IN-17
156304
p. 23

NASA Technical Memorandum 104757

Risk Management in Fly-by-Wire Systems

Karyn T. Knoll

March 1993

NASA

(NASA-TM-104757) RISK MANAGEMENT IN FLY-BY-WIRE SYSTEMS (NASA) 23 p

N93-22703

Unclas

G3/17 0156304

NASA Technical Memorandum 104757

Risk Management in Fly-by-Wire Systems

Karyn T. Knoll
Lyndon B. Johnson Space Center
Houston, Texas

March 1993

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

CONTENTS

Section	Page
Abstract	1
Introduction	1
Description of Fly-by-Wire Systems	1
Risks Inherent in Fly-by-Wire Systems	2
Risk Management in the Fly-by-Wire Industry	5
Configuration Control	5
Verification and Validation	6
Tools	7
Backup Flight Control Systems	9
Risk Management and the Space Shuttle Program	12
References	17

TABLE

Table		Page
1	Flight Control System Summary	15

PRECEDING PAGE BLANK NOT FILMED

Abstract

A general description of various types of fly-by-wire systems is provided. The risks inherent in digital flight control systems, like the Space Shuttle, are identified. The results of a literature survey examining risk management methods in use throughout the aerospace industry are presented. The applicability of these methods to the Space Shuttle program is discussed.

Introduction

Since the development of the Space Shuttle, many other aerospace vehicles have incorporated fly-by-wire technologies in their flight control systems in an effort to improve performance, efficiency, and reliability. Because the flight control system is a critical component of any aerospace vehicle, it is especially important that the risks inherent in using fly-by-wire technologies are thoroughly understood and carefully managed. This paper examines the findings and techniques of other aerospace vehicle programs that have incorporated fly-by-wire technologies in their vehicles and discusses the implications of their experiences to the Space Shuttle program.

Description of Fly-by-Wire Systems

Conventional aircraft use mechanical (sometimes supplemented by hydraulic) linkages to connect the pilot's controls to the aircraft control surfaces. This arrangement provides a direct correlation between the pilot's actions and the behavior of the control surfaces. In a fly-by-wire system, these linkages are replaced by some form of electrical control system containing the aircraft's control laws and flight characteristics. Using this logic, the control system combines inputs from the pilot's controls with those from other aircraft sensory devices in order to produce commands to the control surfaces.

Fly-by-wire systems are generally used in military fighters to make the aircraft more maneuverable. More accurately, the fly-by-wire system enables the use of an unstable airframe providing greater maneuverability. Such an aircraft requires a computer to make adjustments fast enough to counter the airframe's natural instability and keep the aircraft flyable. In transport aircraft, fly-by-wire systems are used to improve fuel efficiency, ride comfort, and safety. These aircraft are often flyable in the event of a loss of the control system, though some require a backup system to provide the connection between the pilot's controls and the aircraft control surfaces needed to enable direct control similar to that of a conventional aircraft. In the case of the Space Shuttle, a fly-by-wire system keeps the vehicle within the correct flight profiles to enable it to reach predetermined targets without exceeding any vehicle limitations.

Fly-by-wire systems are either analog or digital. Analog systems combine inputs from the pilot's controls and from the vehicle's sensors in an analog (using physical, rather than numerical, parameters to carry out its calculations) manner in order to produce commands to the control surfaces. Analog systems tend to be limited in the complexity of the control problems they are capable of handling as well as being inflexible in the breadth of mission objectives they are capable of supporting. In digital fly-by-wire systems, inputs from the pilot's controls and the vehicle's sensors are transformed into

discrete signals which are sent to a digital computer. The flight computer numerically manipulates these signals in accordance with its programming in order to produce output commands. These discrete outputs are transformed into analog signals understandable to the vehicle control surfaces. Because the software in its flight computer can be reprogrammed, a digital flight control system offers a great deal of flexibility in meeting mission objectives. Both the primary and backup flight control systems on the Space Shuttle are digital. For this reason, this paper focuses on concerns pertinent to digital flight control systems.

There are two architectures of digital flight control systems in use today–synchronous and asynchronous. Synchronous systems use a synchronization signal to trigger the execution of specific software tasks. This feature makes it particularly easy to know exactly when each task is running, and adds a degree of predictability to the flight control system that makes it relatively easy to test and analyze. The major disadvantage of synchronous control systems is that this rigid timing structure can be unforgiving to tasks that overrun their allotted execution time or to a large influx of unscheduled tasks, such as inputs from the crew or error handling requirements. Such events can confuse a synchronous control system and cause it to stop functioning. Asynchronous systems assign priorities to tasks and schedule them to run at specific times. With "repeat every" time clauses attached to tasks that need to run cyclically, these clauses reset the scheduled time for the next time the task needs to run. When it is time for a task to run, a check is performed to determine whether or not anything else is running in the CPU and, if so, whether the scheduled task or the running task has the higher priority. The task with the higher priority is given control of the CPU and the other task is allowed to run once it is finished (providing no task with a higher priority comes along in the meantime). Tasks in an asynchronous system may be interrupted by higher priority tasks. Following execution of the higher priority task, an interrupted task picks up from where it was at the time of the interruption. In the case of an overload, low priority tasks may not get to run, but at least their request does not bring down the system. (Asynchronous systems know how to "just say no.") The resiliency of asynchronous systems makes them somewhat less predictable and, consequently, more difficult to test and analyze. (Not surprisingly, the system designed to handle the greatest amount of unpredictability in its environment is the one which runs most unpredictably. This is a consequence of the system's ability to adjust to the conditions of its environment.)

