General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by NASA STI Compliance and Distribution Services

77-1397

AVIONICS SOFTWARE DEVELOPMENT - EXPERIENCES FROM THE B-1 PROGRAM

E. F. Reed

Boeing Aerospace Company
Seattle, Washington

Abstract

The successful development of B-1 Avionics software has stemmed from basic planning decisions at the program's outset. The program manager designated software as a major engineering organization and allocated task and budget responsibility accordingly. The program chief engineer instituted engineering-wide event scheduling and requirements allocation tasks in which software personnel were significant contributors. These planning decisions, though few in number, established software as a bonafide engineering discipline and created the environment for effective software management. The software manager in turn established policies governing milestone definitions, documentation formats, testing approach and software configuration management. With adjustments, these policies are proving to be effective during continued development and flight testing.

Introduction

Since 1972, Boeing has been the Avionics Subsystems Interface Contractor for the Air Force on the B-1 Manned Bomber program. The initial contract was to develop, integrate, and flight test offensive avionics. A significant feature of the avionics system design was the utilization of several equipment items already operational on other Air Force programs. The contract was expanded to include defensive avionics in 1974.

The primary contract objective has been to integrate avionics equipment items into an operational system. Two key engineering tasks in accomplishing integration were the allocation of requirements to hardware and software and the development of the operational software to meet allocated requirements. Software is a critical element in the avionics architecture because it integrates the sensor, command and response data from the many hardware components to provide required system capabilities. Because of its adaptability to changing requirements, software remains a critical element throughout the life of the avionics system.

No formalized software management concept existed at Boeing in 1972. Concepts had been developing on other programs with embedded computers, particularly AGM-69A (SRAM) and E-3A (AWACS). While the software efforts on these programs were quite successful, it was apparent that additional emphasis was needed in certain areas.

1. System and software requirements should be defined and clearly understood before beginning significant software or hardware design effort.

2. Software documentation should be an integral part of phased software development, with format, content and releases keyed to contractual and program events.

3. Phased, comprehensive software testing should include integrated software testing, within a dynamic and realistic environment and with strict configuration controls, prior to release of software for system validation testing.

4. Software development should be integrated with corporate and program level standards and controls, with additional software standards and controls established to permit efficient and reliable response to change requirements throughout the development cycle.

Based on these past experiences and lessons learned, and with support of program management, a software management approach was conceived and implemented. The approach worked well in developing the baseline B-1 Avionics software configuration. With adjustments to the original plan based on experience acquired during baseline development, it is being effectively applied in the continued development and flight test environment where orderly change processing is essential.

System/Software Description

B-1 Avionics is partitioned into offensive (communicate, navigate, deliver weapons) and defensive (penetrate) subsystem groups. Offensive subsystem development was begun in early 1972, with flight testing still in progress. Defensive subsystem development go-ahead was received in early 1974, with flight testing scheduled to begin in early 1979. The following paragraphs describe system hardware and software and briefly discuss the Computer Program Development Items (CPCIs) developed to support B-1 Avionics integration.

The System

As in many modern-day electronics systems, embedded computers are critical components in the B-1 Avionics System. As shown in Figure 1, three computers (avionics control units) form the "heart" of the system. An avionics multiplex system of seven one-megahertz asynchronous serial digital channels provides the "arteries". The computers are Singer-Kearfott SKC-2070 processors with 32K of 32-bit memory, seven bidirectional serial digital channels and two unidirectional parallel digital channels. They are supported by an IBM mass storage unit with 526K of 16-bit words, a Raymond Engineering Data Entry Unit (tape cassette) with 450K of 16-bit words, and a keyboard/display integrated into the Controls and Displays Subsystem. Other system components include LN-15S Inertial Measurement Units (Litton), APN-200 Doppler Radar (Teledyne Ryan), APN-194 Radar Altimeters (Honeywell), APQ-144 Forward Looking Radar (General Electric), and APQ-146 Terrain Following Radar (Texas Instruments). These components were adapted to the multiplex system using Boeing-developed interface terminals.

