SOFTWARE RELIABILITY AND ADVANCED AVIONICS

by GERARD E. MIGNEAULT
NASA Langley Research Center
Hampton, Virginia

SUMMARY

This paper proposes that software is becoming the most safety critical element of the highly reliable avionics systems which will be needed in civil transport aircraft of the future.

The paper first discusses the pressures leading to the use of digital technology, especially computers with software, in future civil transport aircraft. The level of required reliability pertaining to safety is then determined, both as mandated by regulations and as observed in actual practice.

Finally, advanced fault tolerant computers are described. Their reliability is simply analyzed in order to determine the role software will play; it is critical. The level of software reliability required is then examined.

INTRODUCTION

Electronic components are not new to civil transport aircraft. However, in the past such equipment has been predominantly analog in nature. The recent past, the present and the near future constitute a period of transition, a time of change from analog components to components making fuller use of digital technology, and including the use of stored program computers. One can foresee the trend continuing into the future and leading to greater dependence upon stored program computers, and consequently, upon the embedded software. It follows therefore, that software reliability is becoming, and will continue to become, an even more significant factor in the analysis of the reliability of avionics and, accordingly, of the reliability of the total aircraft.

AVIONICS IN FUTURE CIVIL AIRCRAFT

The anticipated wider use of stored program computers in future civil aircraft is a consequence of several more or less obvious factors. More obvious is the pace of development of digital technology: ever greater amounts of computational capability compressed into ever smaller volumes—of less weight, consuming less power, and at less cost both initially and in later maintenance. Thus, other things unchanging, there is an economic advantage to the use of the newer technology devices—on a functional device substitution basis. Since the newer, stored program computer devices are essentially multifunction devices, there is also the potential benefit of reduction of the total number of devices and greater standardization of device types. Moreover, multifunction devices permit priority rankings among functions to be exercised, i.e., the option is more available to the system of choosing which functions continue to perform when the system is faced with a number of failed components. Such an option is no insignificant advantage; it is not available to a system in which each component device is dedicated to a specific function. Thus, multifunction devices provide a system with a greater likelihood of “graceful degradation.” Another more obvious factor is the opportunity for expanding the scope of some functions by virtue of the increased data base and computational power available.

A less obvious factor is the fact that some potential increases in civil aircraft fuel efficiency are dependent upon the availability of reliable, increased computational capability. Such fuel efficiency increases would be possible due to weight and drag savings resulting from reductions in aircraft passive structure. In turn, the structure reductions would become possible by means of “active control” techniques which maintain aircraft aerodynamic stability and reduce peak local loads on aircraft structure (in effect, distribute loads more evenly across the structure) by complex, precise, ever continuing, and possibly differential, “active control” of aircraft control surfaces. Clearly, to the extent to which such computational power was substituted for previously passively provided structural integrity and stability, uncontrolled in-flight interruptions and/or continuing general malfunctions of the increased computational capability would not be tolerated.

In sum, potential economic benefits provide, and will continue to provide, powerful pressures to introduce more digital technology, especially in the form of stored program computers, into future civil aircraft. To many it is almost an article of faith that the introduction of more digital technology into avionics will have a positive effect upon the reliability of the avionics. While the presumption may not be too difficult to demonstrate correctly for the limited cases of device per device substitution, it remains to be determined with acceptable confidence that the maximum performance benefits to be associated with the use of stored program com-
puters can be achieved without an intolerable detrimental resultant effect upon the reliability of the total aircraft.

RELIABILITY REQUIRED OF AVIONICS

The reliability required of a total aircraft system provides a bound on the reliability required of the avionics and the software embedded in it. There are several, not totally unrelated concerns which give rise to aircraft reliability requirements. They are (a) cost of operations for airline operators, (b) disruption of the air transportation system, and (c) safety of occupants of aircraft and individuals on the ground. The concern about disruption of the air transportation system is ordinarily far less prominent than the other two concerns, if it is recognized at all. With its potential for national economic chaos, it could become the source of the very strictest reliability requirements when the nature of design inadequacies and software (un)reliability is considered. However, in this paper as in society in general, safety is recognized as the concern generating the most stringent requirements since clearly the most conspicuously undesirable malfunctions in civil aircraft are those which result in sudden, often spectacular loss of human life.

