Space Shuttle Primary Onboard Software: STS-1 to Operational Use

A.J. Macina*
International Business Machines Corporation, Houston, Texas

The Shuttle primary onboard software and associated computer hardware comprise a data processing complex responsible for virtually all intersystem data flow and vehicle external interfaces. Onboard functions related to guidance, navigation, flight control, systems monitoring, and management, as well as vehicle/crew/ground interfaces, are handled by this primary system. The organization that prepared the software for the Shuttle orbital flight tests emphasized a disciplined design and development phase coupled with a thorough independent verification activity. Current focus is centered on tailoring the software organization, test philosophy, and maintenance and test facilities to the high flight frequencies and diverse missions expected during the Shuttle operational era.

Shuttle Software Overview

THE successful completion of the first orbital flight of the Shuttle (STS-1) in April of 1981 marked the culmination of one of the most complex software development and integration activities ever undertaken. The evolution of the Shuttle's data processing system (DPS) has spanned more than eight years. Along with the orbital flight tests, the development program also included the approach and landing test (ALT) project during which a Shuttle vehicle was launched from a Boeing 747 aircraft and performed the terminal approach and landing portions of the Shuttle mission. Five such ALT flights were performed during August through October of 1977 providing an early test bed for both the DPS hardware and basic software architecture that would eventually be used for the orbital test missions.

With respect to size and complexity, the software for the first orbital flight test of the Shuttle involved eight separately executable programs or memory configurations sharing a common operating system (see Fig. 1). These programs were stored on a mass memory tape device and were loaded into the onboard computers on crew request. One of the eight performed all functions necessary to the launch, ascent, and orbit insertion phases of the mission. Two others provided software to be used during on-orbit operations and included both navigation and control functions as well as system monitoring and management functions. Another memory configuration performed all functions necessary to the de-orbit, entry, and landing mission phases. The remaining four were designed to perform critical prelaunch and on-orbit vehicle checkout procedures. In all, these eight programs, including the software operating system, comprised approximately one-half million 32-bit words of data and executable instructions. All Shuttle applications software modules and a large part of the operating system included in these memory configurations were implemented in the HAL/S high-order language. The language and associated compiler, specifically developed for aerospace applications, are scientifically oriented and especially suited to real-time systems.

The criticality of the software to the Shuttle is emphasized by its numerous interfaces with other vehicle subsystems. There are few functions integral to the Shuttle operation for which the software does not perform some type of computational service. Specifically, the onboard software is responsible for the guidance, navigation, and flight control functions performed during all flight phases. This includes both the gathering of environment and sensor input data and the issuing of commands to the vehicle effectors (engines, aerosurfaces, etc.). The software also handles all vehicle/ground interface functions prelaunch through landing via a direct launch data bus (ground) or telemetry (in flight). In addition, it provides many crew/subsystem interfaces. Other software functions include the management and monitoring of onboard systems, fault detection and annunciation, preflight and pre-entry checkout, and safing procedures.

The number and size of the services performed by the onboard software are not the only factors contributing to its complexity. The requirement for redundancy to achieve reliability has also been a factor. To obtain the required "fail-operational/fail-safe" reliability, the software in certain critical flight phases must execute redundantly in multiple computers. For example, during the STS-1 mission the software for the ascent and entry phases executed redundantly in four of the five IBM System/4 Π Model AP-101 general purpose computers (GPCs), the fifth computer being reserved for the backup flight system (BFS). To achieve this redundancy, an intercomputer synchronization scheme which guarantees identical inputs and outputs from the redundant computers had to be developed. The software needed to support this redundancy requirement is an integral part of the operating system architecture. It provides such functions as computer synchronization at rates in excess of 300 times per second and control of input data to assure that all computers receive identical information from redundant sensors whether or not hardware failures have occurred.

The delivery of the software for the STS-1 and STS-2 missions saw the completion of an onboard system capable of supporting a majority of the basic Shuttle functions. Software development activities from there forward were reduced and restricted to the areas of payload support and handling, and enhancement of on-orbit maneuvering and rendezvous capabilities.

Primary Flight Software Architecture

The main memory capacity of the AP-101 computer (106K, 32-bit words) precludes all eight operational programs (OPS) from being resident at one time. As depicted in Fig. 1, OPS are, in general, associated with specific mission phases in order that computer main memory overlays are avoided

Submitted Oct. 21, 1981; presented as Paper 81-2135 at the AIAA Computers in Aerospace III Conference, San Diego, Calif., Oct. 26-28, 1981; revision received Nov. 5, 1982. Copyright © 1982 by A.J. Macina. Published by the American Institute of Aeronautics and Astronautics with permission.

*Senior Systems Analyst, Onboard Space Systems, Space Shuttle Programs, Federal Systems Division. Member AIAA.

