NASA Technical Memorandum 101703

Initial Flight Qualification and Operational Maintenance of X-29A Flight Software

Michael R. Earls and Joel R. Sitz

September 1989

National Aeronautics and
Space Administration

NASA Technical Memoradum 101703

Initial Flight Qualification and Operational Maintenance of X-29A Flight Software

Michael R. Earls and Joel R. Sitz
Ames Research Center, Dryden Flight Research Facility, Edwards, California

1989

National Aeronautics and
Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California 93523-5000

INITIAL FLIGHT QUALIFICATION AND OPERATIONAL MAINTENANCE OF X-29A FLIGHT SOFTWARE

Michael R. Earls* and Joel R. Sitz*
NASA Ames Research Center
Dryden Flight Research Facility
Edwards, California

Abstract

This paper is predominantly a nontechnical discussion of some significant aspects of the initial flight qualification and operational maintenance of the flight control system software for the X-29A technology demonstrator. Flight qualification and maintenance of complex, embedded flight control system software poses unique problems. The X-29A technology demonstrator aircraft has a digital flight control system which incorporates functions generally considered too complex for analog systems. Organizational responsibilities, software assurance issues, tools, and facilities are discussed.

Nomenclature

ACC	automatic camber control
AHRS	attitude heading reference system
AR	analog reversion
B/U	backup
CCR	configuration change requests
CL	control law
CLP	control law processor
CM	configuration management
DARPA	Defense Advanced Research Projects Agency
DCL	DEC control language
DR	digital reversion
FCC	flight control computer
FSIM	function simulation
FS/CP	failure status/control panel
IBIT	initiated built-in test
I/O	input/output
IOP	input/output processor
ISA	integrated servoactuator
LVDT	linear variable differential transformer
MCC	manual camber control
OFP	operational flight program
PA	power approach
PAC	precision approach control
PCN	program change notice
PDL	program design language
PLA	throttle position (power lever angle)
PL/I	programming language I
Pri	primary
Ps/Qc	static pressure/impact pressure
RAV	remotely augmented vehicle
SEU	system evaluation unit
SIBLINC	scale invert bias logic interface console
STR	system test report
UA	up and away
UDF	unit development folder
V and V	verification and validation
XAIDS	extended aircraft interrogation and display system

Introduction

The first X-29A airplane (Fig. 1) has successfully completed 242 flights, proving the concept of forward-swept wings and several other advanced aerodynamic, structural, and avionic technologies. The X-29A forward-swept wing technology demonstrator is an aerodynamically unstable aircraft requiring a highly augmented software-intensive control system to maintain stable flight. In addition to its aerodynamic insta-

*Aerospace Engineer.

Copyright ©1989 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for Governmental purposes. All other rights are reserved by the copyright owner.

bility, the X-29A airplane incorporates the following technologies:

1. Forward-swept wing planform
2. Tailored composite wing structure
3. Variable incidence close-coupled canard
4. Multisurface control
5. Discrete variable camber

This paper will describe the initial flight qualification and operational maintenance phases of the X-29A program. The initial flight qualification phase ended with the qualification of the software system for first flight. The operational maintenance phase occurred during the flight test program. During this phase several groups of changes were incorporated and the software system was requalified for flight after each change. The flight control system, of which the software is a part, is discussed to provide insight into the complexity of the flight qualification tasks described. The multiple flight control modes, flight control system components, and digital flight control computer architecture are also discussed.

The first flight of the X-29A airplane occurred on December 14, 1984. Initial flight qualification included the activities which assured that the software to be used on first flight, integrated with aircraft systems, was safe (flight-qualified) and that initial mission requirements could be accomplished. Organizational responsibilities, the software development process, and software assurance functions (change control, discrepancy reporting and correction, and testing) during preparation for first flight are discussed.

Operational maintenance included the process by which software system changes were incorporated and the software requalified for flight during the operational phase of the program. This phase started after flight number four, when the U.S. Government took possession of the aircraft and assumed responsibility for flight safety. Operational maintenance topics included changes in organizational responsibility, software development, configuration management and other software assurance topics, time-saving parallel development and test activities, and the importance of a high-fidelity simulation at the user's facility.

Some observations and recommendations are included, based on X-29A experience in development and maintenance of a complex software-intensive system. Management and technical issues during initial flight qualification and operational maintenance phases of the program are discussed.

Generic terms are used to describe the organizations involved in the X-29A program. The airframe manufacturer was Grumman Aerospace Corp., Bethpage, New York. The term "prime contractor" refers to Grumman. Grumman subcontracted to Honeywell, Inc., Minneapolis, Minnesota, for the sensor computer subsystem which included the flight control computers (FCCs), flight software, and control system sensors. The terms "subcontractor" and "developer" refer to Honeywell. The U.S. Government is the "customer" and "user." "User" refers specifically to the National Aeronautics and Space Administration (NASA), Ames Research Center, Dryden Flight Research Facility, Edwards, California, where the flight test program was conducted.

System Description

Flight Control System Modes

The control system modes and their options are summarized in Table 1. Several of the modes incorporated variable gains based on airspeed, altitude, Mach number, and angle of attack to optimize the aircraft's capabilities throughout the flight envelope. Normal mode, with its subsets, was implemented in the digital computers. Degraded modes were included to allow the system to remain in normal mode with degraded performance after certain failures. The name of the direct electrical link mode is misleading because the mode had the capability of stabilizing the aircraft during takeoff and landing—a capability that would not exist in a direct, unaugmented, stick-to-surface control system. Analog reversion (AR) was a backup mode, implemented in the analog computers, that was capable of recovering the aircraft from anywhere in the flight envelope. Digital reversion (DR) was a digital backup mode, which was removed during the flight test program. A more complete description of the X-29A control system modes can be found in Ref. 1.

Flight Control System Components

Flight control system hardware is illustrated in Fig. 2. The canards, flaperons, and strake flaps were utilized for pitch control. Flaperons were also used for roll control, and the rudder was utilized for yaw con-

trol. Crossfeeds between lateral and directional axes were included in the control laws that coordinated the flaperon and rudder commands.

The flight control system was a triplex digital system with a triplex analog backup system. The goal of the system, with its associated failure detection, redundancy management, and fault reaction, was to provide safe return of the aircraft after any two subsystem failures; that is, fail operational/fail safe. The airdata sensors, accelerometers, and primary rate gyros were triplex for the digital control system. A backup set of triplex rate gyros and a triplex set of impact pressure sensors were provided for the backup analog control system. The backup rate gyros were used by the digital system in the event of a primary rate gyro failure. Pilot commands utilized triplex sensors of stick and rudder pedal positions. Single-string inputs to the control system included the attitude heading reference system (AHRS), throttle position (PLA), and the remotely augmented vehicle (RAV) system.

Each digital and analog computer in the flight control system outputs all surface actuator commands. Hardware logic monitored the flight control mode and thereby selected raw commands from either the digital computer or the analog computer. The raw commands were interchanged between FCCs in analog form, and hardware logic in each FCC outputs to the actuators the midvalue of its commands and those of the other computers. Each digital computer provided an ARINC 429 bus output, which contained information on both proportional and discrete internal parameters, to the data acquisition system for downlink telemetry. Only one digital computer provided output to the engine, while another provided output to cockpit displays.

Digital Flight Control Computer Architecture

Each of the three identical FCC boxes contained two digital processors and a backup analog computer. Digital computers were a derivative of an HDP-5301 processor that had been used on several other programs. Figure 3 shows a block diagram of the digital computer system. To achieve the required throughput, digital processing in each FCC was distributed between an input/output processor (IOP) and a control law processor (CLP). Interface functions, performed by the IOP using fixed-point calculations, included discrete and analog input and output; ARINC 429 bus input from air data sensors; ARINC 429 bus output to downlink telemetry system; redundancy management; failure detection; fault reaction; and initiated built-in test (IBIT). The CLP executed the control laws, calculating actuator commands and some cockpit displays from pilot and sensor inputs, using fixed-point and floating-point calculations. Each processor contained 2 k of random access memory and 14 k of electrically erasable, programmable, read-only memory. Synchronization and data exchange occurred between the IOP and CLP through 1 k of common memory.

The digital system operated at an 80 Hz minor cycle, but the majority of the computations were run at 40 Hz. All FCCs ran the same software. The three FCCs were synchronized to real time and to each other at the beginning of each minor cycle. The executive functions were task scheduling, synchronization and data exchange between FCCs, and synchronization and data exchange within an FCC. The executive used a cyclic, single-tasking structure which facilitated timing analysis because of its invariant execution order.

Each FCC had a serial data transfer capability with the other two FCCs. The intercom data rate was 1.0 MHz, with sending and receiving under direct control of the IOP. Sensor and pilot command inputs were exchanged for signal selection and monitoring, so that all FCCs used the same inputs to the control laws. Actuator commands were exchanged to assure that the outputs from the control laws were identical, having used identical inputs. The integrity of each data exchange was tested to ensure that all FCCs performed signal selections from identical data sets.

To implement the cross-channel FCC data exchange and synchronization, the common memory had one word of data and a one-word software flag for each input and output that was processed. This technique allowed the time lag between sensor input and control surface output to be minimized and held constant.