A minimum acceptable level of safety is specified in regulatory agency directives by the use of the expression "extremely improbable" to describe the likelihood of catastrophic events. However, the requirement is subject to interpretation. A typical interpretation by a major airframe manufacturer is the following:

"... a number less than or equal to $1 \times 10^{-9}$ has been imposed to represent the probability of an event designated as extremely improbable. ... Loss of the CCV/FBW function, given a fault-free system at dispatch, shall be extremely improbable."^{2}

There are two points to be noted. First, the qualification "given a fault-free system at dispatch," which is intended to exclude having to consider physically degraded systems in the determination of a system's reliability, begs the issue of software bugs and, indeed, of all questions of design inadequacies. Of course, such flaws are not denied. But they are not included within the reliability computational process. The working hypothesis is that they will be exorcised before operational use of the systems to an extent such that their presence is negligible, or more precisely, such that the frequency of malfunctions of a system due to such residual flaws is sufficiently less than the frequency of malfunctions due to physical degradation to permit them to be ignored in reliability calculations. Conventionally the hypothesis is justified by system verification, i.e., testing, prior to operational use.

The second point to note is that the interpretation applies the requirement for "extremely improbable" events to loss of a specific function critical to the flight of an aircraft rather than to the loss of an aircraft. No apportionment of the (un)reliability among subsystems is indicated; the occurrence of any malfunction from the set of all malfunctions of the stored program computers whose consequences include loss of the CCV/FBW function must, therefore, be an "extremely improbable" event.

There are also in circulation drafts for an FAA Advisory Circular on the topic of system design analysis to generate a consensus explanation of the expression "extremely improbable." One contains the statements:

"Systems, considered separately and in relation to other systems, should be designed ... such that a catastrophic failure condition is extremely improbable. ... Extremely improbable refers to events ... with a mean frequency in the order of $1 \times 10^{-9}$ or less per flight or flight hour. Such events are the loss of a number of lives and/or destruction of the aircraft."^{4}

The last sentence is explicit; the mean frequency magnitude, $1 \times 10^{-9}$, is to be coupled with loss of lives (or aircraft), not with a function. Thus to the extent that malfunctioning computational systems (i.e., stored program computers including the embedded software and firmware and other residual system design inadequacies) can singly cause catastrophic consequences, the occurrence of any malfunction from the set of all such failures must be at least "extremely improbable," and possibly even less likely in order to allow for the apportionment of some of the (un)reliability to other aircraft subsystems—including the human factors.

Finally, statistics on the state of civil aviation safety lend credence to the reasonableness of the interpretations cited above of the regulatory agency safety requirement. Figure 1 contains the history by calendar year of the "average aircraft's average speed" in the recent past. As indicated by the graph, while the period prior to 1974 was a time of transition, the period from 1974 to the present (1978 was the last fully documented year at the time of writing) has been quite stable. The "average aircraft's average speed" has varied from its mean value during the period by no more than approximately 0.3 percent while the total hours flown each year has remained relatively constant, approximately 6.3 (± 7 percent) million hours. Therefore, the period from 1974 to 1978 is here adopted as the base period.

Figure 2 contains the history of the mean frequency of fatal accidents (per million hours flown) per calendar year. During the years of the base period, the mean frequency of such catastrophic events has varied between $0.5 \times 10^{-4}$ to $1.5 \times 10^{-4}$ per flight hour with a mean mean of approximately $0.9 \times 10^{-4}$. Moreover, an examination of individual accident records reveals that the majority of the accidents are not ascribed to equipment malfunctions as the primary cause. Exact proportions are debatable owing to reporting differences; however, it suffices to note that it could be argued that the mean frequency of fatal accidents due primarily to equipment malfunctions was in the range from $1 \times 10^{-4}$ to $1 \times 10^{-5}$ per flight hour during the base period. Certainly, other things unchanged, nothing less safe than what is already available is acceptable.