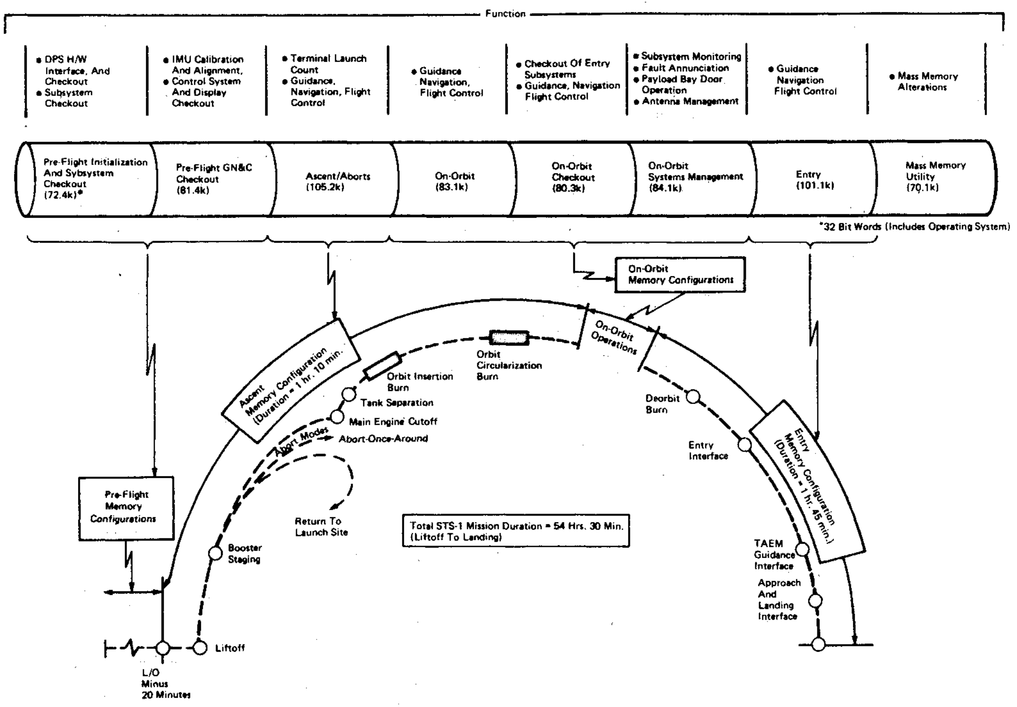

Function
• DPS H/W Interface, And Checkout • Subsystem Checkout
• IMU Calibration And Alignment, • Control System And Display Checkout
• Terminal Launch Count • Guidance, Navigation, Flight Control
• Guidance Navigation, Flight Control
• Checkout Of Entry Subsystems • Guidance, Navigation Flight Control
• Subsystem Monitoring • Fault Annunciation • Payload Bay Door Operation • Antenna Management
• Guidance Navigation Flight Control
• Mass Memory Alterations
Pre-Flight Initialization And Sybsystem Checkout (72.4k)*
Pre-Flight GN&C Checkout (81.4k)
Ascent/Aborts (105.2k)
On-Orbit (83.1k)
On-Orbit Checkout (80.3k)
On-Orbit Systems Management (84.1k)
Entry (101.1k)
Mass Memory Utility (70.1k)
*32 Bit Words (Includes Operating System)
On-Orbit Memory Configurations
On-Orbit Operations
Ascent Memory Configuration (Duration ≈ 1 hr. 10 min.)
Orbit Circularization Burn
Orbit Insertion Burn
Tank Separation
Main Engine Cutoff
Abort-Once-Around
Abort Modes
Deorbit Burn
Entry Interface
Entry Memory Configuration (Duration ≈ 1 hr. 45 min.)
Pre-Flight Memory Configurations
Return To Launch Site
Total STS-1 Mission Duration = 54 Hrs. 30 Min. (Liftoff To Landing)
Booster Staging
TAEM Guidance Interface
Approach And Landing Interface
Liftoff
L/O Minus 20 Minutes

Fig. 1 STS-1 Shuttle mission profile and operational flight programs.

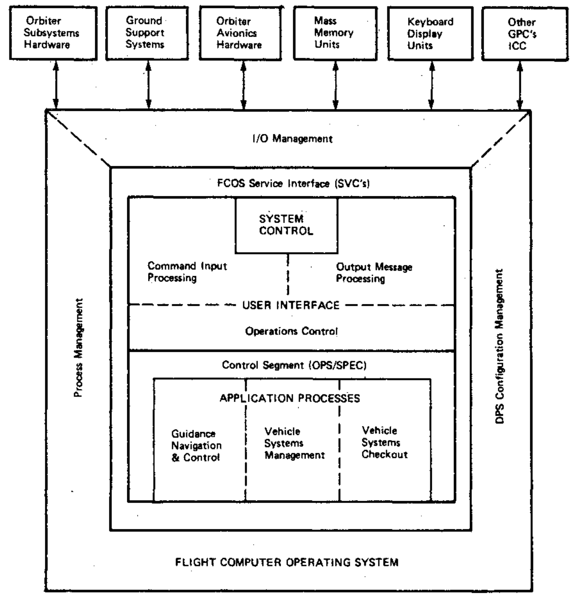

Orbiter Subsystems Hardware
Ground Support Systems
Orbiter Avionics Hardware
Mass Memory Units
Keyboard Display Units
Other GPC's ICC
I/O Management
FCOS Service Interface (SVC's)
SYSTEM CONTROL
Command Input Processing
Output Message Processing
USER INTERFACE
Operations Control
Control Segment (OPS/SPEC)
APPLICATION PROCESSES
Guidance Navigation & Control
Vehicle Systems Management
Vehicle Systems Checkout
Process Management
DPS Configuration Management
FLIGHT COMPUTER OPERATING SYSTEM

Fig. 2 Software architecture.

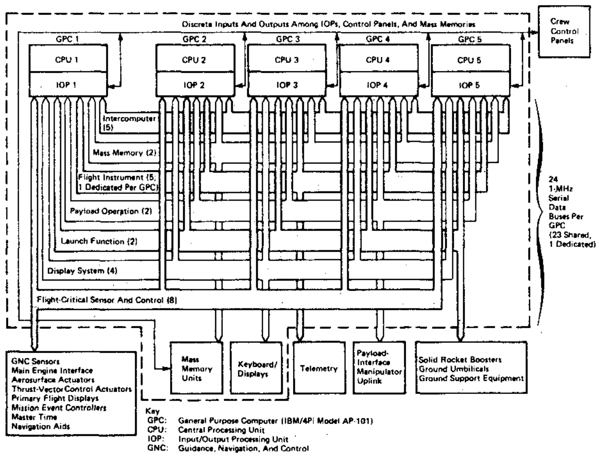

Discrete Inputs And Outputs Among IOPs, Control Panels, And Mass Memories
Crew Control Panels
GPC 1
CPU 1
IOP 1
GPC 2
CPU 2
IOP 2
GPC 3
CPU 3
IOP 3
GPC 4
CPU 4
IOP 4
GPC 5
CPU 5
IOP 5
Intercomputer (5)
Mass Memory (2)
Flight Instrument (5, 1 Dedicated Per GPC)
Payload Operation (2)
Launch Function (2)
Display System (4)
Flight-Critical Sensor And Control (8)
24 1-MHz Serial Data Buses Per GPC (23 Shared, 1 Dedicated)
GNC Sensors
Main Engine Interface
Aerosurface Actuators
Thrust-Vector Control Actuators
Primary Flight Displays
Mission Event Controllers
Master Time
Navigation Aids
Mass Memory Units
Keyboard/ Displays
Telemetry
Payload-Interface Manipulator Uplink
Solid Rocket Boosters
Ground Umbilicals
Ground Support Equipment
Key
GPC: General Purpose Computer (IBM/4Pi Model AP-101)
CPU: Central Processing Unit
IOP: Input/Output Processing Unit
GNC: Guidance, Navigation, And Control