The computers also executed many built-in tests that were designed to ensure that the flight control computer hardware and software were performing properly. These tests were performed when the system was powered up and operating, in both the IOP and the CLP. A set of pilot-initiated built-in tests checked the flight control system sensors, displays, indicators, and actuators.

Reference 2 contains a detailed description of the digital FCC architecture.

Initial Flight Qualification

Organization and Responsibility

The X-29A program was sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) and managed by the U.S. Air Force Flight Dynamics Laboratory. The aircraft was built by the Grumman Aerospace Corp., who had responsibility for flight safety until the aircraft was delivered to the government. NASA's Ames-Dryden Flight Research Facility conducted independent reviews and analysis of pre-delivery activities and participated in aircraft ground tests. Grumman subcontracted to Honeywell, Inc., for the sensor computer subsystem, which included the FCC hardware and software. The prime contractor was responsible for providing the customer with a flight-qualified aircraft, including FCC flight software.

Software Development

It is not economically feasible to replace the advantage of a well-managed, structured software development approach with testing; it would require an infinite amount of testing to gain confidence in software produced using a poor development process. The subcontractor was responsible for the software development process, with formal design reviews held by the prime contractor and customer. The organizational structure of the software development group consisted of two major subgroups. One group was responsible for the control law implementation, the other for the input/output (I/O) implementation. Both groups had lead engineers who reported to a software manager responsible for the technical and administrative aspects of the software development process. It is customary in complex flight-critical applications to assign the functions of design, code, and test to separate groups; however, due to resource constraints, all of these functions were performed by the same personnel.

A major factor in developing quality software is the use of software standards and practices during software design, code, and test. The software development guidelines used were internal subcontractor guidelines based on DOD-STD-2167. Figure 4 presents an overview of the subcontractor's software development process and illustrates how the development process was partitioned into distinct phases. The figure also shows the relationship between the hardware and software development. The process is shown to proceed sequentially through the phases, but iterations to the requirements, design, code, and test procedures did take place. Once a baselined phase was complete and a change had occurred, the requirements and the software design were updated, and, if already baselined, the code and test procedures were also updated and reexecuted as part of the phase then currently in progress. Reliable software was obtained by implementing good engineering discipline in the software development process.

A major challenge of any software-intensive development effort is the apparently never-ending changes to requirements. Fundamental to the design of a flight control system for a technology demonstrator is the ability to accommodate changes in the requirements with minimum resources. The deficiency in most software development life cycles is the assumption that a perfect set of requirements exist before the design phase begins. The subcontractor contended with this deficiency in three major ways:

1. Requirements the X-29A airplane had in common with other digital flight control system architectures were applied to minimize the need to reestablish those requirements and to take advantage of off-the-shelf software and hardware design.

2. Experienced systems and lead software engineers reviewed customer and mission requirements and determined what the customer "system objectives" were and translated those objectives into a formal design specification. The translation process consisted of written text in block diagram form and was based on their systems and software experience and close interaction with the customer. The block diagrams were especially useful in the test and maintenance phases of the program.

3. The change control process was well structured and key documents (related software development plans, design specifications, and test plans) were baselined to easily accommodate changes to the customer requirements.

Each flight control computer had two Honeywell HDP-5301 digital processors. The software was written in HDP-5301 assembly language which consisted of a 59-instruction-set architecture. It was a unique instruction set and did not represent a microprocessor standard. Calculations were mixed between fixed- and floating-point operations of single and double precision. The processor had a mode to prevent regis-

ter overflow. The operational flight program (OFP) in each FCC consisted of 205 total modules (approximately 29,000 instructions) utilizing approximately 2200 variables and 3000 constants. Execution of the OFP in each processor was controlled by a software executive rate structure as opposed to an interrupt-driven control. The memory organization of the OFP consisted of localized variables and constants used only internal to a module. It also had common data repositories used by two or more modules for major functions such as signal selection and redundancy management, mode logic and fault reaction, and pilot IBITs where common variables were held.

Each module contained a module header which identified the module by name and number, described the function of the module, listed in tabular form by date the module change history along with the author, and listed the inputs and outputs. The design for each software module was written in a loosely structured program design language (PDL). The module design was included with each module as part of the comments.

Once the software engineer had clear design specifications, the coding process itself was a relatively straightforward task. To assist in keeping the code structures used by different engineers consistent, the project software team established coding guidelines. These guidelines suggested ways to implement various code structures in an efficient manner. Assembly language macros from other projects using the same programming language were also used to aid in the coding process. Once the requirements were defined, the module was designed and coded, and the module test was written. A more detailed description of the tests and test philosophy follows in the initial flight qualification software assurance section of this paper.

All of the information applicable to each module was kept in a unit development folder (UDF) in a centralized library. The UDF contained the module design, module code, module test procedure with results, and applicable engineering information that might be of interest to an engineer designing or coding an interface with that particular module. The folder also contained redlined masters of previous versions of the module as it was changed in the development process. This allowed a reviewer, for instance, to easily trace how a change was incorporated.

Along with the formal design specification that was used to code from, there was also a software description document that described various important aspects of the software development process and key information for the operational maintenance phases. This document contained the memory map for each processor (what was used and what was spare), how to assemble and link the code for each processor on the engineering workstation, descriptions of major functions implemented in the software, the design for each module, data dictionaries, variable set/used tables, and a graphical representation of the throughput and worst case timing analysis for the operational flight program.

Tools

There were numerous software tools that were used to minimize development time and the probability of human error. The tools utilized are listed and described in Table 2. Because of the size and complexity of the operational flight program, the tools used to help the developer organize information about the code were the most advantageous. These included the set/used tables, data dictionary tool, memory map tool, the Bender chart tool, and the workstation tools.

The Bender chart tool was specially useful because of the graphical manner in which it conveyed information that was otherwise difficult to describe. An example of a Bender chart is shown in Fig. 5. The Bender chart tool shows the module execution frequency (horizontal width) and the minor frames in which the module was executed (at bottom of chart). It also presents the worst case timing of each module (vertical length) and shows resynchronization and processor rendezvous points.

The timing tool was useful for two reasons. First, an important aspect of any real-time application is worst case module execution time. This is needed to make sure available throughput is not exceeded. Second, because of the synchronous nature of the system, channel-dependent paths in the code needed to be balanced. The timing tool assisted the engineer in determining the execution time of a given path using the HDP-5301 source code as input.

The set/used tables allowed the developer to quickly trace a variable's span of influence across the modules in which it was used.

Another class of tools that proved to be extremely helpful were tools that assisted the tester. These included the data integrity tool, overflow analysis tool, and function simulation (FSIM) tool.

The data integrity tool was a programming language I (PL/I) program that used the HDP-5301 source code as input. It verified that memory reference instructions dependent on processor mode (that is, fixed- as opposed to floating-point mode) were used after the processor had been set to the appropriate mode. Gain scheduling using linear interpolation in fixed-point mode made it necessary to use the overflow analysis tool to protect from inadvertent overflow. This tool checked adjacent gain table breakpoints and verified that the gain scale factor was large enough to accommodate differences.

The FSIM tool was an interactive program in which the user would enter various types of control law elements (that is, lag filters, integrators, time delays) in a given path and analytically compute the gain and phase margins at discrete frequencies for that path. These outputs would be used to compare with actual open loop frequency response results.

An extensively used tool at the user facility for software system testing was the extended aircraft interrogation and display system (XAIDS). It was a multiprocessor system which performed several functions: terminal emulator to the system evaluation unit; patch generation and transmission to flight control computers; and display of ARINC 429 bus data from the flight control computers on a terminal in engineering units, as well as output of the information as analog signals. The ARINC 429 display and conversion was particularly useful, since it was the only way to display internal software parameters in real time.

The tools used on the X-29A program played an important part in developing quality software.

Software Assurance

Software assurance is a series of processes whose goal is to deliver error-free, maintainable software. It includes software design standards and practices (described above), configuration management, reviews, test philosophy, software verification and system validation, and media production and control (that is, object tapes and source tapes).

Configuration Management (CM). This process assures that the system configuration and problem status are known, documented, and traceable. It is an essential process throughout the life of a complex system, especially one which is software intensive. Configuration management consists of two major processes: (1) change control, and (2) discrepancy reporting and correction. Change control is the process by which design or implementation changes are defined, implemented, tested, and incorporated in a new baseline software release. Discrepancy reporting and correction is the process by which design or implementation anomalies are reported, assigned criticality, and closed out after correction (through the change control process).

The CM process of each contractor interfaced with the other such that software configuration and discrepancies were known and traceable. The prime contractor used a software control board to authorize control system design changes, trace software releases, and track discrepancy reports. Its activities were limited to the flight control system hardware and software. The software developer used internal documentation to track discrepancies and perform change management. The customer configuration control board operated in parallel with the contractor software control board, in an advisory capacity only.

Reviews. Reviews were held at all levels of development. Figure 4 shows the developer review guidelines. Module design, code, tests, and test results were subject to peer review. Integration tests and results were also reviewed by lead engineers. The prime contractor, whose personnel were resident at the software developer's facility, conducted a formal review of the module tests. The customer, whose personnel were resident at the prime contractor's facility, reviewed tests conducted by the prime contractor. Both the contractor and the customer held flight readiness reviews.

