

ASSURING AGAINST SYSTEMATIC FAULTS USING ARCHITECTURE AND FAULT TOLERANCE IN AVIATION SYSTEMS

Squadron Leader D.W. Reinhardt

Royal Australian Air Force
Deputy Senior Design Engineer
Avionics C-130H/J
RAAF Richmond, NSW 2755, Australia

derek.reinhardt@defence.gov.au

Professor J.A. McDermid OBE FREng

Head of Department of Computer Science
University of York
United Kingdom

john.mcdermid@cs.york.ac.uk

Abstract

The failure circumstances of complex aviation systems involving technologies such as software are dominated by systematic faults. However systematic faults are often poorly resolved by the coupling of software assurance with traditional system safety methodologies.

This paper examines the treatment of systematic faults and failures in a number of actual aviation systems, including civil and military Automatic Flight Control Systems (AFCS) and Flight Management Systems (FMS). The results of this examination are contrasted with the fail safe design criteria underpinning the 14CFR 25.1309 airworthiness requirements for civil aircraft certification, and commonalities in architectural and fault tolerance treatments of systematic faults and failures are identified.

Using the identified commonalities, and examining how architecture and fault tolerance contribute to bounding the uncertainty of the effects of systematic failures on a system, a framework is proposed for quantifying the assurance of safety architecture in aviation systems. The assurance framework provides a direct measure of the degree of the system's fault tolerance against systematic faults and failures, and thus infers the system's suitability for use in the presence of aircraft level failure conditions of differing severities.

Keywords: Architecture, Assurance, Aviation Systems, Fault Tolerance, Safety, Software Assurance, Software Safety, Safety Critical.

1 Introduction

In complex aviation systems involving technologies (e.g. software) whose faults are dominated by a class of faults referred to as systematic faults, there are substantial challenges at providing assurance that these faults do not lead to unacceptable aircraft failure conditions.

Most safety and assurance standards for software systems in the aviation domain deal with systematic faults through the identification and allocation of software safety requirements and through the specification of software integrity levels or design assurance levels. However there are numerous limitations (refer to [JTM07],

[McD07], [McK06], [NTS06], [Wea03]) with the mechanisation of the current assurance frameworks that limit their effectiveness at providing robust assurance that systematic faults do not lead to unacceptable aircraft failure conditions. This is primarily because of limitations in the treatment of requirements validity of the system and software, as well as in the direct provision of evidence that the behaviours of the system and software are acceptable with respect to safety. The current standards are also limited in the ability to determine the impact on the aviation system of shortfalls against the criteria of the standards.

This paper examines these challenges for aviation systems and proposes an alternative framework for the assurance of systematic faults in aviation systems. The framework that is proposed is intended to be compatible with the existing standards' approaches, even if the standard were not to explicitly adopt the architectural assurance framework proposed in this paper.

2 Background

In order to understand the sources of limitations in the effectiveness of the current assurance frameworks for systematic faults in aviation systems, it is necessary to understand what systematic faults are, and why their treatment is inherently different to that of other sources of failure. The following sub-sections provide insight into systematic faults and failures, and then examine in detail the sources of limitations in current standards frameworks in this context.

2.1 Systematic Faults and Failures

Unlike random failures, which are caused by items and components wearing out, systematic failures are produced by requirements, design and implementation faults introduced by errors or omissions made by developers and manufacturers (i.e. humans, tools, systems or processes) during system development or manufacture, or by human error or omission during operation or maintenance. Once such a systematic fault is resident within a system, then it will always be activated when the particular set of circumstances and system state transpire. Unfortunately it is also difficult to predict the occurrence of systematic failures caused by such faults, because:

- complex systems have complex interactions with their operating environment
- complex systems have significant numbers of internal states both explicit and implicit;
- there is uncertainty as to the number of unknown design faults, and there may be limitations in the understanding of the operational environment or context; and
- as it is generally not possible to predict when a particular set of circumstances will occur, in association with the vulnerable transition criteria between system states (both temporal and spatial) in existence due to the latent design fault, then it is generally not possible to produce statistical information from which to determine failure distributions, reliability and probability.

Thus the typical means of determining likelihood, based on probability of failure for random failure, does not translate to systematic failures, and a different approach is required. [McD07] distinguishes this as epistemic uncertainty (i.e. imperfect knowledge of the system or the stochastic model), rather than aleatoric uncertainty (i.e. 'randomness' which can be characterised by a stochastic model). In simple

terms, shortfalls in knowledge of the distribution of systematic failures makes predicting a probability of systematic failure worthless.

Unlike traditional hardware, all failures associated with technologies like software (i.e. from faults in requirements, design, or implementation) are systematic (e.g. if the fault is present, it is just waiting for the right set of circumstances to activate the fault). No element of the software is 'wearing out' in the traditional sense. [McD01] identifies that software failures arise most often from:

- discrepancies between documented requirements specifications and the behaviours needed for correct and safe functioning of the system; and
- misunderstandings by software developers about the software's interface with the rest of the system.