Fig. 3 Space Shuttle data processing system.

during critical portions of the flight. Each of the eight OPS or memory configurations consist of two basic parts: the system software, which is resident and identical in all eight OPS, and the applications software (see Fig. 2). For transitions between OPS, the system software remains unaltered, while the applications software, depending on the transition requested, may be partially or totally overlaid.

System Software

Major elements of the system software include the management and control of the flight computer's internal resources and external interfaces. This function is performed by the Flight Computer Operating System (FCOS). A second element, system control (SC), provides software initialization and software/hardware reconfiguration (memory overlays, data bus switching, etc.). A third function, user interface (UI), handles all communications between the software and its various ground and onboard users.

FCOS

The Flight Computer Operating System uses a multitasking/priority queue structure. It schedules and allocates central processing unit (CPU) resources in response to requests from applications software or external subsystems. These resource requests are made through standard supervisor calls (SVCs) embedded in the applications code, timer-initiated interrupts for cyclically required functions, or externally driven interrupts signaling incoming data from other subsystems. In servicing these various interrupts and SVCs, the process management function assures that the highest priority process with work to perform is given control of the CPU.

The FCOS function also manages all input and output, as well as the interface between the two physical units that comprise each computer; that is, the CPU and the input/output processor or IOP. Although considered a part of the general purpose computer, the IOP is actually an array of 24 separate processors each controlling the I/O traffic on one of the 24 data buses connecting each computer to various ground and onboard systems (see Fig. 3).

Probably the most unique function of the FCOS software is the synchronization of redundant computers during critical flight phases.[1] As mentioned, this synchronization occurs over 300 times per second. It is accomplished by "stop-points" embedded in both the system software and the applications. Upon reaching one of these "stop-points," a given computer will cease normal execution of programs and attempt to determine if all other computers in the set have reached the identical point. To make their determination, each computer receives synchronization information via dedicated discrete lines from all other computers. A given member of the set will wait only 4 ms for all other members to reach the "stop-point." Computers not arriving within the time limit are considered failed and are removed from the redundant set. Removal involves termination of all intercomputer communication with the offending member along with annunciation of the "failure-to-sync" to both crew and ground controllers. Actual power-down of a faulty computer is a manual function performed only by the crew.

System Control (SC)

Computer initialization, reconfiguration and control of the data processing system hardware and data bus network is performed by the SC software. Initial program loads (IPLs), program overlays from the mass memory unit, and data bus switching are examples of the services provided.

User Interface (UI)

Support of the four crew (CRT) display units is provided by the UI software. These units and associated input keyboards comprise the primary interface between the crew and various software functions and hardware subsystems. In all, more than 60 different display formats are available on request throughout the eight operational programs. In addition to the CRT displays, the UI software also controls vehicle-to-ground interfaces. These are: the launch data bus (LDB), a direct channel between the onboard computer and the launch processing system at the Kennedy Space Center, and the network signal processor (NSP), an onboard device connected to the computer via the bus network, which receives commands and data from the Mission Control Center at the Johnson Space Center in Houston.

Applications Software

The Shuttle onboard software has been developed to support three primary preflight and in-flight applications. The first, guidance, navigation, and flight control, determines vehicle position, velocity, and attitude, performs subsystem redundancy management, and provides the crew with the displays and data entries necessary to control the avionics system. This software also issues all engine and aerodynamic surface commands from lift-off through rollout. GN&C software is resident in five of the eight OPS programs of Fig. 1.

The second major application, systems management (SM), monitors the performance and configuration of orbiter subsystems. It provides fault detection and annunciation of anomalies to both crew and ground. Other functions planned for later flights include payload deployment and retrieval operations. For early flights the SM software will be resident in two of the eight OPS of Fig. 1. In later flights, this software will be confined to on-orbit operations.

The final application, vehicle checkout (VCO), provides support for the testing, integration, and certification of Shuttle subsystems during vehicle preparation, prelaunch countdown, and during the orbit coast period just prior to entry. VCO software is contained in three of the OPS programs and is co-resident with both GN&C and SM applications.

STS-1 Software Development and Verification

Goals and Objectives

The organization responsible for the development and test of the Shuttle software evolved from previous space software development programs including Apollo and Skylab. The most beneficial experience, however, was gained during the approach and landing test (ALT) program. Although this system was considerably smaller and the flight environment benign in comparison to the STS-1 mission, valuable software design, test, and managerial experience directly applicable to the STS-1 activity was gained.

The primary objectives of the STS-1 software organization can be summarized in three key points. The first and most obvious is to:

1) Develop software which *adheres to the letter of the customer's requirements.* This refers to formal requirements documentation.

2) Assure that the software performs in accordance with the *customer's operational expectations* for both nominal and off-nominal conditions. This refers to operational requirements explicitly or implicitly identified in crew procedures, operational mission profiles, etc.

3) To provide software which is *"error-free."*

The first two goals are easily quantifiable since specific test scenarios can be developed to cover all documented software requirements and one-to-one mapping maintained and reported. The third goal of producing error-free software is a more nebulous objective when applied to complex systems and, as experience has shown, can only be asymptotically approached. How well a software development and test organization does in achieving this goal depends on how effectively it is structured to address the following areas:

1) Early identification and application of programming standards and techniques with subsequent checks on the fidelity with which these have been followed.

