{fc} = V'_{g_X}$ or $V^*_{g_X}$, the FCD register shall be updated each major cycle during FCD by the computer value of equation (34), (37) and (47). No such update is required when V'_{g_Z} is the FCD variable.

VER. Same as above. (No real verification.)

REQ. Adjust axial engine (offset axial thrust vector from c.g.)

VER. Reference to MFS message of "Pre-position with K_{57ee}, K_{571ee}".

REQ. Preposition axial engine [Flow, P. 27].

VER. No specific trace of requirements dictated by flow chart.

REQ. $t - t_{431} \geq K_{d_{434ex}}$ Inhibit Centerline Control and Proper Roll engines

VER. Traced to PD; no real verification.

REQ. During chaff dispensing, the two roll engines nearest the chaff dispenser in use, R_1 and R_4, or R_2 and R_3, shall be inhibited, depending on whether the chaff dispenser at 0 deg or at 180 deg, respectively, is in use.

VER. Referenced to VIDEO plots of Thruster Status (4 plots total).

COM. Implicitly verified via above that the proper pair of roll engines are inhibited. By looking at VIDEO, one can conclude that neither roll engine came on during the restricted period; however, this could be due to merely to the fact that neither roll engine was ever required to be turned on by the control equations. One the other hand, the roll engines are always commanded on by control in pairs; if only one engine comes on during the chaff/thrust restricted period, it may be assumed that the other was inhibited by additional program logic.

REQ. Inhibit Centerline Control and 2 roll engines [Flow, P. 27].

VER. No specific trace of requirements dicated by flow chart.

REQ. $t - t_{431} \geq K_{d_{434ex}}$ Set C_{IV_6} for Limited Pitch Control Inhibit.

VER. Traced to PD. No verification.

REQ. $K_{DS_\theta} = K_{\theta_j} C_{IV_5} + |C_{IV_6}|$. . . The constant, C_{IV_6}, shall be zero except during chaff dispensing when its sign shall depend on which chaff dispenser is being used.

VER. Traced to PD.

REQ. Set biased control deadspace constant and resume normal attitude control. [Flow, P. 27].

VER. No specific trace of requirements dictated by flow charts.

Example 3 - Tau Problem

System Considerations

The Weapon System has an overall accuracy requirement measured as CEP. In turn, the G&C system has its own accuracy requirements determined by budgeting the overall requirement. One factor affecting G&C errors is the timing accuracy of the D37 Computer. Timing affects two error sources: PIGA/platform calibration and in-flight calculations. Of these two, calibration errors are much more sensitive (20 X) to timing errors. Therefore, the primary emphasis is to minimize the timing error in calibration in order to minimize overall G&C errors.

The MM III tau mechanization was carried over from MM II and involves a two-fold requirement: First, to calculate the ratio of computer time to the external precision time source; and secondly, to use this ratio (tau) to correct the timing errors during calibration. There is no requirement to correct in-flight timing errors since the resulting impact error is expected to be small.

The original mechanization was developed/programmed for MM II around 1964. The original documentation of the "requirements" was by internal letter. The mechanization was subsequently documented in the MM II -211 Program Requirements Document and then in the MM III Wing VI -11 PRD.

Problem Description

In programming the tau calculations for the original Wing VI MM III ground program, a programming error was made related to a subtle complexity of the D37 computer's operation during "off-line" multiplication. The result of this error in terms of overall program operation was a bias in the calculation of the tau ratio which

Problem Description (continued)

was very small, or zero, when the computer timing was relatively accurate, i.e., a "nominal" system. The bias in the computation became relatively much larger as the system deviated from the nominal, i.e., tau deviated from one.

The programming error was not discovered during checkout, verification, or system testing of either the original program or the (-11) revision to the program for which the tau computation remained unchanged.

A discrepancy was ultimately noted in analyzing calibration data from a system at VAFB for which the tau ratio was unusually "off-nominal". The programming error was subsequently discovered by attempting to reproduce/explain the discrepancy on a TXO site. The impact of the error in terms of overall system performance is obviously quite small and would not jeopardize system performance criteria limits.

Program Requirements:

The original requirements from which the tau mechanization was programmed for MM II were apparently a combination of verbal and informally documented information. The requirements as documented for the MM III Wing VI -11 program are given below. This documentation was, of course, not used as a source of programming information for the function but it was used as the reference for testing the function during verification.

11.2 COMPUTER TIME CALIBRATION

Purpose

The purpose of computer time calibration is to calculate Tau (τ), the ratio of computer time to external precision time, and to detect and report precision time failures.

Required Functions

1. The Left Half Word of R.O, which increments by 1 each 8 word times, shall be used as a Fine Time Counter. This Fine Time Counter is reset to a binary 100 000 000 00 by a precision time pulse. A precision time pulse occurs once every 1 ± .0000005 seconds and is 400 to 560 μsec in duration.

2. Calculate Rn which is the "Nth" determination of the number of computer eight-word time pulses accumulated between precision time pulses from the ground equipment.

$$Rn = R.O(n) - R.O(n-1) - \Delta R + C_{nom}$$

R.O(n) = present value of R-loop sample

R.O(n-1) = previous value of R-loop sample

ΔR = number of 8 word time pulses that should be accumulated in R.O during a 0.96 second sampling period = 1536.

Cnom = the nominal number of 8 word time pulses that should be accumulated between precision time pulses; = 1600

N = the Nth sample, where the sample period is defined to be 0.96 sec.

If the solution of the above equation exceeds the gross check limit of 1 pulse (625 μs) the updating of Tau shall be bypassed.

3. If the gross check limit of 1 pulse (625 μs) is not exceeded, a value of C'(n) shall be calculated as:

$$C'(n) = C'(n-1) + \beta \left[C'(n-1) + Rn\right]$$

where: C'(n) = difference between the computer time and precision time in fine time pulses.

C'(n-1) = previous value of C'n.

Rn = value calculated for item 2. above.

11.2 COMPUTER TIME CALIBRATION (Continued)

Required Functions (Continued)

β = -1/100 shall be used in the τ computations until gyro biasing has been initiated.

= -1/600 shall be used in the τ computations after gyro biasing has been initiated.

4. If C'(n) exceeds a limit of 30 ppm, and the system is not in Initial Alignment, MSR 62 (Precision Time Failure) shall be set.

5. Tau shall be calculated as follows and shall be used for correcting pulse rate calculations during PIGA calibration.

$$\tau = \frac{C'(n) + Cnom}{Cnom}$$

Program Testing

Since this example represents an error which was not detected during the program testing activity, it is pertinent to ask why it was not discovered and how testing could be designed to discover this type of error.

- No specific accuracy/precision requirements are specified in the PRD for the tau equations.
- No specific testing of the precise equation solution was performed.
- System inputs were used to "stimulate" the tau function during testing. Since the effect of the error was only significant for an "off-nominal" system, it statistically is not likely to show up for any small sample of systems.
- The error could have best been detected by either detailed examination of the program with regard to computer "idiosynchrosies" or specific testing of the equations involved using test values which represent the range of conditions for which the calculation is designed.

-11 Checkout

The tau routine was not specifically tested during checkout of the -11 revision since it was not changed from the previous program.

-11 Verification

For separate tests were performed on the tau function as a part of the verification of the -11 revision. These tests are summarized below.

Test #1

Purpose: Assume that initial B is -1/100 and B after start of gyro biasing is -1/600.

Assume that the initial alignment indicator is reset to "not initial alignment" at least 24 minutes after startup.

Procedure: Run an initial alignment sequence, verify changing and value of B by reading it from computes memory and verifying gross timing when it changes.

Test #2

Purpose: Assume that updating of tau is performed when the gross check limit of 1 pulse is not exceeded and that the updating is bypassed if the limit is exceeded.

Assume that the equation for updating tau is mechanized as depicted in the PRD.

Procedure: Records R.0 (n-1), C'(n-1), and tau from computer memory; forces gross check failure by inhibiting precision time input for 30 secs. Apparently verifies equations and proper response to gross check failure (tau not updated) by examining printout from this run. Also verifies that status bit (fine check failure) is not set by gross check failure.

Sufficient data was recorded to do a bit for bit verification of the equations for at least this test condition, but this was not done apparently because of schedule pressure and lack of motivation since this calculation had been used for such a long time.

Tests 3 & 4

Purpose: Assume that the tau alarm cannot be set, even though tau fails the 30 PPM check before the fine Beta bias has been used for 9 minutes.

Assume that the tau alarm is set when the 30 PPM check fails and fine Beta has been used for 9 minutes.

Procedure: Verifies response to fine check failure by forcing gross check limit to zero with key-in and examining status bit before and after initial alignment.

Comments on Verification

- Selection of β value tested from system level; i.e., run alignment and monitor value.
- Approach to equation verification uses "system inputs", i.e., let program run with precision time input and computer (R.0) counter running and monitor all values (as opposed to status, prepared test case). The fact that the equations were not really verified is due to lack of effort not approach.
- The two limit checks were verified in a fairly gross sense. The gross check was verified by inhibiting the precision time input for "$>$30 secs" and verifying that the gross check was failed. (this doesn't verify the precise limit of the gross check).

  Likewise, the 30 PPM check was only verified to the extent that it would pass under "normal" conditions, and that if the check failed (forced by changing the 30 PPM constant to 0) the proper responses would be set. It is possible, but unlikely, that some additional manual eyeballing was used to verify the constant.)

The responses to limit check failures were checked at system level i.e., create system mode, force failure, observe status responses. (MSRs).

Example 4 - Fault Isolation in Missile Test

System Considerations:

- The missile test requirements are designed to serve two purposes: 1) insure that the missile is flightworthy, and 2) provide isolation of faults to a replaceable AVE end item. The general philosophy used in determining flightworthiness is to test those missile functions which are necessary/critical to the flight mission and which are not exercised as a result of normal in-silo operation. The fault isolation requirements are stated as 95% confidence of isolating 95% of the failures. This is stated as a design "objective" and not a specific requirement. Isolation is to one of the following subsystems: R/S, PBPS, MGS, or downstage (booster).