Risks Inherent in Fly-by-Wire Systems

The flight control system is a highly critical component of a fly-by-wire system. To keep the vehicle flying safely (or, in most cases, flying at all), it is crucial that both the hardware and software composing the system keep functioning properly. The types of malfunctions to which a digital flight control system is susceptible may be considered as falling into two broad categories: operational failures and the infamous generic software failures.

Operational failures refer to cases in which properly designed and coded software fails to run correctly during operation of the vehicle either because of a hardware failure in the flight computer or because of some type of accident corrupting the software. All aircraft systems, whether hardware or software, are susceptible to this type of failure.

Flight control systems guard against this type of failure in a manner common to their hardware counterparts by providing redundancy. In the case of a flight control system, this generally means identical computer units running identical copies of the software. The idea here is that, even if one or more unit fails, the remaining units will be able to provide the required functionality.

Generic software errors refer to errors in the software code itself. Because this type of error is reproduced in all redundant units, it is of special concern to fly-by-wire system designers and much effort is spent in attempts to detect and guard against it.

In the case of a serious generic software error, the flight control system may respond so erratically that the vehicle is driven into a state in which the system is unable to control the vehicle (as in the case of an erroneous gain causing the system to diverge into a loss of control state), or the flight control system may stop processing tasks altogether (as in the case of an error that causes the operating system's timer to be inaccurately set).

Generic software errors can have a variety of causes. One form of generic software error is the requirements error in which the requirements defined for the system do not adequately reflect the actual environment encountered in flight. Another form of generic software error is the implementation error in which the software does not meet requirements. This type of error may be caused by the choice of an incorrect algorithm to meet the design requirements, a coding error, or a compiler error which causes the code not to get implemented as written. Implementation errors are generally the easiest to find, yielding most readily to analytical, auditing, and testing techniques.

Perhaps the most insidious form of generic software error is the system-level error in which it is not so much a specific error in any single component, but rather the interactions between the components of and/or the inputs to the flight control system that cause a malfunction to occur. System-level errors may be considered as falling into two broad categories:

- Timing independent errors, in which a combination of inputs and/or environmental conditions that was not anticipated by the system designers and testers results in a malfunction; these errors are observable whenever the specific combination of inputs and conditions is encountered.

- Timing dependent errors, in which an anticipated combination of inputs and conditions, occurring in a specific timing window, produces an unexpected result. (Note: Since the Space Shuttle flight control system uses a minor cycle of 40 ms, this system could be considered to possess 25 timing windows per second; one begins to see the impossibility of testing all combinations of inputs in all timing windows). Timing related errors are generally the most difficult to reproduce and, therefore, are the most resistant to testing; the shear magnitude of possibilities for the occurrence of timing errors makes them equally resistant to analytical techniques. It is also possible to experience errors caused by a combination of inputs occurring in a specific timing window which are not anticipated. Such a case would be a subset of timing dependent errors since their detection and analysis would be subject to the anomalies associated with this type of error.

Essentially all of the generic software errors encountered in this literature survey had system-level causes. Examples include:

- X-29 Air Data System Single Point Failure: After more than 200 flights, a serious design error was detected in the redundancy management logic of the X-29 Research Aircraft which had been present since its 38th flight. Simulations showed that a single-point failure (the loss of an air probe) would have caused the system to apply an incorrect gain resulting in the aircraft becoming unstable and departing stable flight. Fortunately, this mishap did not occur in flight, but for over 162 flights the aircraft was in danger of being lost if the loss of an air probe would have occurred. The X-29 Research Aircraft was grounded until its software was fixed (Mackall, 1989).

- HiMAT Gear Deployment Anomaly: A timing change in the ground software supporting the HiMAT Remotely Piloted Research Vehicle, coupled with the failure of one uplink receiver, resulted in the landing of the aircraft with its landing skids retracted. The timing change had been thoroughly tested in simulations under nominal conditions, but the loss of the uplink receiver was not anticipated and had an undesirable result (Mackall, 1989).

- AFTI/F-16 Flight 44 Anomaly: According to this NASA TM, "The interaction of many design characteristics and a unique flight condition resulted in the divergence of the three computers' output commands to the control surfaces causing the redundancy management software to declare all three flight computers failed" (MacKall, 1989). When the pilot engaged full rudder while slowing the aircraft down, the system's triplex asynchronous sensor architecture caused the aircraft's three sensors to differ enough in the values they sensed that the redundancy management software was unable to handle the divergence. This condition caused each flight computer to declare the other two failed and to independently send its own outputs to the control surfaces. Additional redundancy management in the actuators resulted in the hardware selecting Channel C, the third of three redundant channels, for operational purposes. This allowed the pilot to land the aircraft in a simplex mode, although without knowing the real condition of the flight control system at the time. (In debriefing, the pilot stated that he flew in with his finger over the backup control system's engage button, ready to engage if the aircraft were to start to lose control.) In the post-flight investigation it was decided that if this condition were ever encountered on another flight, the future procedure would be to engage the backup system since the simplex mode landing had placed the aircraft in danger of being lost if Channel C failed.

- JAS-39 Gripen Crash: In a January 1989 issue of Aviation Week and Space Technology, a two page advertisement for the JAS-39 Gripen contains a quote by the SAAB aircraft test pilot, "On the simulator, we have pushed the aircraft to the edge. At the beginning we crashed the aircraft occasionally to find the limits to which the aircraft could be taken while we were on the ground, but having found and checked out the borderlines in the simulator has given me great confidence in the safety of the aircraft. Personally, I felt I knew exactly what the Gripen would be like to fly" (Holmstrom, 1989). On February 2, 1989, the Gripen flipped over on landing and crashed with its canopy side down as the result of a control stick input

that the flight software was unable to handle. The cause of the crash was ruled to be a software error that eventually delayed the program for eighteen months.

