VALIDATION OF NSSC-I SOFTWARE FOR THE HUBBLE SPACE TELESCOPE

Glenn Foley and Jan Owings

NASA, Goddard Space Flight Center, Greenbelt, Maryland

Abstract

The Hubble Space Telescope, to be launched in March, 1990, contains two primary onboard computers. The spacecraft support systems computer, the DF-224, performs attitude determination and control and manages spacecraft structures and mechanical systems. The payload computer, the NSSC-I (NASA Standard Spacecraft Computer, Model 1), is dedicated to the command and data handling of the five scientific instruments.

This paper describes the simulation and test methods used to ensure that the NSSC-I flight software properly carries out its requirements for control of scientific observations and for monitoring the health and safety of the payload. The hardware and software test environment is discussed. The different kinds of tests (unit, special real time, and stress tests) that are performed before the software is incorporated into the flight system are described, and the kinds of errors that each test category is best suited to find are listed. Finally, the limitations of the tests are discussed, and current plans for enhancing the test environment are outlined.

Background

When the astronauts of STS-31 launch the Hubble Space Telescope (HST) from the orbiter Atlantis, years of work by NASA and its contractors will be brought to fruition, and the astronomers of the world will see further into space and time than ever before. The first of NASA's great observatories to be launched, the HST consists of a 2.4-meter aperture Ritchey-Chretien telescope with five scientific instruments. Two cameras, two spectrographs, and a photometer will be used to observe both stars and extended objects with a precision never before achieved.

The HST program is a cooperative effort by NASA, the European Space Agency, and the Association of Universities for Research in Astronomy. This paper only addresses the roles of a few of these contributors. The Marshall Space Flight Center (MSFC), which is the lead NASA center for the mission, has directed the development and integration of the spacecraft. Two principal contractors, the Perkin-Elmer Corporation and the Lockheed Missiles and Space Company report to MSFC. The Goddard Space Flight Center (GSFC) is responsible for managing the development of the scientific instruments and for operation of the HST after launch. The view of the HST mission presented here will focus on a part of the GSFC responsibilities, and will discuss other elements of the project only as necessary for an understanding of the onboard software that controls the payload.

The HST has three major components: the optical telescope assembly (OTA), built by Perkin-Elmer; the support systems module (SSM) from Lockheed; and the payload, managed by the GSFC with sensors developed by several universities and aerospace firms, and the payload control system provided by the International Business Machines Corporation. Figure 1 shows the relationship among these subsystems.

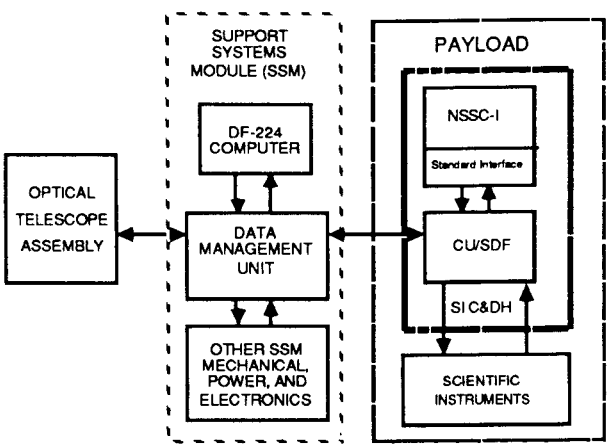

Figure 1. HST Major Subsystems

In addition to the primary and secondary mirrors, the OTA includes thermal and optics control electronics and the fine guidance sensors (FGS). The FGS allow the telescope's pointing control system to lock onto a

target with a pointing accuracy of about 100th of an arc second and with a stability of about seven 1000ths of an arc second. There are three FGS, but since only two at a time are necessary for guiding the telescope, the third will serve as a sixth scientific instrument for astrometry.

The SSM contains the data management system, including the central computer for the spacecraft, the DF-224. The structures and mechanical systems, pointing control, power, and thermal systems are within the SSM as well. The SSM data management system interfaces to the payload's Scientific Instrument Control and Data Handling (SI C&DH) module. In addition, the payload includes the five scientific instruments (SI). Within the SI C&DH, the Control Unit/Science Data Formatter (CU/SDF) and the NASA Standard Spacecraft Computer, Model I (NSSC-I) manage commands and telemetry within the payload.