- The primary changes to the missile test functions from MM II to MM III were the additional test/fault isolation requirements related to the PBPS and R/S. The MM II test philosophy and test sequences were carried over to MM III.

- Specific MM III missile test/fault isolation mechanizations were determined by extending the MM test approach to the PBPS which conserved computer memory by allowing common routines to be used for testing the three boost stages and the PBPS. PBPS operating characteristics were obtained rather informally from the responsible PBPS engineers within Autonetics. Fault isolation logic was derived from a failure mode and effects analysis using predicted component failure rates and MM II experience.

- Missile test/fault isolation mechanization details were modified as a result of extensive "reliability improvements" studies/analysis which was being conducted on MM II during the time of the early MM III development. These modifications resulted primarily from analysis of MM II field data relative to missile test failures.

Program Requirements

- Initial MM III missile test was developed for Block II Force Mod Ground Program. This mechanization did not include PBPS fault isolation. It was an extension of then current MM II missile test to include a basic test of the PBPS functions. The Block II requirements were based on rather preliminary and incomplete data on PBPS operating characteristics. Since this program was strictly R&D, there was not strong motivation to include fault isolation.

- The missile test mechanization was reviewed and changed for the Block III ground program. This change resulted from a change in personnel responsible for the missile test software requirements, better data on PBPS operation, and more pressure on establishing "operational" requirements. The Block III mechanization still did not include PBPS fault isolation due to memory constraints and the fact that it was still an R&D program.

- The PBPS fault isolation requirements were added at the block IV (operationsl) configuration. A memory budget (very small) was established for this addition which essentially constrained the mechanization to extending the fault isolation technique used on the downstage hardware

to the PBPS. This technique involved compile the results of the test samples into fault patterns which would then be examined to select the "most probable" failure (in terms of replaceable end item) No specific analysis was performed to determine whether the test mechanizations met the 95%/95% design objective.

- Program requirements were supplied to the programmers in an informal manner, usually an internal letter though, in some cases, verbally. These requirements were subsequently documented in the Figure A's for the initial versions of the Wing VI and Force Mod ground programs. At the -11 revision to the Wing VI program, it was groundruled that missile test not be effected; hence, the PRD developed for the -11 program does not contain missile test requirements.

- The fault isolation requirements are summarized on the following charts:

BASIC FAULT ISOLATION TECHNIQUE

- APPLY TEST COMMANDS IN A PREDETERMINED ORDER
- SEQUENTIALLY SAMPLE PERTINENT MONITORS
- COMPILE SAMPLES TO FORM FAULT PATTERNS
- FAULT PATTERNS PROCESSED AS A FUNCTION OF
  - HARDWARE FAILURE PROBABILITIES
  - MMII FIELD EXPERIENCE
- REPORT FAULTS BY SETTING APPROPRIATE BITS IN THE MSR/MOSR WORD
- ISOLATE TO END ITEM PER T.O. PROCEDURES

F/C POWER FAULTS

- F/C ELECTRONICS POWER TURN "ON" (WS-133B)

STAGE 2-3 HYDRAULIC POWER MONITOR Z14C USED IN PLACE OF FAULT ISOLATION MONITOR X8C

IF FIRST SAMPLE OF Z14C IS FALSE, HYDRAULIC POWER "TURN ON" IS BYPASSED AND THE FOLLOWING ALARMS EXIST:

D37 MSR	FAULT P/O	REMARKS
03	503	F/C ELECTRONICS POWER ABSENT (Z14C = FALSE)
01	501	F/C HYDRAULIC POWER ABSENT SINCE IT WAS NOT "TURNED ON" (X13 = X14 = FALSE)
21	521	F/C ELECTRONICS GROUND POWER LIMIT (M3) NOTE: THIS ALARM WILL NOT BE SET IF Z14C MONITOR FAILED.
22	522	F/C HYDRAULIC GROUND POWER LIMIT (M2)

F/C POWER FAULTS

- F/C HYDRAULIC POWER "TURNED ON" IF 1st Z14C SAMPLE = TRUE
- POSSIBLE ALARM CONDITIONS (WS-133B)

D37 MSR	FAULT P/O	MONITOR	REMARKS
02	502	H_3 = FALSE	F/C ELECTRONICS AND HYDRAULIC ON COMMANDS NOT ISSUED TO C225
01	501	X_{13} = X_{14} = FALSE 2nd Z14C = TRUE	BOTH HYDRAULIC PRESSURE MONITORS BEING FALSE SET OR THE 2nd Z14C MONITOR SAMPLE BEING TRUE SET INDICATES HYDRAULIC POWER LOW OR ABSENT.
21	521	M_3 = FALSE	F/C ELECTRONICS GROUND POWER LIMIT
22	522	M_2 = FALSE	F/C HYDRAULIC GROUND POWER LIMIT

F/C POWER FAULTS

F/C POWER TURN "ON" (WS-133A-M)

POSSIBLE ALARM CONDITION:

D37 MOSR	FAULT P/O	REMARKS
31	56	F/C HYDRAULIC POWER LOW OR ABSENT ALARM SET BY: 1. HYDRAULIC PRESS. MONITORS $X_{13C} \cdot X_{14C}$ = FALSE 2. HYDRAULIC POWER MONITOR Z_{14C} 1st AND 2nd SAMPLE PATTERN OTHER THAN TRUE AND FALSE RESPECTIVELY.
21	46	G&C COUPLER CONTROL FAULT. C53E MAY HAVE FAILED TO ISSUE F/C POWER "ON" COMMAND. ALARM SET BY X_{5C} = FALSE

MISSILE ORDNANCE DISCRETE/FINE COUNTDOWN FAULTS

NO-GO RESPONSE TO THE FOLLOWING COMMON CHECKS:

WS-133B		WS-133A-M		REMARKS
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	
08	508	29	54	COMPUTER FAILURE SET BY: 1. CRITICAL DISCRETE Z23C SET TRUE 2. FINE COUNTDOWN TEST FAILURE
11	511	38	63	CONTROL AND DISCRETE UNIT FAILURE SET BY: 1. CROSSFIRING OF DISCRETES 2. OUT OF SEQUENCE 3. IMPROPER HAZARDOUS CURRENT MONITOR Y1C AND Y14C STATUS 4. PBPS FAULT ISOLATION MONITOR X7C IN THE TRUE STATE 5. ANY DISCRETE TRUE SET STATUS DURING DISARM CHECK

MISSILE ORDNANCE DISCRETE FAULTS

DOWNSTAGE/PBPS DISCRETE FAILURE NO-GO RESPONSE TO FALSE STATUS OF THE FOLLOWING DISCRETES DURING ARMED CHECK.

WS-133B		WS-133A-M			
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	DISCRETES	CURRENT MONITORS
10	510	37	62	GAS GENERATOR 1 AND 2 (D26A)	Y_{3C} & Y_{4C}
				STG. 1-2 IGN/SEP. (D13A)	Y_{6C} & Y_{7C}
				STG. 2-3 IGN/SEP. (D14A)	Y_{8C} & Y_{9C}
				GAS GEN. 3-4 (D28A)	Y_{10C} & Y_{20C}
				T. T. A&B (D16A)	Y_{11C} & Y_{12C}
11 & 13	511 & 513	27 & 38	52 & 63	PBPS PROP. PRESS. (D24A)	Y_{19C}
				PBPS PROP. OUT (D23A)	W_{15C}
				STG 3/PBV ELECT. DIS. (D29A)	W_{19C}
				STG 3/PBV MECH. DIS. (D27A)	W_{23C}

MISSILE ORDNANCE DISCRETE FAULTS

R/V OR UPSTAGE DISCRETE FAILURE NO-GO RESPONSE TO FALSE STATUS OF THE FOLLOWING DISCRETES DURING ARMED CHECK

WS-133B		WS-133A-M			
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	RE-ENTRY SYSTEM DISCRETES	CURRENT MONITORS
19	509	39	64	R/S 1 (D21A)	W_{16C}
				R/S 2A & 2B (D15A)	W_{13C} & W_{14C}
				R/S 3 (D19A)	Y_{23C}
				R/S 4 (D11A)	Y_{15C}
				R/S 5 (D17A)	Y_{16C}
				*R/S 6 (D14B)	Y_{21C}
				R/S 7 (D10A)	Y_{24C}
				R/S 8 (D20A)	W_{22C}
				R/S 9 (D11B)	Y_{16C}
				R/S 10 (D10B)	Y_{18C}
				R/S 11 (D18A)	W_{21C}
				R/S 12 (D17B)	W_{24C}

*HARDWARE CHANGE ON BLOCK IV + SYSTEM

R/S NO. 6 (D14B) DISCRETE

- HARDWARE COMPARISON:

SYSTEM	Y_{21C} MONITOR	Y_{1C} HAZARDOUS CURRENT MONITOR
BLOCK IV	VOLTAGE TYPE	NONE
BLOCK IV+	CURRENT TYPE	VOLTAGE TYPE

- FAULT ISOLATION PROCESSING OF R/S NO. 6 DISCRETE FAULT

  IS: BLOCK IV TAPE = G&C FAULT (MSR 11/MOSR 38)

  CHANGE TO: BLOCK IV AND BLOCK IV+ TAPE = R/S FAULT (MSR 9/MOSR 39)

- LIST OF CONSTANT CHANGES BETWEEN BLOCK IV AND BLOCK IV+

  CYCLE SUM CONSTANT

  HAZARDOUS CURRENT MASK

  CHECK SUM CONSTANT

- BLOCK IV TAPE USED WITH IV+ HARDWARE RESULTS IN INCOMPLETE TESTING OF R/S NO. 6 FUNCTION
- BLOCK IV+ TAPE USED WITH IV HARDWARE RESULTS IN FAILURE OF ORDNANCE DISCRETES TEST