The next section examines some of the techniques employed to prevent, detect, and correct errors in fly-by-wire systems. The final section then examines the applicability of these techniques to risk management of the Space Shuttle flight control system software.

Risk Management in the Fly-by-Wire Industry

There appears to be a universal agreement among the sources surveyed that system complexity is the biggest driver influencing the risk associated with the development and operation of a fly-by-wire system. Complexity increases both the amount of local problems (i.e., problems isolated within a single component) likely to be embedded in the system (this appears to be a linear effect) and the amount of interaction occurring between the components composing the system (this appears to be an exponential effect).

The complexity of a program is determined by a combination of the complexity of the particular system for which the program is responsible and the management of the program. To a certain extent, the requirements of the system will limit the minimum amount of complexity achievable by the program. Program management will either achieve or increase this complexity level as a byproduct of its own ability to focus as it defines and implements program goals.

Some of the major techniques and tools employed in the management of the risk (which includes the activity of determining the amount of risk acceptable to the program based on program goals and resources) and the handling of risk-producing factors (such as the generic software errors discussed above) associated with fly-by-wire systems are presented below.

Configuration Control

Because of the number of variables involved in the development and operation of real-time avionics systems, even the testing of well-designed software is a monumental task. It is therefore no surprise that there is a widely held tenet in the fly-by-wire industry that no amount of testing can make up for a poor design process. For this reason, a good configuration control process appears to be essential to the production and maintenance of quality software.

The process of developing and modifying baselined documents by means of board decisions on change requests (CRs) and discrepancy reports (DRs) appears to be the standard for a configuration control process. Changing the functionality of the software and fixing discrepancies discovered in the software both involve modifying the software in a manner that requires the performance of the same impact analyses, implementation activities, and testing. Therefore, a single configuration control process, managed by a single board, is used to handle both fixes and new functionality changes to the software of a flight control system. The need to manage flight control software as an integrated system over its entire life cycle appears to far

outweigh any considerations related to whether a particular modification is being made to fix a discrepancy (often an operations phase function) or to add new functionality (technically, a development phase function) to the vehicle.

Well-structured documentation, clear and consistent accountabilities associated with the analysis, implementation, and verification responsibilities involved in making changes to the system's software, and good (formal) communication between all of the parties involved in the process are considered to be the essential elements of a reliable configuration control process.

During the operational phase of a program, the configuration control process generally takes on an additional significance since, at this time, it seems to be an almost universal practice to have configuration management replace any system engineering process that may have been in existence. Rather than treating system-wide analysis and integration as a distinct function for which a designated organization, the system engineering and integration office, is responsible, the system-level evaluation to a proposed software modification is now accomplished by distributing copies of the CRs/DRs to the managers responsible for each component of the system. These managers then assess the impact of the proposed change on their individual components. Interestingly, one memorandum found that flight system programs sometimes substitute this component-oriented approach to system-level analysis and integration for the distinct system engineering function even during the development phase of the vehicle, although almost always to the detriment of the overall program. The memorandum concludes, "A system designed as an integrated system, including all interactions and interfaces, has a reduced level of difficulties in testing and operation" (Regenie, 1986). Because system-level problems are the nemesis of software maintenance, attention given to the methods employed to perform system-level analyses seems to be beneficial at every phase of a program's life cycle.

Verification and Validation

Verification, the process of assuring that the integrated software meets specifications, and validation, the process of assuring that the software, when integrated into the flight system, cause the system to function as required, and are extremely important activities used to control the risks associated with fly-by wire systems. Only flight test itself is capable of providing a more direct indication of just how well the system will operate and, until the system has been thoroughly verified and validated, flight testing is too risky a venture. Additionally, verification and validation offer greater opportunities for data gathering and software execution analysis than does an actual flight test.

Integrated software with hardware simulations, which are analogous to those performed in the Space Shuttle program's Shuttle Avionics Integration Laboratory (SAIL), are considered essential to the validation of fly-by-wire systems. Most programs use some type of "aircraft in the loop" simulator in order to provide this capability. Although such a facility is expensive to construct and maintain, being as one source put it, "more like an airplane than like a simulator," it is generally felt that the fidelity provided by this type of simulation enables the identification and resolution of problems to an extent that justifies its expense (Szalai, 1978). This type of simulation has become so

standard in some segments of the fly-by-wire industry that the Ames Research Center's Dryden Flight Research Facility (ARCDFRF) has constructed a generic Integrated Test Facility which provides the core simulation capabilities and interface functions required to turn any of the ARCDFRF's research aircraft into a simulator.

The X-29 Research Aircraft program found it useful to develop a verification chart which provides the program with an indication of the type and amount of testing required to support any modifications to the software. Possible types of software modifications ranged from simply changing the value of a constant to modifying the code to an extent that required movement of surrounding code. The X-29 program chart attempted to identify the minimum amount of verification needed to insure confidence in the reliability of the software system in light of the risk associated with each type of modification.

It is widely accepted that the complete verification and validation of real-time software systems such as those found in fly-by-wire systems is impossible. The number of possible combinations of inputs to a system, especially when combined with the number of possible timing windows, is too numerous to be identifiable. Additionally, the time required to verify all of the identified test cases is prohibitive. The X-29 Research Aircraft (which has a flight control system approximately one-tenth the size of the Space Shuttle's) has approximately 90 inputs. To verify normal, null, and extreme failure instances of each input would require 2.49x10e38 years at fifteen minutes per test case (Sitz, 1990).