* CCV/FBW = Control Configured Vehicle/Fly by Wire
Hence, for avionics a maximum mean frequency of catastrophic malfunctions of $1 \times 10^{-9}$ per flight hour over the lifespan of aircraft is asserted to be the reliability requirement. The magnitude $1 \times 10^{-9}$, however, is a source of difficulty for it makes a reliability estimate with useful confidence bounds virtually impossible to obtain by conventional system verification prior to operational use because of the number of trials and elapsed time required. In particular, the working hypothesis mentioned above must be justified by some other means—if it is to be relied upon.

A solution often referred to involves the concept of (aircraft) systems of greater reliability constructed from subsystems of lesser reliability (reference 5 for the theoretical notion)—for example, the use of back-up systems. But, in addition to the still present difficulty of credibly estimating the extremely high reliability of the decision logic implemented to switch to a back-up system, the notion requires the use of alternate systems, external and redundant to avionics. It is precisely such systems which advanced avionics is intended to obviate—if promised benefits are to be realized. Therefore, the notion is considered inconsistent with the intent of advanced avionics.

**FAULT TOLERANT AVIONICS COMPUTERS**

Computer architectures have been developed specifically in anticipation of the need to satisfy the safety requirements implied by the expression “extremely improbable” discussed above and in anticipation of the data processing needs of future civil aircraft. The reliability of the physical component devices available now and in the foreseeable future, devices such as processors, memories, power supplies, etc., whose mean time to failure (MTTF) parameters are realistically in the range from $10^2$ to $10^5$ hours, implies mean frequencies of failure in the range from $1 \times 10^{-2}$ to $1 \times 10^{-1}$ per hour for conventional, fault intolerant computer systems constructed from such devices. What is somewhat experimental in the referenced architectures is the attempt to attain extremely high system reliability by means of fault tolerance.
achieved by the use of redundancy, error detection achieved by voting among redundant components, and reconfiguration—all performed internally.

An aside is needed at this point to ensure consistent interpretation of the terms “failure,” “fault,” and “error.”

Confusion can arise as a result of the “software” trend, evidenced by recent articles on software reliability, of using the words in a manner reversed from the usage generally adopted for hardware. The following meanings are used here for both hardware and software:

A failure is the event when something causes a device, component, system, algorithm, etc., to change its state from one in which it performs its intended function to one in which it does not. The something which causes the change may or may not be known. After the failure, the device, component, etc., is called a failed or faulty device, component, etc. Any higher level system of devices, etc., which cannot perform its function because a subdevice, sub-etc., is failed is also called failed or faulty.

A fault is the particular condition or flaw in a failed device, etc., which differentiates it from its unfailed state.

When the function or output of a device, etc., differs from its intended function or output, that difference is called the error. In data processing systems, error means bad or wrong data. An error is all that can be detected internally to a computing system. A higher level system which contains a failed device, etc., emitting errors yet continues to perform its function is said to be fault tolerant. An accumulation of errors may well be the cause of a failure of a higher level system.

Thus a physical device fails when it “breaks down.” Thereafter it contains a fault. A system designer or software programmer can create a design or software containing a fault; in this sense the designer or programmer failed. A fault may or may not be active; when it is, one or more errors result. A fault is latent, transient, intermittent or permanent dependent upon the manner in which it generates errors. A software bug may not surface until some time after a system has been in operation, i.e., it may be latent. A bug may cause a data error only occasionally in response to specific, infrequent input data patterns, and may thus appear intermittent. Customarily a software bug is regarded as a permanent fault, remaining in the system ever after from the moment of its creation by a programmer. However, it is possible for a bug, having given rise to a data error, to disappear from an operational system—in which case it appears as a transient; the resulting bad data may or may not be attenuated in further processing. As an example, consider the common occurrence of failing to preset a variable at system start up. Thus, software bugs can appear to share the possible attributes of hardware faults.