2) Establishment of a comprehensive test plan driven by customer requirements and operational usage.

3) Establishment of tests, audits, and code inspections not specifically driven by customer requirements but necessary to prove the design error-free.

4) Early definition of requirements for the simulation test bed(s) and other support software accompanied by a thorough test plan and subsequent configuration control.

5) Configuration control of the incremental build and integration of the evolving software system.

6) Configuration control of the implementation and retest of the software changes resulting from requirements upgrades and discrepancy corrections.

Organizational Structure

The organization chosen to address each of these key items is divided into five functional areas: requirements analysis and system architecture, software development, system integration and build, independent verification, and customer support and field test. Each represents a line department although all are not equal in size or position within the organizational hierarchy.

Software Design, Development, and Integration

The first three functions found in most software organizations are responsible for the design, implementation, integration, and module-through-system-level test of the software. The requirements analysis group, a "front-end" organization oriented toward applications software, is made up of engineers and programming specialists familiar with avionics systems. Their participation started with involvement in the early formulation of the software requirements by NASA and the prime contractor, Rockwell International. Their role was to assess the feasibility of implementing the requirements into the data processing hardware and software. A second objective was to gain an in-depth understanding of the chosen avionics design to guide their subsequent system-level software design activities. This understanding of both the avionics design and the software structure also proved invaluable to requirements analysts in their later role as advisors to the programmers implementing the detailed software design.

The system architecture organization performs a role similar to the requirements analysis group with their efforts primarily directed toward the operating system software. In addition, their responsibilities included the sizing and measurement of the evolving software package (CPU utilization, memory size, and timing measurements).

Shuttle software development is divided into two major organizations—one responsible for the operating system and associated user interface software, and the second responsible for the "applications" software (e.g., navigation software, flight control software, etc.). These organizations, assisted by the requirements analysts and system architecture personnel, design and code the software at the module or subroutine level. They also perform initial integration by coupling individual modules with an executive structure to produce overall subsystems such as the guidance, navigation, and flight control software. Testing is performed at each level of development from the module level using driver programs, through the initial integration level.

The final integration of applications programs with the operating system is performed by the system integration group. It is the responsibility of this group to collect new or updated software modules from the development organizations, compile or assemble them, and use the resulting object code to update a master software system library. This master system is then link-edited and passed through a mass memory build program which converts it to a form that can be used to load a Shuttle onboard mass memory tape unit. The combined integration and development organizations perform additional system level tests after the final systems have been built. These include nominal flight simulations in a multicomputer environment as well as hardware/software interface tests.

As a part of their development and integration responsibilities, these three organizations define, apply, and enforce programming standards. Both during the design phase and after implementation into code, formal inspections assure that the software adheres to all baselined programming standards. The development and integration groups also maintain configuration control during all levels of design, coding, integration, and testing. This control pertains not only to the software package but also associated documentation such as requirements baselines, design descriptions, discrepancy tracking, and integration and system build documentation.

Independent Verification

Independent verification is a separate line organization with the ultimate responsibility of assuring that the software is error-free. The verification group maintains an equal status with the software development organization, and derives its authority by remaining completely independent with respect to management and personnel.[2,3]

The philosophy of the verification group is based on the premise that the software is untested when received. This is a key factor in the development of their test plan. Because their role is essentially that of a pseudocustomer, they maintain an adversary relationship with the development groups. Verification receives the software at the end of the design, development, and integration cycle and subsequently subjects it to a complete detailed and system level validation program.

Verification testing is divided into two major categories: detail/functional testing and performance or system-level testing. The detail/functional tests address the question of whether the software meets the letter of the requirements and is basically divided by functional area (e.g., ascent guidance programs, entry navigation, etc.). Performance or system testing is directed at how the system as a whole will perform in both nominal and off-nominal or stress situations.[4] As a result, the performance verification is divided by mission phase; that is, ascent/aborts, on-orbit, entry. The majority of verification testing is performed in the Software Development Laboratory (SDL) using actual Shuttle data processing system hardware. A small amount of verification testing is performed in other facilities, such as the Shuttle Mission Simulator (SMS) and Shuttle Avionics Integration Laboratory (SAIL), in order to take advantage of their extensive flight hardware and crew interfaces. Descriptions of these test facilities are provided later. A prime objective in both types of verification is to perform all testing on the total unaltered (or "unscarred") software system rather than a modified or partial system.

Execution of test cases is only one aspect of the responsibilities of the verification analyst. Associated with each test case is a standardized checklist that must be filled out before the test case is considered completed. One item on this checklist is the code inspection of all software modules covered by the case. This "mental walk" through the software has proven to be especially fruitful in uncovering particularly subtle discrepancies.

Verification's organizational independence and its assumption that the software is untested when received have both been emphasized earlier. In addition, two other attitudes play a key role in the verification group's approach to testing. Since verifiers are considered systems analysts and not just "testers," they are responsible for attempting to diagnose problems and propose solutions when possible. This expedites software development's analysis and correction of alleged software discrepancies. This responsibility, however, is tempered by the overriding directive that "...when in doubt—assume the software to be at fault." This is the key to providing error-free software. A second attitude addresses the requirements and can be simply stated as "...the requirements are not to be considered infallible." These attitudes and assumptions cast verification in the role of the conscience of the project and foster a definite but healthy adversary relationship between the verification group and the software development organizations.

Customer Support and Field Test

The customer support and field test activity occurs in parallel with verification. This organization provides a number of services to software users at customer test sites (simulators) and on the actual Shuttle vehicle. These include installation and checkout of the newly released software; problem analysis, trouble-shooting, and implementation of temporary solutions; customer education and customer/software organization liaison; test site unique software alterations; customer test program support; crew training program support; and mission support.

Software Development and Test Life Cycle

Figure 4 describes the typical software development life cycle and depicts the interaction of the five functional areas listed above. For STS-1, the cycle started in 1974.

