D2

N85-16891

SHUTTLE AVIONICS SOFTWARE
TRIALS, TRIBULATIONS AND SUCCESSES

O. L. Henderson
HONEYWELL INC.

ABSTRACT

The decision to use a programmable digital control system on the Space Shuttle engine was a new application of digital control in the early 1970's. The use of digital control was primarily based upon the need for a flexible control system capable of supporting the total engine mission on a large complex pump fed engine. The mission definition included all control phases from ground checkout through post shutdown propellant dumping.

The flexibility of the controller through reprogrammable software allowed the system to respond to the technical challenges and innovation required to develop both the engine and controller hardware. This same flexibility, however, placed a severe strain on the capability of the software development and verification organization. The overall development program required that the software facility accomodate significant growth in both the software requirements and the number of software packages delivered.

The above requirements became a serious challenge to the software development facility. This challenge was met by reorganization and evolution in the process of developing and verifying software. The resulting process consistently provided high quality software on schedule throughout the balance of the shuttle program.

INTRODUCTION

In March, 1972, NASA selected the Rocketdyne Division of Rockwell International to design and develop the Space Shuttle Main Engine (SSME) for the reusable Space Shuttle. The engine development program was managed by NASA/Marshall Space Flight Center (MSFC) and was supported by various engineering laboratories within the Science and Engineering Directorate.

A digital computer control concept was selected as the basis of the engine control system. Digital control was selected over the more classical engine control concepts of valve sequencing or analog computer control because of the following advantages.

The engine was still in the early stages of development and the digital controller allowed modified operational sequences and functional changes through software updates. Time consuming and costly controller hardware modification could be avoided as the engine matured. The concept of software modification provided flexibility and adaptability during all stages of the engine development and use.

The digitally based systems also demonstrated the most economic means of providing fast and flexible control and sophisticated monitoring and redundancy management schemes for the complex operational sequence of the engine.

The above attributes of the system were reflected in the requirements for a software development and test facility which could respond quickly and accurately to the needs of a dynamic engine development program. The initial organization defined for this facility had previously demonstrated iteself as being effective for the typical embedded software program in a digital controller. The magnitude and scale of software development required for the Main Engine Program soon made it evident that the "classical" organization was not up to the task.

The following paragraphs describe the early problems and the solutions developed to provide the required quality software needed to support the engine development program.

DISCUSSION

EARLY HISTORY AND CHALLENGES

The initial organization defined for the software development was as shown in Figure 1. This organization was typical for an embedded software program delivered with a hardware unit in the early 1970's.

The software group had a vertical organization for design coding and testing of software programs. Requirements were from two sources: the controller hardware/software relationships from the project systems group; and the engine system software needs from the customers software requirements specification. The organization had a strong knit team approach and was staffed with experienced software designers who had the background to efficiently develop and test a software program. This approach had generally worked and was cost effective on the embedded software programs delivered in previous control system applications.

The organization appeared to fit the program. The initial design definition was going to lead to the Flight Configuration. The controller hardware being developed was the projected flight control configuration and there was only one defined deliverable software configuration.

The major challenges recognized relative to the controller software at this time were:

1. The engine being developed was the most complex high performance engine ever built. Due to its high operating pressures, temperatures and turbopump speeds very precise and rapid control of critical engine functions were of prime importance.

2. Failure of a control function during engine hot fire development testing could result in catastrophic damage. The system hardware was redundant to provide safety against failure. The software, however, was identical in the redundant channels of the controller. A software error that was fatal in the first channel would also be present in the second channel.

The above requirements dictated a conservative software design with significant amounts of selfchecking and the ability to respond rapidly to a control perturbation. It was also recognized that this software would require extensive testing and verification prior to use in an engine hot fire situation.

The philosophy on the testing was to do three levels of test. The first at the software module level using a host machine. The second at the software system verification level in the controller hardware while the hardware was linked to a hybrid simulation of the engine and vehicle interfaces. The final testing was to be performed at the NASA Marshall Hardware Simulation Laboratory where the software again in the controller hardware was verified with an engine simulation containing the actual engine control sensors and actuators.