The referenced computing systems are designed to detect and contain errors and isolate faults in physical components at the level of processors, memories, etc.—generally. Detection requires at least comparison; containment and isolation require a plurality. Necessary algorithms are implemented in hardware and software. When components are deduced to be faulty, they are ignored in future computations by the unfailed components—not unlike ostracism within a social system. Functions previously performed by components since failed are distributed among the unfailed components; if there is insufficient computing capacity remaining, those functions least important to the flight of an aircraft are discarded. The process continues until insufficient resources remain to perform minimum computations and the system cannot support flight control.

The state transition diagram in Figure 3 is a simplified representation of the scenario above, incorporating the essential approximating assumptions made in reliability analyses to date of highly reliable fault tolerant avionics computing systems. The analyses have been more complete and searching than this simple representation—accounting for various component types, not all interchangeable, having different propensities for failure, etc. Yet the additional refinements of analysis do not significantly modify the conclusion below.

The prime assumption in the reliability analyses is that the elemental failures at the physical level in any given component occur independently of the occurrence of other such failures in other components. It is assumed, and every effort is made to ensure, that the environment is controlled such that “massive,” system-wide failures do not occur. For example, avionics systems must be protected from lightning discharges having such system-wide effects. The diagram in Figure 3 represents this assumption by restricting degradation solely to a state with exactly one fewer component.

A second assumption is that the frequency of nearly si-
multaneous errors (resulting from different failures) is su-
ficiently small to be neglected in the count of system failures. 
To the extent to which this assumption is not correct, the 
algorithms which perform detection and containment of er-
ors and isolation of faults are inadequately designed for they 
cannot cope with many combinations of multiple failures 
concurrently. Clearly, the greater the latency time between 
the creation of a fault and its manifestation as an error, the 
less justified the assumption. Yet the assumption is main-
tained for its mathematical convenience and for lack of suf-
ficient hard data (to date) to support alternate models of 
behavior. The instances when the assumption is not correct 
are accounted for and represented in the diagram by “cov-
erage” parameters, conditional probabilities that, given a 
failure, transition to a correctly reconfigured and operating 
state is successfully accomplished.

A third assumption is that, once reconfiguration has been 
performed, any further errors generated by the faulty com-
ponent are prevented from propagating outside predetermined 
containment boundaries and thus prevented from 
causing secondary failures. This assumption is also repre-
sented in the diagram by means of the restriction of degra-
dation solely to a state with exactly one fewer component; 
in addition, analyses normally constrain component failure 
rates to be independent of system state.

Finally, analyses of the avionic computer systems have 
conventionally neglected software and design faults—hy-
pothesizing a system fault-free, the bugs exercised by much 
testing and program correctness proving and perhaps even 
entirely avoided by application of disciplined management 
and program development techniques. It should be noted 
that, because of this “decoupling” of the software (un)reliability from the process of estimation of computer 
system reliability, the notion of a required software reli-
bility becomes disassociated from the context of the appli-
cation. Denied this direct, measurable relation to an appli-
cation, rather than remaining simply a characterization of 
software’s merit, the notion is often associated with com-
parisons and orderings of methods for implementing soft-
ware (e.g., preferences for certain program structures, for 
estimating number of bugs remaining in code, etc.).

SOFTWARE AND DESIGN LOGIC AS SYSTEM 
ELEMENTS

While the number of faults (flaws) remaining after careful 
development and testing remains problematical, what is im-
portant in the context of an application and reliability are 
the frequency with which faults are activated and the se-
verity of the consequences of the errors generated (empha-
sizing again the context of the application). If software pro-
grams (and design logic) are considered as system elements, 
possible sites of residual faults, interacting with other more 
tangible components and capable of leading to avionics com-
puter system failure, then the real consequence of software 
malfunctions can be evaluated and the reliability required 
of software can be stated.