DOWN STAGE FAULTS

CONTROL SYSTEM PATTERN RECOGNITION FOR STAGE 1, 2, AND 3

WS-133B		WS-133A-M							
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	D37 RATE	X_{8C}	V_N	V_R	V_S	V_L
NONE	NONE	NONE	NONE	0	•	0	0	0	0
8	508	29	54	1	*	*	*	*	*
11	511	38	63	0	1	*	*	*	*
				0	0	1	*	*	*
11	511	28	53	0	0	0	1	1	0
&	&	&	&	0	0	0	0	1	*
12	512	38	63	0	0	0	0	*	1
				0	0	0	0	1	1
				0	0	0	1	0	*
12	512	28	53	0	0	0	1	1	1

REMARKS:

* = "0" OR 1
"0" = OK
"1" = FAULT
V_N = NULL
V_R = RATE
V_S = STEADY STATE EXTEND
VL = LAG

PBPS VALVE/STAGE 2-3 ROLL FAULTS

NO-GO FAILURE REPORTING FOR ANY FAILURES DETECTED DURING PBPS VALVE TEST OR STAGE 2-3 ROLL TEST

WS-133B		WS-133A-M		
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	REMARKS
11	511	27	52	MSR 11 OR MOSR 38 = CONTROL AND DISCRETE UNIT FAULT
&	&	&	&	
13	513	38	63	MSR 13 OR MOSR 27 = STAGE 2-3 ROLL FAULT
				ABOVE FAULT SETTING IDENTIFIES VALVE OR DISCRETE FAILURES FROM PBPS ACTUATOR FAILURES.
13	513	27	52	STAGE 2-3 ROLL FAULT

PBPS ACTUATOR TEST FAULTS

PBPS ACTUATOR FAILURE FLAGGING FOR EITHER PITCH OR YAW CHANNEL

WS-133B		WS-133A-M		FAULT PATTERN PER CHANNEL							
D37 MSR	FAULT P/O	D37 MOSR	FAULT P/O	V_N	X_N	V_R	X_R	V_S	X_S	V_R'	X_R'
NONE	NONE	NONE	NONE	0	*	0	*	0	*	0	*
15	515	45	70	1	1	1	0	1	1	1	0
				1	1	1	0	0	0	1	0
				0	0	1	0	1	1	1	1
11 & 15	511 & 515	38 & 45	63 & 70	ALL OTHER PATTERNS							

NOTES: "1" = FAULT AND "0" = OK

* = "1" OR "0"

X_N, X_R, X_S, AND X_R' ARE FAULT MONITOR CHECKS AT NULL, RATE, EXTEND POSITION AND RETRACT RATE

V_N, V_R, V_S, AND V_R' ARE VOLTAGE CHECK AT NULL, RATE, EXTEND POSITION AND RETRACT RATE

Program Testing

- Since the missile test routines were not changed during the development of the -11 program, they were not tested as a part of the verification activity.

- The primary verification/checkout of the missile test functions on the programs prior to the -11 involved the use of DSIM. Tests were also run on the TXO site as a part of checkout and demonstration. These consisted of a complete "go" run, and forced failures to verify the primary failure status indications. The following letter summarizes the testing using DSIM.

1.0 Introduction

In order to check the Missile Test Routine in this tape, the DSIM program has been used extensively. This letter documents the final DSIM runs made for validation purposes.

2.0 DSIM Description

DSIM is a 360 program which simulates D37D operation, instruction by instruction and prints out (with trace on) changes to the A register, timing information, etc.

Recent additions to this program (such as the SRD instruction) have made it possible to simulate all Missile Test instructions (in this tape).

Additionally, subroutines have been developed which simulate the necessary inputs to obtain a 'GO' run. The subroutines simulate all inputs required in Missile Test: discrete feedbacks; actuator and computer feedback voltages (dynamics are simulated as ramp functions); PIGA inputs to the V-loop; C53 operation; etc. These subroutines are being documented in TM242-001 (D. V. Smith, D'641-80).

3.0 Detailed Simulations

Two detailed runs (trace on) were made. One simulates a GO run and one a NO-GO run (with repeated Missile Test). The NO-GO run was made without the input subroutines.

From these runs, the following items have been checked:

A. Individual Timers
B. Keep-Alive Timing (1.5 sec min.). This is an automatic subroutine function when trace is on.
C. Output and Input Sequencing
D. Correct character outputs
E. Correct voltage outputs

3.0 Detailed Simulations (continued)

F. Correct program routing
G. Overall timers in Missile Test
H. Correct 'GO' results in all NO-GO words
I. Correct program flow for a normal 'GO' run (including semi-somnus) and through a repeated Missile Test
J. Overall time (45 sec. max.)

4.0 NO-GO Simulations

The NO-GO tests (trace off) were made to check NO-GO routing, alternate routing, that proper MOSRs are set (and reset), that Missile Test is repeated on a NO-GO and that Missile Test exits properly for all failures. NO-GOs were forced similar to the detailed test plan which was developed for use in the lab. Verification that proper MOSR bits were set was done from the output dump. MOSR words and Fault Data Words were dumped at the end of each simulation. Items accomplished by this run:

A. Missile Test was simulated 137 times through as many different alternate routes as possible.

B. Fault data and MOSR bits were dumped after each run and analyzed for proper response.

C. LAT/CSD Test failure routes were checked (X2 and X4 True/False, X5 False).

D. F/C Electronics (Z14 False) and F/C Hydraulics (Z14 True) monitor failures were simulated. X5 (coupler) failure was forced. F/C Power failure was simulated.

E. Semi-somnus test bypass was checked and test failure was forced.

F. Control system test failures in all three stages and each actuator were forced. Both upstage and downstage patterns were forced and failure in combination with X8 (Fault Monitor) was checked.

G. Computer charge rate failures were forced.

H. All critical discretes were forced true and false, as was Z23. Single crossfire checks were not made.

I. Stage 2 and 3 roll failures were forced.

4.0 NO-GO Simulations (continued)

J. All PBPS valve failures and axial failures were forced individually. The six "PBPS Fault" patterns were forced.

K. X6, X7, and X8 fault monitors were forced true and false.

L. Internal monitor (computer) failures were forced.

M. Both Local and Remote branches were checked.

N. A Fine Countdown Failure was forced.

O. After these runs, the "saved hot loops" were dumped and verified for no change as were the four hot channels.

P. Missile Test run time was verified to be consistent.

Q. Repeat decisions were checked.

R. Exit decisions were checked.

S. MOSR reset in Missile Test was checked.

5.0 Tracing to Final Tape

All tests were made on the octal deck output of cut 12. No Missile Test changes were made between cut 12 and 14. Cut 14 is the Final Tape now scheduled for demonstration.

6.0 Conclusions

It is concluded from these tests that the program routing and function have been verified.

Based on the results of these runs and previous tests made, detailed validation of Missile Test is considered complete and need not be repeated in the lab.

The results of these DSIM runs will be retained in the unit file and are available for review by anyone concerned.

7.0 Recommendations for Tape Demonstration

For Missile Test routine, I would recommend one 'GO' run and one 'NO-GO' run (with repeat) during final tape demonstrations. This will exercise most of the Missile Test program and verify hardware compatibility.

EXAMPLE 5 - P-PLUG SAMPLING

System Considerations

Due to considerations of weapon system security, particularly inadvertent or unauthorized launch threats, launch commands, ELC's, must be validated by the ground program using two code words stored in a physical plug, the P-plug, which can be sampled under D37 control.

Due to the sensitivity of this code information, it must be protected from undue "exposure" to potential unauthorized access.

Program Requirements

The primary program requirements related to the P-plug sampling are the detailed algorithm for validating the ELC's.

A current leakage problem in the P92 hardware on Minuteman II caused a situation where erroneous P-plug samples could occur. A specific sequence of computer instructions was designed (and imposed as "requirements") to work around this problem. This sequence minimized the time of the first P-plug sample and maximized the time between the first and second samples. Two separate hardware changes were made in the P92 to correct the original current leakage problem, with the second fix making the program timing restrictions go away. However, because the P92 fix was not a retrofit change, the program timing restrictions had to be maintained to make the systems with "old" P92's work properly.

The Minuteman III computer (D37D) had a similar though unrelated, problem which affected P-plug sampling. In order to get a valid sample, a double sampling routine was required. However, the timing restrictions of Minuteman II do not apply.

Both the Minuteman II and Minuteman III sampling problems are of such a nature as to be sensitive to the particular codes, computers, and P92 being used, and will not necessarily result in an erroneous sample for a given combination of the three.

The general concern over "exposure" of the P-plug code information resulted in two program requirements: (1) the two words of the code are not to be contained in D37 memory simultaneously, and (2) the code is to be sampled and held in D37 memory as little as possible. Though recognized as specific program requirements, they were not documented in either the Figure A, Part I or the -11 PRD.

The -11 PRD requirements involving the double sample are given below:

Page 3063 (under Execute Launch Command Processing)

> The P-plug No. 1 and 2 codes are obtained by the following sequence:

a. Issue DOB 25 (enables interrogation of No. 1)
b. Sample P-Plug No. 1 code for at least 2 consecutive word times
c. Issue DOB 26 (enables interrogation of No. 2)
d. Sample P-Plug No. 2 code for at least 2 consecutive word times.
e. Issue DOB 0

Program Testing

The tests run during the -11 verification are summarized below:

Page 271, Test No. 6 under Cable Execute Launch Command

The test consists of issuing two ELC's and verifying the proper resultant system modes. ("system" level testing, i.e., proper system response implies proper processing/sampling of P-plug codes.)

One of the stated objectives is to "assure that the P-plug codes are sampled in the proper order and satisfy the PRD timing requirements." However, only the sequence of discrete outputs (DOB's) is examined to verify that the sequence DOB 25, DOB 26, DOB 0 occurs. (It is not completely obvious why this sequence is verified.)

The double sample requirement is not explicitly tested and since this requirement was related to a "marginal" condition, failure to actually program the double sample would not necessarily result in an erroneous P-plug sample for a given P-plug/G&C combination.

The requirements related to code exposure are not explicitly (or implicitly for the matter) verified.