Software-related incidents and accidents have still occurred when the software satisfied its specification and when the operational reliability of the software was perceived to be very high [McD01]. This is due to:

- requirements that specify behaviour that is not appropriate from a system perspective;
- requirements that do not specify some particular safety behaviour and therefore the developers have made invalid assumptions about those particular behaviours; or
- software that has unintended (and unsafe) behaviour beyond that which is specified in requirements.

Because of these causes, software faults are often observed as being intermittent, affected by changing conditions, recurrent under the right conditions, partial (i.e. not all software faults activate detectable and identifiable errors, or propagate to immediate failures), and all these factors create challenges in assuring that such faults and failures can not occur.

For these reasons, it is not possible to statistically predict the probability of systematic failures, and thus for technologies such as software it would seem that we are left without a means to quantify the associated risks using traditional risk assessment methodologies (i.e. risk being a function of consequence and likelihood).

2.2 Current Standards Approach

Instead of providing a structured assessment of the likelihood of systematic failure for software (which has been established in Section 2.1 to be difficult), current standards employ the concept of assurance. Assurance standards provide confidence in the behaviours of the software by examining key product and process attributes and evidence across the development and verification life-cycle. The proposition is that the more rigorous the exposition of these attributes, and the greater the body of evidence supporting these attributes, the more likely it is that software faults are not introduced into the software or system, or they are detected and corrected throughout the development and verification process. Hence, the assurance approach implicitly shapes the distribution of systematic failures such that while the distribution is still not known, the likelihood of systematic failure realising a safety related failure effect in the system is low within the constraints of the intended (and understood) operating context.

However the shortfall with the traditional assurance approach is that the evidence still only supports an implicit argument about the likelihood of a systematic failure realising a safety related failure effect, and they don't directly provide any information about the behaviour of the software when a systematic fault is activated. The following sections examine the specifics of this problem in further detail.

2.2.1 Existing Software Assurance Frameworks Don't Address Architecture

The safety process as described by [ARP4754] is shown in Figure 1. The key steps in the process associated with the formation of the system and software architecture are:

- the development of system architecture using architectural requirements determined from Preliminary System Safety Assessments (PSSA); and
- the allocation of item requirements to software based on the system architecture and PSSAs.

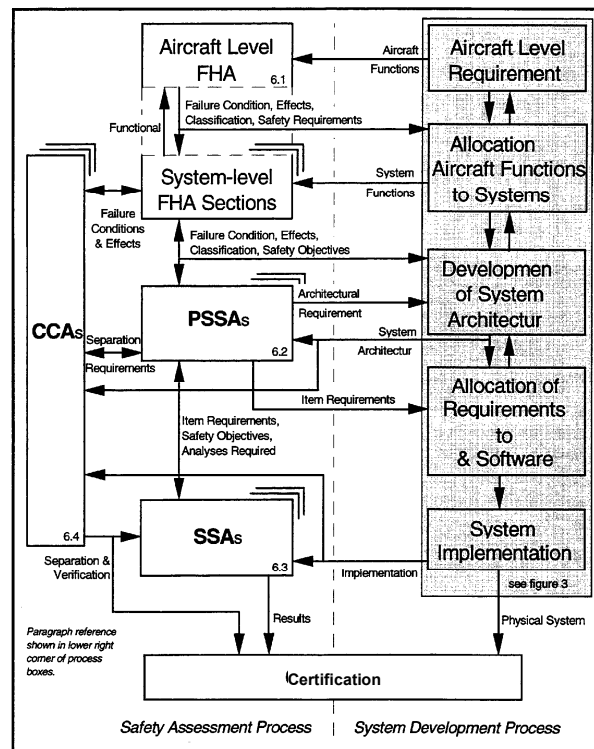


Figure 1: Safety Assessment Process Model [ARP4754]

Clearly the PSSAs are a key contributor to the formation of the system architecture under [ARP4754]. PSSAs are usually an inter-related set of System, Sub-system and Component level Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA). They are structured around demonstrating the predicted likelihood of system level failure conditions identified in the preceding analysis. However, because of the probability/likelihood focus of these combined analyses, they are often only capable of providing a binary perspective (handled / not handled) on treatments such as fault tolerance and architecture on systematic failures.

Fault tolerance is the ability for a system to detect an error, fault or failure condition (as defined by [ALR04]) and then undertake a level of reconfiguration/handling to prevent the fault or localised failure propagating to a failure at the sub-system

boundary, or a system hazard at the system level. Architecture in this context is used to provide these layered detection and handling mechanisms and thus achieve fault tolerance.

Once a treatment is allocated to an item (such as a software component), and the assurance level assigned, there are limitations to further consideration of the architecture and systematic failures within the objectives of standards such as RTCA/DO-178B [Rei08]. For example, RTCA/DO-178B, requires development of high level requirements and low level requirements, including derived requirements. RTCA/DO-178B also requires that the software architecture is developed, and there are also links to architectural assessment. However, the requirements are not from the perspective of the architecture's contribution to preventing systematic failures. There is substantial scope for improvements to be made as to how architecture is addressed, particularly from a product behavioural validity perspective. Furthermore, redundancy, design diversity, and layered detection and handling mechanisms (software, software implemented, hardware implemented, line replaceable unit (LRU) and system) within these architectures clearly introduce behaviours in a system that are far from merely binary, and require different approaches to assure.