Test Philosophy. Flight software undergoes two types of testing activity during the process of flight qualification: verification and validation. Verification is the process by which determination is made that the software performs as specified. It is accomplished by devising individual tests for each specified software task, conducting the test, and observing that the task was accomplished according to the specification. Validation is the broader task which seeks to determine if the system, of which the software is a part, performs adequately to accomplish the flight requirements. Failure modes and effects tests (both open loop and closed loop) are among the techniques used in software validation. In these tests, failures are artificially induced and a proper system response to those failures is verified.

The critical nature of the software (capable of causing loss of life or aircraft) dictated that a maximum

level of testing within the resources of the project be accomplished. System design reflected the criterion that the aircraft be able to return safely after any two failures (fail operational/fail safe). Verification tests included software system integration aspects as well as individual software module testing. Validation tests concentrated on the most critical elements of the software, that is, control laws during normal operation, fault reaction, mode logic, and sensor redundancy management. To minimize the risk of undetected software errors, independent verification and validation (V and V) tests were run in addition to the contractor's V and V testing. The philosophy was to gain system experience in the most realistically simulated environment possible to increase confidence that the system would perform as expected in the real world environment. The possible known combinations of real world stimuli are too numerous to analyze completely. The more a system is tested in a simulated environment, the more unlikely it becomes that generic faults will be discovered during flight test. The use of experienced software and systems personnel in these areas of the process was very cost effective.

Software Verification and Validation. Verification and validation tests were performed by the contractors in the initial flight qualification phase.

The subcontractor partitioned his verification testing into three phases: (1) module testing, (2) internal independent review, and (3) systems testing. The rationale for phase one module testing focused on two major points. First, the module test was to verify that the code reflected the intent of the module design. The module test did not test the function of the module as part of the system as a whole. Second, the module test exercised all code paths at least once and, where applicable, tested minimum and maximum input values. In the case of complex modules, the module test contained a description explaining the test cases. Phase two consisted of a subcontractor's internal independent general review of the software structure, module interfaces, system requirements, and planned test coverage. Phase three consisted of the systems-level tests with the software and hardware integrated. System performance was tested against the system requirements. The majority of the system-level tests run on the control law processor consisted of open loop frequency responses of the control law paths. Other tests run were gain table X-Y plots, variable warmstart analysis (occurred when a computer was reset), and an overflow analysis.

Most validation tests were performed using a six-degree-of-freedom hardware-in-the-loop simulation at the prime contractor facility. This simulation included three FCCs using the flight software under test, a failure status control panel similar to that in the aircraft, and in some cases electro-hydraulic actuators driving simulated control surfaces. Flying quality evaluations of the unfailed system were conducted at specified points throughout the flight envelope in all of the control modes. Failure modes and effects tests, which tested flying qualities as well as system response and annunciation during mode changes and simulated control system hardware failures, were conducted. Time history and frequency responses were generated. Simulation results were compared with those predicted analytically, and data on stability margins was obtained. A five-degree-of-freedom simulation, with the actual aircraft in the loop, was used for some validation tests. For example, during limit cycle tests, the control system gain margins were examined.

Independent Software Verification and Validation. The customers (NASA, Air Force, and Navy) performed independent V and V test sequences using NASA Ames-Dryden's six-degree-of-freedom hardware-in-the-loop simulation, as shown in Fig. 6. This simulation included three FCCs, a failure status control panel, and an analog computer model of surface actuators. Some contractor tests were repeated, and customer-generated validation tests were run. These tests were of the same types as those run at the contractor facility, but they used different hardware, simulation system, and test procedures. Testing from a second perspective, using a different test facility, minimizes the risk of undetected errors. This risk reduction is the value of independent V and V. For example, it was discovered during these tests that certain dual-rate gyroscope failures to null caused loss of aircraft if the first failure was undetected.

Operational Maintenance

Changes in Organization and Responsibility

Official delivery of the aircraft, with a limited flight envelope, occurred after flight number four. The customer then assumed responsibility for the aircraft, and for flight safety. The contractors continued participation in design changes, and were responsible for design and V and V of the full envelope control system. Except for the first major modification done to the control laws (full envelope release), validation testing was the

responsibility of the customer and verification testing was shared by contractor and customer.

Software Development

When changes to the design specification were received by the developer, he chose a method of implementation based on the complexity and form of the change being requested. This decision affected the testing process both at the developer and customer facilities. The four methods of implementation that were used will be discussed in order of increasing complexity. The testing path for each of the implementation methods is illustrated in Fig. 7.

Overlay. Only simple changes of values of constants were incorporated by overlay. In this method no source code was changed. The FCC program was not reassembled. Flight control computer program memory was loaded with the baseline release. The FCC memory was then altered to incorporate the changes, and the contents of its program memory (not just the changes) were re-recorded to a new tape containing the new software release. This type of implementation required the least amount of retest. Overlay changes were retested on each subsequent release until a release was implemented by relink and tested.

Permanent Relink-Without-Relocation. Only simple constant changes were incorporated by permanent relink-without-relocation. In this method source code was changed, but the change did not result in any code relocation. Thus, no instruction addresses changed from the previous release. Affected modules were reassembled and all modules were relinked. This method required the same amount of testing as an overlay change. No retest of the change was required on subsequent releases as the change was permanent.

Relink-Without-Relocation. Simple structural code changes were generally implemented by using this method. Source code was changed in such a way that code downstream of the change was not relocated. The changed modules were reassembled and all modules were relinked. Added code was put in a designated unused area and called from the modified module by way of branches inserted in-line so as not to relocate any other code. This method was used when it was desired to minimize the level of testing. It is second to relink-with-relocation in terms of the amount of testing required. Changes implemented by this method were retested when the code located in the patch area was put in-line by way of relink-with-relocation.

Relink-With-Relocation. In this method, source code was changed and all modifications were made in-line. The affected modules were reassembled, and all modules were relinked. The term "with relocation" indicated that in the process of adding or deleting code, other code was relocated. As a result, the addresses downstream of the change were no longer the same as in the previous release. This method was used for major structural changes and when a new, unadulterated baseline was desired. As in all changes made by relink, object code was downloaded to the program memory in the FCC and the program memory contents were then recorded onto a cassette tape. Of the four methods used to implement a change, this method required full regression testing.

Tools. Various tools which have become available since the initial software development, along with those discussed in the initial development section of this paper, were used during software development and test at the developer's and the customer's facilities. These are listed in Table 2 as used in the maintenance phase.

The configuration control tool was a developer software library manager which tracked module source code revisions and prevented any module from being modified by two programmers at the same time.

The spare memory tool was used by the developer to document unused program memory. This information provided statistical data on percent memory usage, as well as specific information on memory available in patch areas for use in a relink-without-relocation.

Checksum tools were used by the developer and the customer to partially verify that the download of object code to flight computer memory from the assembler/linker was successful. Checksums were calculated from the output of the linker for comparison to those calculated by the system evaluation unit (SEU) from flight computer memory.

The SEU interface utility is a personal computer program, developed by the customer but also used by the flight software developer, in which the personal computer is a front end to the system evaluation unit. Normal and enhanced terminal functions are performed; however, additional menu-driven capability includes saving the terminal keystrokes and responses on a disk file, creating flight computer patches, saving them to disk file, and sending them to the flight computers (in any combination of channels), starting and stopping the processors, and computing checksums.

New tools to enable the customer to do the source comparisons required in releases produced by relink-with-relocation include tape reading tools, which unpack and sort developer source code tapes so the information can be saved on disk, and a source file comparison tool, which does the source comparison between two disk files and then prints the differences.

The overplot tool allows the customer to plot test data over reference data on a hard copy. Time histories and frequency response gain and phase are routinely overplotted as part of software system validation.

The XAIDS was equipped to retransmit ARINC 429 bus data from the three FCCs to the simulation computer on a single 1553 bus.

Software Configuration Management

Configuration management responsibility was assumed by the customer upon aircraft delivery. Prime contractor internal configuration management remained active, but with decreasing project responsibility. Subcontractor configuration management remained essentially unchanged.

Configuration Control Board. The configuration control board met at the customer facility and was chaired by the customer project manager or a designee. The board was composed of technical experts from all program disciplines involved in flight safety and research. Membership included representatives of all governmental and contractual organizations involved with the aircraft. Disciplines included flight operations, flight controls, simulation, aerostructures, loads, aerodynamics, instrumentation (including onboard systems, real-time data display, and data reduction), and flight planning.

The board performed the same functions as the contractor software control board. These functions included change control, discrepancy reporting and corrective action, and risk assesment. However, in addition to those functions, the board also controlled the entire aircraft, control room, data reduction, and simulation.

Change Control Process. The X-29A process is illustrated in Fig. 8. Changes were generated due to discovered problems (in which case a discrepancy report was written) or because of the addition of new mission requirements. Configuration change requests (CCRs) could be written by anyone and were submitted to board membership in advance of the regularly scheduled meeting. All items handled by the board were identified and entered into a database system to facilitate traceability. The board had several options on the disposition of new change requests: (1) hold, (2) return for analysis, (3) disapprove, or (4) approve. It could also cancel previously submitted requests. Items "held" could be resubmitted at a later date. Items returned for analysis required clarification to convince the board that the proposed change was safe or even necessary. Disapproved changes required no further board action. The board made technical recommendations to project management on the disposition of CCRs. However, management could overrule decisions they considered beyond the means of project resources. Anyone on the board could recommend that a change not be incorporated.