Due to the size, complexity, and evolutionary nature of both the requirements and design of the software, it was recognized early that the ideal development cycle could not be strictly applied and still satisfy overall program objectives and customer needs. Due to schedule constraints, the ultimate users of the software could not wait for the complete package to be developed, verified, and delivered. Since the software was such an integral part of the overall system, checkout of customer test simulators and crew trainers, and more important, the buildup of the actual Shuttle orbiter could not begin without it.

A system release approach was devised for STS-1, which met the objectives by applying the ideal cycle to small elements of the overall software package on an iterative basis. This approach was based on incremental releases. The releases were first separated into flight phases or memory configurations; that is, entry, ascent, and vehicle checkout. The

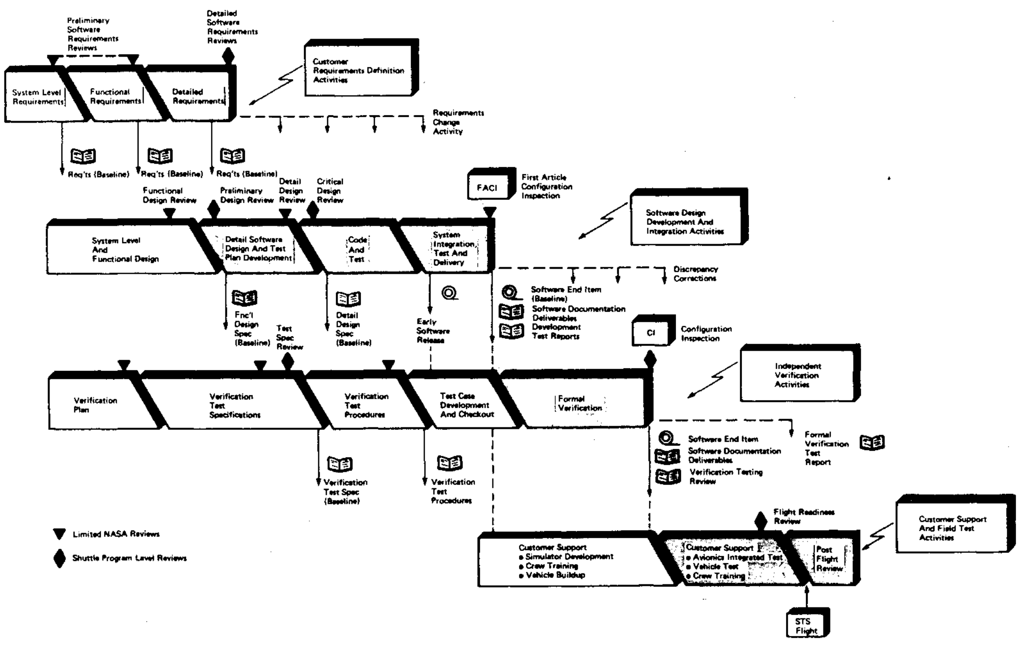
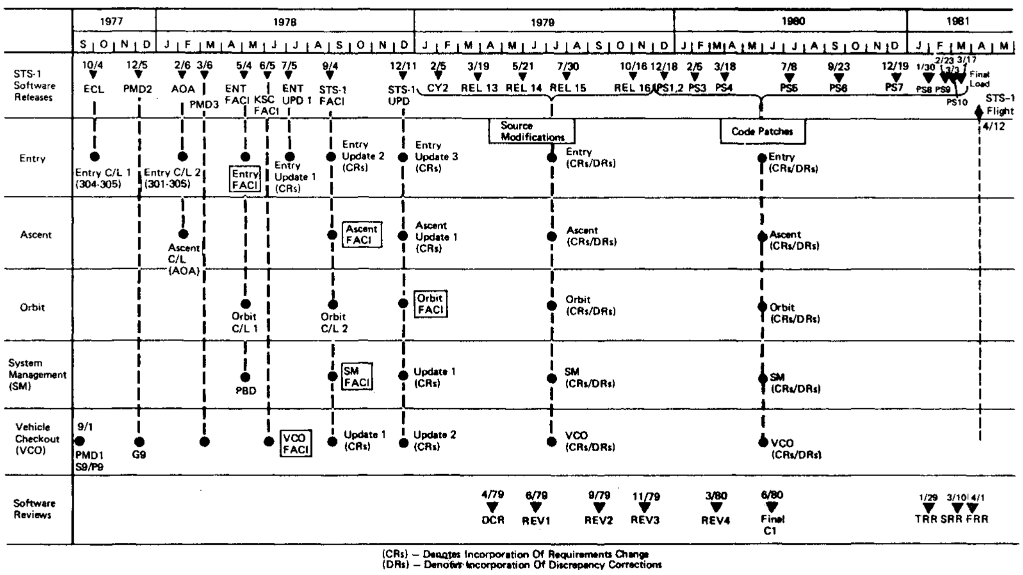

Preliminary Software Requirements Reviews
Detailed Software Requirements Reviews
System Level Requirements
Functional Requirements
Detailed Requirements
Customer Requirements Definition Activities
Requirements Change Activity
Functional Design Review
Preliminary Design Review
Detail Design Review
Critical Design Review
FACI
First Article Configuration Inspection
Software Design Development And Integration Activities
System Level And Functional Design
Detail Software Design And Test Plan Development
Code And Test
System Integration, Test And Delivery
Discrepancy Corrections
Software End Item (Baseline)
Software Documentation Deliverables
Development Test Reports
CI
Configuration Inspection
Independent Verification Activities
Verification Plan
Verification Test Specifications
Verification Test Procedures
Test Case Development And Checkout
Formal Verification
Software End Item
Software Documentation Deliverables
Verification Testing Review
Formal Verification Test Report
Flight Readiness Review
Limited NASA Reviews
Shuttle Program Level Reviews
Customer Support
• Simulator Development
• Crew Training
• Vehicle Buildup
Customer Support
• Avionics Integrated Test
• Vehicle Test
• Crew Training
Post Flight Review
Customer Support And Field Test Activities
STS Flight

Fig. 4 Software development life cycle.