Although the impossibility of completely verifying a flight control system is well understood by the fly-by-wire community, the accepted folk wisdom in the industry prescribes testing as many instances as possible. For this reason, testing tools and techniques designed to automate, or at least make more "user friendly," everything from the set up of test cases to the collection and analysis of test data are growing in popularity. This trend is also being promoted by the development of new technology capable of supporting these new ways of doing business.

Tools

Advances in computer technology have greatly influenced fly-by-wire systems both by increasing the functionality which the flight control system itself is capable of providing, thereby leading to the design of more complex systems, and by increasing the number and type of tools available to aid in the development and maintenance of these systems. Tools to support the entire spectrum of activity associated with a fly-by-wire program are increasingly available.

Categories of engineering tools include:

- Software development tools (code editors, compilers, link editors, and so forth),
- Quality assurance tools (limit checkers, data integrity tools, comparers, and so forth) which serve to assure that the code meets specific criteria, and
- Analysis tools (debuggers, memory mappers, timing tools, and so forth) which provide engineers/analysts with insight into the structure and execution of the code.

Automation is often used wherever tasks of a well-defined, repetitive nature are identified. Many of the tasks associated with the production of the software products and with the performance and analysis of standard verification test cases are automatable. The operational phase of any flight program, when the vehicle has been sufficiently well-defined to enable configuration of any supporting tools, is especially ripe with opportunities for automation.

Artificial intelligence (AI) tools, intended to provide flight control system engineers with expert knowledge in computerized form, are also beginning to appear. Articles describing the benefits of AI tools tended to be written by the tool developers themselves, whereas sources describing the implementation of specific fly-by-wire systems stressed the usefulness of analytical tools which helped provide them with insight into the structure and operation of the flight software. Participants in the X-29A Research Aircraft program state, "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" (Duke, 1986). This situation may be due, at least in part, to the relative newness of AI tools. The rightful place of these tools, based on the contributions they are capable of making in a real program environment, will most likely be unfolding over the next several years.

Although the advantages tools offer in the development and maintenance of fly-by-wire systems are incontestable, it must be realized that, like all software-intensive devices, tools bring their own set of problems and concerns to the program employing them. Some of these are discussed below.

Tools require their own development, validation, and maintenance efforts. With major tools, such as flight simulators (which are one form of verification and validation tool), these efforts can become programs in and of themselves requiring their own set of management and programmatic resources. The responsibilities, resources, and interfaces required to support the tools used by a fly-by-wire system can require almost as much attention as the flight system itself.

Additionally, since tools are software systems, it must be remembered that they are subject to many of the same kinds of errors as the flight software. This additional potential source of error must be accounted for whenever troubleshooting errors in the system. In order to realistically scope the resource requirements of a fly-by-wire system, it is necessary to realize that a certain amount of the program's resources will be used to deal with problems in the tools as well as in the flight system itself (including the manpower that will be used in determining whether a particular problem was caused by an error in the flight software or by an anomaly in the tool).

Certain tools, most notably compilers and cross-compilers, represent an additional risk in that they manipulate the flight software and, in so doing, introduce yet another possible source of generic software errors. One source describing safety concerns related to the NASA Vertical Short-takeoff-landing Research Airplane program, states, "a decision to use a high-level compiler in developing the code... created a software safety problem; there was no known practical way to guarantee that the compiled and installed machine-level code would be free from the one-in-million (10e-6 (catastrophic) errors/hour) chance of having a data sensitive, catastrophic bug" (Dunn,

1991). Applying a patch to compiled code is also somewhat riskier than patching assembly language code since the extra transformation that the compiler applied to the code must be accounted for in the engineering of the patch.

As software-intensive systems become more prevalent, so do the number of generic tools available to support a variety of these systems. While the use of such commercial off-the-shelf (COTS) technology can offer considerable cost savings to a program over the in-house development of similar tools, these savings must be weighed against the specific needs of the program considering their use. Two methods exist for adapting generic tools to the specific systems they are intended to support: standardization and configurability. A standardized tool presents a defined interface to the systems it is intended to support. Whether or not a standardized tool can be of use to a particular program is determined by the impact of adapting the system to the interface provided by the tool. A configurable tool provides a user with some means of adapting the tool to each specific system it supports. Whether or not a configurable tool is of use to a particular program is determined by the amount of effort required to configure the tool to a specific system compared to the amount of effort required to develop a tool customized for the system.

Another concern that needs to be accounted for when determining the tools to be used by a given program is the effect these tools are likely to have on the personnel using them. This concern is especially relevant with automation and AI tools. It can be detrimental to provide so much automation (or, in the case of an AI tool, so much expertise) that the tool user loses valuable insight into the process being executed. Automation can, if not properly managed, turn into a double-edged sword of sorts in that it can allow a relatively untrained person to execute a process, such as the setup and execution of software verification test cases, under normal circumstances while requiring twice as much knowledge and analysis in the event of a problem (was it the flight system or was it the tool?). The limited amount of knowledge gained in the execution of an automated process under normal circumstances often does not support the development of an understanding of the system that will be required for identifying and dealing with any problems that may arise. In the case of AI tools, a slightly different problem presents itself. If not properly designed, AI tools can interfere with the optimum analyst/machine division of labor. The analyst may be required to spend time dealing with computer concerns (including the fact that the computer may want the analyst to interface with it in a manner that is disruptive to the way in which he would naturally approach the solution to a problem in his subject area) while the computer attempts to do the analytical job that would otherwise be performed by an expert analyst albeit without the cognizance of the situation that a human would bring to the solution of the problem.