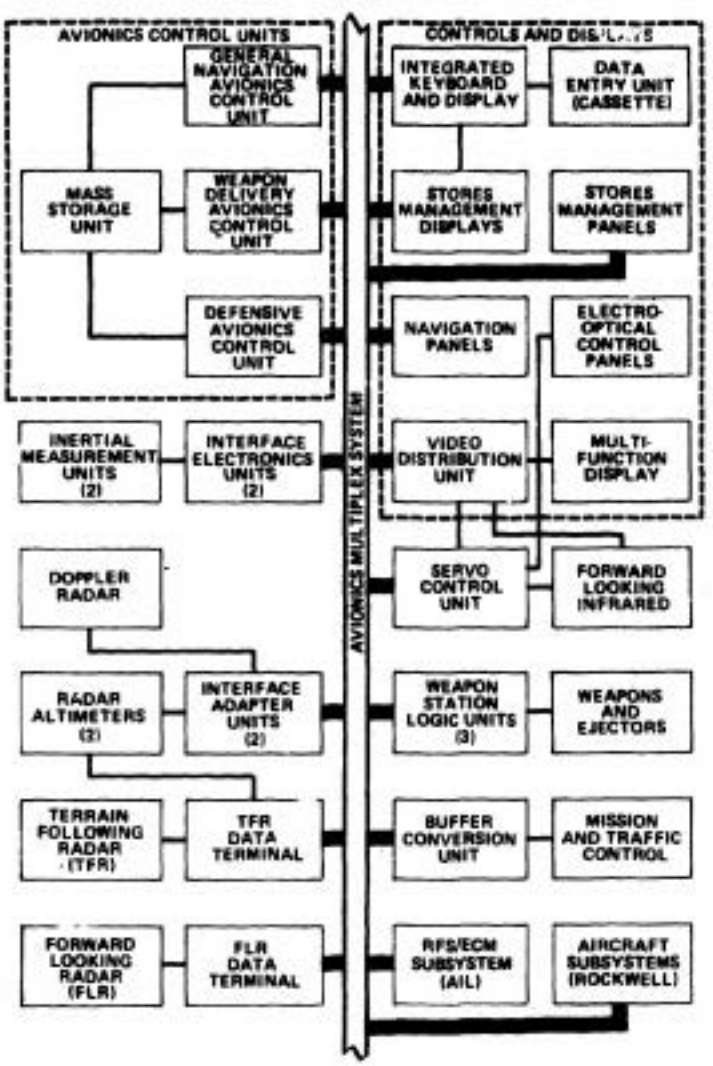


Figure 1: Avionics System Block Diagram

Operational Software

The operational software in the avionics control units integrates the diverse components and subsystems into an effective avionics system. As an example of subsystem operation, Figure 2 shows the five offensive software functions with their major external and internal interfaces.

The executive function initializes the system, performs pre-operational system validation, and manages real-time input-output, interrupt handling, and task scheduling. The executive in one of the control units acts as the controller of the avionics multiplex system. Features include major/minor frame task allocation; program/data roll-in from mass storage or data entry unit; mission essential software backup; and nuclear circumvention control.

The controls and displays function services CRT displays, keyboard, and control panels; processes sensor inputs; and performs mission sequencing. Operator decisions are aided by displayed logic trees. Panels and displays are backed up by integrated keyboard and multifunction display.

The navigation function services the inertial measurement units; performs dead reckoning, inertial and Kalman-aided navigation; and issues avionics steering commands. Independent 17-state Kalman filters support the two inertial measurement units.

The weapon delivery function coarse aligns and fine aligns the missile inertial platforms; computes impact points and launch footprints; controls weapon launch/jettison sequences; and performs simulated launches for test and training. Up to 24 SRAM missiles can be aligned and independently targeted. Stores selection and launch control operate in either manual or automatic mode.

The avionics integrated test function performs system status monitoring, interfaces with the aircraft integrated test system, and provides special test functions.