NSSC-I Functions for the HST Mission

The NSSC-I computer architecture, based on a 1974 design described in "Development and Application of NASA's First Standard Spacecraft Computer," [2] includes a fixed-point arithmetic central processing module (CPM) and 64K words of random access memory. HST real time flight software residing in the NSSC-I includes two basic components, the flight executive (Exec) and SI application software. All of the NSSC-I flight software is written in NSSC-I assembly language. A preprocessor written specifically for the HST mission allows specification of some higher level selection and repetition logic in the code, thus automating syntax for frequently used constructs.

The Exec software is described in detail in the *Multimission Modular Spacecraft Onboard Computer Flight Executive Technical Description.*[1] It was developed for the Solar Maximum Mission and Landsat programs and modified for use on HST. A major responsibility of the Exec is the processing of commands to the HST payload. Commands may be initiated from spacecraft operators on the ground, stored command sequences loaded into NSSC-I memory, or other software residing in the NSSC-I. Commands may be directed to HST's science instruments ("turn on high voltage" or "open shutter," for example), to the CU/SDF (such as "enable communications interface"), or they may be software commands to the NSSC-I Exec itself ("dump an SI data log").

The Exec is also responsible for scheduling the execution of the other software processors in the NSSC-I. Software may be activated periodically, asynchronously by request, or asynchronously by event (such as arrival of science data from an SI).

The NSSC-I communicates with HST's other main flight computer, the DF-224, via processor interface tables (PITs) exchanged every half second. Each computer's PIT includes a toggling "I'm OK" bit used to monitor the health of the other. The Exec also uses the PIT to request science data tape recorder usage and spacecraft pointing offsets on behalf of SI application software.

Exec software reports science and engineering data from the SI C&DH to the SSM for transmission to the ground or staging in onboard tape recorders. Every half second, the Exec samples SI remote interface units or areas of NSSC-I memory for values to be reported, and it calculates some critical values itself for inclusion in the engineering data stream. Telemetry values collected can be checked against selectable high and low limits, and if the limits are violated, the Exec can request other software actions (such as commanding an SI to a "safe" state when a hazardous condition is detected). The Exec can also initiate transmission of processed SI science data, NSSC-I memory dumps, or ancillary science data.

Finally, the Exec periodically performs diagnostic self-tests to ensure the integrity of the NSSC-I and its software. These tests include a memory checksum calculation on code to assure software has not been overwritten or corrupted, software processor "infinite loop" detection, an NSSC-I instruction set test, and the ability to detect overloading of the CPM.

The NSSC-I SI application software is tailored to the requirements of each instrument. Generally speaking, this software performs functions such as configuring the SI mechanisms and optics for use, acquisition of astronomical targets, observations with the instrument, health and safety monitoring (power consumption, temperature, bright object protection, etc.), and processing and quality checking science data from the instrument.

Modification of the Exec for use on HST and implementation of the SI application software was done by IBM. In January 1986, responsibility for maintenance of all the NSSC-I software transitioned to the Flight Software Systems Branch (FSSB) of the Goddard Space Flight Center. The FSSB role includes enhancement of the flight software, correcting errors,

and implementing changes to meet the evolving requirements of the spacecraft. Modifications to the software are grouped into baselined releases, and it is the process of validation of each of these new releases to which the rest of this paper is dedicated.

Life Cycle

Requirements Analysis

A request for a modification to the NSSC-I flight software begins by submission of a program trouble report (PTR) form to the Configuration Management Office (CMO) of the HST Project at the Goddard Space Flight Center (HSTP-G). The PTR documents the problem or reason for enhancement and assesses the criticality of the change. Before the development of each release begins, the flight software manager of the HSTP-G directs the FSSB to assemble a list of prospective flight software changes. The FSSB responds with a document which summarizes all active PTRs and includes an analysis of the need for each change, estimates of the resources which would be required to implement and validate each change (computer and manpower), an assessment of impacts each change would have to other ground and flight subsystems in the project, and a recommendation for the inclusion or deferral of each proposed change. Creation of this PTR summary document involves interaction with the initiator of the PTR, the SI development teams, and the spacecraft user/operations community, as well as experimentation with possible software design modifications. A major focus during this phase is to assure well-defined and agreed upon requirements, including coordination with all affected flight and ground elements. After this analysis is provided to the HSTP-G, a configuration control board (CCB) meeting is held. It is at this time that the HSTP-G reviews the proposed modifications and directs the FSSB whether to include or defer each change in the next software release. This decision is based upon the PTR summary status sheets and input from project managers, spacecraft engineers, instrument development teams, spacecraft operators, science planners, and the science user community. The new baseline is composed of the previous release and those changes which are approved at the CCB meeting.