Once CCRs were written, software program change notices (PCNs) were generated from the CCRs by on-site contractors who had the most detailed knowledge of the control system architecture and specification documents. Program change notices were software design documents which had two functions in the X-29A program: (1) to provide detailed software design specification changes, and (2) to authorize implementation of those changes. These documents were presented to the configuration control board in two different ways: (1) for information only (no approval required) if the design changes mirrored the CCR from which it was derived; or (2) for official approval, if the design differed from the initiating change request. Program change notices were identified by the customer using a hybrid numbering system which consisted of a customer version number with a contractor version number appended (that is, PCN 012-2015).

A PCN was delivered to the prime contractor who, in turn, delivered it to the software developer. The software developer then incorporated the change using the production method agreed on by the customer and on-site contractor personnel. An engineering change notice containing information on the design, implementation, testing accomplished, cross-reference to the PCN for traceability, and developer's identification of the new software release was generated. The engineering change notices were then delivered with the final version of the software release.

If a new release was to be generated by overlay only, the customer had the option to create the new object tape in-house and begin V and V testing. This tape would later be compared bit for bit to the final release tape sent to the customer after the developer had completed required verification testing. If the tapes

did not compare bit for bit, all testing done on the customer-created tape would be invalidated and the process would begin again with the developer's final tape. When the new release was generated by a method other than overlay, the customer would receive a preliminary object tape when the developer had sufficient confidence in the implementation of the new release.

A new software release was delivered to the customer by way of the prime contractor. The customer then renamed the release to correspond with the scheme inherited from the prime contractor. On receipt of the preliminary tape, the customer would start the V and V process, as shown in Fig. 7. Meanwhile, the developer completed the verification tests at his facility and either delivered a final tape or authorized the customer to relabel the preliminary tape as a final tape.

Customer V and V would continue until all required tests were complete. System test reports (customer test results which had been reviewed by a cognizant engineer) were then presented to the configuration control board for test acceptance and configuration change request closeout. It was the duty of the customer software manager to ensure that all configuration change requests implemented in a new release had been closed before the release was flown.

Discrepancy Reporting and Corrective Action. Discrepancy reports were generated when an occurrence was not expected or was considered unsafe by cognizant personnel. Discrepancies could be unexpected test results or a problem uncovered during normal flight operation. They were submitted to and processed by the configuration control board in the same fashion as configuration change requests. Discrepancy reports were presented to the configuration control board when: (1) the discrepancy report was initially submitted for acceptance (agreement that it was in fact a discrepancy) and assignment of criticality; and (2) for closeout, after corrective action and testing had occurred, or when another justification existed for closeout.

Criticality was assigned to each discrepancy report based on the possible effect the discrepancy had on flight safety and research mission accomplishment: Criticality of 1, flight critical, indicates possible loss of life or the aircraft; criticality of 2, mission critical, indicates that a research objective could be lost; and criticality of 3 was assigned to those which fit neither criticality of 1 nor criticality of 2. Discrepancies with criticality of 1 had to be closed or (if the discrepancy was sufficiently improbable, difficult to fix, and able to be monitored) put on a project accepted risk list before the aircraft was flown. The accepted risk list was reviewed at a briefing prior to each flight. Lesser criticality software discrepancies did not prevent the aircraft from flying and were fixed at a convenient time. Mission critical discrepancies would keep the aircraft from flying if pilot procedures could not compensate for the problem.

Closeout of discrepancy reports was based on changes which had gone through the change control process and had been verified to correct the discrepancy. Discrepancy reports were also closed out because they were no longer applicable. For instance, reports of discrepancies occurring in a certain flight control mode were closed out when that mode was removed from the control system. If a discrepancy could not be corrected for any reason, that discrepancy report remained open.

There was a separate, but sometimes overlapping, discrepancy mechanism utilizing the aircraft logbook, which was used for routine maintenance and hardware replacement.

Software Assurance

Reviews. Software developer internal reviews continued as before in the operational phase. Prime contractor reviews of developer verification tests were ended. Flight readiness reviews of new software were generally included as part of the technical briefing held before the first flight with a new release. The technical briefing also included a description of changes and a summary of any open discrepancies.

Test Philosophy. The scope of the software developer verification tests was based on the method of production by which the modules were changed. As in the initial flight qualification of any new software release, the more system tests that were performed, the more the confidence in the system grew.

The customer always conducted a functional verification test on each change to close the CCR from which it was authorized. It was never assumed that the contents of a delivered object tape to be used for flight was correct. The tape contents were loaded into FCC memory. Memory was then verified to have the expected numbers (per documentation supplied) in the checksum locations, and the rest of memory was checksummed with the results compared to the expected checksums. This process verified that the object

tape media had not been altered during transit from the developer. After a bit-for-bit comparison to the previous software release, a second flight object tape was created from memory loaded with the delivered copy. This comparison verified (for most methods of software production) that the correct changes, and only the expected changes, were incorporated in the new release. The magnitude of validation testing, and the decision whether to perform independent V and V, were based on: (1) safety of flight implications of a design, specification, or implementation error in the changes; (2) the number and complexity of changes; and (3) the method of production.

Software Verification and Validation. The scope of the developer's verification testing ranged from module tests only for nonflight critical software changes, to module and full integration tests for flight critical changes.

Customer testing is illustrated in Fig. 7, Software/system verification and validation process. The first three steps shown in the figure were always done, followed sequentially by V and V tests based on method of production, optional validation steps based on safety implications and complexity of changes, a mandatory five hours of simulation by a project pilot using the new release, and ending with a document releasing the new software for flight. A brief explanation of the tests noted in Fig. 7 follows.

New Tape Verification Process was an object tape media test which assured the customer that the tape contents were the same as noted by the developer during tape recording. It also verified that the proper changes were implemented (except for tapes produced by relink-with-relocation). It included copying the original tape to have a second flightworthy tape for backup, pending successful completion of V and V testing. A nonflight simulation tape was recorded, in which a patch was included. This patch disabled instructions to write to FCC nonvolatile memory, which had hardware constraints on the number of writes before mandatory replacement. This was considered important since simulation computers could, after acceptance testing, be used on the flight vehicle. The second purpose of the patch was to disable monitoring of a nonvital-simulated sensor, the attitude heading reference system, when the simulation was not in operation. Without the patch, this simulated sensor was usually declared failed when the simulation was reset, causing operational problems and altering failure mode test results.

Standardized Test Matrix verified in a gross manner that the new software communicated with the FCC hardware over a large subset of flight computer inputs and outputs. Regardless of the method of production, each change was functionally tested to close its CCR, using the CCR and the PCN to define the test requirements. In all production methods, previous overlays were retested. Relink-with-relocation required retest of software coded in-line for the first time, and two steps to verify that the correct software changes were incorporated. The first step was to produce the new release at the customer's facility from a source tape received from the developer. The object code was bit-for-bit compared with the object tape received from the developer, which verified the integrity of the source tape. Source code from the verified tape was then compared with the source code of the last release. The differences verified that the proper changes were incorporated.

The change documentation lifecycle is complete; however validation testing of the release containing the change must be accomplished successfully before it can be used on the aircraft. Validation tests dependent on method of production included time histories and frequency responses, where relink-with-relocation required the full set. In practice, the full set of time histories and frequency responses was run on all new releases where time histories are required, since those tests have been partially automated.

Additional time histories and frequency responses have been required on releases with specific control law changes. The use of failure modes and effects tests, and the scope of the tests, depended on safety implications and complexity of the changes.

When all V and V was successfully completed, a software release document was prepared which authorized the new software release for flight. It described all changes incorporated in the new release, listed relevant CCRs, PCNs and system test reports (STRs), and briefly described tests accomplished by the developer and by the customer. All discrepancy reports generated during testing were listed with their criticality and their status (open or closed). The software release document was published as an X-29A internal document and distributed to project personnel. For each flight day, which may consist of several flights, a flight tape release form was signed by the customer software manager, the aircraft operations engineer, and an inspector. It contained flight numbers for which it was valid, the release identification of the software to be

used, the checksums, as well as statements that all required testing had been accomplished, that the tape was suitable for flight, and that the tape was loaded properly on the aircraft FCCs.

Independent Software Verification and System Validation. Three major control system changes since aircraft delivery resulted in independent V and V efforts. Those tests were devised and run by government personnel (when the V and V was done by the prime contractor for expanded envelope), by the prime contractor, and by non-NASA government personnel.

Parallel Activities

The change process was shortened by subcontractor and customer working in parallel. The greatest time saving was in software test. The software developer sent a preliminary object tape to the customer after partial testing, allowing customer V and V tests to be run in parallel with the remaining developer verification. When a problem requiring a second preliminary release occurred, full retest was generally not required at either the developer or customer facility; the practice of sending preliminary tapes saved time even if changes had to be made. At times, V and V could also be accomplished in parallel with an independent V and V effort.

The customer could produce a new object tape by overlay in parallel with the developer generating the same release by overlay or permanent relink-without-relocation. Integrity of the customer-generated tape was ensured by a bit-for-bit comparison with the tape received from the developer. Customer V and V could begin as soon as the procedure for overlay tape generation was complete. Changes of this type which also required minimum validation testing have been implemented, tested, and released for flight in less than one week.