Backup Flight Control Systems

In quantifying the definition of safe and reliable software for experimental manned flight work, the Ames Research Center (ARC) budgets a maximum failure target rate of 10e-6 catastrophic errors/hour for the entire flight control system. The term "catastrophic" is defined as meaning any event resulting in death and/or major property loss or personnel injury and/or significant property damage as defined in MIL-STD-882B, "System Safety Requirements." Experience with flight control systems

indicates that systems of the size those ARC deals with can be expected to enter initial flight testing with a residual catastrophic error rate of 10e-3 to 10e-4 (catastrophic) errors/hour. These estimates appear to have remained approximately constant over the history of digital fly-by-wire systems having appeared in both one of the earliest (Szalai, 1978) and one of the most current (Dunn, 1991) sources examined (although the earlier source suggests trying for a rate of 10e-9 errors/hour). These figures, along with the lack of any existing analytical method for reliably guarding against generic software errors and the impossibility of completely verifying a flight control system, appear to have made backup flight control systems a standard in fly-by-wire aircraft. Table 1 provides a list of both the primary and backup flight control systems found in a number of fly-by-wire aircraft.

The question of just how dissimilar a backup system needs to be in order to provide adequate protection against a generic software error is a subject of much debate among fly-by-wire aircraft designers. If one assumes that a backup system should be at least dissimilar enough not to contain the same error itself, then an analysis of the types of generic software errors to which the primary flight control system is most susceptible would be useful in providing insight into the requirements for the backup.

In defining the dissimilarity requirements for a backup flight control system, there does not appear to be any universal feeling concerning the need for the use of dissimilar hardware although the use of redundant hardware, either triplex or quad-redundant, is becoming a standard. This would indicate that, in the case of the hardware components of a flight control system, operational failures, rather than generic errors, are the larger concern. All sources agree, however, on the need for dissimilar software for the backup system whether that software runs in its own flight computer or is resident in the primary channels.

There does appear to be considerable agreement among all sources on the need to keep the backup flight control system as simple as possible. The resident backup software (REBUS), an experimental backup flight control system tested on NASA's F-8 digital fly-by-wire (DFBW) research aircraft, used less than one-tenth the memory required by the primary flight control system (Deets, 1986). In no case was a backup flight control system found to be required to provide a vehicle with anything other than true backup types of functions. This emphasis on simplicity in the design of backup systems appears to be driven by the same concern repeatedly expressed with regard to the design of primary flight control systems–that system complexity has repeatedly proven to be the nemesis of system reliability and maintainability. Additionally, accomplishing the transition from the primary flight control system to the backup is recognized to be a difficult and critical problem. Consequently, it is prudent to allow the backup system to concentrate all of its resources on the accomplishment of transition and backup functions in the event that its services are required. The single exception to the simplest possible backup system rule appears to be in the event that the backup may be called upon to provide extended flight envelope protection at the time of transition. In attempting to determine the optimum tradeoff between degraded-mode complexity and envelope-protection capability, the AFTI/F-16 Research Aircraft program states, "Our experiences have illustrated increasingly high penalties associated with failing to cover transitions to degraded-mode flight.... Aircraft control deteriorates so rapidly with unstable airframes that the advantages of backup

simplicity are quickly overshadowed by risks of aircraft loss or severe operational constraints imposed to enable safe transition to the backup" (Ishmael, 1984). The memorandum concludes that the optimum amount of complexity allowable in the design of a backup flight control system is enough to "provide equivalent envelope protection during the transition to degraded flight control."

As mentioned above, the ability to successfully engage a backup flight control system when needed to recover an aircraft from the consequences of "hitting" a generic software error in the primary system is widely recognized to be the most difficult issue associated with the use of these systems. A backup system's job is complicated by the fact that it is not possible to know just when, or under what circumstances, its services will be required. Two major concerns affecting a backup system's ability to successfully assume control of an aircraft involve the availability of the data the backup system needs in order to takeover and the condition of the aircraft as it is handed over from the primary system. If, for example, a particular generic software error results in the vehicle being driven into a region outside its flight envelope, the backup system must possess the capability to bring the aircraft back into a stable configuration upon its assumption of control of the vehicle.

The data availability problem is dependent on the actual data required to define the state of the vehicle and on the architecture of the vehicle, including its flight control system. Data availability is dealt with by identifying the specific data the backup flight control system needs to know as it assumes control of the vehicle and by providing some means for the backup to attain reliable values of this data upon its engagement. The vehicle condition problem is handled by providing sufficient logic in the backup flight control system in order to enable it to handle whatever conditions it is likely to encounter during operational flight. This is part of the AFTI/F-16 Research Aircraft program's "flight envelope protection requirement" discussed above. Implicit in the solution to these engagement problem concerns is the need for the designers of a fly-by-wire program to have a clear concept of just what they expect the backup system to be able to accomplish. Exactly what is the program's definition of "saving the vehicle?"

In an effort to deal with the possibility that a generic software error could cause a situation requiring action sooner than the pilot can react, the Ames Research Center's REBUS program has experimented with the automatic engagement of a backup flight control system (Deets, 1986). REBUS runs resident in each of its host aircraft's primary flight control computers. The F-8 DFBW Research Aircraft has a triplex primary flight control system which made REBUS a triplex backup system. REBUS was automatically engaged (by introducing a generic software error into the primary flight control system) 22 times during six test flights under a variety of flight conditions. Since REBUS possesses a "return to primary" capability, engagement of the REBUS system could be tested multiple times during a single flight. There were 18 returns to the primary system during the testing of REBUS. The feasibility of an automatically engaged backup flight control system is determined by the fly-by-wire aircraft designer's ability to accurately identify the proper transfer criteria associated with the encounter of a generic software error in the primary flight control system. The decision to use an automatically engaging backup system is a trade-off between the designer's ability to accurately identify the types and consequences of the generic software errors

to which the system is susceptible and the pilot's ability to accurately recognize and react to the same.