(CRs) – Denotes Incorporation Of Requirements Change
(DRs) – Denotes Incorporation Of Discrepancy Corrections

Fig. 5 Flight software release schedule.

first drop for each release represented a basic set of operational software and provided a structure for adding other capabilities on later deliveries. The development of the full set of baseline capabilities for each release culminated at a first-article configuration inspection (FACI) point which marked the beginning of the independent verification effort for that release.

The advantages to the customer are obvious, but this approach also provided certain advantages to the software organization. It fostered a gradual development and test approach, exposing the software in small increments to both verification and field users. This allowed early identification of software discrepancies and eased problem isolation so often difficult in large fully integrated systems. The incremental approach also exposed the software and data processing system to actual flight hardware at a much earlier point in the program.

The STS-1 development program had 24 interim releases of the software in the period October 1977 to the STS-1 flight in April 1981 (see Fig. 5). Although full software capability to support the first flight was provided after the ninth release in December 1978, additional releases of the software were necessary to accommodate the continued requirements changes and discrepancy correction activity inherent in large, complex, first-of-a-kind software systems.

An Integrated Test Approach

The key to the success of the incremental release technique was a method for project-wide control of all testing, an integrated test approach. The verification group was not the only organization involved in testing. Both the development and integration groups tested the software during the incremental buildup of capabilities. This testing proceeded in parallel with verification of earlier releases. In order to insure total coverage and consistency across the project, test execution and documentation standards were implemented. These included review and approval procedures involving both IBM and NASA.

A key element of this integrated test plan was development of a management approach which emphasized a hierarchical

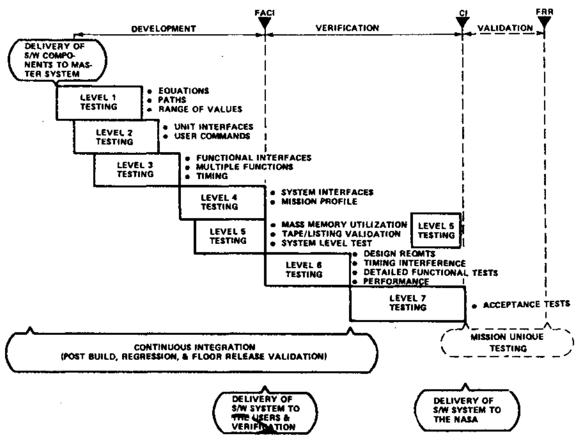


Fig. 6 Levels of testing.

ordering of development tests that allowed for continual integration of program parts as they were developed, and a systematic sequence of evaluation tests on the parts as well as the flight software system as a whole (see Fig. 6).

During the development period, compilation units were added to the master system via the system build process which was invoked periodically. Each master system update was tested to determine the preservation of the software's basic capabilities. Also, more detailed level tests were used to determine the quality of newly added capabilities. The former testing was termed "regression testing," the latter "new capabilities testing." All specific test plans were documented in an integrated test plan which covered all phases of the testing process. Seven unique phases or levels of testing, described in the following paragraphs, were performed on the various interim software releases using the Software Development Laboratory (SDL).

Level 1 Testing (Unit)

During the development activity, specific testing was done to insure that the mathematical equations and logic paths provided the results expected. These algorithms and logic paths were checked for accuracy and, where possible, compared against results from external sources and against the system design specification (SDS).

Level 2 Testing (Functional)

The level 2 facet of the development testing was similar to level 1. However, level 1 testing described above was expanded to modules that interfaced with each other in the total functional environment and were required to satisfy a specific user input command. It combined modules which, by design, operated in conjunction with each other and tested them as a function against the SDS and the requirements.

Level 3 Testing (Subsystem)

Level 3 testing demonstrated the ability of a subsystem to execute nominally in a simplex flight computer environment (e.g., fly an ascent trajectory, or perform self-test of the vehicle hardware). These tests were the first real indicators of the software performance as an integrated system. All facets of the applications programs, from the integrity of the algorithms to its interface with the system software, were exercised. Completion of level 3 tests was one of the key milestones in the path to releasing a system for verification and field usage.

Level 4 Testing (System)

Level 4 testing, performed by the software integration group, exercised control logic interfaces, operational program (OPS) transitions, and display processing in a multiple-flight computer environment. This was the phase during which the first onboard mass memory image containing all eight operational flight programs (see Fig. 1) was integrated and tested.

Level 5 Testing (Release Validation)

Prior to delivering the software to field users, the level 4 tested end item was loaded into an actual Shuttle hardware mass memory and tested in one of the NASA simulation/training facilities. This was to verify that the delivered software would function in the most realistic hardware environment available.

Levels 6 and 7 Testing (Independent Verification)

Levels 6 and 7 refer to testing performed by the independent verification group. Level 6 tests encompass all of the testing performed by verification, and level 7 is that subset, defined by mutual IBM/NASA agreement, as the software acceptance tests to be presented to the general Shuttle community at configuration inspection (CI). Both test plans and results were subject to NASA review and approval. As previously mentioned, verification constitutes a complete retest of all software capabilities at both the detailed and system levels. No assumptions were made about the quality or correctness of the software based on testing performed in levels 1-5.

Description of Test Facilities

Shuttle program test requirements necessitate use of three facilities for verification of the Shuttle avionics system: the Software Development Laboratory (SDL), the Shuttle Avionics Integration Laboratory (SAIL), and the Flight Systems Laboratory (FSL). A fourth facility, the Shuttle Mission Simulator (SMS), although not specifically involved in testing the avionics system, does play a major role in exercising the integrated avionics design through its extensive crew training activities.

The Software Development Laboratory (SDL), located in Houston at the Johnson Space Center, is the primary facility for the development, integration, and verification of the primary onboard software system. It was developed and is maintained by IBM, the software contractor, under contract to NASA. Its capabilities will be discussed later in this section.