This overall test philosophy proved to be correct and remained throughout the software program development stages.

There was a third and significant challenge to the software development. This challenge was not initially recognized for either its scope or magnitude. The problem evolved from one of the significant advantages of using the programmable digital controller. The control system allowed relative inexpensive flexibility in accomodating changes in the engine and associated control scheme. This led to a staged evolution of the overall system development. Innovative changes to the engine system could be planned and implemented without schedule loss due to controller modifications.

The difficulty was in the demand this placed on the software development and verification operations. The changes and evoluation required ever increasing software growth in program size and sophistication. This had been anticipated but the actual growth was between 100 to 200% over early projections. The growth was primarily due to the fact that as the engine/controller system matured it was recognized that a more efficient and sophisticated control capability could be accomodated by the basic digital control concept. The most severe demand on the software development, however, was due to the fact that engine development dictated a controller and associated software be available from the earliest stages. The early availability need resulted in a semi-hardened controller hardware design and placed the rapid change requirements on the software. This was further aggrevated as both the engine systems and the controller hardware definition changed as the engine matured. The above created an incremental or multilevel approach to the engine testing and resulting controller software needs.

Reasonable progress on the program dictated that there would be several iterations of both the engines configuration and engine controllers as we progressed to the final operational stage. These iterations had changing and or conflicting software requirements. Also several versions of the software configurations were required simultaneously to service the various engine test stands and development laboratories.

The above scenario resulted in the requirement of simultaneous development, testing and maintenance of two to three basic versions of the operational software in the same facility with the same technical staff. Each basic version of the software had 20 to 30 revisions which required verification, certification and shipment.

The realities of the situation soon made it evident that the software organization was not meeting the immediate or long term needs of the controller program or the engine development program.

The initial concept of software development could not handle the environment. We saw a degradation of the quality of the software documentation and numerous errors in the shipped versions of software. Serious cost overruns and schedule slips developed. There was a growing concern we would provide software that could result in a catastrophic engine failure.

DEVELOPED SOLUTION

The basic problem with the existing organization was that it had not been structured to handle the large software development operation required to deliver and maintain simultaneously several software definitions for a single target control system.

The problem was reviewed and broken down into the following elements:

1. Organization: The software design, development and test responsibilities had been organized as a tight knit team with overlapping and fuzzy responsibilities. The group worked well on a typical single string embedded software development but was inefficient for the large embedded program requiring multiple simultaneous program design cycle iterations.

   Their was a need for a better definition of the responsibilities and interface between the Systems group and the Software Design group. Software Design was accepting direction from both the customer through the Customer Software Requirements Specification and from Systems through the Controller Systems Requirements Specification. This direction at times conflicted or overlapped due to the different stages of engine system and controller development.

2. Change Control: Software requirements were changing rapidly. Change and problem control were not rigidly documented and became suspect when the volume of changes and problems increased. The test plan or test tracking system defined could not handle the heavy volume of test and retest that was building up on the program. The software design documentation specification structure was formalized and specific but the above problems caused this documentation to degrade significantly in both its accuracy and the ability to meet shipping schedules.

3. Quality Control: There was a need for an independent group identified to review and pass on the accuracy or quality of the software or its progress throughout the design process. Quality was left to the individual designers for their portion of the effort. Although the individual effort was good, the continuity across the program was missing and overall quality suffered.

4. Management: The design process had grown so complex and large that even though individuals in the organization felt that they were accomplishing their tasks currently and reporting to be on schedule, the overall development was missing all of the major milestones. The problem was isolated to the fact that intermediate check points or gates in the design process were non-existant. The designers did not understand they had a problem until the last step which was to test their design, against a program requirement, in the verification test facility.

The above reviews and problem identification resulted in a reorganization of the software development groups and the development process. The reviews identified that the environment of the project required multiple software programs in various stages of maturity to be in the development pipeline simultaneously.