EXAMPLE 6 - PHASE REGISTER MONITOR TEST

System Considerations

The general missile test requirements as stated in the weapon system design criteria is to test those computer, ground system, and missile control functions which are not normally tested or exercised sufficiently to recognize a failure which would prevent a successful launch or flight. There is also a specific numerical requirement stated for the probability of success for terminal countdown and flight.

The phase register is a three flip-flop register whose function is critical to the flight mission. While the phase register functions are utilized during other phases of missile test/ground operation, the computer monitor (feedback) of this register is only used functionally during flight.

Program Requirements

The requirement to test the phase register monitors was added at the Block IV change point. It was suggested by a software system engineer who discovered it more or less by accident (just happened to think of it). It was presented to SAMSO/TRW at the Critical Design Review and they agreed to adding this test. Apparently, an analysis was made to determine the probability of a failure in these monitors which was relatively low but very little additional memory was required to add the test, so it was included.

The specific test requirements are very straightforward consisting of several sequences of loading, sampling, and testing the phase register contents.

Program Testing

The testing of the missile test functions using DSIM is described under Example 4. Since missile test was not tested as a part of the -11 verification it is not known what specific testing if any was performed on this function.

EXAMPLE 7 - COARSE ZETA

System Considerations

In order to use the GCA as an IMU azimuth reference, its angular position relative to the inertial platform must be known precisely.

The initial Minuteman III mechanization for initially aligning the GCA used the physical position stop on the GCA as an indication of relative GCA/platform position.

During the early Minuteman III development, the GCA servo electronics were redesigned as a result of an analysis conducted during the Minuteman II reliability improvement program. As a result of this new servo hardware, it was much more difficult to derive accurate, repeatable measurement of GCA position at the stop.

The new GCA servos were scheduled in at the Block III change point, and a new software mechanization was included for initial GCA alignment. The following letter describes the mechanization change:

Subject GCA Initial Alignment

Introduction

This letter is intended to inform thos concerned of the proposed method of initial alignment of the GCA for Minuteman III, Block III systems. The method will be compared with the present method and rationale for the differences discussed.

Discussion

The present method of alignment is to slew the GCA to its CCW stop and then CW a fixed number of Heading Data Converter counts (ϕ_{Aj}). This method requires a fairly accurate setting of the GCA stop and very good stop repeatability, to have confidence in aligning the GCA to the same position each time.

With the addition of Level Detector #4 to the stable platform, a reference other than the GCA sliding stop has been provided. Level Detector #4 was added to the platform to provide a means for measuring the angle between the normals of mirrors #1 and #2, with the GCA at its reference position. This angle is referred to as zeta (ζ). The measurement of ζ is necessary for SAT calibration; however, a rough measurement of ζ also can be used to align the GCA. That is, it can be used to tell when the GCA is aligned or how much to slew to correct the alignment. Hence, the GCA can be slewed to an initial angle close to the desired angle, by using the stop as a reference then a measurement of ζ made and compared to the initial ζ. If the ζ's agree, alignment is completed. If not, the difference in ζ's indicates without ambiguity how far and which direction to slew, assuming, of course, the same ζ stability as required for SAT.

An example of how the alignment will function should help in clarifying the method. After IMU power turn-on, the platform will be slewed and stabilized to LD #4. The GCA will then be slewed to the clockwise stop. Within 60 ms of contact with the stop, the computer detects the stop. It then zeros the right half of R.2 (GCA position register) and reverses the direction of slew. The commanded slew angle after stop detection is called β_{AP} and will be approximately 90°. After stabilization LD #2 and LD #4 will be roughly aligned. The stabilization tolerance for the β_{AP} slew will be three counts to assure that a stable null exists which will never be rejected, no matter where the stop is located or where right-half, R.2 is zeroed.

After stabilization of the GCA is confirmed, LD #2 is selected and the platform is leveled to that bubble. The number of yaw gimbal counts received as the platform is torqued from LD #4 to LD #2 is a rough measurement of ζ. If no previous measurement of ζ has been made, the GCA is aligned when the magnitude of ζ is less than or equal to three counts. If a previous measurement of ζ is available, the previous ζ is subtracted algebraically from the current ζ and the GCA is aligned when this difference is less than or equal to two gimbal counts in magnitude. If the ζ check is failed in either case, the GCA is slewed to correct ζ and β_{AP} is updated accordingly. In the case of no previous measurement, the corrective slew angle will be the multiple of four counts nearest to ζ. In the case of a previous ζ available, the corrective slew angle will be the multiple of four counts nearest to the difference in ζ's. A discussion of the tolerances and calculations involved is given in enclosure (1).

The advantages of this method are more reliable GCA alignment, no accurate stop setting necessary, precise stop repeatability not necessary, no complicated computer program necessary to control the servo at the stop for accurate measurement.

The only disadvantages that have occurred to the author are the additional time necessary to rough measure ζ (via yaw gimbal counts) to confirm or correct GCA alignment and the programming necessary to accomplish this. This programming is roughly equivalent to that necessary at the stop for accurate measurement.

Questions or comments concerning this mechanization should be directed to the author.

Program Requirements

The original requirements for the Coarse Zeta change were transmitted informally to the programmer, verbablly, be internal letter, and from the factory programs.

The requirements from the -11 PRD are shown on the following pages. These requirements appear under "Operational Modes - Alignment Mode."

6.3.1 ALIGNMENT MODE

Required Functions

1. Initial Alignment

   1. Platform Slew to Roll Gimbal Stop

      During Initial Alignment the platform shall be slewed into the CW roll gimbal stop and then slewed off the stop through an angle α_{AC} (nominal) = 105.5° to an approximate North heading.

      The functions necessary to perform platform slew are detailed in Paragraph 9.1.1.

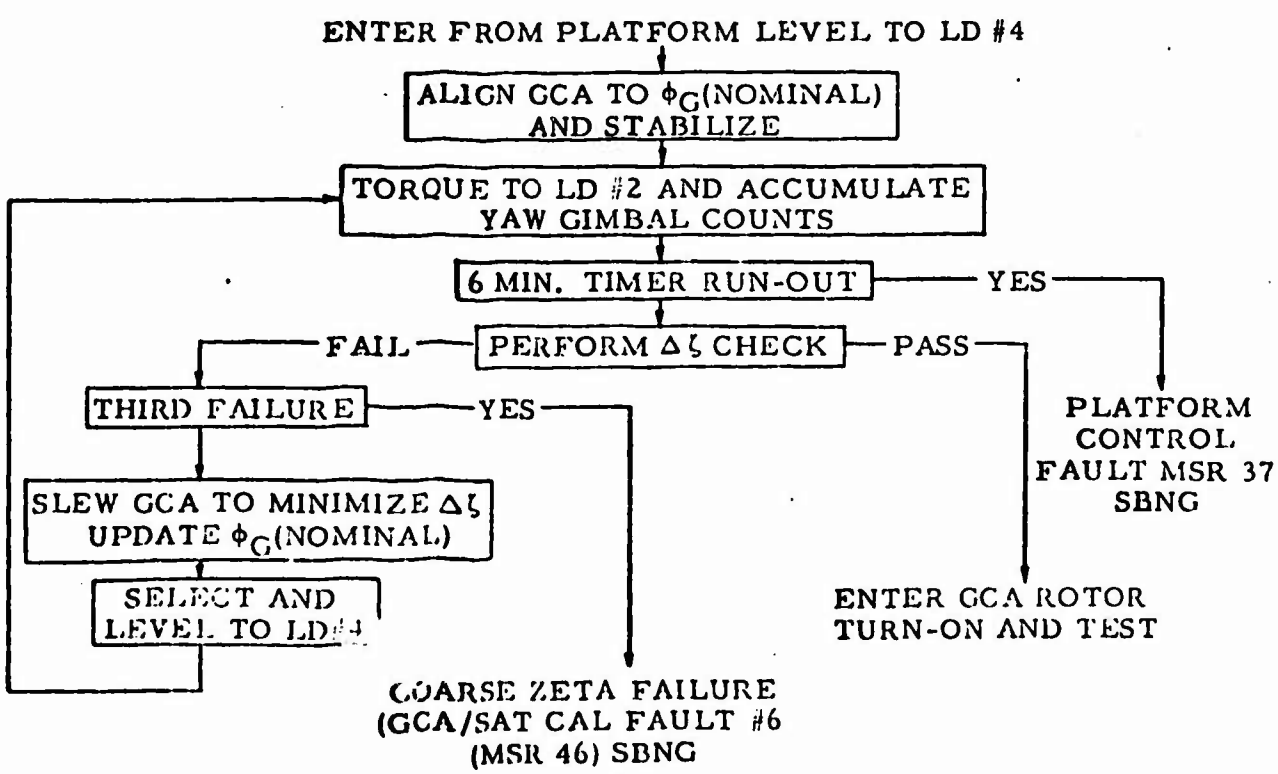
   2. Platform Leveling to LD #4

      In preparation for GCA alignment the platform should be levelled to LD #4.

      This shall be accomplished by slewing the platform about the platform pitch axis towards the CCW stop. If the stop is detected, or if rotating through 90° without sensing the stop, the platform shall be slewed back 45° about pitch in a CW direction. LD #4 shall then be used to level the platform in pitch and yaw.

   3. GCA Initial Alignment and Coarse Zeta

      The sequence of operations shall be as presented below.



6.3.1 ALIGNMENT MODE (Continued)

Required Functions (Continued)

1. Initial Alignment (Continued)

3. GCA Initial Alignment and Coarse Zeta (Continued)

GCA ϕ_G Alignment -

This mode shall be entered under the following conditions:

- X_p at a nominal North heading and pitched over leveled to LD#4.
- Position of the GCA with respect to the platform is unknown and is to be determined.

The GCA shall be slewed to the CW stop then CCW through ϕ_G(nominal) = 92.64° and stabilized.

The functions necessary to perform GCA slew are detailed in Paragraph 9.2.1.

Coarse Zeta (ζ) Alignment -

After slewing the GCA through ϕ_G(nominal), LD#2 shall be selected and the platform shall be torqued at maximum rate about Zp from LD#4 to LD#2.