The Shuttle Avionics Integration Laboratory (SAIL), also located at the Johnson Space Center in Houston, is responsible for avionics system integration and hardware/software certification. It is primarily used by NASA and Rockwell International, the Shuttle integration contractor, to support elements of their avionics verification plan. The SAIL was designed as a simulation laboratory where avionics hardware (or simulations of the hardware), flight software, flight procedures, and ground systems are fully integrated and tested. Its integration and test role for STS-1 emphasized the prelaunch, ascent, aborts, and orbit insertion phases of the Shuttle mission.

The Flight Systems Laboratory (FSL), located at the Rockwell facility in Downey, California, has capabilities similar to those of the SAIL. Its role in the Shuttle program is also similar in that it is used by NASA and Rockwell to perform avionics system integration and verification for the on-orbit, de-orbit, entry, and landing phases of the mission. Both the FSL and SAIL are full six-degree-of-freedom flight simulators with man-in-the-loop capability.

The Shuttle Mission Simulator (SMS) is NASA's primary training facility for Shuttle crews. Located at the Johnson Space Center in Houston, it provides a realistic environment for training crews in all mission phases from prelaunch through landing and rollout. The SMS consists of two simulators, both employing actual Shuttle data processing system hardware. One, the "moving-base" simulator, provides cockpit motion consistent with vehicle dynamics; the second is a "fixed-base" simulator. Both provide extensive visual displays in addition to a direct link to the Mission

Control Center, also in Houston. Although not directly involved in software testing, the SMS does provide an invaluable benefit in this area. Its extensive program of crew training under off-nominal conditions tends to subject the software to a wide variety of stress situations. This provides added confidence in the performance and capabilities of the system.

The SDL Facility—The Software Test Bed

The SDL serves a dual role in the Shuttle test program. The Primary Avionics Software System is both developed and verified in the SDL. Because of this dual role, the laboratory has been designed with capabilities to analyze, verify, maintain, and control Shuttle flight software. Additionally, the SDL is used to integrate all software elements into an orbiter-compatible tape used to load the Shuttle mass memory tape unit.

The SDL provides six major functions with which to perform development, verification, and integration activities. These functions are a program management facility; a mass memory build facility; a simulator; a postprocessor; a documentation, analysis, and statistics system; and an array of preprocessors.

The Program Management Facility (PMF) provides the programs and control needed to build, maintain, and track program data and create deliverable systems. The mass memory build facility provides the utilities by which Shuttle software elements are mapped into the format required by the orbiter's mass memory units. The mass memory build also creates universal patch format (UPF) tapes used by all Shuttle test sites to update onboard mass memory units.

The preprocessors provided by the SDL convert data required for flight into forms that can be used by the flight software. These data include mission-dependent initialization data (I-Loads), display formats, systems-management measurement tolerances, and downlist telemetry data.

The documentation, analysis, and statistics system provides users with the data necessary to develop, analyze, and maintain the evolving software system.

The SDL simulator and postprocessor functions provide the primary facility used in the development and verification tests of the software system. It provides a realistic closed-loop, six-degree-of-freedom simulation of the Shuttle operational environment and avionics hardware. Available Shuttle hardware elements include multiple AP-101 general purpose flight computers along with their associated input/output processors, and the crew display electronics units (DEUs) and CRTs. All flight hardware is interfaced via a special-purpose flight equipment interface device (FEID). This device not only provides the hardware-to-hardware and hardware-to-host computer interface, but also the means by which the simulation is monitored and controlled.

The FEID allows the user to control simulation execution at the flight software instruction or data location level. Users have the capability to "stop" a multiflight computer simulation on time, event, or upon execution of a particular instruction or data location reference within the flight software. While the simulation is stopped, capabilities are available via direct memory access to alter or collect data from the flight computer main memory. Since the simulator is also stopped during this period, faults may also be introduced external to the flight computer in the simulated hardware or vehicle environment. Upon completion of the faulting or data collection activity, the simulation can be restarted without loss of continuity in instruction execution or I/O. This ability to stop on *any* user-specified software instruction and access or alter the contents of the flight computer without affecting the simulation is a unique feature of the SDL and is not available in the SMS, FSL, or SAIL simulators.

The major outputs of the simulator are log tapes, which contain the flight software commands issued and the data received or transmitted. The log tapes also contain diagnostic data such as data snaps, dumps, or traces requested by the user. The SDL postprocessor provides a data reduction and manipulation capability for the log tapes. Postprocessor capabilities allow users to specify the subset of logged data to be processed, its format, special computations that are required, and whether a comparison should be made with other simulations.

Shuttle Operational Environment

Software for the early STS missions has emphasized development of new capabilities. The software organization was one geared toward basic design development and verification of an untried system. Test facilities have been tailored to be flexible with generalized diagnostic features, provisions for extensive user intervention, and emphasis on the hardware/software interfaces. Early test philosophies have emphasized painstaking thoroughness to the point of being redundant. Design and code reviews, detailed testing, system level testing, followed by complete reverification by an independent organization have all been necessary to assure crew safety and mission success for the early flights. These initial tools and techniques, however, will not be cost effective or responsive enough to meet the demands posed by the Shuttle flight manifest in the operational era.

Shuttle Flight Manifest

The current flight manifest highlights two factors over the next five years—increased mission frequency and varied mission objectives and payloads. The typical Shuttle flight will last from 1 to 7 days in near-Earth orbit. Upon return the orbiters and boosters will be refurbished and refitted. For the orbiter, the procedure is projected to take less than 30 days. Recent estimates predict 12 flights in 1984, with the number per year increasing to as many as 30 as the Shuttle becomes fully operational.

The basic software architecture and applications programs to support the majority of defined missions will have been developed and verified with the completion of the tenth mission. Although the intent is to have the software remain nearly constant after that point, in reality it will continue to change throughout the operational era. Changes will be the result of three activities: 1) continued *development* of new or enhanced capabilities; 2) *maintenance* (i.e., correction of discrepancies); and 3) mission *reconfiguration*.

Although software requirements are still being refined in some applications areas, development of new or enhanced capabilities is expected to dramatically decrease as the operational era is approached. Likewise, as the software stabilizes and attains a certain level of "shelf-life," changes resulting from discrepancies in requirements, design, or implementation will also decrease. Even though development and maintenance activity is expected to be minimal in the operational era, significant software data alterations are planned from flight to flight. These data changes are termed "reconfiguration."