Dryden Simulation Facility

Facility Description

One of the best investments that the X-29A program made was in the simulation capability at NASA Ames-Dryden Flight Research Facility, where the flight test program was conducted. The major cost impact was the hardware-in-the-loop simulation in which certain flying qualities can be evaluated, and where a large subset of the possible subsystem failures could be simulated using actual flight hardware and software. This capability enabled the customer to perform large-scale independent V and V testing prior to first flight and prior to expanded envelope flight, and to perform the V and V testing of changes made during the flight test program.

This simulation was also used to test preliminary software designs, and for pilot training. It included the simulation computer (equipped with a large number of discrete and analog inputs and outputs), three FCCs, a failure status control panel (an X-29A cockpit panel), an interface console (signal conditioning, and test points tied to all pins of the FCC signal connectors for all three channels), an analog actuator model (analog simulation of the surface actuators, including failure detection and operational modes), a SIBLINC (signal conditioning for simulated sensors), and the XAIDS (a microcomputer device which emulated a terminal into flight computer test hardware, and which displayed in engineering units the digital data outputted from the three flight computers for downlink).

All-FORTRAN real-time and batch simulations were used for control system checkout and to produce reference data against which flight software was validated. The real-time simulation required simulation cockpit hardware and limited simulation computer input and output interfaces; and the batch simulation required no external hardware. They were independent implementations of the same specifications from which the FCCs were coded.

Automated Testing

An initial capability of automated testing was resident in the simulation facility. It is envisioned that this capability will be greatly expanded when Dryden's Integrated Test Facility, now under construction, becomes operational.

The test capability included many useful features. Commands to the simulation program could be inputted from a test script and saved as a text file on disk rather than from a terminal. Each analog and discrete input and output of the simulation could assume three independent values, changing value at times read from an internal clock whose resolution was 12.5 μs. A sine wave sweep generator, whose beginning and ending frequencies and amplitudes were controllable, could be summed with any of the analog inputs or outputs. Using these capabilities, simulated switches could be operated, simulated sensors could be failed, and frequency response data could be obtained.

The X-29A program incorporated automated test features as they became available, and when hardware constraints allowed. Interface modifications to the

hardware-in-the-loop simulation were incorporated to maximize the ability of the simulation computer to control analog and discrete inputs to the flight control system hardware used in the simulation. The Standardized Test Matrix (the test of flight control computer inputs and outputs) and time histories and frequency responses, whose overplots to reference cases were used to determine correctness of control system operation, were partially automated. Failure modes and effects tests, in which simulated subsystem failures were introduced during real-time operation at known flight conditions, have not yet been automated; however, with the proposed incorporation of an outerloop autopilot, automation would become practical.

Reduction of test data was done on a computer other than the one used for real-time simulation. Data transfer between the simulation computer and the one used for data reduction is a technical problem currently in work. Ideally, one computer would control the test execution, data transfer, and data reduction in an automated sequence. This has been accomplished in a limited fashion, but the capability is not yet operational.

The verification of proper changes in releases produced by relink-with-relocation involved assembly and relink of source code, and source comparison with a previous release. These operations used the software production and source compare capabilities resident in the simulation facility.

A personal computer was used to speed the process of FCC hardware acceptance testing, by calculating pass/fail status, and recording data on disk. It was also programmed as a terminal emulator and front end to the FCC interface hardware, the SEU. In addition to the normal functions of a dumb terminal, a record of terminal inputs and outputs could be saved on disk, and software patches could be written, saved on disk, and sent to the flight computers.

Observations and Recommendations

The X-29A program was strictly a research effort; therefore, some of the problem areas were unique to that environment. Most were typical of problems which may be encountered in any critical software development and maintenance effort.

The organization having responsibility for hands-on operation and maintenance of a complex software system (that is, a user organization) should, from program inception, recognize several management and technical issues. Most of these issues were recognized by the X-29A program. Issues will be stated in the context of a generic program which has a software developer organization and a user organization.

All parties should agree to documentation and configuration management standards at the beginning so configuration control can be passed from the developer to the user on delivery. The X-29A situation of two numbering systems for PCNs, and for software releases, illustrates this problem.

In order to facilitate design and test activities, initial system requirements should include system test interfaces which provide access to hardware and software internal parameters. These parameters should be randomly accessible without interrupting real-time operation, and be displayed as either raw data or engineering units. The capability to stop the processor at will should be implemented, with multiple safeguards against inadvertent halts during normal operation. The test interfaces should have the capability of test automation.

Initial Flight Qualification Phase

It was reaffirmed during operational maintenance that some of the most helpful coding guidelines are as follows. Straightforward coding is more cost-effective than coding that is difficult to understand and to modify. This is true even if the unorthodox code uses hardware or software features that make it more efficient. Documentation which adds to the readability of the code (PDL) and meaningful comments is highly desirable. Variable names should be chosen with care and should reflect the variable's function.

Configuration management activity (change control and discrepancy reporting) at the developer facilities should be closely monitored by the user. Developer and user documentation should be compatible to allow smooth transition of responsibility from the developer to the user at delivery.

Technical personnel from the user organization should actively participate in developer system integration, and in V and V test activities. This was done with the X-29A program, and the knowledge gained was invaluable. User and developer should realize up front that integration and testing, with the design changes they will most certainly produce, consume more resources (time and money) than is usually allotted for them.

If the system is of a critical nature, an independent V and V should be considered. This concept was used

successfully in the X-29A program. It provides greatly increased confidence in the software system. Total independence infers a different organization generating and executing the tests using a different facility with a different implementation of the same simulated external environment. In practice, an independent V and V can be useful with some aspects of commonality with the V and V activities.

Operational Maintenance Phase

The following recommendations refer to the user's facility.

Skilled and knowledgeable technical people, including both user and developer personnel, should be assigned to the operational site and work together as a project team with a minimum of organizational barriers. Every effort should be made to have systems and software people who worked on the development continue at the customer site, and if applicable at the developer facility, during the operational maintenance period. Contractor personnel assigned to Dryden were valuable and respected members of the X-29A project team.

Rigorous configuration management is required, including a configuration control board on which project management and representatives from all disciplines and organizations of the technical team participate. Successful implementation of the configuration management process relies on a dedicated and knowledgeable configuration control board, working with project management who realize the importance of the process. An undocumented, but extremely important, function of the configuration control board is dissemination of information to the many disciplines involved in operation of the aircraft; not only the effects of changes, but knowledge of the deficiencies of airborne and ground support systems. All members of the project team should understand and be involved with the discrepancy reporting, investigation, and corrective action process. Care must be exercised in use of parallel discrepancy reporting and correction systems, such as the aircraft workbook on the X-29A project. A "routine" maintenance item, such as a throttle lever adjustment, may impact operation of a control mode, such as precision approach control (PAC). As part of the change control process, the user organization should have and exert full authority to direct the software implementation process to ensure compliance with the decisions of the configuration control board. This implies direct control over the developer organization. The user on the X-29A program had no direct contractual ties to the developer, only with the prime contractor. At times, the configuration management process was compromised by the prime contractor modifying or delaying implementation of approved changes.

Facilities for software system V and V should be provided at the user site, utilizing as a minimum the actual computer hardware and software. The test facility should provide individual control of system inputs, and capability of monitoring and recording in engineering units system inputs and outputs. It should be designed with the maximum automation allowed by the system under test. In the case of a flight control system, this requirement is met by a hardware-in-the-loop closed-loop simulation in which all flight computer inputs are individually controlled and outputs go to actual or simulated flight hardware. Simulation constraints should not force modifications of flight code to enable testing. From a technical viewpoint, the long-term gains from such a facility far exceed the cost of its implementation and maintenance. Most of these criteria were met by Dryden's X-29A hardware-in-the-loop simulation. Expanded capability will be available when Dryden's Integrated Test Facility becomes operational. System test capability will range from hardware-in-the-loop simulation (with additional automated test capability) to aircraft-in-the-loop (iron bird) simulation. The six-degree-of-freedom iron bird simulation will monitor actual control surface positions and simulated engine thrust to provide to the aircraft systems simulated sensor information, which in some tests is summed with the outputs of aircraft sensors. Flying qualities, with and without failures inserted, can be evaluated with the maximum amount of flight hardware and software active.

The user should obtain the capability to develop software in-house. If practical, software development should transition to the user's facility at delivery, with on-site developer and user personnel sharing the responsibility for development. If software development is done off-site, source code should be transmitted to the user on a medium which can be utilized by both the developer and the user. The software development capability is then required as part of the verification that the proper software changes were incorporated in a new software release.

If unnecessary features exist in critical software, it may be cost-effective to eliminate those features. A flight control mode, DR, was removed from the X-29A

control system. The benefits of the DR mode were overshadowed by the cost in V and V test resources required to retest it at each new software release.

Late in the flight program on the first X-29A, it was discovered during validation testing that at certain flight conditions a within-tolerance air data failure (at those conditions a failure to null) caused the aircraft to become unstable. This pointed out a deficiency in the test philosophy up to that time. Previously, the simulated aircraft flying qualities had been tested under nominal conditions, and with single and multiple subsystem failures to null or hardover. It became obvious that a series of tests with subsystem outputs in error by just under the failure tolerance should be required. This type of testing has since been implemented on the X-29A aircraft, and is recommended for other applicable systems.