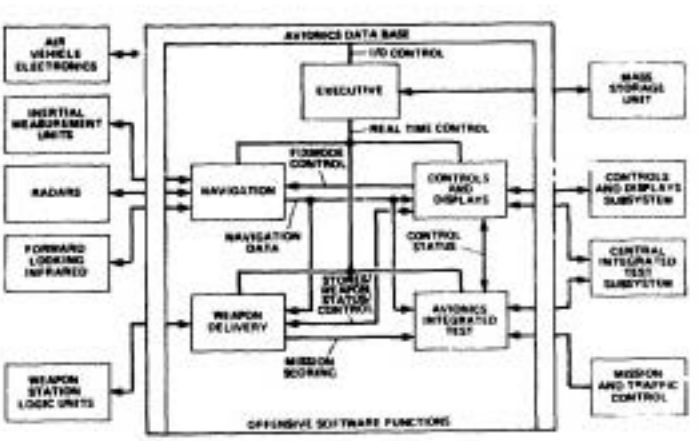


Figure 2: Offensive Software Interfaces

Test Station Software

To support controlled and repeatable testing at all levels, extending from individual software modules to the total avionics system, a computer-controlled test station was developed. This test station interfaces to the avionics system as shown in Figure 3. It includes a B-1 Subsystems Simulator (B-1SS), a Weapon Stores Simulator (WSS), and a Defensive Subsystem Simulator (DSS), each utilizing a Harris 6024/5 24-bit computer. B-1SS software models aircraft flight characteristics, selectively models avionics subsystems, and acquires and records data from the avionics control units. WSS software models weapon stores and their characteristics, and selectively models weapon station logic units. DSS software models the interface to the Radio Frequency Surveillance/ Electronic Countermeasures Subsystem. Shaded areas denote equipment which can be included or simulated depending on the desired test configuration. This permits flexibility in test design and test planning.

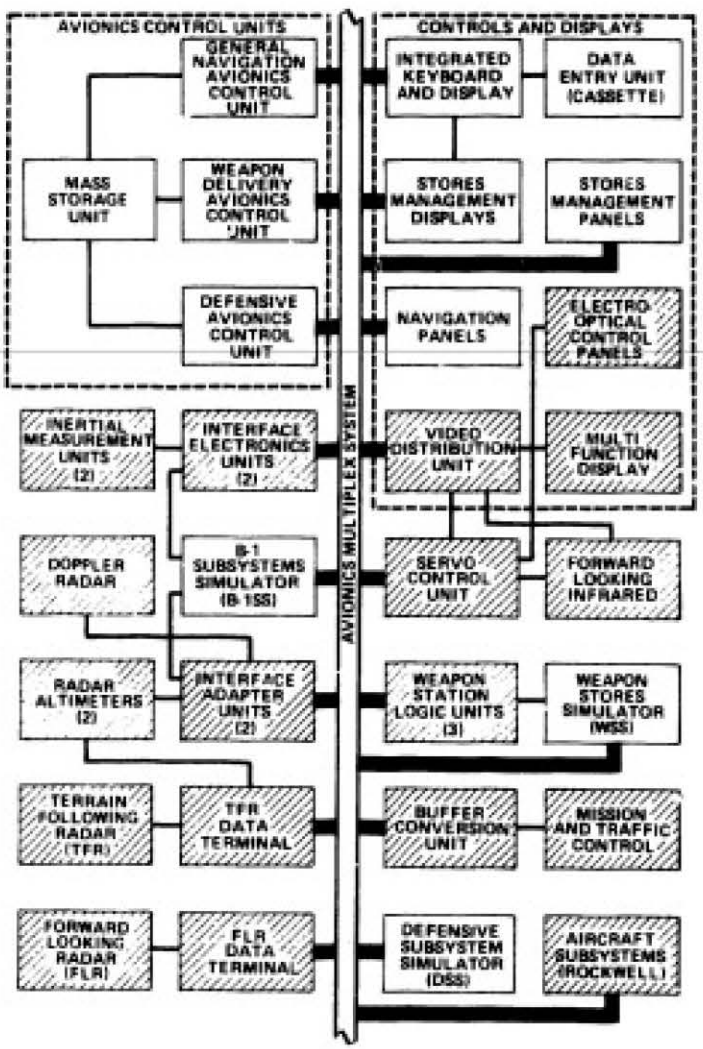


Figure 3: Test Station Block Diagram

Support Software

To facilitate development of operational and test station software, a comprehensive set of support software was provided. This software, designed to operate on most IBM 360 and IBM 370 computer models, includes JOVIAL/J3B compilers (SofTech, Inc.) and Boeing-built language processing tools (assemblers, linkers, loaders), simulators (instruction-level, statement level, environment), and configuration management tools.