The approach of the FSSB HST NSSC-I team is to assure product quality through many activities performed at each intermediate phase in the software development process. The events involved in each phase will now be described to illustrate the methods employed for detection of problems as early as possible in the life cycle.

Design

Because it is necessary to understand the motivation for and implication of each change in order to create the PTR summary document, requirements are generally well understood and documented by the time the CCB meets. Thus, the next action is to determine the final design or design modification for each change. This involves further iteration between the NSSC-I software team, the PTR originator, and interested project managers and engineers. When an agreement is reached, a final design review is held in which the designer of the modification leads a line-by-line walkthrough of the FORTRAN-like program design language (PDL) for the affected software. PDL is the main tool used to communicate and preserve the NSSC-I flight software design. It is maintained for the Exec code and each SI's application software and is included as part of every delivery. This design tool formats the logic expressed, validates the syntax of control structures present, and provides cross references for data items and logic segments. The forum that the walkthrough provides allows all persons involved in the change a final opportunity to scrutinize the approach taken. Re-examination of the new design in this arena with the emphasis on "the big picture" often uncovers details overlooked in previous design iterations.

Implementation

The developer modifies the appropriate NSSC-I source modules and generates a unit test plan for verifying the logical correctness of the affected code. The developer makes a copy of the existing source code (stored in controlled libraries for each baseline) in his or her user account and performs all modifications to this version. The implementation phase culminates at a code inspection in which the implementor's peers review and comment on the work being done. The inspection aids in detection of errors by introducing the fresh perspective of others on the development team. Inspections are not only the easiest way to detect simple logic and syntax errors, but they also act as a way to enforce coding standards and practices, ensuring consistency with existing code and software efficiency. The developer must distribute a unit inspection package to the team (and any invited special interest guests) at least two days prior to a scheduled

inspection. The package contains a summary of the change, new source code, differences between the existing and modified source, new PDL, differences between the old and new PDL, and a unit test plan. The developer must also present a listing of the assembler output for any modified code. Assembling the affected modules provides a syntax check as well as providing memory requirements for the change.

The formal inspection requires a reader, moderator, and the author (implementor). Remaining members of the team participate as inspectors. The reader walks the inspection team through the change and provides background where necessary. The moderator records statistics of preparation time spent by participants and also records comments given by the team during the inspection. The author is present to answer specific questions only -- the idea being that the inspection materials should allow the audience to comprehend the content without intervention. After the inspection, the implementor has one week to disposition each item recorded by the moderator. The implementor is then ready to perform unit testing.

The purpose of unit testing is to exercise every path in the software which has been created or modified as part of a change. This implies that multiple test cases are required to exercise all possibilities in selective and repetitive branching, as well as exercising boundary and mid-range values on calculations. Unit testing may be performed with either an NSSC-I software code simulator development tool or on an actual engineering model NSSC-I with special driver software as the developer prefers. The unit test plan specifies required preset memory and register values, range of logic to execute (with intermediate results needing verification), and post-test memory and register values to validate. Unit testing is best suited for detecting simple logic errors, with the NSSC-I enforcing greater discrimination than is afforded at code inspections. Failures at the unit test level may require a code reinspection if deemed severe enough by the baseline development task leader. After each module has been successfully unit tested, it is promoted to a configuration controlled "test" account where it is available to other developers for integration with the rest of the flight software.

Integration Testing

All of the NSSC-I code -- the Exec and the SI application software -- form one bootable load module. Thus, integration test development can be more challenging than in the conventional software environment where one can assume the operating system will support the development process. As usual, however, the purpose of the integration test period is to demonstrate that the requirements levied by the approved changes have been met without perturbing the existing system. Integration tests may be special tests composed solely for validating the subject change, or they may be existing test procedures modified to exercise and stress the area of interest. Formal acceptance testing (AT), performed prior to delivery, consists of successful execution of the integration tests together with baseline functional and system regression tests. When a change is significant enough to impact a large portion of NSSC-I activities, a system regression test will be updated to incorporate use of the modified code. This assures that the new capability will be validated with every future release.