Flight-to-Flight Reconfiguration

Since it was not feasible to design a self-contained, unchanging software system to meet the needs of all projected Shuttle missions and payloads, certain areas of the software were generalized and made easily changeable with only minimal impact to the integrity of the overall system. If one divides the software into three basic categories: logic (code and algorithms), constant data (unchanging values such as the value of π), and mission reconfiguration data; the logic and constant data comprise that part of the software which is intended to remain unchanged in the operational era. Mission data, however, will be used to "reconfigure" the software for particular missions and payloads.

Origin of Mission Reconfiguration Data

Flight-to-flight reconfiguration data, which are parameter values rather than code or logic, arise from three sources: variations in mission profiles; changes in vehicles or vehicle configuration; and variations in payloads or payload carriers.

Reconfigurable mission profile data include such parameters as launch site characteristics, ascent and abort trajectories and targets, orbit profiles, entry targets, landing site characteristics, and, in general, all data required by the guidance, navigation, and flight control system which can vary from mission to mission.

Since nearly all information paths within the Shuttle are at some point handled by the data processing system, the flight software is designed to be reconfigured with data describing the precise configuration of the particular orbiter utilized. Such vehicle data include data bus network addressing information for various line-replaceable units (LRUs) and sensor calibration information (e.g., inertial measurement unit factory calibration data, accelerometer scalings and biases, etc.). Other vehicle-dependent data include mass properties and engine characteristics. Although the intent is to maintain commonality among all Shuttle vehicles, some variation will be a reality in the operational era and has been provided for within the definition of "reconfigurable" data.

Payloads and payload carriers can have a wide-ranging influence on reconfiguration of the software. Inactive or autonomous payloads may only affect such data as vehicle mass properties or remote manipulator arm deployment or retrieval sequences, whereas more complex payloads and carriers may require any or all of the following: specialized telemetry downlink data; unique fault detection and annunciation; and specialized payload/carrier crew display formats and data entry and control capabilities.

Software Implementation of Reconfiguration Data

Implementation of mission-reconfigurable data has stressed automation. Mission profile data and data specific to vehicle configuration are commonly referred to as I-Loads (initialization loads). I-Load requirements are provided to the software organization by NASA on magnetic tape. The tape is input to a Software Development Laboratory facility which applies the I-Load data directly to an image of the Shuttle's mass memory device, thus minimizing manipulation of the existing software system, i.e., recompilation and relink-edit. The automated process also includes extensive verification and documentation features.

Update of other reconfigurable data, such as formats for telemetry downlink, parameter limits for fault detection and annunciation, and specialized crew displays for payload monitoring and control, have also been automated. Preprocessors are used to apply data changes supplied by NASA on requirements tapes. Unlike the I-Load procedure, update of these data do in some cases involve the recompile of certain areas of the flight software. These areas, however, tend to be well defined and of fixed length, thereby minimizing change to the overall software structure.

Upgrade of Facilities and Test Techniques

The SDL, although an extremely versatile development tool, cannot, as presently configured, support software reconfiguration at high flight frequencies. The need for upgrading the SDL to a Software Production Facility (SPF) for the operational era was recognized early in the STS program. Enhancement of both hardware and software has already begun. The hardware upgrade primarily involves more powerful host computers, while software upgrades consist of the conversion of existing support software and development of entirely new facilities to complement the altered flight software modification and test philosophy dictated by the operational era.

Since enhanced computer resources cannot be expected to handle alone the vastly increased workload posed by an operational Shuttle fleet, new support facilities will stress automation throughout the reconfiguration process. Where found to be cost-effective, automation will replace current manual procedures, from the preprocessing of incoming data, through its incorporation into a deliverable software system.

Automation is also a major objective in development of a test philosophy for the operational era. This objective is tempered by the realization that, although reconfiguration only involves "data," these data are part of a complex interactive real-time software system, and is therefore capable of altering overall Shuttle performance.

In that light, studies are currently in progress which address verification procedures and management for the operational era, Shuttle system performance sensitivities to reconfiguration data, and test selection criteria.

Another concept being proposed as a means of increasing the responsiveness of the software organization to reconfiguration is data standardization. This entails placing limits on the uniqueness of individual missions. An example involves imposing constraints on mission planners, such that ascent trajectories could only be selected from a standard set established during an initial series of flights. However, standardizing and cataloging or grouping sets of reconfiguration data is an approach that requires the review and support of the entire STS project. Since extensive resources are expended in mission planning, benefits from such an approach could be realized in almost all aspects of the Shuttle program.

Conclusions

Early development of the Shuttle primary software emphasized several key factors common to the success of any complex interactive software system. These include: early involvement in the definition of requirements; standardized programming techniques; an incremental test approach which includes both detailed and system level independent verification; adequate simulation facilities utilizing operational hardware; and last and most important, strict configuration control during all phases of development and testing. However, operational use of the Shuttle software dictates more cost-effective methods for software data modification. Current plans emphasize automated techniques to reconfigure the software for the varying missions and payloads of the operational era.

References

[1]Sheridan, C.T., "Space Shuttle Software," *Datamation Magazine,* July 1978.

[2]Clemons, J.F., "Verification of the Onboard Flight Software Developed for the NASA Space Shuttle Program," IEEE Eighth Texas Conference on Computing Systems, Dallas, Texas, Nov. 1979, pp. 6B-1 to 6B-5.

[3]Macina, A.J., "Independent Verification and Validation Testing of the Space Shuttle Primary Flight Software System," NSIA/AIA/USAF-SD/NASA Mission Assurance Conference, Los Angeles, Calif., April 1980, pp. 518-535.

[4]Valrand, C.B., Treybig, J.H., Behnke, K.L., and Culley, K.B., "Performance Verification of the Space Shuttle Onboard Software," 4th Digital Avionics Systems Conference, St. Louis, Mo., Nov. 1981, pp. 446-458.