Figure 4 shows some of the salient characteristics of the software developed by Boeing for the B-1 Avionics system. Note that over 75 percent of the flight software is in a high-level language.

SOFTWARE TYPE / STATISTIC	OFFENSIVE FLIGHT SOFTWARE (OFS)	DEFENSIVE FLIGHT SOFTWARE (DFS)	SUPPORT SOFTWARE (SS)	TEST STATION SOFTWARE (B-1SS, WSS, DSS)
COMPUTER	SINGER SKC 2070 (2)	SINGER SKC 2070	IBM 360 OR IBM 370 (MANY MODELS)	HARRIS SLASH 5
LANGUAGE	JOVIAL (75%) ASSEMBLY (25%)	JOVIAL (80%) ASSEMBLY (20%)	AED (EXTENDED ALGOL) 99% ASSEMBLY 2%	ASSEMBLY
OPERATING SYSTEM	REAL TIME EXECUTIVE (BOEING DESIGN)	REAL TIME EXECUTIVE (BOEING DESIGN)	STANDARD OS OR VS (MANY VERSIONS)	REAL TIME EXECUTIVE (BOEING DESIGN)
STATEMENTS (LINES OF CODE)	52,000	17,000	125,000	135,000
EXECUTABLE INSTRUCTIONS	93,000	40,000	275,000 (INCLUDING LOCAL DATA)	125,000
DATA BASE	30,000	8,000	DYNAMICALLY ALLOCATED	88,000

Figure 4: Software Summary Statistics

Software Management Approach

The B-1 software management approach is heavily influenced by the management decisions which put it into effect. These decisions were surprisingly few in number; but they created an environment which established software as a bonafide engineering discipline participating in the cost and schedule tracking systems, the primary communication channels of engineering, and the rigorous design approach traditionally reserved for hardware. The management decisions were made on three levels; the program manager, the chief engineer, and the software manager.

Program Manager Decisions

The most important decision was in reporting relationships. The importance of software in integrating the system made it essential that software have a strong voice in system design decisions. Accordingly, the program manager had the software manager report to the chief engineer on an equal level with the system engineering manager and hardware design manager. This relationship provides two important advantages. First, it provides the software manager with visibility of all engineering department activities. Second, it gives him a voice in key engineering decisions. In developing a large system with embedded computers, software participation in system architecture and software requirements allocation decisions is critical.

Along with this reporting relationship went an allocated budget and task responsibility. This required the software manager and his subordinates to plan the development tasks carefully

to schedule tasks and events at a detailed level, and to specify measurable evidence of completion of each task. Weekly reports of budget and schedule status and monthly calculation of the earned value parameters provided visibility and accountability of software as an entity.

Chief Engineer Decisions

Detailed task definitions are essential to a clear understanding of program-level milestones and the interrelationships of tasks among organizations. The chief engineer directed the first engineering activity to be the development of an integrated task and event network. This network defined program-level (top-down) schedule constraints, and identified the necessary engineering organization (bottom-up) commitments to support them.

This network was essential to the software planning process. First, it provided the dates on which other organizations required software design data. Second, it defined demand dates for other organizations to supply written requirements such as algorithms and interface definitions. By participating in the development of the network, the software manager was assured that realistic commitments had been established. Likewise, the chief engineer was assured that the software development plan supported program-level constraints. This allowed a detailed plan for the software activities to be developed, giving direction to the software development effort and providing a baseline from which later adjustments could be made.

The chief engineer assigned the critical task of requirements allocation to a multi-discipline task force. This task force, made up of senior level personnel from systems engineering, hardware design and software, defined the B-1 Avionics System by means of system logic diagrams, hardware/software/system block diagrams, system data flows, and preliminary hardware/software requirements allocation. Thus, prior to any hardware or software design activity, each engineering discipline had a common understanding of the overall system logic and the allocation of system requirements. Also, software personnel had the opportunity to interact with other engineering disciplines and to influence the system design. The system logic, block diagrams, data flows and requirement allocations were used as source data for development of software interface requirements, data requirements and preliminary software design.

Software Manager Decisions

The software manager's basic decisions, documented in a Software Product Assurance Plan, provided the foundation for orderly and controlled software development. These decisions, relating to milestone definition, documentation, test approach, and configuration management, are individually discussed.