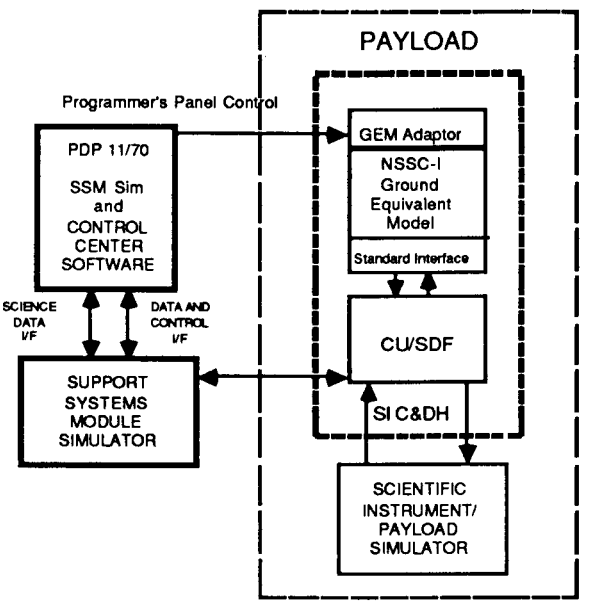

Figure 2. STIF

The environment used for system level testing is the Software Test and Integration Facility (STIF), shown in Figure 2, which consists of a general purpose computer and special spacecraft simulation hardware. The general purpose computer hosts a user interface similar to a spacecraft operations control center, science and engineering data collection and analysis software, simulation software for selected spacecraft functions, and driver software for communication with the special hardware. The special hardware includes engineering models of the NSSC-I and CU/SDF, a

science instrument/payload simulator, and a minimal simulation for the SSM subsystem of the spacecraft. System level tests are written as operational procedures executed at the STIF's command console. The STIF simulation hardware and software closely resemble the actual spacecraft in orbit from the perspective of the SI C&DH. Thus, the NSSC-I software can be exercised in a test environment faithful in timing and interface behavior to that present on HST.

Formal Acceptance Testing

The harsh and remote environment in which the NSSC-I flight software performs its critical real time role motivates the extensive effort expended during AT. First and foremost, one wants to avoid introducing potential safety hazards to the health of the payload of the $1.5 billion HST spacecraft. Many of the SIs have optics and detectors which can be damaged by exposure to bright light sources or extreme temperatures, thus reducing the success of the mission. Another need for added confidence in the software arises from the limited access available to a spacecraft system. Troubleshooting an anomaly in flight is subject to many more constraints than in a ground-based system. Because of the nature of the spacecraft, one must rely solely on engineering and memory dump data reported by the HST to ascertain the symptoms and cause of a problem. This is further complicated by the fact that communication contacts for receiving telemetry from and issuing commands to the spacecraft are limited to only a fraction of each orbit. This effectively reduces the visibility of the spacecraft.

Performance of formal AT takes about 60 hours for a typical release. Two people are required at all times to conduct AT (though quality assurance observers and other interested parties are frequently present during testing) -- the test operator (TO) and the test conductor (TC). Shifts of TCs and TOs are selected from the members of the development team; however, one may not be a TC for a test that he or she authored. It is the duty of the TO to make all entries at the STIF command console, and to configure all necessary switches and selectors on the special hardware. The TC directs the TO in actions to perform and assures that they are done properly. The TC is also responsible for performing all data validations required during and after a test. Any deviations from planned procedures or results are recorded by the TC on a deviation report form and must be resolved before formal delivery may occur. Another function of the TC is to assure that the proper version of the flight software load module is being tested. A load module is identified primarily by its checksum -- a value (unique to approximately 1 in 2^{18}) computed by logically exclusive or-ing together all NSSC-I memory words which represent machine language code (this is also the value used by the Exec in its memory checksum diagnostic self-test described earlier). This precaution assures that one does not inadvertently test a previously released baseline load module or an intermediate one generated during the development process.

As stated, tests run during formal AT are of three varieties. Special integration tests are written specifically to functionally validate changes being incorporated into the new baseline release. System regression tests are performed to assure that none of the existing requirements and capabilities of the NSSC-I flight software have been compromised by the new additions and to assure that prescribed CPM margins are not exceeded. Baseline functional tests also act as regression tests, but concentrate on stressing the NSSC-I's hardware interfaces. The baseline functionals also act to demonstrate the soundness of the special hardware and test system.