Risk Management and the Space Shuttle Program

In comparing the Space Shuttle to the fly-by-wire systems examined in this survey, three factors immediately stand out:

- Size: The flight control system of the Space Shuttle is approximately an order of magnitude larger than those of the aircraft examined in this study.

- Flight-to-Flight Reconfiguration: The need to re configure the flight software for each mission appears to be unique to the Space Shuttle. Other fly-by-wire vehicles make changes to the software in the flight control system at the equivalent of the OI (operational increment) level, when new functionality is desired. The resulting flight control system software then supports all flights of the aircraft, without further adjustment, until the next OI. The requirement to customize the Space Shuttle's software, with its related engineering, verification, and validation implications, for each flight introduces an additional risk to the Space Shuttle program, beyond that experienced by other fly-by-wire systems.

- Role of the Backup Flight Control System (BFS): Unlike the backup flight control system of other fly-by-wire vehicles, the Space Shuttle's BFS is expected to provide a number of functions, such as system management during ascent, abort, and entry modes, that are primary to the operation of the spacecraft. BFS also provides some requirements, such as a restart capability, that would otherwise be the responsibility of the primary flight control system. Additionally, the design of the Primary Avionics Software System (PASS) had to be modified in order to accommodate the presence of the BFS and to provide for its data exchange needs. Although the original program philosophy intended that the BFS was to be removed following the flight test phase (defined as the first four flights) of the program, the design of the flight control system was not entirely consistent with this philosophy. That is, the PASS we have today is not the same PASS we would have had if the BFS did not exist. Far from being a mere backup system, the BFS is an integral part of the Space Shuttle's flight control system.

To a large extent, the Space Shuttle program uses many of the same risk management techniques as other fly-by-wire vehicle programs. Some of the conclusions and recommendations that suggest themselves as a result of this survey of the experiences of others involved in the study, development, and operation of fly-by-wire systems are discussed below.

As with any large, complex system, system-level problems are a special concern to the Space Shuttle program. A study of the common causes of software DRs could be made to verify this statement, although there does seem to be a general agreement among the software community as to its accuracy. For this reason, any tools or methodologies that provide insight into the system-wide functioning of the flight software would appear to be a promising investment for the program. Traditionally, system-level analyses during the operational phase of a program are performed by

means of the configuration control process. CRs and DRs are sent to all of those responsible for a component of the system. Recipients evaluate the impact of the proposed modification on the component for which they are responsible. But this process cannot be expected to catch all of the possible interactions occurring in a system the size of the Space Shuttle. In his article, "The Bug Heard Round the World," Jack Garman provides an excellent example of the type of interaction that can easily slip through the configuration control review process (Garman, 1981). In the incident described in that article, a bus initialization calculation used the same data register to perform its calculations that a telemetry phasing routine depended upon to perform a timing calculation. Neither party responsible for these components could really be expected to know that its component had an impact on the other. After all, this did not involve an interface between the two and it would be unreasonable to expect either party to understand the internal implementation of the other's component. Nonetheless, this generic software error stopped the launch of STS-1 and cost the program one launch abort plus the costs associated with analyzing the problem. Possibly this problem could have been avoided if the data register usage had been identified as a system function (or component) and the program had an agent responsible for its analysis and impact evaluation. As part of a risk management exercise, the Space Shuttle program may want to consider identifying system-wide functions that may not be adequately evaluated as part of the component managers' analyses alone. Agents responsible for evaluating these functions may then be appointed or tools capable of analyzing them may be procured. A corollary to this suggestion is that the program may want to identify any unintentional overlaps in programmatic accountability as well as identifying the types of gaps mentioned above since having multiple parties accountable for the same function can have the same effect as having no one responsible.

The Space Shuttle program may benefit from a trend analysis of the data available in its configuration control databases. Such an analysis could possibly provide insight into interactions between software changes and discrepancies to which the Space Shuttle software is susceptible. New insight may also be gained with respect to the program's use of software changes to fix hardware problems. Additionally, the program may want to examine the manner in which it has been distributing its efforts and resources between true operational activities and new development, including upgrades work. Such an examination may lead to a cognizance of "de facto" program philosophies that could provide useful guidelines for those involved in the management of Space Shuttle software. Or, possibly, areas in which there is a lack of a consistent philosophy, leading to "wishy-washiness" in the decision-making process, might be identified. An examination of the amount of programmatic effort that has gone into new development work during the operational phase of the Space Shuttle program might provide insight into the stability of the system that would be of use to those involved in making decisions in which system maturity is a factor. The possibility of identifying development work currently being managed as part of an operational effort also exists. Consideration can then be given concerning the benefit that might be realized from moving such work into a development type of environment. Finally, work identified as being truly operational in nature could be examined for any automation potentials it may possess.

Another area that could potentially benefit from an examination and clarification of programmatic philosophy is flight-to-flight software verification. Currently, there is some potential for confusion between the role of the verifiers and that of the principal function managers with respect to the accountabilities associated with the analysis of discrepancies uncovered during testing. Are the verifiers expected to be able to analyze discrepancies in the flight software and its I-Loads and, if so, what are the responsibilities of the principal function managers and I-Load owners with respect to this area? Who is responsible for identifying and analyzing discrepancies in the simulation software? A clarification of the program's expectations from each of the participants in this activity would reduce the risk associated with unclear responsibility specifications. Additionally, operational verification may present opportunities for automation.