Measurable Milestones. The first and perhaps most basic decision was the definition of measurable milestones throughout the software development activity. A milestone was defined to be the completion and release of a tangible output such as a deliverable document, completed code, or a completed test.

Documentation System. The second decision was the choice of software documentation format. Because completed documents were to provide some of the measurable milestones by which development progress was to be measured, a documentation system which coincided with the key developmental events was desirable. A milestone system similar to SAMSO 61-47B was selected because it provided a structure which focused on major development events. At each phase of software development, a document is defined:

Milestone 1	Software Functional Requirements
Milestone 2	Software System Concept and Test Plan
Milestone 3	Interfaces and Data Definition
Milestone 4	Detailed Design
Milestone 5	Updated Detailed Design (as built)
Milestone 6	Test Procedures and Test Results
Milestone 7	Users Manual
Milestone 8	Top Assembly Drawing

Documents are phased with contractual (SDR, PDR, CDR) and program (code complete, software verification complete, system validation complete) events. This enhances the software manager's ability to track progress with measurable milestones.

The Milestone documentation system was used for test station software and support software as well as for operational software.

Test Approach. The software manager's next major decision was to maximize testing of software in its system environment but without actual sensor hardware. This type of testing permits the static and dynamic relationships between software modules to be examined and software functional requirements to be verified. The test station previously described supports this mode of testing.

In module-level verification testing (MVT) modules are first tested individually for nominal operation. They are then integrated one by one into a full software complement, using previously tested software as the environment for the module under test. This testing is conducted by design group personnel.

Inter-module compatibility testing (IMCT) is accomplished with a full complement of software, but no sensor hardware. Repeatable test conditions are provided to exercise the system dynamically in its full range of operating conditions, including off-nominal conditions. This testing is conducted by personnel outside the design organization.

System Validation Testing (SVT) utilizes actual sensors as well as sensor models. It is conducted by the program-level test organization with Quality Control buy-off. Software errors are minimized at this, more costly, test stage because of the previous comprehensive test phases.

Configuration Management. The fourth major decision was to impose rigorous configuration control on software and its documentation. All software and documentation is entered in a Computer Program Library (CPL). Once entered, any change to software or documentation must be authorized by software management. This policy is carried out by personnel outside the design groups. These configuration control personnel were given the responsibility of ensuring CPL integrity, and of implementing a system to provide controlled yet expeditious changes to the CPL. The resulting change control system requires that control forms be used to define and authorize any change prior to its release.

The Software Problem Report (SPR) is used to identify a software deficiency (software which incorrectly implements a required software function). The SPR is used when the problem is one of implementation and not of requirement definition. It can be generated by design or test personnel; when generated, it is entered into the software configuration management accounting system. This is an automated reporting system providing weekly status to designers. The SPRs for which each designer is responsible are identified along with required incorporation date. An SPR is closed when a correction approved by both design group and test group supervisors is received by the CPL librarian. The SPR mechanism provides for controlled correction of software errors after the software is entered in the CPL.

The Design Change Request (DCR) is used to control modifications to allocated system requirements. Any released design can only be changed by DCR. Associated coding changes are also authorized by the DCR. An even more important use is to identify all module revisions required to implement a system change which imposes new software requirements. A single allocated requirement change is translated into one or more DCRs identifying and authorizing each required software module change.

Software Organization. The B-1 Avionics Software organization shown in Figure 5 follows naturally from the above management decisions. In addition to the groups responsible for implementing the software, a separate Software Test Group is assigned overall testing responsibility, and a Configuration Management Group is assigned change control responsibility. These two groups form a staff which enables the software management to ensure software reliability and baseline integrity.