Though the types of tests run during AT emphasize different concerns, the composition and methods used by the tests are similar. Test procedures consist of real time sequences of directives to the ground system (which may in turn result in commands to the spacecraft). These are run in conjunction with stored command sequences loaded into NSSC-I memory. A typical test may simulate a science observation, target acquisition, or SI turn-on against the background operational activities of the NSSC-I. One test is dedicated to the detection of hazardous conditions with the various SIs, protective actions taken ("safing"), and recovery of the hardware and software to operational modes.

The procedures can validate some NSSC-I activities in real time, and may also specify logged data items to be verified by the TC after a test has executed. Commands generated by the flight software are captured in the STIF and are verified against predictions. Elements of science and engineering data logged during the execution of a test are also compared to predicted values as part of test validation.

In general, tests are designed to provide CPM loading that is heavier than the actual operational worst case. A special background processor installed in the NSSC-I flight software during system testing counts and records unused CPM cycles. These data are analyzed to

assure the CPM loading profile does not exceed prescribed margins.

Complete sets of system level tests, such as those used in AT, executed in an environment with a faithful spacecraft simulation help assure the robustness of the product software. This is essential in minimizing the possibility of invalid or harmful commands being issued to SIs, lost observing time, and lost science data.

Upon successful completion of formal AT, the new versions of all modules modified for the baseline release are promoted from the "test" account and replace the old versions in a configuration controlled "ship" account from which a delivery is prepared and the current version of the flight software is maintained. Though the magnitude of the NSSC-I software development is not small, the development environment is sufficiently intimate and contained to successfully allow computer account protection to be the sole method of configuration control -- no commercial configuration management tool is employed.

Delivery Procedures

The FSSB delivers several products to the HSTP-G including the baseline description document, the new executable flight software load module and symbol table, all current NSSC-I source code and intermediate assembler and loader products, PDL for all of the flight software, and copies of the unit and system tests used to validate the release. The HSTP-G has the responsibility of archiving the new software and distributing it to the spacecraft developers at Lockheed and the HST operations control center at GSFC.

Within 30 days of delivery, the FSSB must deliver an Engineering Test Report (ETR) to the HSTP-G. The ETR includes an inventory of all hardware and software used during AT, those changes included in the new baseline, the log maintained during AT by the TCs, and all deviation reports generated during AT with their resolutions. The ETR contains more detailed information than the baseline description presented at delivery, and serves as a permanent record of the details of formal AT.

Finally, the FSSB supports testing of the new baseline on the HST at the Lockheed spacecraft integration and test facility. Operating the new software on flight hardware allows the highest fidelity testing without HST actually being in orbit. In general, the instrument development teams and SI C&DH hardware teams compose real time tests which execute through the integration and test ground system at Lockheed to exercise new regions of code in their respective areas of responsibility. This is very similar to integration testing on the STIF with the added confidence of operating against the real SI and spacecraft hardware. The FSSB role is to aid in development of the spacecraft tests as well as assisting in data and anomaly analysis.

Enhancement of the Test Environment

While the flight software is always tested in a real time environment, some elements of the simulation are not entirely faithful. For example, the NSSC-I performs turn-on for SI power supplies and monitors the telemetry to verify that power and voltage constraints are not violated during the process. The flight software developer writes a test in which his or her code issues the proper turn-on commands in sequence. The hardware used in the simulation, however, can only output static telemetry values. This forces the developer to generate a synchronized update procedure for the payload simulator data store to modify the telemetry so it appears that the commands effected a hardware change. Otherwise, the flight software will "safe" the instrument being monitored, effectively ending the simulation.

The need to perform such "open loop" simulations as described above inevitably requires sacrificing fidelity of timing or data quality in a test. It is desirable to have an enhanced payload simulator able to update the simulated telemetry in response to commands it detects from the NSSC-I. Such is the purpose of the Monitor and Science Instrument Simulator (MASIS) currently being developed by the FSSB. MASIS will replace the payload simulator in the STIF configuration and will allow more faithful "closed loop" simulation of engineering and science data content and timing. MASIS will also have sophisticated "sky map" modelling capabilities to faithfully simulate SI acquisition of astronomical targets in a closed loop fashion.