Following the X-29 Research Aircraft program's example, the Space Shuttle program may consider creating a verification and validation matrix containing the software functions affected, such as timing, guidance, flight control, annunciation, and so forth, along one axis and the type of software modification (I-Load patch, code patch, source code change, and so forth) along the other. The relative risk associated with each type of modification and the verification required to support the change (or fix) would be specified for each entry in the matrix. Such a matrix could be used as a tool providing guidance in the disposition of CRs and DRs, thereby promoting consistency in the program's handling of the same.

A final observation is that there is no reason to believe that the Space Shuttle is any more immune to generic software errors than is any other fly-by-wire system. In fact, errors of this type have been uncovered during simulations. Additionally, the need for flight-to-flight reconfiguration and the continuous upgrading of the Space Shuttle's software (OIs) make the Space Shuttle an especially dynamic system. (We never have, and never will fly the same software twice.) For this reason, it appears to be advisable for the Space Shuttle to retain a backup flight control system. Additionally, the program may want to consider the feasibility of simplifying the system in order to enhance its reliability and reduce its maintainability requirements. Any effort spent on enhancing the system's engagement reliability would also appear to be well spent.

Table 1. Flight Control System Summary

Aircraft	Primary Flight Control System	Backup Flight Control System	Source
F-16a	quad-redundant analog	none	Carl S. Droste
F-18	quad-redundant digital	analog/mechanical	Jay Miller
B-2	quad-redundant	none (primary restart capability)	William B. Scott
F-8 NASA DFBW	triplex digital synchronous	triplex analog	Victoria A. Regenie
REBUS	N/A	resident in primary FC digital asynchronous	Dwain A. Deets
AFTI/F-16	triplex digital asynchronous	triplex analog independent	Victoria A. Regenie
F-18 HARV	1 research flt. cntl. system (for thrust vector control) interfaced with f-18 nominal flt. cntl. system	F-18 flt. cntl. system	Joel R. Sitz
HIMAT RPV	ground-based mainframe digital asynchronous	simplex onboard digital asynchronous	Victoria A. Regenie
X-29	triplex digital asynchronous	triplex analog plus backup mode internal in primary computer	Joel R. Sitz

Table 1. Flight Control System Summary (continued)

Aircraft	Primary Flight Control System	Backup Flight Control System	Source
X-30	quad-redundant digital	none (?)	Aviation Week and Space Technology
X-WING	quad-redundant digital synchronous	backup control software resident in primary FC	William R. Dunn
C-17	quad-redundant	limited mechanical	Edwards Air Force Base
BOEING 777	triplex digital/analog	analog/mechanical	William B. Scott
CONCORDE	analog	mechanical	J. Mac McClellan
A310 AIRBUS	partial fly-by-wire system	backup software resident in each primary FC	Dwain A. Deets Flight International
A320 AIRBUS	duplex digital	triplex digital limited mechanical	Flight International Len Morgan
A340 AIRBUS	triplex digital	duplex digital dissimilar HW/SW	Flight International
V-22	triplex digital	simplex analog	Walter L. Ballauer

References

Architecture Refinement for A340 FBW. *Flight International*, vol. 139, no. 4270, June 5-11, 1991, p. 38.

Ballauer, Walter L.; Leet, John R.; Mitchell, James; and Eck, David R.: Testing the Tiltrotor Flight Control System. *Aerospace Engineering*, vol. 11, no. 6, June 1991, p. 37-40.

Burgess, John; and Skrzycki, Cindy: Phone "Blackout" Started In a Single Circuit Board. *The Washington Post*, June 29, 1991, p. A1.

Burgess, John; and Skrzycki, Cindy: Phone Sleuths Tackle Elusive Mystery. *The Washington Post*, July 3, 1991, p. A1.

Burgess, John: Tiny "Bug" Caused Phone Blackouts. *The Washington Post*, July 10, 1991, p. A1.

Ceruzzi, Paul E.: Beyond the Limits. The MIT Press, (Cambridge, Massachusetts) 1989.

Computer Error Cause of Rocket Failure, Pentagon Says. *The Houston Post*, August 27, 1991, p. A4.

Deets, Dwain A.; Lock, Wilton P.; and Megna, Vincent A.: Flight Test of a Resident Backup Software System. NASA TM-86807, Jan. 1986.

Description of an Experimental Expert System Flight Status Monitor. NASA TM-86791, Oct. 1985.

Disbrow, James D.; Duke, Eugene L.; and Regenie, Victoria A.: Development of a Knowledge Acquisition Tool for an Expert System Flight Status Monitor. NASA TM-86802, Jan. 1986.

Dornheim, Michael A.: X-31 Flight Tests to Explore Combat Agility to 70 Deg. AOA. *Aviation Week and Space Technology*, vol. 134, no. 10, March 11, 1991, pp. 38-41.

Droste, Carl S.; and Walker, James E.: The General Dynamics Case Study on the F-16 Fly-By-Wire Flight Control System. AIAA Professional Study Series, vi, 1985, p. 113.

Duke, E.L.; Hewett, M.D.; Brumbaugh, R.W.; Tartt, D.M.; Antoniewicz, R.F.; and Agarwal, A.K.: The Use of an Automated Flight Test Management System in the Development of a Rapid Prototyping Flight Research Facility. NASA TM-100435, May, 1988.

Duke, Eugene L.; Regenie, Victoria A.; and Deets, Dwain A.: Rapid Prototyping Facility for Flight Research in Artificial Intelligence-Based Flight Systems Concepts. NASA TM-88268, Oct. 1986.