The software development was reorganized by borrowing from the structures and documentation requirements that had been proven for multiple phase hardware development cycles. Specifically the development was changed to break the overall process up into smaller operations which could be identified and managed independant of the proceeding or following operation. Completion criteria and control documentation was then identified for each of the operations. These changes allowed the tracking and management of the multiple programs in process.

The software development was reorganized as shown in Figures 1 and 3. This organization provided solutions to the specific problems we had identified in the previous paragraphs. Originally it was felt that the change in organization and added controls would significantly increase the cost of the operation, however, after some evolution and refinement this organization proved to be able to deliver higher quality software and software documentation at a comparable cost and always on schedule.

Referring to Figure 2, it can be seen that the new organization contained two new groups, Test and Reliability. Also the roles of Systems and Software Design were modified to fit the concept of the software development facility operating as a software factory or production line.

The reorganization and the addition of several procedures and methods did not significantly change the overall functions performed during the software development. The changes were intended to break the overall development of the process into manageable and traceable subunits with definitive completion criteria. Each subunit process had a controlled source of incoming information and in turn became the controller source of incoming information for the next step of the process.

The following paragraphs will describe the changes that were made and the problems that were solved by these changes.

Systems Design

The Systems design group was identified as the primary customer interface. All customer direction and software design requirements flowed through the systems group. This provided a focal point for the design requirements definition for the various software packages being developed. The above procedures provided a unified and controlled interface between the software process and the customer and hardware needs and revisions.

The second significant change relative to the system group was that they were given the responsibility for the generation of the Verification Test Requirements for the software. This document defined the test or retest required for a base program or for a revision. Generation of these documents significantly improved the quality of the software testing. The software was not tested against a design requirement and not the software design interpretation of the requirement.

Systems was also given the responsibility of defining the retest required against a given operational program if that program was revised and updated. The organization now had all external interfaces and software requirement documentation, both design and test,focused in one group. This change eliminated the earlier conflicting requirements and definitions from reaching the design floor. The Systems group integrated all design and test requirements into documents they controlled and maintained, the software designers and software testers could work to a single consistent source of direction.

Software Design

The Software Design group was also reorganized to provide a process which had small identifiable design steps that the process development could progress through. These steps were designed such that completion of a given step or function was identifiable and had a testable criteria for completion. The Software Design process now included the detail design, code and walk through of the design, testing of the design module and integration of the module into the operational program as independant functions.

The functional and detail design were provided by the software design engineers and the changes were documented via a software change memo. This memo identified the module being designed in response to the specific design requirements and also incorporated the new and/or modified software documentation required for the change.

The change memo was then used as the design print for coding of the changes. It was also used as the document that directed change of both the source code for the operational program and the program design documentation.

The software change memo provided a definite reviewable traceable definition of a piece of the software design. Although it required considerable time and detailing by the designer it allowed the use of junior grade engineers and/or technicians to do the coding, the production of the software source change and the update of the source configuration library. It also provided a document that could be reviewed against the design requirement for correctness by the reliability group. After review and completion of coding and mark-up of the redline source configuration, the design walk through was completed. Members of software design, software production, verification and systems group used the walk-through as a vehicle to verify the code, met the design and the design met the intent of the requirements.

The code was then incorporated into the source and the module was tested as an entity by the software group. It was then incorporated in the source being prepared for the test floor and the software designer and coders would use abbreviated test documentation to perform integration testing of the change. This integration testing was the final debugging of the particular change by the software group and determined if the module was ready to be handed over to the test team for verification. After completion of integration testing the new test source program was presented to the Verification Test Group. They would perform a standard verification acceptance test of the new program. This standard test checked the critical paths and timing of the operational programs nad was the criteria for acceptance of a new software version for detailed testing on the test floor.

Verification Test

The verification testing group was one of the two new sections created in the software organization. Their responsibilities included the generation of detailed cookbook type test procedures to be used in the testing of the software. The verification test group was also responsible for identification, scheduling and control of all equipment and documentation required to create and control the test facility. Their primary function, however, was to perform the actual verification of the software program and revisions.