While torquing from LD#4 to LD#2 the change in yaw gimbal counts shall be accumulated as $\zeta_{(n)}$. If torquing to LD#2 is not accomplished within 6 minutes the system shall exit to Standby No-Go and MSR 37 shall be latched.

When leveling to LD #2 is completed the following check shall then be performed using the accumulated yaw gimbal counts converted to OER counts

$$\left| \zeta_{(n-1)} - \zeta_{(n)} \right| = \Delta\zeta \leq 2 \text{ OER counts}$$

where:

initially $\zeta_{(n-1)} = \zeta_{(factory)}$.

If this fails the GCA shall be slewed to minimize $\Delta\zeta$. The slew angle ϕ_{SL} (used to update ϕG refer to 9.2.1) shall be set equal to $\Delta\zeta$ (OER counts) to the nearest OER null.

6.3.1 ALIGNMENT MODE (Continued)

Required Functions (Continued)

1. Initial Alignment (Continued)

3. GCA Initial Alignment and Coarse Zeta (Continued)

The platform shall then be torqued from LD#4 to LD#2, again, another ζ accumulated and the $\Delta\zeta$ check repeated.

If the $\Delta\zeta$ fails three times then the system shall enter standby No-Go with MSR 46 (GCA/SAT Cal Fault #6) latched (Coarse ζ failure).

4. GCA Rotor Turn-On and Test

Platform Leveling to LD#1 -

The platform shall be slewed 90° CCW about the pitch axis and leveled to LD#1. The roll axis shall be held to the Xp-North heading.

GCA Rotor Turn-On -

Gyrocompass rotor power shall be applied in the following sequence:

a. 40v overvoltage applied for 6±0.5 seconds via application of DO3B.

b. Reduce the voltage to normal output by removing the overvoltage discrete DO3B.

c. Delay for 240±0.5 seconds, during which G6B4 gyro biases shall be established.

"Turn-On Test"-

a. An additional torquing rate of 10 pulses/0.12 sec shall be applied about the platform Yp-axis, the other two axes (Xp and Zp) shall remain in the free inertial mode. At the same time a restoring torque shall be applied in an attempt to cage the gyrocompass.

b. For a period of 15 seconds the state of th. G/C pickoff ($Z11_c$) shall be sampled every minor cycle to establish a reference stating using the last sampled state as a reference.

c. For the next 70 seconds the state of the pickoff shall be sampled to the reference state.

Program Testing

The new coarse zeta mechanizations (and operation of the new GCA servo hardware) was originally tested in the lab using a D37C (Minuteman II) computer and a breadboard servo module. A problem was discovered and corrected in the design of the new servo.

The mechanizations and new servos were further tested through use in the IMU factory prior to delivery of the initial Block III R&D program.

The testing conducted as part of the -11 verification is described below: (Under Operational Modes - Initial IMU Alignment.)

The following tests cover the entire alignment sequence not just coarse zeta.

Test No. 1

Purpose

Verify sequence of events and timelines during "nominal" alignment sequence

Method

1. A nominal alignment sequence is executed on the SSL.

2. The value (octal) of a particular computer scratchpad location (E.3) is automatically printed by the simulator each time it changes value; simulated "real time" is recorded also.

   Platform/GCA position is recorded each time a platform/GCA slew is completed, as determined by the discrete output signal which controls slew.

3. The scratchpad location printed contains the variable information which controls a program "branch" used to sequence the program through the alignment functions. Events during Coarse Zeta are checked by verifying delta times between various values of E.3, sequence of E.3 values, and GCA position after each slew.

Test No. 2

Purpose

Verify response to faults, commands, and transients during initial alignment.

Method

"Restart" dumps are constructed at significant points during the alignment sequence. Using these restarts to re-initialize the simulation, various

error conditions are simulated including circumvention transients. The following are examples of the induced failures:

1. Slew timer runout

2. Circumvention prior to and subsequent to stop detection

3. GCA overrate

4. Leveling Timer runout

The failures are simulated by forcing computer interface signals to failed states. Circumvention is simulated by "pulsing" the computer detector input.

Test No. 3

Purpose

To verify the 3-time failure logic by Coarse Zeta.

Method

Simulation is initialized to the start of the GCA alignment sequences.

While the program is performing the Coarse Zeta measurement, the GCA position input is "forced" to an incorrect reading.

The program attempts this sequence 3-times (each time, the simulator forces a failure). After the third try, the proper No-Go response is verified.

EXAMPLE 8 - SA0 SUBMODE

System Considerations

The Minuteman II and Minuteman III systems use a silo-implaced autocollimator as a reference for azimuth alignment of the IMU. The Minuteman II IMU contained a gyrocompass assembly which was used as a secondary (and degraded) azimuth reference if the autocollimator was not functioning. This mechanization required a highly accurate physical implacement of the autocollimator.

Prior to Minuteman III, a gyrocompassing/gyrocompass calibration technique called Self-Alignment Technique (SAT) was developed at Autonetics. This mechanization, requiring modification to the IMU hardware, provided a sufficiently accurate azimuth estimate to potentially eliminate the need for an autocollimator. A feasibility study was funded, Minuteman II IMU was modified and software developed to test the SAT mechanization which proved successful.

The SAT mechanization was included in the baseline Minuteman III requirements but the autocollimator was still used as the primary azimuth reference with GCA data being used to refine the azimuth estimates. Autocollimator implacement procedures were not relaxed. This was apparently due to lack of confidence and field data on the SAT mechanization and some disagreement as to autocollimator implacement accuracy.

The SA0 mode (and SA3 command) requirements were added at the Block IV (operational) change point. This mode prevented use of GCA data during initial system operation, prior to calibration of some GCA parameters. The change was specifically requested by SAC at the Critical Design Review and was apparently motivated by lack of confidence in the SAT mechanization even though available test data indicated that the mechanization was adequate.

The SA0 mode/SA3 command requirements were deleted on the (-11) revision. By this time, mid-1971, considerable field data on the SAT mechanization was available from system operations. Also, a review of autocollimator implacement accuracies indicated that implacement errors were larger than anticipated in some instances; i.e., the GCA accuracy was more predictable. Reliance was subsequently placed on the GCA data, and autocollimator implacement procedures were relaxed.

Program Requirements

Initial SA0 mode requirements were worked out informally with SAC, SAMSO and TRW at (and subsequent to) the Block IV CDR. Very general requirements were added to the Figure A, Part I. The detailed requirements were developed by Autonetics software system engineering personnel and transmitted informally to the programmer. These requirements are described below.

SA0 Mode

Purpose

The purpose of the SA0 submode is to provide a method for using an accurately aligned autocollimator as an azimuth reference when entering Strategic Alert from Initial Alignment.

Description

When in the SA0 Submode the system will enter Strategic Alert with $\phi_z = 0$. Upright gyrocompassing will continue normally, however, ϕ_z will not be updated by gyrocompassing data until a successful SAT calibrate has been completed. At that time the system will exit from SA0 and enter the SA1 submode. Gyrocompassing checks are still performed in SA0 (ΔP, $\Delta\phi$, etc) and a resultant GCA failure will exit the system from SA0 to the SA3 submode.

A seismic event will also cause exit from the SA0 submode and entry into the normal seismic avoidance with $\phi_z = 0$ being updated the Q6 torquing pulses and gyrocompassing data upon exit from free-inertial.

Figure 1 presents the events for sequencing the system to Strategic Alert in the SA0 submode. The following has been provided for:

1. SA0 is enabled during Ground Initialization when the Disable Discrete (Dd) is set true.

2. A local command is provided to inhibit the SA0 mode when the alignment accuracy of the autocollimator becomes unreliable.

3. If the QC Rotor Test is failed, the system will exit SA0 and enter Strategic Alert in SA3.

Figure 2 presents system operations when in the SA0 mode of Strategic Alert. The following has been provided for:

1. Upon detection of a seismic the system exits from SA0 and enters the normal seismic response routines with $\phi_z = 0$.

2. GCA failures cause exit from SA0 and entry into SA3 with $\phi_z = 0$.

3. Normal exit from SA0 is accomplished upon successful completion of a SAT Cal. At that time the SA1 mode is entered with $\phi_z = \phi_n - \phi_3$.

Locally Commanded SA3 Submode

Purpose

The purpose of this mode is to provide a method for commanding the system directly into SA3, at any time during alignment, whenever the GCA becomes unreliable.

Description

When SA3 is commanded at the beginning of the alignment sequence, the system will proceed to Strategic Alert, after autocollimator acquisition, in SA3 with $\phi_z = 0$. Gyrocompassing is inhibited and the system can only enter SA4 as an alternate submode.