Concluding Remarks

The X-29A program illustrates that a complex, software-intensive, software-critical system can be successfully developed, flight qualified, and maintained under less than ideal programmatic conditions. Multiple organizations with diverse backgrounds were intimately involved in the development and change processes. The limited manpower in the software area during development was further reduced during operational maintenance.

All participants in software system development and maintenance recognized the importance of the following software assurance processes: software design standards, reviews, change control, discrepancy reporting and correction, and verification and validation testing. The software system test facility at the user facility was utilized in the development of design specifications as well as for verification and validation testing.

Tools used by and developed for the X-29A program reduced the probability of error and increased the productivity of software development and test personnel. The autotest capability proved to be very useful in shortening the time required to complete numerous tests which were always repeated following a control system change. The lack of a centralized project database to store the vast amounts of information associated with complex software-critical systems hindered both the development and maintenance phases.

The success of the X-29A software system is the result of several factors: Overall good cooperation of the prime contractor and the software developer with NASA Ames-Dryden Flight Research Facility, the user organization; the technical expertise of the control law designers and the software developer; the technical excellence and teamwork exhibited by the various contractor and customer software and systems personnel assigned to Dryden; the high-quality software validation test facilities at the prime contractor facility and at Dryden; project management support of the configuration management process; and project management support of the software test team.

References

[1] Chacon, V., and McBride, D., *Operational Viewpoint of the X-29A Digital Flight Control System*, NASA TM-100434.

[2] Honeywell, Inc., Military Avionics Division, *Machine Language Program, Forward Swept Wing* (FSW Program Performance Specification), DS–Pt. 1, rev. Aug. 31, 1987.

Table 1. The X-29A flight control modes.

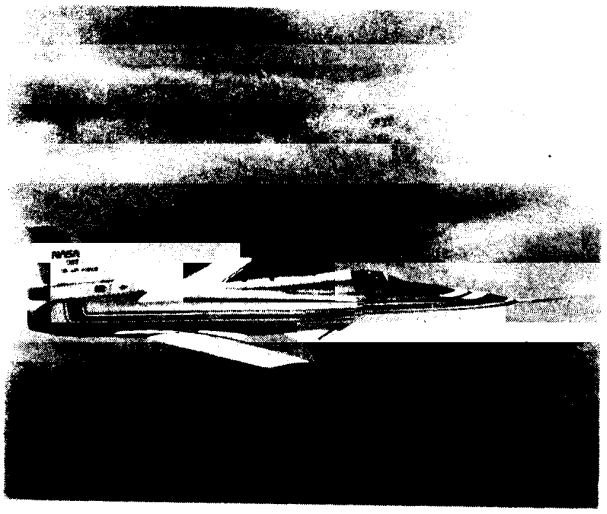
Control modes	Option	Function	Description
Normal		Normal flight	Variable gains based on airspeed, altitude, Mach number, angle of attack. Digital implementation. Pilot selectable.
	ACC		Automatic camber control (ACC) continuously trimmed flaps to maintain optimum performance. Pilot selectable.
	MCC		Manual camber control (MCC) positioned the flaps at discrete positions and held them fixed as long as the canard was operating within its acceptable range. Pilot selectable.
	Speed stability		A pilot aid to maintain constant airspeed. Automatic below a defined airspeed, pilot selectable above.
	Degraded modes		Adaptations of the normal mode control system to loss of certain sensors. Automatic.
Normal PA		Takeoff and landing	Power approach (PA) gains. Automatic when pilot selects landing configuration.
Precision approach control		Precision landing	Autothrottle mode which controlled airspeed, allowing the pilot to control flight path angle with the pitch stick. Pilot selectable in normal.
Digital reversion		Normal flight	A backup mode using reduced sensor complement. Pilot selectable and automatic downmode from normal with certain failures.
	UA		Up-and-away (UA) gains. Pilot selectable.
	PA		PA gains. Pilot selectable.
Direct electrical link		Ground operation	Control system mode automatically selected when weight was on any wheel.
Analog reversion		Analog backup	Backup control system implemented in analog hardware (the only mode not implemented using software). Pilot selectable or automatic downmode from digital modes with certain failures.
	UA		UA control system with gains scheduled on impact pressure. Pilot selectable.
	PA		PA control system with fixed gains. Pilot selectable.

Table 2. Tools used during software development.

Tool	Phase used in	Description
5301 assembler	Development and maintenance	Assembles 5301 source code.
5301 link editor	Development and maintenance	Links 5301 object code to create flight code.
Downline load tool	Development and maintenance	Transfers flight code from μVAX to flight computer.
Workstation tools	Development and maintenance	DEC Control Language (DCL) programs to facilitate human interface with 5301 assembler and link editor on the μVAX II workstation.
Data dictionary tools	Development and maintenance	These tools helped create and maintain a listing of all the control law processor variables. Units, scale factors, data types, and description.
Data integrity tool	Development and maintenance	This tool traced program execution in the source code and verified processor mode (floating point, single precision, double precision, etc. ...) before execution of a memory reference instruction.
Overflow analysis tool	Development and maintenance	Tool to look at adjacent entries in gain tables to verify that the difference between them would not cause an overflow in the processor.
Set/use table	Development and maintenance	Listing of which module sets and/or uses a variable.
Bender chart	Development and maintenance	Graphical display of processor throughput and worst case module execution time in μs.
Memory map tool	Development and maintenance	Generates listing of all variables and their addresses.
Timing tool	Development and maintenance	Computes path execution time using source code.
Spare memory tool	Maintenance	Looks at assembled version of code and determines locations of spare memory.
Checksum program	Maintenance	Computes checksums of linked flight code to compare to the checksums generated by the SEU. The SEU is the interface to the flight control computers.

Table 2. Concluded.

Tool	Phase used in	Description
Configuration control tool	Maintenance	An online configuration management tool to safeguard flight code from accidental updates, etc. . . . Acts as a "librarian" to the flight code modules.
Function simulation (FSIM)	Development and maintenance	Generates phase and gain values at discrete frequencies given elements in a software path.
SEU interface utility	Maintenance	Macintosh program for a front end to the SEU.
Overplot tool	Maintenance	Generates control files for the plotting program to overplot 5301 and FORTRAN time history runs.
Autotest capability	Maintenance	Automatically collects time history and frequency response data for plotting. Automatically generates sensor failures to check redundancy management.
XAIDS	Maintenance	Extended aircraft interrogation and display system. User interface to the SEU. Also converts ARINC 429 bus data to 1553 data for use by simulation computer.
Source comparison tool	Maintenance	Compares two large source files and prints differences.
ASCII file tools	Development and maintenance	Tool to take out tabs, put in tabs, various other ASCII file manipulations to facilitate printing, archiving (on tape), and editing of large files.
Tape reading tools	Maintenance	Unpacks and sorts Honeywell source code tapes.



EC 88-0093-009

Fig. 1 The X-29A airplane.

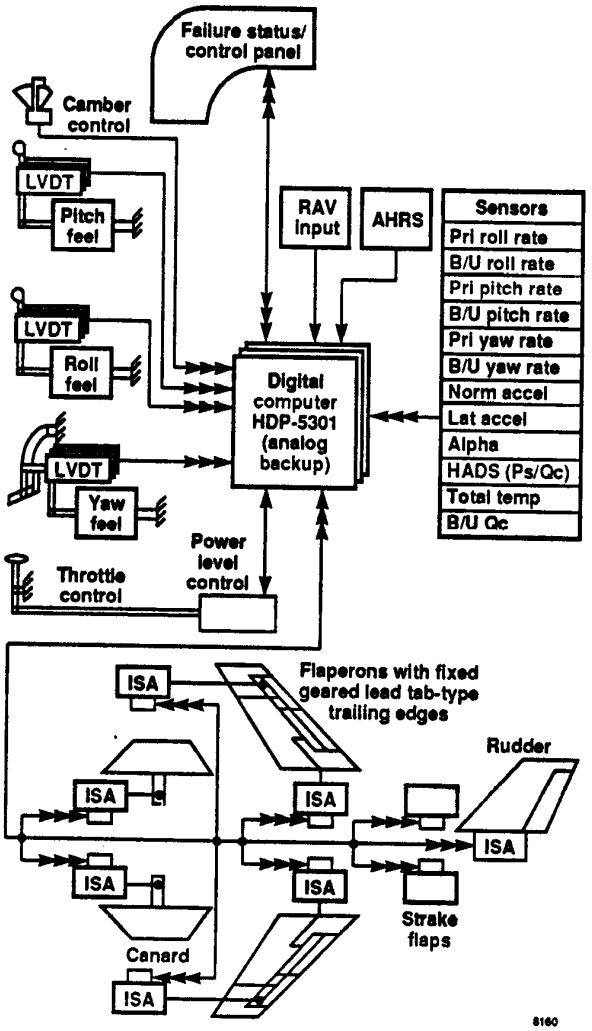


Fig. 2 The X-29A flight control system hardware.