A fatuously simple representation of software behavior is 
illustrated in Figure 4a; it illustrates a difference between 
hardware and software. Unlike the case for hardware in 
which redundancy is provided by replications of compo-
nents, simply replicating software in replicated components 
only replicates any faults; consequently, errors occur in rep-
licate sets and the notion of error detection and fault tol-
erance by comparison and majority voting is defeated. (The 
same is true of design logic.) Yet software fault tolerance 
techniques, of which B. Randell and his colleagues at the 
University of Newcastle-upon-Tyne have been leading in-
novators, exist. They attempt to provide “redundancy” by 
means of alternate, secondary algorithms and “acceptance” 
tests to detect errors. Such concepts appear applicable to 
avionics; minimal additional time and memory usage are re-
quired. Accordingly the state transition diagram in Figure 
4a represents the behavior of fault tolerant software on the 
assumption that successful recovery from a software error 
error is followed by return to an initial (software) state. That is, 
unlike hardware, software may not degrade. The rationale 
for the assumption is that a fault responsible for a software 
error has always lain latent; presumably it will do so again 
after the date or conditions which activated it have passed. 
A recovery parameter, analogous to the “coverage” param-
eters mentioned for hardware above, can be used to account 
for the possibility of not detecting or recovering from all 
software errors. Figure 4b is an equivalent but simpler repre-
sentation.

Studies of system failures due to software have been pub-
lished, e.g., some recent data indicating that for one special 
application and for one failure mode a hypothesis of expo-
nentially distributed system failure times due to software was 
not tenable,10 but there is no credible, empirical evidence 
for the selection and justification of any complex, general 
model of system failures due to software11 let alone due to 
general design flaws. Figure 4 is not intended to suggest that 
a simple model is sufficient for analysis and prediction pur-
poses.

The diagram in Figure 5 is a combination of Figures 3 and 
4b to represent a total system comprising hardware and soft-

ware (and design logic). An initial $n$ redundant hardware component system can fail by either

1. suffering a hardware component failure and degrading (with conditional probability $c_r$) to an $n-1$ hardware component system which can in turn fail, or
2. suffer a component failure from which it cannot recover (i.e., with conditional probability $1-c_r$), or
3. suffer a software error from which it cannot recover. $\mu$ represents the rate of fatal software errors; $1-k$, the conditional probability of not being able to handle an error.

Clearly, each and every path to failure is bounded by the $1 \times 10^{-9}$ reliability requirement. In particular, for $\lambda$ and $\mu$ interpreted as mean frequencies of hardware and software critical malfunctions per hour, $n\lambda (1-c_r)$ and $\mu(1-k)$ must each be less than approximately $1 \times 10^{-9}$. For reasonable and realistic values of $n$ (3 to 10) and $\lambda (\sim 10^{-4})$, $0.999997 \leq c_r \leq 1$, in a sense, a requirement on design logic. No credibly reliable estimates for $\mu$ are available, hence it can only be required of software that $\mu (1-k) \approx 10^{-9}$.

Since, and if, $\mu$ and $(1-k)$ both pertain to the same kind of failure, i.e., software bugs and design flaws, one might speculate that they are similar in magnitude and that a credible demonstration that $\mu$ and $(1-k)$ are each approximately $10^{-4}$ or $10^{-5}$ is the reliability requirement for avionics software.

CONCLUSION

To the extent to which the assumptions stated above approximate the real world, hardware can be replicated until required reliability is achieved, but the same is not true of software. Hence, software is the critical element of highly reliable (fault tolerant avionics) computer systems; the problem of design inadequacies is considered to be the same as the software problem. Since data on software error rates (in the precision implied necessary by the model above) are lacking, it is currently not possible to predict with "credible" confidence that a highly reliable software system will indeed satisfy its reliability requirement. This leaves an avionics system with embedded software in an uneasy state but points quite clearly to the area of needed research.

REFERENCES

Languages

This year the "Languages" technical area focuses on practical matters. The four sessions are all concerned with language issues arising from real world considerations.

"Ada, Where it Stands Now" will assess the current status of the new Department of Defense programming language for embedded computer systems. Embedded computer systems are those which interface with non-computer devices such as satellites and submarines. "MUMPS" is considered by its devotees to be an instant data base implementation system. This session will spread the word. "High Level Languages for Microprocessors" explores the availability, outlook, and problems of powerful languages for little computers. "Pascal in the Real World" demonstrates that a well designed language can be used to solve real world problems.

If a common theme emerges from these sessions, it is that it is possible to design, implement and use clean, powerful and "academically" acceptable languages for the nitty gritty jobs which must be done.