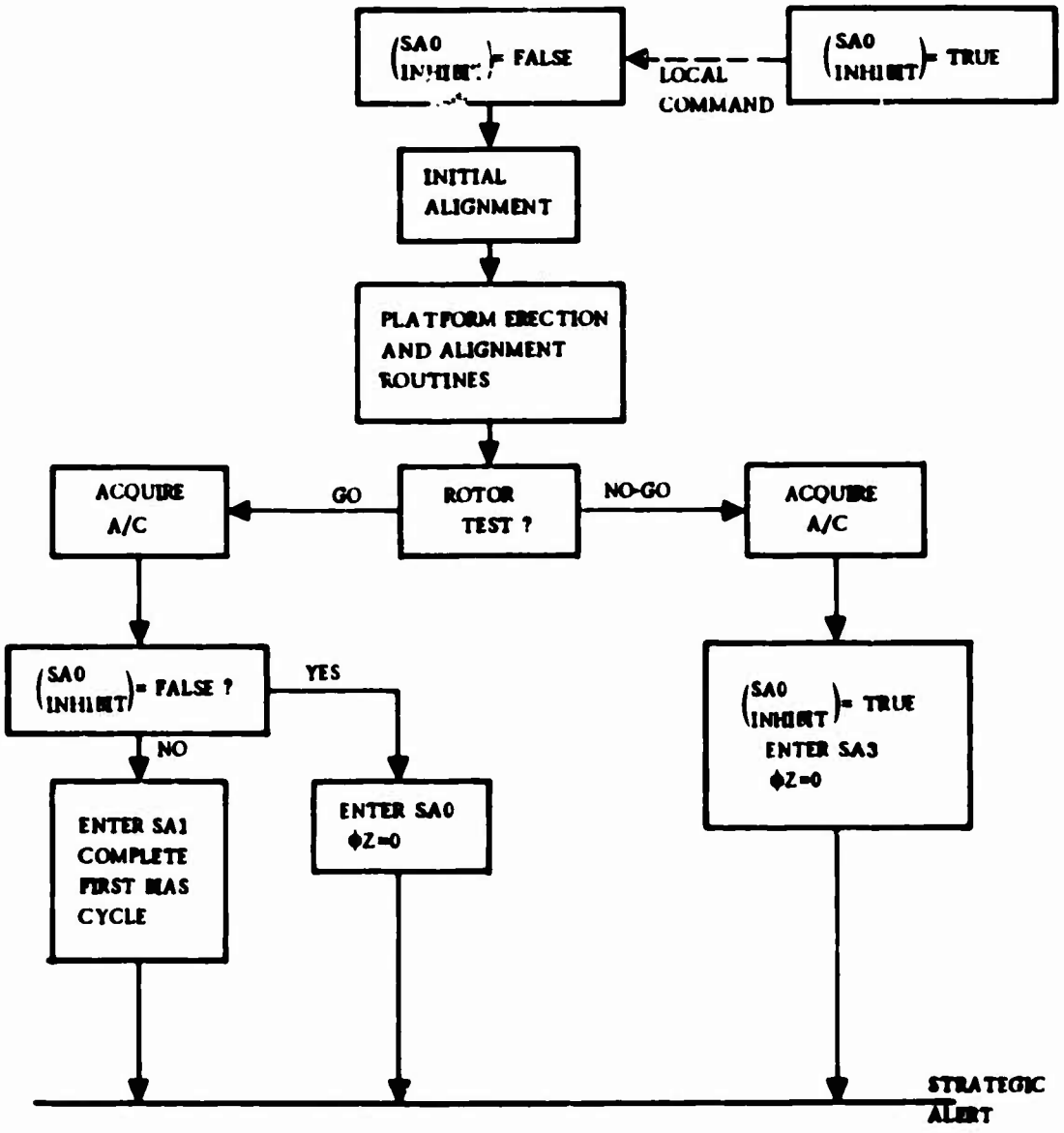


Figure 1. System Sequence to Strategic Alert in SA0 Submode

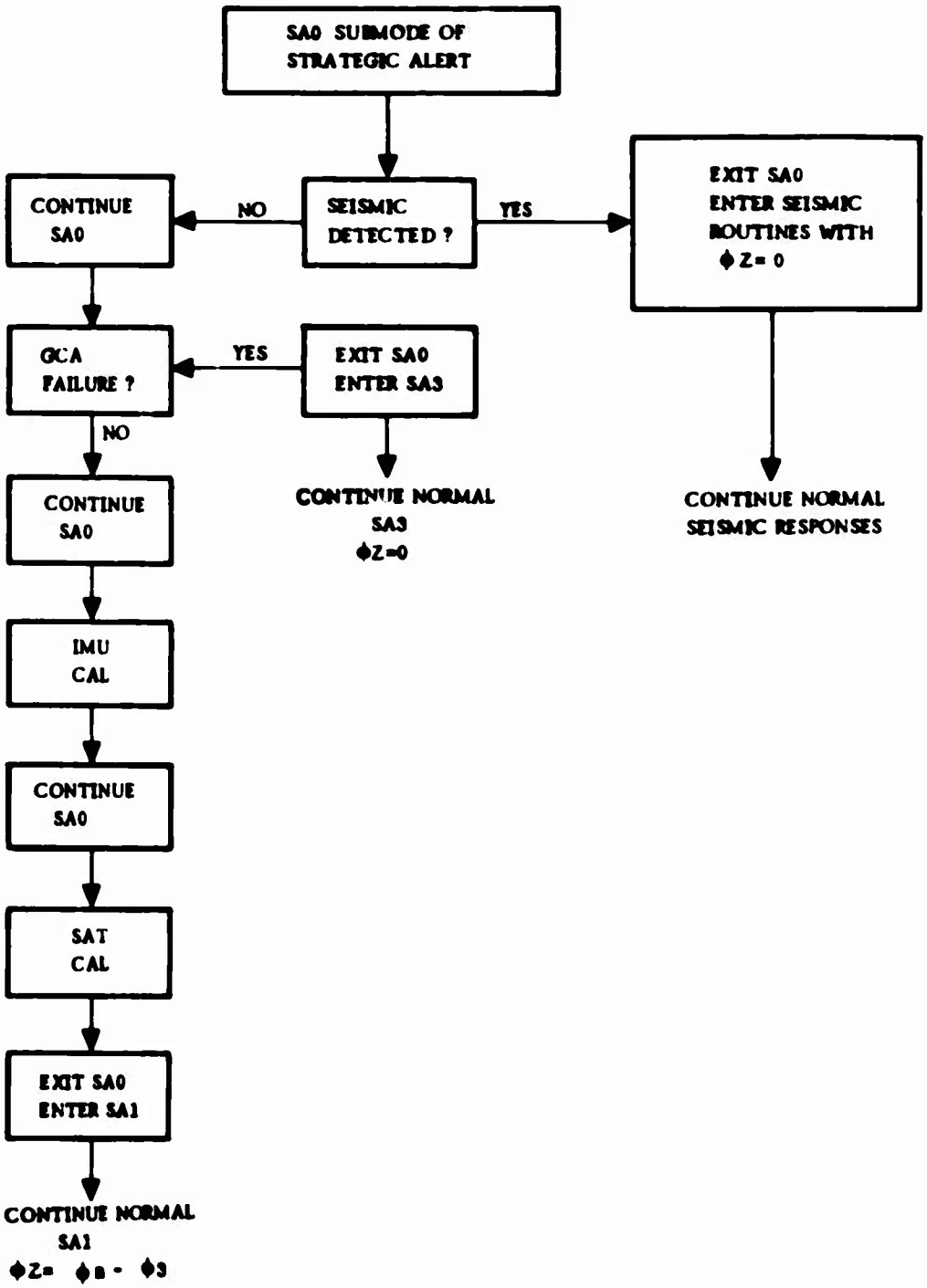


Figure 2. SA0 Operation in Strategic Alert

The rotor test will be attempted in subsequent alignments if SA3 is not commanded.

This sequence is presented in Figure 3.

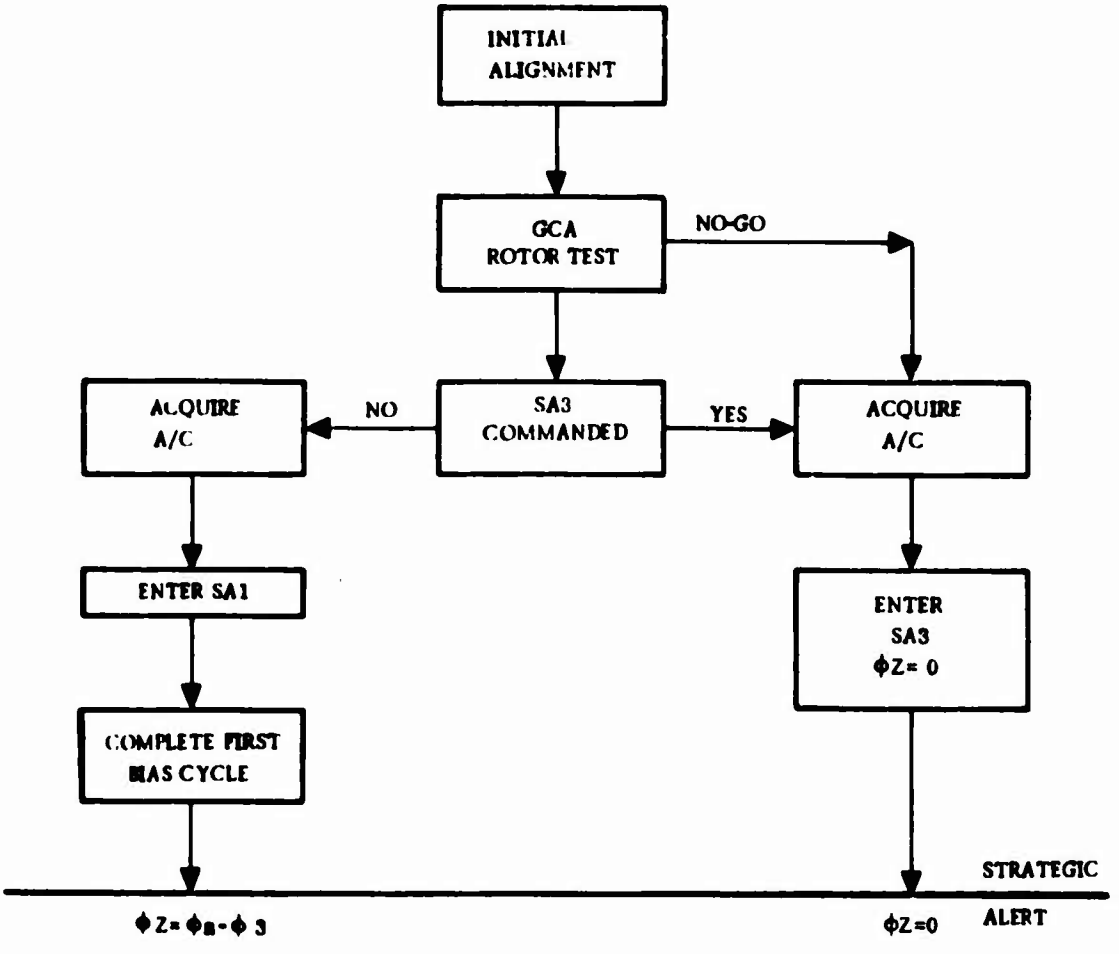


Figure 3. Locally Commanded SA3 Submode

Program Testing

The SA0 mode (SA3 command requirements were deleted at the -11 revision. No specific testing was performed to verify the deletion or to analyze the effects of deleting these functions.