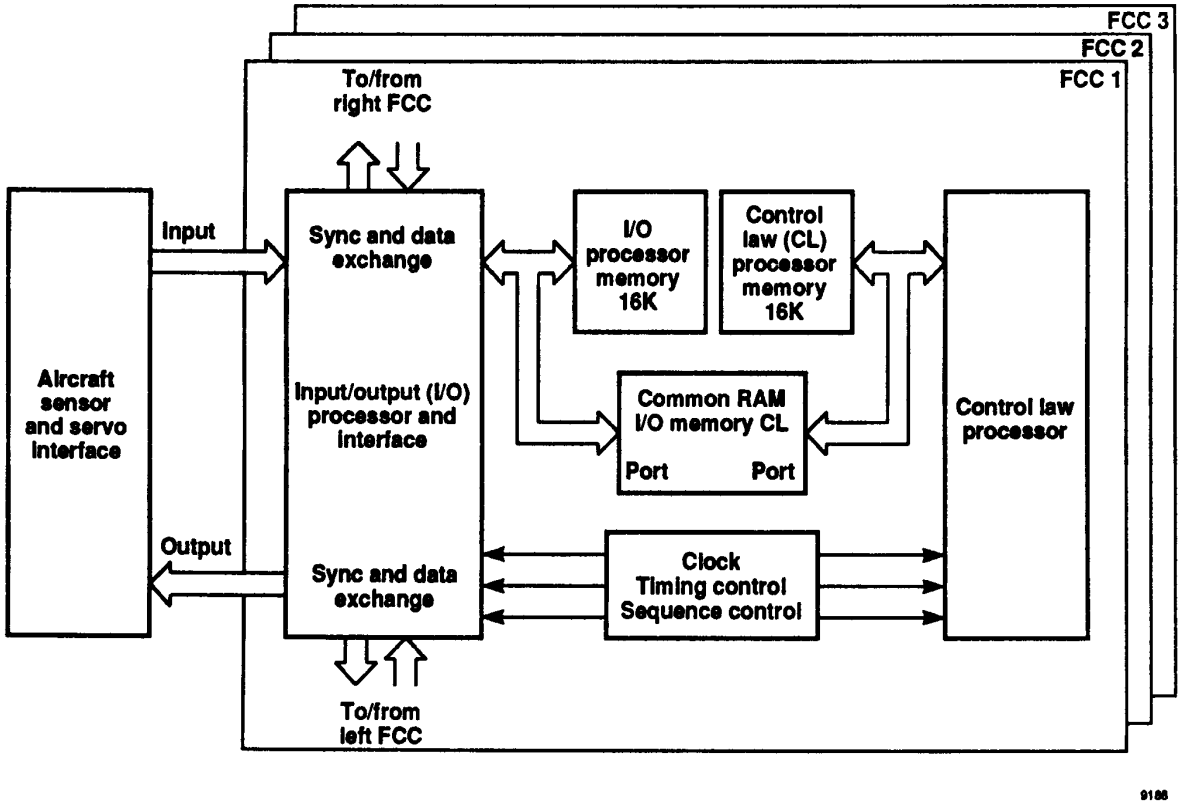


Fig. 3 FCC block diagram.

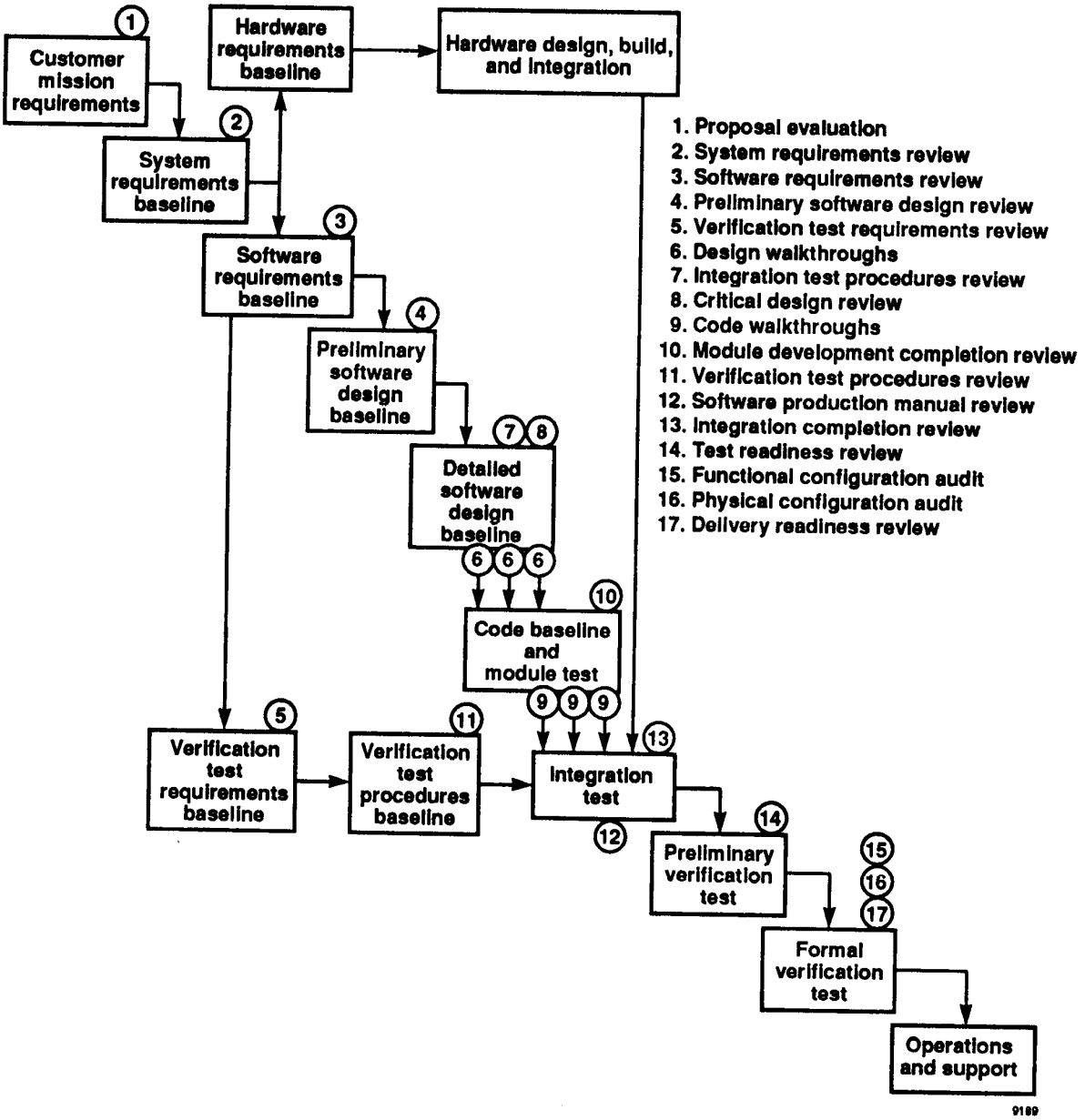


Fig. 4 Honeywell software development process and reviews.

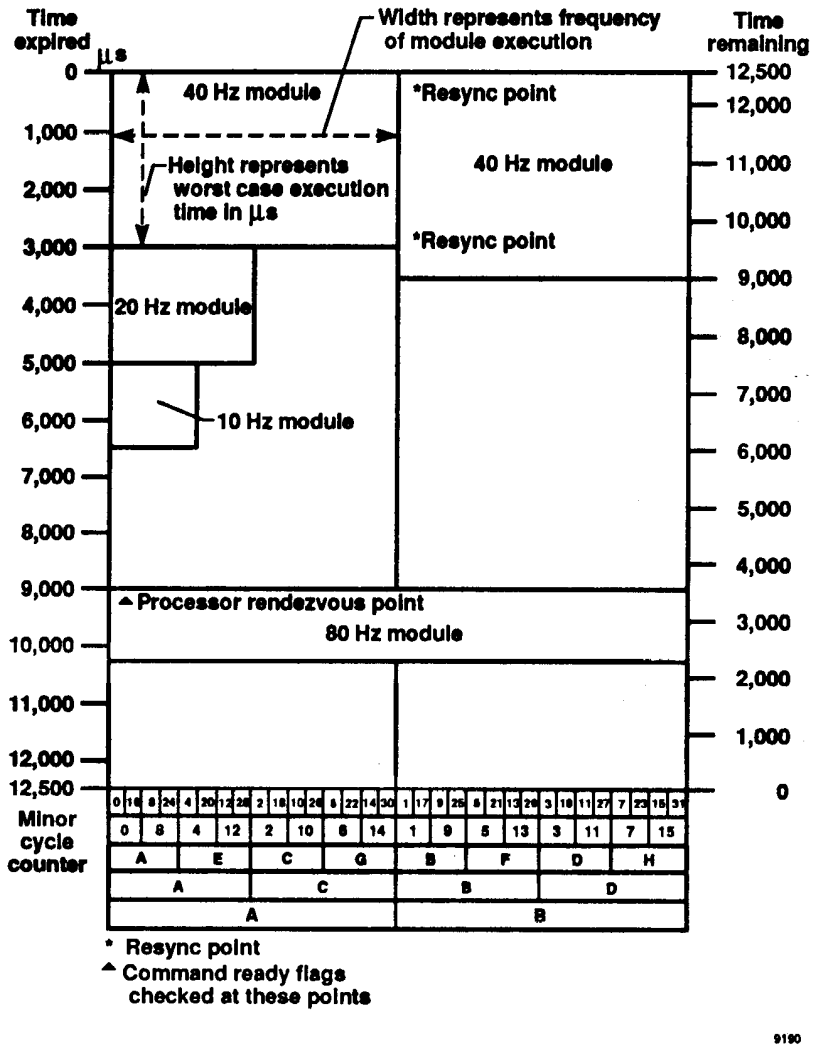


Fig. 5 Example of Bender chart.