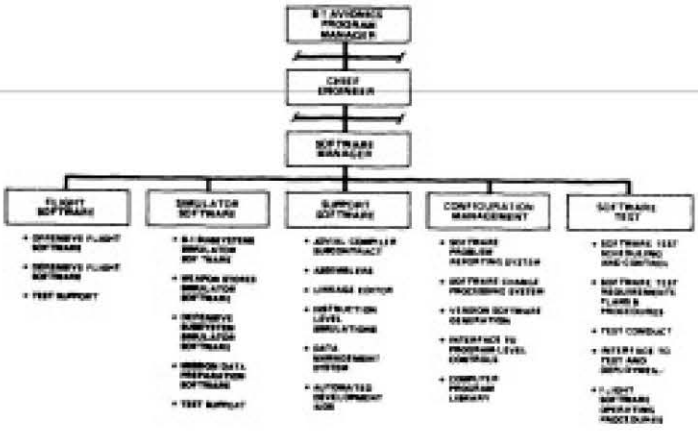


Figure 5: Software Organization

Software Management Emphasis During Flight Test

A software baseline was completed and validated by laboratory testing in April 1975 using the software management disciplines and controls described in the previous section. The laboratory validation proved that the software baseline satisfied the allocated system requirements in a simulated environment.

The cost, schedule and technical performance of the software organization during the baseline development verified the management approach and established a solid management and software baseline for support to the avionics flight test program. Because the purpose of flight testing is to validate the implementation of the basic system assumptions, it was recognized that software errors, algorithm deficiencies, and need for basic requirements changes could be found on a continuing basis. Changes to the integrated software would be required to implement many of the required system corrections.

This section discusses the framework of multiple configurations, long range scheduling, change and resource control which was established to accommodate software changes during flight test while preserving software reliability and configuration integrity.

Projected Needs

Following initial baselining, multiple configurations and large numbers of changes were anticipated -- a natural consequence of a major system validation and flight test program. The multiple configurations were required to support:

1. special navigation system testing using a C-141 aircraft;

2. an offensive system configuration for flight testing on the third B-1 aircraft;

3. an offensive/defensive configuration for flight testing on the fourth B-1 aircraft;

4. a configuration for each of eight B-1 production lots (since cancelled).

Anticipated change activity included new requirements generated by the Air Force; internally generated implementation changes; and quick turnaround changes in direct support of flight test. In practice there have thus far been 29 new customer requirements, 329 internally-generated changes, and 74 quick turnaround changes.

Long Range Scheduling

The scheduling of changes is key to both configuration control and support to system

development. The need for a firm long range software change schedule was recognized. Blocks of time for phased development were allocated and keyed to program events which require revised software configurations.

The time blocks were broken down into design, coding, module verification testing, inter-module compatibility testing, and system validation. This applied to test station, and support software as well as operational software. These time blocks could be realistically scheduled because they were essentially a repeat of the flow time required for baseline development. The software organization was thus able to identify change impact and plan the support required for each configuration, as soon as the unique functional requirements for that configuration were defined.

Committing and Incorporating Changes

The long-range software change schedule structures the program activities affecting software. When a change is authorized, its effectivity is defined in terms of the program event to be supported. Each change is assessed by the software organization for its impact to the software program; the lines of code to be revised or added; the design time; and the test time. Requirements for revision to the support or simulator software are identified at this time, and their design and test time is estimated. Manpower and test equipment requirements are defined. The appropriate software change block is identified, and available flow time is assessed for adequacy. All software changes are committed to appropriate time blocks based on the recognition and analysis of these factors and constraints.

As changes are committed to a time block, the configuration management group maintains a log of the changes and the configurations affected. Any software errors to be fixed are included for commitment to the block. Other pertinent data such as program change number, status of incorporation, and required retest for each change is identified. This log tracks the design time and test time committed in a block. When the time block is filled, no more changes can be accepted.

The block change log is the instrument for coordinating software development efforts. Only accepted and approved changes are allowed for the block design/test task. Thus, designers make only the proper changes, and test planners have early visibility of required testing.

Tracking Block Changes

The task from this point through validation of the new software version is a repeat of the activities of the baseline development process. The engineers are familiar with the software and the process; the sequence is therefore straight-forward for each individual change. Managing the simultaneous incorporation of many changes in a block (285 total module revisions in the current block) is also straight forward but requires a good plan and careful tracking.

The block change effort is monitored by the software manager via a tracking plan prepared by the design group supervisor. This plan identifies the number of individual changes to be completed each week of the design, code and test effort. Statusing is done on a 0/100 percent basis (a task is considered zero percent complete until it is one hundred percent complete). Because the software change blocks are also tracked at the program level, progress on completing the plan is reviewed by the program manager weekly.