EXAMPLE 9 - TARGET NUMBER CHECK

System Considerations

The Minuteman II and Minuteman III Weapon Systems must have the capability of being targetted to more than one target. Also, it must be possible to change the target assignment for each missile remotely. This is accomplished via the Preparatory Launch Command (PLC) sent from the Launch Control Facility to the Launch Facility. There are two types: PLC-A and PLC-B. The former command specifies an Execution Plan (which includes target number and launch delay time) prestored in computer memory, and the latter specifies the target number and delay time directly as part of the contents of the message. The use of PLC's as described above provides a certain amount of targetting flexibility and allows quick changes of war plans without having to reload target tapes and execution plan tapes at the LF.

The Minuteman II system has eight target sets, and PLC's can specify any target number from zero to seven via three bits in the message. However, the Minuteman III system has only three target sets even though the PLC can still specify eight targets. The Minuteman III software is supposed to ignore any target numbers greater than three.

A programming error in the original Block IV, Wing VI ground program caused target No.'s 4, 5, 6, and 7 to be processed as 1, 2, 3, and 1, respectively. This error was discovered accidently by an Autonetics programmer in the course of analyzing the program instruction sequence while investigating an unrelated problem subsequent to delivery of the program.

Program Requirements

The Figure A for the Block IV program specified the following requirements:

Receipt of PLC with Improper Target Number

If a PLC specified, directly or indirectly (via the loaded Execution Plan Tape), a target number which is higher than possible for the given "front end," or for which the target constants have not been loaded, the DCU will not change selection of the target or perform fuzing and no external reply will be made.

However, the programming for this function was apparently derived from the corresponding function in the Minuteman II, Wing VI ground program and though the Minuteman III program "masked" out the extra target number bit of PLC, no specific test included to recognize an improper number.

The program error was corrected in the -11 revision to the Wing VI ground program. The requirements in the -11 PRD are specified as follows:

The target number indicated by bits T2-T3 of the target data word is either 1, 2, or 3. If a target number 4 through 7 is indicated, the PLCA shall be ignored.

Program Testing

It is not known exactly what testing was performed in this area on the original Block IV program. It may be speculated that the reason that the problem was never found in previous testing is that improper targets do not exist in the execution plans used during testing and were never commanded via PLC-B's. After the problem was corrected, tests were run to command all eight targets to assure that 1, 2, and 3 were accepted and 0, 4, 5, 6, 7 were rejected.

The -11 verification contained tests for rejection of PLC-A specifying execution plans for targets 4, 5, 6 and 7. No tests were written for target 0 since the original tape excluded that target already. No tests were done for PLC-B's or for radio commands due to the processing commonality in the problem. The tests were designed to test the program fix only and were optimized with respect to limiting the number of tests. This was accomplished by identifying program commonality in the PLC processing.

The tests were performed by modifying (with a Key-in) the target number stored in one of the execution plan data words. The system (computer, IMU, LCF, simulators) was operated in a normal "Strategic Alert" mode and transmission of a PLC-A message was simulated followed by status request and status response messages. The status responses were examined to verify that no change had occurred as a result of the PLC-A. This procedure was repeated three more times modifying the execution plan data to specify target No. 5, 6, and 7, respectively.

Additional testing of the function was performed as part of the formal demonstration of the -11 program. Tests verified that PLC-B's specifying targets 1, 2, and 3 were accepted and targets 0, 4, 5, 6 and 7 were rejected. No tests were done for PLC-A's or radio commands. PLC-A's were not demonstrated because it would have been necessary to modify the execution plan constants in memory, and memory modification is not normally permitted during a formal demonstration. Radio commands were not tested due to processing commonality between cable and radio PLC's.

EXAMPLE 10 - GCA SLEW/FUZING "RACE" PROBLEM

System Considerations

The primary operational mode of the Minuteman weapon system is called Strategic Alert. This mode indicates an "all system go" condition and is necessary in order for a launch to occur. The Strategic Alert mode may be interrupted when a target or execution plan change is commanded. This results from the receipt of a PLC message (see description under Example 9). The primary reason for this interruption is the time required to change the fuzing of the re-entry system.

A specific time limit is specified in the G&C design criteria which limits the allowable time out of Strategic Alert. This limit is a function of the number of RV's which need to be fuzed, and was derived from analysis of the time that would be necessary under worst case conditions.

A program error in the Block IV, Wing VI ground program caused this specified time limit to be exceeded by a factor of three under a specific set of circumstances. The situation resulted from internal program interference between the GCA slew program routine and the PLC processing routine and would occur when a PLC-A or PLC-B requiring only R/S fuzing and no IMU realignment was received during the time that the GCA was being slewed through 180 degrees. The probability of this set of circumstances occurring was determined to be on the order of 0.01. However, the problem was observed during system testing at VAFB.

Program Requirements

The time requirement specified in the design criteria was never documented directly in the Figure A, ICD, or PRD. However, these documents specify details of the fuzing routine which, when combined in a worst case fashion, agree with the total time requirement. These details cover signal transmission and reception rates, allowable and required program delay times, permissible number of fuzing attempts, and tolerances on rates and times. (It should be noted that these requirements per se were being met by program even with the program error.)

It is not clear whether the original programmer was aware of the specific criteria time limit but in any case, the error was not related to programming a specific function, but to the unplanned interaction of two program routines. Therefore, being aware of the time constraint would not have avoided the error.

Program Testing

No one knows for sure why this problem was not discovered in earlier testing. It can be speculated that either it never occurred (based on low probability of occurrence) or that it occurred and went un-noticed (either no-one paying specific attention or not smart enough to recognize the delay as abnormal).

Testing of the -11 revision which corrected the problem consisted of setting up the problem conditions (specific PLC during GCA slew) and verifying that the return to Strategic Alert occurred within the specified time. This testing was done as a part of the checkout activity.

There was no specific testing conducted during the verification for this program change. The reason is probably because the verification tests were written against the requirements in PRD rather than against the ECS items. The tests which were run exercised the PLC, fuzing, and GCA slew functions independently but not in the specific combination related to the problem.

The verification testing for the -11 revision did include some tests for "coexistence" or lack of interference of program functions. In particular, the PRD includes a matrix of major system modes and command message types which specifies mode/command compatibility. Testing was included to verify each block of the "coexistence matrix." However, in the case of the coexistence of GCA slew and R/S fuzing, the slew and fuzing functions are not "high" enough level modes to be included. Hence, no direct testing was performed to verify these interactions (or lack of interaction).

EXAMPLE 11 - HAF FUZING (AND GROUND POWER FAULT DURING FUZING)

System Considerations

The capability for delivering and detonating a warhead at high altitude was added to the Minuteman III system at the Block IV change point.

The Minuteman system has an overall safety requirement which specifies the probability of a "faulty launch" or "Class A incident," i.e., delivering an armed warhead to a point outside a specified range of the intended target.

The Block IV mechanization contained a potential safety problem such that if an error occurred during a retargeting operation, it was possible to cause a faulty launch (see statement of problem under program requirements).

Program Requirements

The basic high altitude fuzing (HAF) requirement as it affected the ground program was to identify the R/V fuzing message as either HAF or non-HAF.

The ECS description of the problem corrected at the -11 revision is included below.

Title of Change

Setting Infinite Hold for R/S Ground Power Fault During High Altitude Fuzing Attempt.

Statement of Problem

In the present tape, detection of an R/S Ground Power fault (power does not come on when commanded) results in exiting the fuzing routine and setting an alarm. As a result, the R/S remains partially or completely with the fuze settings for the previously selected target(s), but the missile is otherwise targeted to a new target set. Under these conditions, if an R/V previously fuzed for low altitude (ground or air burst) is now targeted to a target requiring high altitude fuzing, it will most probably overshoot the target by more than ____ naut. mi. (faulty launch).

Analysis of the C163 Ground Power circuits (using actual and predicted failure rates) shows that the probability of failure is approximately an order of magnitude greater than the basic Faulty Launch criterion number. In the absence of any system inhibits (e.g., procedures that prohibit launching a missile which has an alarm condition), this failure mode becomes the dominant path in the Faulty Launch Fault Tree.

Justification for Change

This change will prevent launching a missile when a fuzing safety problem exists.

Description of Proposed Change

If an R/S Ground Power Fault occurs during the R/S fuzing sequence, MSR 2 (Signal Converter Failure) will be set and the fuzing attempt will continue. If the R/S Ground Power is not on, the fuzing attempt will fail and a check is made to determine if the fuzing message is for high altitude burst. A HAF fuzing error results in the program setting Δt equal to infinity and no launch mode. If R/S Ground Power came on despite the monitor indication and the fuzing attempt is successful, the program will continue and no further action will be taken.

Program Testing

Minuteman contractors perform a safety analyses for faulty launches via fault tree analysis. This is a statistical type of analysis which is based on the combination of failure probabilities of various elements of the weapon system. A particular failure mode or situation is factored into the overall analysis only if it is identified and included in the fault tree model. This problem was not found via this analysis because it was not part of the model.

The safety problem was noticed by a programmer during Block IV development. The problem was brought to the attention of SAMSO/TRW but it was decided not to change the mechanization due to an assessment that the probability of occurrence was very low. Subsequent to delivery of the Block IV program it was decided that the problem was significant enough to correct.

The -11 verification included two tests which verified response to a fuzing failure with and without a ground power failure.

EXAMPLE 12 - DATA TRANSRECORDER (DTR)

System Considerations

The DTR hardware, consisting of a magnetic recorder/printer, was originally developed by Autonetics for the Minuteman II system by direction from OOAMA. The purpose of the DTR addition was to provide additional data on operation of systems in the field and in maintenance/troubleshooting.

The DTR capability was added into Minuteman III at the Block IV change point. However, it was only added to the Force Mod ground program since due to several memory constraints in the Wing VI program.

Program Requirements

The DTR capability was added to the Minuteman III, Wing VI program at the -11 revision. The requirements were initially established via an informal SAC request to SAMSO and thence to Autonetics. Initially, discussions were by telephone between SAMSO and Programmers. Later, the subject came up in a meeting involving both SAMSO and OOAMA.