Dunn, William R.; and Corliss, Llyod D.: How Safe Is Control Software? NASA Tech Briefs ARC-12710, vol. 15, no. 6, June 1991, p. 126.

Earls, Michael R.; and Sitz, Joel R.: Initial Flight Qualification and Operational Maintenance of X-29A Flight Software. NASA TM-101703, Sept. 1989.

Edwards Air Force Base, CA.: Second C-17 Mission Cut Short Following Flight Control System Faults. *Aviation Week and Space Technology*, vol. 135, no. 12, Sept. 23, 1991, p. 24.

Evans, Martha B.; and Schilling, Lawrence J.: The Role of Simulation in the Development and Flight Test of the HiMAT Vehicle. NASA TM-84912, April, 1984.

Garman, John R.: The Bug Heard Round the World. *ACM Software Engineering Notes*, vol. 6, no. 5, Oct. 1981.

GNC GPC Lockup on OPS Transition (PASS). FSW DR 103049, June 12, 1989.

Gripen Crash Delays Flight Test Program. *Aviation Week and Space Technology*, vol. 130, Feb. 13, 1989, p. 22.

Holmstrom, Stig: I felt that I knew exactly what the Gripen would be like to fly. *Aviation Week and Space Technology*, January 1989. (advertisement)

Ishmael, Stephen D.; Regenie, Victoria A.; and Mackall, Dale A.: Design Implications From AFTI/F-16 Flight Test. NASA TM-86026, Jan. 1984.

Kharj, Al: F-16As Prove Usefulness in Attack Role Against Iraqi Targets in Desert Storm. *Aviation Week and Space Technology*, vol. 134, no. 16, April 22, 1991, pp. 62-63.

Mackall, Dale A.; and Allen, James G.: A Knowledge-Based System Design/ Information Tool for Aircraft Flight Control Systems. NASA TM-101704, Oct. 1989.

Mackall, Dale; McBride, David; and Cohen, Dorothea: Overview of the NASA Ames-Dryden Integrated Test Facility. NASA TM-101720, May 1990.

McClellan, J. Mac: Wired. *Flying*, vol. 117, no. 5, May 1990, pp. 48-51.

Miller, Jay: McDonnell Douglas F/A-18 Hornet. *Aerofax Minigraph 25*, Aerofax, Inc., P.O. Box 200006, Arlington, TX, 76006, 1988.

Morgan, Len: New Age Airbus. *Flying*, vol. 117, no. 5, May 1990, pp. 42-46.

OPS 0 to OPS 3 Wait State (BFS). FSW Discrepancy Report 106198, April 18, 1991.

Regenie, Victoria A.; Chacon, Claude V.; and Lock, Wilton P.: Experience With Synchronous and Asynchronous Digital Control Systems. NASA TM-88271, Aug. 1986.

Scott, William B.: C-17 First Flight Triggers Douglas/Air Force Test Program. *Aviation Week and Space Technology*, vol. 135, no. 12, Sept. 23, 1991, pp. 21-22.

Scott, William B.: 777's Flight Deck Reflects Strong Operations Influence. *Aviation Week and Space Technology*, vol. 134, no. 22, June 3, 1991, pp. 52-58.

Sitz, Joel R.; and Vernon, Todd H.: Flight Control System Design Factors for Applying Automated Testing Techniques. NASA TM-4242, Oct. 1990.

Skrzycki, Cindy; and Richards, Evelyn: Computer Failure Disables Pa. Phones. *The Washington Post*, July 2, 1991, p. D1.

Skrzycki, Cindy: Phone Outage Focuses Attention on Technology. *The Washington Post*, June 30, 1991, p. H1.

Stettler, M.; Dannenhoffer, J.; and Kenger, L.: Grumman Aerospace Corporation, Bethpage, New York; and Larson, J.: Honeywell, Inc., Minneapolis, Minnesota: Architecture of the Flight Control System of the X-29 Advanced Technology Demonstrator. April 1985. (white paper)

Sweden's First Gripen Prototype Destroyed in Crash on Landing. *Aviation Week and Space Technology*, February 6, 1989, p. 25.

Szalai, Kenneth J.; Jarvis, Calvin R.; Krier, Gary E.; Megna, Vincent A.; Brock, Larry D.; and O'Donnell, Robert N.: Digital Fly-By-Wire Flight Control Validation Experience. NASA TM-72860, Dec. 1978.

Tomayko, James E.: Digital Fly-By-Wire: A Case of Bidirectional Technology Transfer. NASA Tech Briefs, Nov./Dec. 1986. pp. 45-48.

Whitaker, A.; and Chin, J.: Grumman Aerospace Corporation, Bethpage, New York: X-29 Digital Flight Control System. (white paper)

777's Flight Control System Configured For Good Handling and Ease of Pilot Transition. *Aviation Week and Space Technology*, vol. 134, no. 22, June 3, 1991, p. 58.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Technical Memorandum

4. TITLE AND SUBTITLE
Risk Management in Fly-by-Wire Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Karyn T. Knoll (NASA JSC)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lyndon B. Johnson Space Center
Houston, Texas 77058

8. PERFORMING ORGANIZATION REPORT NUMBER
S-700

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, D.C. 20546

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
TM 104757

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unclassified/Unlimited

Subject Category 17

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A general description of various types of fly-by-wire systems is provided. The risks inherent in digital flight control systems, like those used in the Space Shuttle, are identified. The results of a literature survey examining risk management methods in use throughout the aerospace industry are presented. The applicability of these methods to the Space Shuttle program is discussed.

14. SUBJECT TERMS
Flight Software, Flight Control Systems, Risk Management, Spacecraft Software

15. NUMBER OF PAGES
23

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102