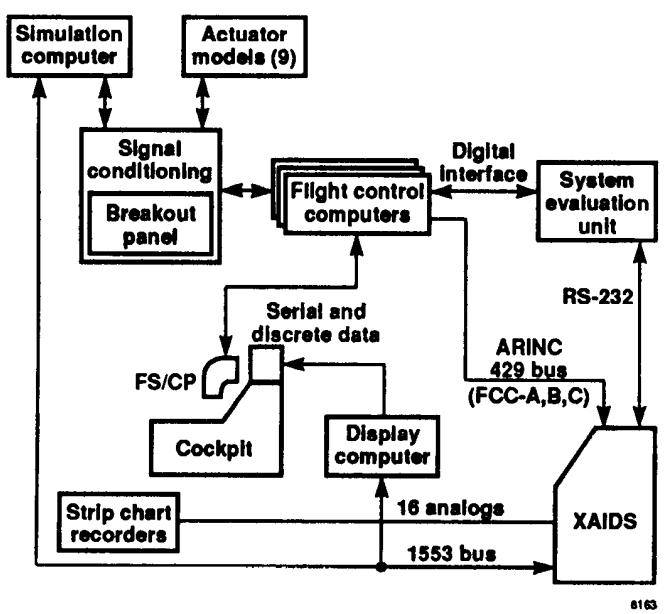


Fig. 6 The X-29A hardware-in-the-loop simulation.

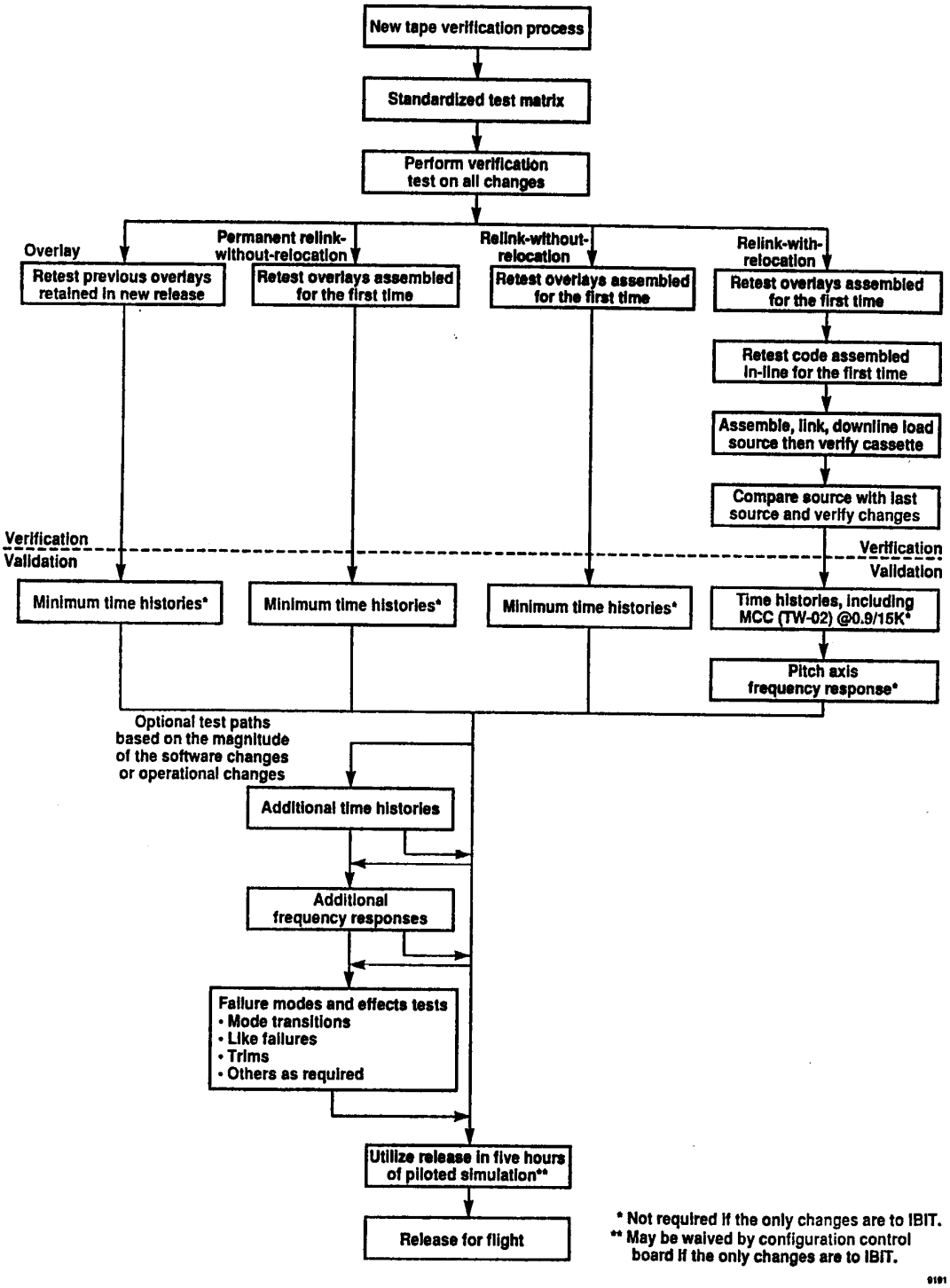


Fig. 7 Software/system verification and validation process.

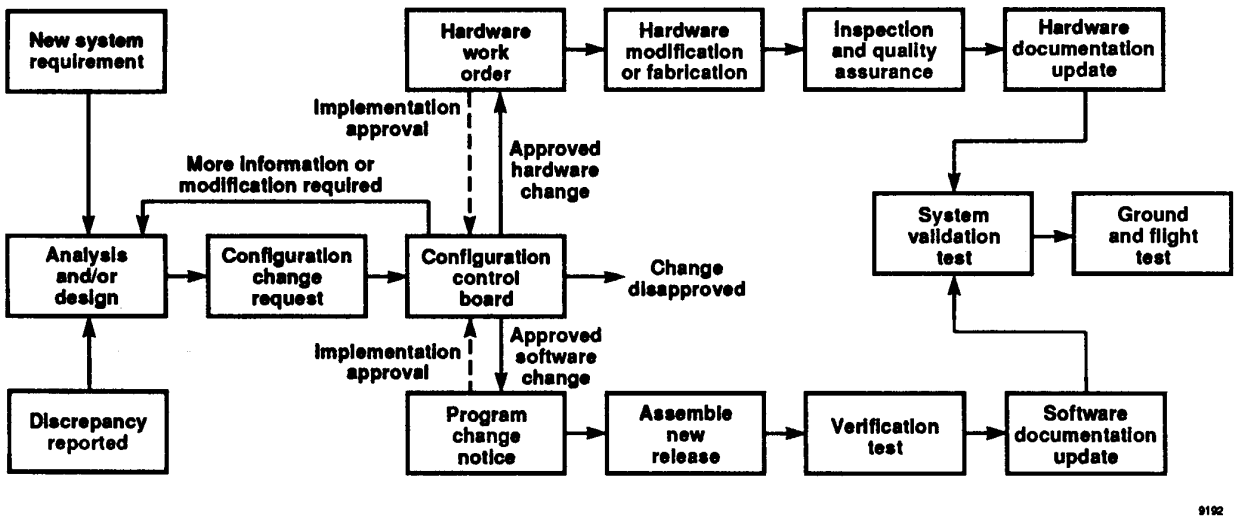


Fig. 8 Change management process.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-101703	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Initial Flight Qualification and Operational Maintenance of X-29A Flight Software		5. Report Date September 1989 6. Performing Organization Code
7. Author(s) Michael R. Earls and Joel R. Sitz		8. Performing Organization Report No. H-1558 10. Work Unit No. RTOP 533-02-51
9. Performing Organization Name and Address NASA Ames Research Center Dryden Flight Research Facility P.O. Box 273, Edwards, CA 93523-5000		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum 14. Sponsoring Agency Code

15. Supplementary Notes

Prepared as AIAA 89-3596 for presentation at AIAA Guidance, Navigation, and Control Conference, Boston, Massachusetts, August 14–16, 1989.

16. Abstract

This paper is predominantly a nontechnical discussion of some significant aspects of the initial flight qualification and operational maintenance of the flight control system software for the X-29A technology demonstrator. Flight qualification and maintenance of complex, embedded flight control system software poses unique problems. The X-29A technology demonstrator aircraft has a digital flight control system which incorporates functions generally considered too complex for analog systems. Organizational responsibilities, software assurance issues, tools, and facilities are discussed.

17. Key Words (Suggested by Author(s)) Software verification and validation Software assurance Digital flight control system development Digital flight control system maintenance		18. Distribution Statement Unclassified — Unlimited Subject category 05	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 27	22. Price A03

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, VA 22161-2171.