Another disadvantage of the STIF test environment is its general purpose computer and control center user interface. The technology of the general purpose computer is more than 15 years old. The machine may be considered "temperamental" by the most lenient of standards. For example, temperatures which exceed a relatively moderate limit often cause the computer to halt. The simple fact that the computer is old makes it difficult to maintain system software, find

replacement hardware, and locate experienced maintenance personnel. Furthermore, the user interface employed on the STIF is a modified version of a 1975-vintage standard control center software system used for spacecraft operations at GSFC. Since the HST operations control center software was not yet available at the time the STIF was developed, using an existing product was an expedient approach to getting an NSSC-I flight software test environment on-line. However, in the ensuing years the actual HST control center software has matured and been validated for controlling the spacecraft. Flight software developers are thus required to be familiar with both test and operational ground systems, each a formidable entity. In addition, if one wanted to recreate a scenario for troubleshooting an anomaly that occurred in operations, any procedures or command loads used would need to be translated between the syntax of each system's user interface.

These problems are addressed in the design of the Extended Software Test and Integration Facility (ESTIF) under development by the FSSB. When complete, ESTIF will host a modified version of the actual HST real time ground system software on a newer general purpose computer. The HST ground system software will be adapted to include versions of the special hardware communications and science and engineering data analysis software rehosted from the STIF. ESTIF will allow NSSC-I flight software development and testing as well as anomaly troubleshooting in a more reliable, operations compatible environment to effectively support the 15 year mission of the HST.

Conclusion

Ultimately, test capability is limited by the fidelity of the simulation that can be run, the knowledge of the test team, and the investment that one is prepared to make in the test process.

The HST payload is complex. As we have seen, it contains several quite different hardware components which operate in a semi-autonomous manner under control of the NSSC-I. The test system itself reflects the intricacy of the spacecraft. The level of detail that must be assimilated is daunting to the novice. Thus, a programmer may erroneously assume that the test hardware has malfunctioned (as noted above, the equipment is unreliable) when in fact, a code error that is data or timing dependent has appeared.

The HST NSSC-I software has almost 70,000 lines of code produced at a cost of about 50 man years effort. Of these, 28,000 are non-comment lines. The executive software has been used for several NASA missions using the Multimission Modular Spacecraft; however, the MMS executive code was substantially modified for HST. The remainder of the code is new. From the first delivery that included SI support in May, 1983, through the present, 279 PTRs have been closed in twelve releases of the flight software. Some of these are enhancement requests, but the majority were written to document errors or new requirements. Since January, 1986, when the FSSB assumed responsibility for the NSSC-I software from IBM, 135 PTRs have been received by the Configuration Management Office. Of these, the CCB accepted 97 to be incorporated in the flight software. Figure 3 shows the number of errors and total changes in the last four software releases.

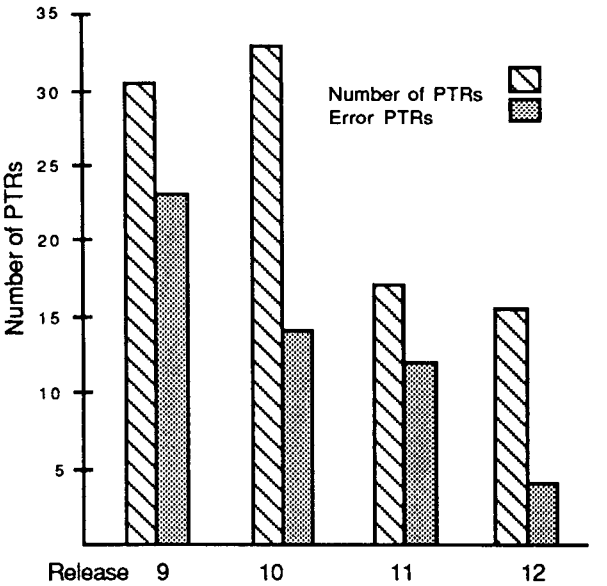

Figure 3. Error Trend

The process used for developing and testing code as described is a sound one, but the effort is intense and there has never been a delivery that was allowed "enough" time. One must inevitably make a judgement as to how much regression testing is sufficient. The investment in preparation, execution, and analysis of test results is substantial, but the test of its adequacy finally will be in the successful operation of the complete system on orbit.

References

1 NASA, Goddard Space Flight Center. *Multimission Modular Spacecraft Onboard Computer Flight Executive Technical Description*, S-700-56, July 1982.

2 Trevathan, Charles E., *et al.* "Development and Application of NASA's First Standard Spacecraft Computer," *Communications of the ACM*, Volume 27 Number 9 (September 1984), 902-913.