The verification was completed as a two step process. The first testing of a new software program was called the dry run. The dry run test purpose was to identify any problems with the software update code and test procedure or test implementation. Problem resolution or debugging was not allowed during Dry Run Test. When a problem was encountered an Internal Software Note (ISN) was generated and the testing continued through the procedure. The identified problems were then assigned to the responsible groups and worked off without interrupting the test flow.

After completion of Dry Run Testing and correction of all identified problems, the formal verification testing was initiated. This formal testing did not allow deviation from procedures or expected results. Problems that occurred during this test would require a formal review and retest after correction. No outstanding problem was allowed after formal verification test completion unless it had been accepted via the customer through a Software Waiver.

The verification test group solved three problems that had plagued the earlier development. These were:

1. Under the original organization Software Design was to do their own verification testing of the software. Experience with that organization showed that the designers were not generating their test procedures to verify that the software met the design requirements. The procedures were generated to test that the software worked the way the software designer had designed it. The new organization utilized test personnel to generate detailed test procedures from test requirements documentation. These detailed cookbook procedures were purposely generated with a minimum knowledge of the actual software design. The procedure writer used the software design documentation only to find the names of subroutines or data words required for a particular test sequence.

2. The second problem solved was the original organization put an unreasonable amount of schedule pressure on the software designer. The designer had to design, code and debug his programs. He was then required to generate a formal test procedure and accomplish the verification testing. This effort did not utilize the software designers major talent in the most efficient manner. He spent too much time generating test procedure and performing verification testing of the software. The new organization solved the problem by allowing the test group to develop test procedures in parallel with the software design and also relieved the software designer of test responsibility. The software design group could go on to the next design cycle while the test group verified the current design.

3. The third problem solved was the parallel operation significantly collapsed the amount of time required to process a change. Paralleling the design operation and the generation of test procedures provided adequate margin to allow the consistent delivery of software updates to the Marshall Simulation Laboratories and the Engine Test Stands.

Software Reliability

The final group in the new organization was a small group of reliability engineers who functioned as a product assurance organization for the software operation. Their duties included review and approval of the software change memos against the design requirments. They reviewed all test procedure generation against the test requirements. Verified that all required documentation updates and design walk throughs had been completed prior to release of software to a source update. Verified that all test paragraphs had been completed and the data was acceptable. Identified and tracked all open problems against a source update through the use of Internal Software Notes (ISN) and the ISN log.

The Internal Software Note was the key document in the tracking of the entire software development effort, including the testing. While many of the ISNs are generated by the test teams, they may be written by any member of any of the four groups who detects a potential problem in a specification, the program, a test procedure, a flow chart, test equipment or anything else that may affect the software or its verification. Groundrules are that only one problem is documented on each ISN and that every problem, proposed software change, procedure chance, etc., must be documented in an ISN. New ISNs are collected each day and are reviewed by the ISN board.

The board is chaired by Reliability and includes members of the other three groups. They determine the most likely cause of the problem and assign the ISN to one of the four groups for action. At these daily board meetings all other outstanding ISNs are reviewed to determine if any are completely resolved and can be closed or should be transferred to another group for further action. Approval of all four groups is required to close an ISN.

Following the meeting, the additions, changes and closures are entered into a computer program and an ISN log is printed showing all open ISNs and the groups responsible for action. The reports highlight the number of known problems requiring resolution prior to a shipment and any group that is developing a significant backlog.

Reliability was tasked with maintaining the ISN log and a log on the status of open and completed testing. Software update could not be shipped from the facility until these two logs had been closed through resolution of the open problems or a request for outstanding problem waivers from Rocketdyne.

CONCLUSIONS

The process described above evolved from a need for extremely reliable software as a part of a critical shuttle control system and early difficulties with this software development effort. A significant measure of the success of the developed process is that the earlier experienced problems did not exist through the balance of the development program. Since implementing the process, each software update and accompanying documentation was delivered on or ahead of schedule.