The initial discussions took place about midway between PDR and CDR. The subject was discussed at CDR with SAMSO, TRW, OOAMA, SAC. This resulted in some conflict of requirements, i.e., OOAMA asked for a more sophisticated and complete DTR routine similar to the one in Minuteman III Wing VI. It soon became apparent that this would cost more computer memory than was available. SAMSO and TRW did not really care one way or another, and SAC only wanted a simple, local mode-only routine. SAMSO finally edicted the latter approach, but did not give Autonetics a final decision until 2-3 months later, resulting in a 2-month schedule slip in delivery of the -11 program.

The detailed -11 DTR mechanization was derived from knowledge of the existing DTR routines and information from SAC personnel. The first ECS drafted by System Engineering did not match up with the programmers ideas of what was required. Consequently, programmer and System Engineer called SAC directly to firm up the detailed requirements. Later, it was necessary to talk to SAMSO and NSA to get clarification on dumping certain secure code information to the DTR.

The requirements from the final version of the ECS are included as follows.

Title of Change

Addition of DTR Capability

Statement of Problem

A means is required to obtain data from DCU memory for field maintenance and system diagnostic purposes. Present methods for obtaining data in the LF (fault data readout words) provide inadequate data in some situations.

Justification for Change

The proposed change will provide access to and recording of all available variable data in the DCU memory when the system is in the Local Mode of operation. This will partially satisfy the long-term data collection objectives of the Minuteman Bench Test Program.

Description of Proposed Change

Capability will be provided to output (dump) DCU memory via the character output lines in a format compatible with the Data Trans-Recorder (DTR) input requirements. The essential features of this change are as follows:

a. Data will be dumped in the Local Mode only.

b. Each dump will consist of the entire contents of the four DCU 128-word hot memory channels (40, 42, 44, and 46).

c. An automatic dump will occur after each startup (Master Reset with IMU power off) except for: (1) the first startup subsequent to a tape fill, and (2) when the CSD(M) arming code exists in hot memory.

Prior to the dump, only the essential system initialization functions will be performed such that a minimum amount of hot channel sectors are changed. Subsequent to the dump, the remaining initialization functions will be performed.

d. C166 Keyboard Command No. 12 will be designated "DTR Dump Command". Execution of this command will result in an immediate dump.

e. The normal Local Communications functions will be inhibited during a dump and will be restored after completion of the dump.

f. A commanded dump will be interrupted by a transition from Local to Remote.

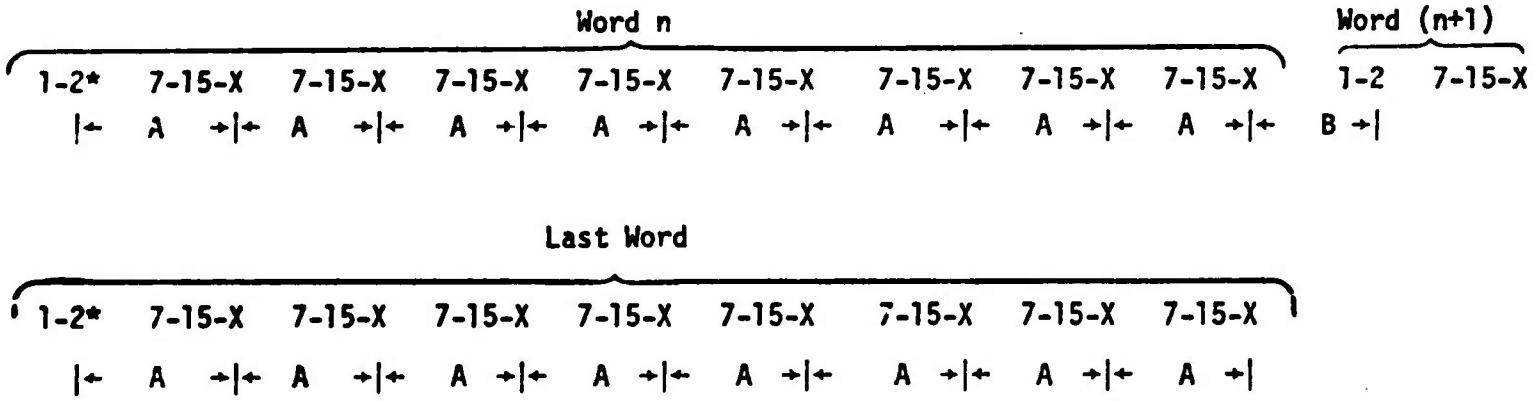
g. A commanded dump will be executed during the Alignment, Calibration, Strategic Alert, Computer Standby, and Standby No-Go Modes. All functions of these modes will continue uninterrupted during the dump and the system will remain in the original mode after the dump.

h. Data will be dumped in octal format, i.e., the most significant bit of the four data bits will be zero.

i. At least one octal number will be dumped each minor cycle (120 ms).

j. The display lights on the C166 will <u>not</u> be blanked out during a dump.

k. Data output format, COA assignments, and interface timing will be in accordance with the following figure:



Time (A) is the rate at which the DTR can accept data characters (X = Data Character). Each character output must be in an "on" state for a minimum of two word times to assure sampling by the DTR.

Time (B) is the period required for the data word (8 data characters) to remain in the output register of the printer to assure proper printout.

CODE	TIME
A	$\geq$ 5 ms
B	$\geq$ 60 ms

* The 1-2 character set may be reissued each time a data character is transmitted.

DATA TRANSMISSION TIMELINE

Program Testing

During -11 checkout, tests were run to verify the requirements as detailed in the ECS such as to assure all data from each memory channel was outputted, response to interrupts, data format. Also, program major/minor cycle timing integrity and non-interference with other program functions was checked. As a result of checkout tests, it was noticed that certain types of interrupts could disrupt the data format and destroy the usefulness of the data. These situations were discussed with SAMSO and it was explained that they were the result of the routine being "simple," i.e., routine did not attempt to provide safeguards against all interrupts. SAMSO OK'd the whole thing and agreed that such problems would be handled procedurally.

The tests run during the -11 verification are described as follows.

14.0 Data Transrecorder (DTR)

Test Number 1.

Purpose

To verify that an automatic DTR dump is inhibited for the first startup subsequent to a tape fill.

Trace to PRD

Paragraph 14.0, Function No. 3.

Test Conditions

Computer in tape fill mode.

Test Procedure

1. Fill Pen Change Tape.

2. Enter Compute.

Required Analysis

Verify no DTR dump occurs.

Data Transrecorder (DTR)

Test Number 2.

Purpose

To verify that an automatic dump occurs after an initial startup.

Trace to PRD

Paragraph 14.0, Functions 2, 3, 5, and 10.

Test Conditions

Initial startup, not first since tape fill.

Test Procedure

1. Prior to startup, record contents of memory channels 40, 42, 44, and 46.

2. Enter Compute.

3. Record Character Outputs (COA's).

4. During DTR dump, issue all Local Commands via C166.

5. Observe Display Lights on C166 during dump.

Required Analysis

1. Compare DTR dump data with Hot Channel data recording prior to startup.

2. Verify that C166 display lights blink during dump.

3. Verify that dump continues when Local Commands are issued.

4. Verify that system is in Computer Standby Mode after dump.

Data Transrecorder (DTR)

Test Number 3.

Purpose

To verify that Hot Channels are dumped in response to a Local Command.

Trace to PRD

Paragraph 14.0, Functions 1, 2, 4, 5, 6, 7, and 10.

Test Conditions

Local Mode and the following:

1. Computer Standby Mode.

2. Standby No-Go Mode.

3. Alignment Mode.

4. IMU Calibration Mode.

5. SAT Calibration Mode.

6. Strategic Alert Mode.

Test Procedure

For each test condition, perform the following:

1. Record contents of memory channels 40, 42, 44, and 46.

2. Issue C166 Command No. 12.

3. During DTR dump execute all Local Commands via C166.

4. Observe display lights on C166 during the dump.

5. Record COA's.

6. After dump is complete, record system status (i.e., Alignment, Strategic Alert, etc) and contents of channels 40, 42, 44, and 46.

7. Re-initialize to original Test Condition.

8. Issue C166 Command No. 12.

9. Record COA's.

10. During dump, switch to Remote.

11. Record system status.

12. Issue C166 Command No. 12.

Required Analysis

1. Compare DTR dump data with recorded Hot Channel Data and verify data are identical except for locations that would normally change over the duration of the dump.

2. Verify that C166 display lights blink during dump.

3. Verify that dump continues when Local Commands are issued.

4. Verify that system is in original mode after each dump.

5. Verify that each dump is terminated at Local to Remote Transition, and that system remains in same mode.

6. Verify that no dump occurs after Step 12.

Data Transrecorder (DTR)

Test Number 4.

Purpose

To verify that CSD(M) arming codes are not dumped to the DTR.

Trace to PRD

Paragraph 14.0, Function 3.

Test Conditions

Remote, Enable Commanded State.

Test Procedure

1. Switch to Local.
2. Issue C166 Command No. 12.
3. Record COA's during DTR dump.
4. After dump is complete, re-initialize to original test conditions.
5. Force entry to Critical No-Go.
6. Enter Compute.
7. Record COA's during DTR dump.

Required Analysis

Verify that memory locations 42-137 and 42-140 do not contain CSD(M) arming code.

Data Transrecorder (DTR)

Test Number 5.

Purpose

To verify that normal program mode transitions will occur during a DTR dump.

Trace to PRD

Paragraph 14.0, Function 7.

Test Conditions

Local, within five minutes of the completion of the following modes:

1. Alignment
2. IMU Calibration
3. SAT Calibration
4. Computer Standby

Test Procedure

Perform the following for each Test Condition:

1. Issue C166 Command No. 12.
2. After completion of DTR dump, record system status.

Required Analysis

Verify that system is in Strategic Alert after completion of dumps for Conditions 1, 2, and 3 and in Alignment after dump No. 4.