The management tools used to monitor technical and management performance of the software organization during development of the baseline are also used during block changes. The two most significant are:

Technical Performance Measures (TPM) which track the effect of changes on computer memory and timing. The impact on TPM parameters is assessed for each change by the design group and reviewed monthly by the program manager;

Earned Value which compares the actual and budgeted cost of work performed. This is also reviewed monthly by the program manager and provides early visibility if software development problems exist.

Managing Multiple Configurations

The method of controlling the content of each flight software configuration is set up through the compile/assembly process. For operational flight software, one baseline configuration is established in the CPL. This baseline contains all the programs required by any of the several multiple configurations. Many of these are identical for all configurations; many require code differences for each configuration; and some programs are not required for some configurations. A particular configuration of software is constructed by selecting the required programs and the required unique code for those programs from the CPL baseline through the use of processor control cards in the compile/assembly process. This permits a minimum CPL and therefore the minimum number of lines of code to maintain.

Quick Turnaround Changes

Software changes are sometimes required on an urgent schedule to support the next B-1 flight. These changes can't wait for the next scheduled change block; therefore, a temporary change is made to the program object code.

The implementation of such a change is carefully controlled by the full application of a disciplined approach. Every step of authorization, requirements definition, design, test, formal engineering release, and revalidation is repeated for each such change required to the object code. At the completion of the revalidation of the change, a merge tape is prepared from the computer memory. This tape contains the operational flight software object code, including the revised code. The merge tape is loaded back into the computer to

assure that the program will load and to verify the checksum. This tape is placed under formal control and sent to remote base for installation on the aircraft.

This process of providing a new program tape with the merged object code changes has an important advantage over the more common practice of entering object code changes directly into the computer at the remote site by either manual means or with separate change tapes. The tape, including the revision, is loaded directly into the computer on the airplane and the risk of improper installation is avoided. The software organization retains full control of the entire software load; and no doubt as to its content exists.

Special Software Programs

Tracking and incorporating the large number of changes required in the operational software during flight test presents opportunities for innovations in the use of the computer to reduce the manual labor required for the many menial tasks related to changing software. The B-1 Avionics software organization has developed programs to assist in document generation and revalidation of revised baseline software.

Document generation programs provide the capability to generate the data base documentation of Milestone 3 and the flow charts of Milestone 5 from operational program source cards. These programs combine actual program features -- data declaration statements for Milestone 3 program branch points for Milestone 5 -- with program comments to generate formatted output.

Revalidation programs provide the capability to compare operational program versions at the source code level, the object level, and as allocated to computer memory. Comprehensive listings of all differences detected are printed. Analysis of the differences detected between revised software and the validated baseline from which it was derived ensures that no unexpected changes are included and provides explicit information used to derive software retest requirements.

Results

The B-1 Avionics has been in flight test since March 31, 1976, completing 46 flights through September 16, 1977. Throughout this test period no flights have been delayed or terminated due to software. Results achieved in flight test verified that the B-1 Avionics System met or exceeded all major system level requirements.

Since 1972 no program level schedule milestones have been missed and an underrun cost performance was achieved.

These achievements have been acknowledged by the customer with three consecutive unprecedented 100% incentive award fees since 1975.

These results lead us to conclude that the software management approach to software development on the B-1 Avionics Program has been effective and has made a significant contribution to the program's success.

Observations

The development of large complex software systems on schedule, within budget and meeting specifications, depends as much on the software organization reporting relationship which the program manager directs as on technical ability to design and code software. Because the B-1 Software Manager reports to the Chief Engineer, he is allocated separate budget and therefore must develop detailed plans and schedules to accomplish the software task within the budget. The monthly earned value calculation provides the Software Manager with the incentive to run a disciplined organization, require well defined requirements and establish carefully thought-out documentation, test and configuration control policies.

The problems of managing large embedded computer software systems during continued system development are amenable to solution by long range planning and tracking of large numbers of changes. An area which needs further development is in the use of special techniques to relieve the software engineers of the menial tasks associated with software development.

Acknowledgements

I wish to thank R.L. Roe and R.L. Wilson for moral support and their valuable contributions to this paper. Thanks are also due to Dr. M.T. Holden for her assistance in editing and rewriting.