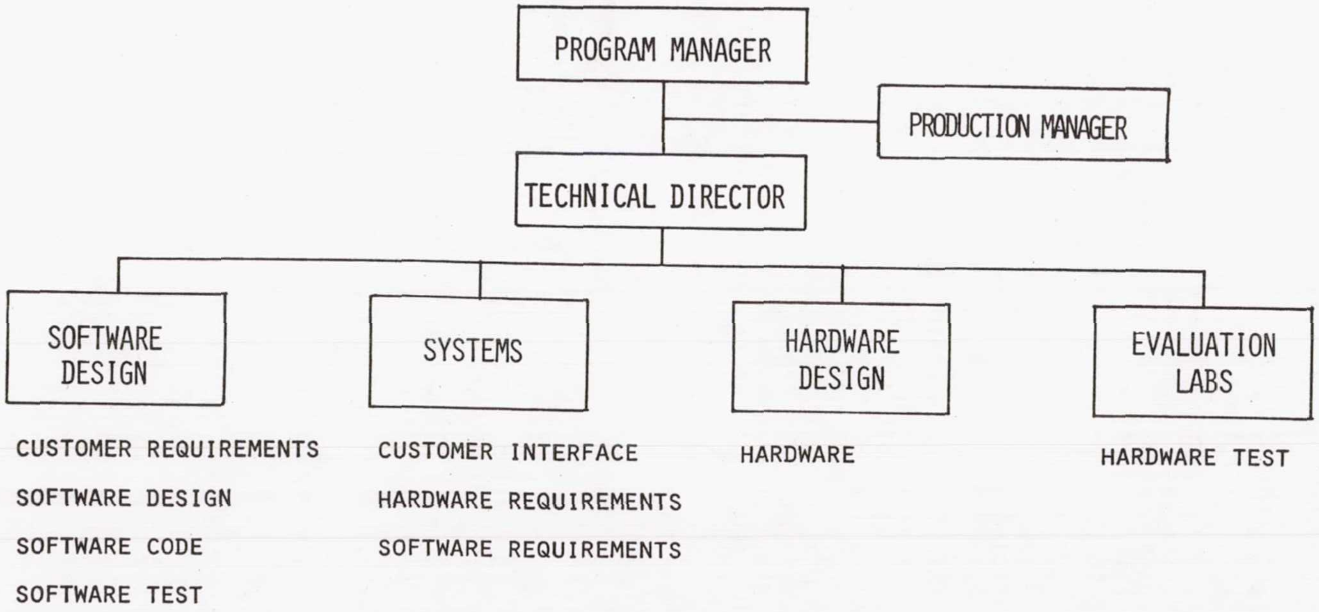
The vast amount of engine hot fire time that the controllers and the associated software have successfully supported demonstrate that the developed process has consistently provided high quality operational programs. Successful completion of the Shuttle launches further confirms that the software development program is sound.

It has been said that the real benefit of our exploration of space is not the exploration itself, but the development of techniques needed to manage large, complex processes where success is a must rather than a goal. The development of this software design system is a case in point of the above statement. The key elements of this success -- planning, organization, tracking and discipline are applicable to all software processes. The concept of distinct groups with well defined responsibilities, periodic status reviews, documentation control approvals and reviews, a formal tracking procedure and, providing authority consistent with responsibility, are not unique in their application to the SSMEC program. The need for these elements of their effective implementation was learned through experience on the program. The real benefit from this experience is the continued use of these concepts on future software development and verification programs.

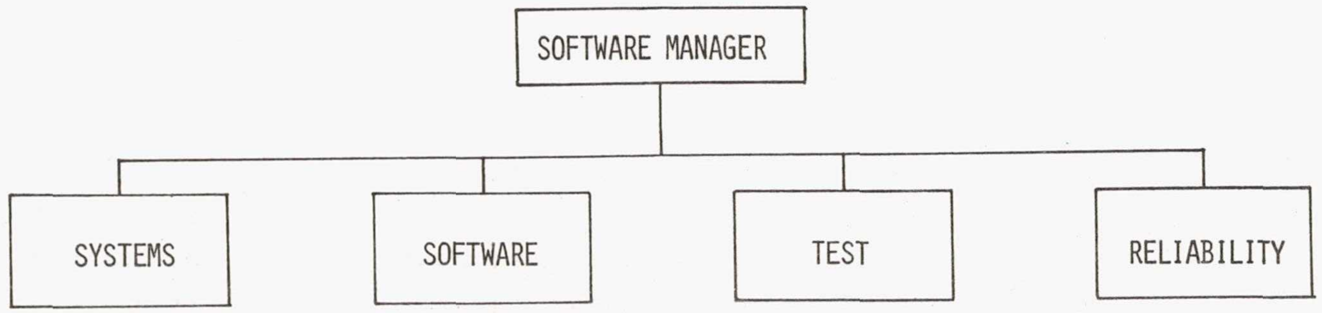
MAIN ENGINE CONTROLLER

<u>INITIAL ORGANIZATION</u>

FIGURE 1



SOFTWARE DEVELOPMENT ORGANIZATION FIGURE 2



SYSTEMS

- . DESIGN SPECIFICATION
- . TEST REQUIREMENTS
- . HARDWARE INTERFACE
- . CUSTOMER INTERFACE
- . RETEST DEFINITION

SOFTWARE

- . DETAILED DESIGN
- . DOCUMENTATION
- . CODE
- . PRODUCTION
- . PROBLEM RESOLUTION

TEST

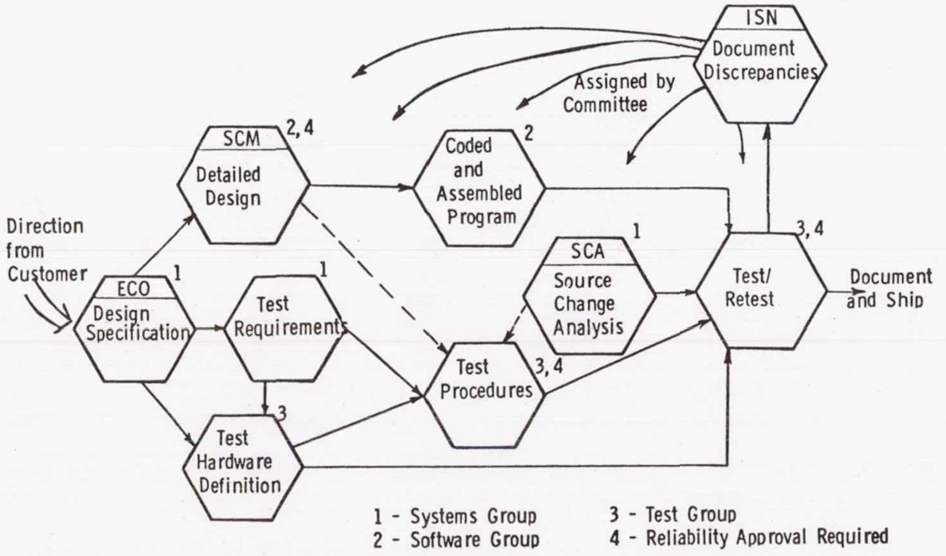
- . TEST PROCEDURES
- . DEFINE TEST HARDWARE
- . CONDUCT TESTS
- . DOCUMENT TESTS

RELIABILITY

- . APPROVAL OF;
  - DESIGN SPEC
  - DETAIL DESIGN
  - TEST PROCEDURES
- . VERIFY TEST RESULTS

PROJECT WORK FLOW

FIGURE 3



REFERENCES

1. W. T. Mitchell, "Space Shuttle Main Engine Digital Controller", Conference on Advanced Control Systems for Aircraft Powerplants, Proceedings No. 274, North Atlantic Treaty Organization, 1979.

2. R. M. Mattox, J. B. White, "Space Shuttle Main Engine Controller", NASA Technical Paper 1932, November 1981.

3. W. T. Mitchell, R. F. Searle, "SSME Digital Control Design Characteristics", Space Shuttle Technical Conference, NASA, JSC, June 28-30, 1983.