One notable exception in RTCA/DO-178B with respect to architecture is partitioning integrity. However, partitioning integrity is predominately about preventing the effects of one software component affecting the function of another software component. It is not about providing an appropriate architectural fault tolerant treatment in the occurrence of a systematic software failure's effects propagating within a system's control and data flows.

Therefore, existing assurance frameworks don't comprehensively address architecture with respect to the behaviour of the system under the occurrences of systematic failures. The explicitness of the treatment of systematic faults gets lost in the interface between the safety analysis (which is likelihood focussed) and traditional assurance methodologies which have limitations in their consideration of architecture in this context.

These limitations have been previously identified by [Pum99] and [Wea03] who proposed approaches centred around the explicit provision of evidence with respect to an arguably complete taxonomy of software failure conditions, and their associated treatments (absence or detection/handling, many of which require architectural implementation). However, neither [Pum99] or [Wea03] set out explicitly how their approaches might be integrated with the existing assurance standards.

This paper proposes that assurance frameworks should explicitly integrate requirements for architectural treatments to systematic faults along with their more traditional objectives.

2.2.2 Existing Assurance Frameworks Don't Provide Quantification of the Degree of Fault Tolerance within a System and its Software

Section 2.2.1 established that the existing approaches didn't provide explicit visibility of treatment of systematic faults. Using this context, the extent to which quantification of the degree of fault tolerance within a system and its software can be explored.

For example, if a certification authority or aircraft operator was to be given the information that their digital flight control system:

- had several relevant catastrophic failure conditions,
- was triply redundant to guard against failures (and inevitably support a robust likelihood argument with respect to random failures), and
- each software component was assured to level A (as RTCA/DO-178B requires);

then how much of this information helps either the certification authority or aircraft operator to understand the fault tolerance behaviour of the system under the occurrence of faults? Actually, very little!

While the example is slightly contrived, as the:

- claimed role of the redundancy would usually be evident to the certification authority in the safety analysis, and
- the flight manual would give some indication of the failure behaviours of the system as it reconfigures itself under fault conditions to the operator;

the certification authority and operator are still missing several quite importance pieces of information. The operator and certification authority don't know:

- how many systematic faults (known and unknown) the system is tolerant against?,
- which classes of systematic faults the system is tolerant against (under what conditions)?, and
- to what extent any redundancy or other highly publicised fault tolerance mechanisms might be violated by the occurrence of systematic faults?

The certification authority and aircraft operator also have limited knowledge (i.e. uncertainty) as to the precise behaviour of the system under these fault conditions, which would be necessary in providing the timely operator response to the source of the fault.

This paper asks the question: shouldn't an assurance framework for aviation systems inherently quantify the answers to these questions? Furthermore, the treatments to such faults should be quantifiable with respect to the extent of detection and handling mechanisms in their failure paths. If a system is truly tolerant against systematic faults, then it should be possible to provide a measure of just how many and what types of faults it is tolerant against. These factors are then obviously a key determiner in the system's suitability of differing consequences and severities in aircraft level failure conditions.

3 Examination of Actual Aviation Systems

So far this paper has identified a number of shortfalls with the existing standards frameworks with respect to systematic faults and failures. However, irrespective of the standard's framework, it is worth examining the degree of fault tolerance against systematic faults actually achieved in a host of representative aviation systems. This will permit any commonalities (positive and negative) in the treatment of systematic faults to be determined.

Two different types of critical aircraft systems (Automatic Flight Control Systems (AFCS) and Flight Management Systems (FMS)) have been considered across a range of civil and military aircraft including the B777, A330, C-17, C-130J and F/A-18A/B. Information on the architecture and design features employed from these systems has mostly been obtained from information in the public domain on these systems. Where this has been insufficient, then additional behaviours and treatments have been inferred using flight manuals, pilot briefing notes and maintenance publications. The main sources of information are identified in the references of this paper.

3.1 Flight Control Systems

Annex A Table 6 provides an overview of the flight control systems of several civil and military aircraft with the purpose of identifying those design features that might provide detection and handling of systematic fails and failures within the system. Specifically the Boeing 777, A330 / KC-30A, C-17 and F/A-18A/B are considered.

3.2 Flight Management, Navigation

Annex A Table 7 provides an overview of the flight management / navigation systems of several civil and military aircraft with the purpose of identifying those design features that might provide detection and handling of systematic fails and failures within the system. Specifically the Boeing 777, A330 / KC-30A, C-130J and F/A-18A/B are considered.

3.3 Aviation System Analysis

Eliciting the fault tolerance design features from Annexes A and B, and classifying and consolidating the features according to [HiM01] and [Ham01], Table 1 summarises the generic classes of fault tolerance mechanisms employed by the examined systems. Section 5 examines these features in further detail, however to provide context to this examination, it is necessary to firstly introduce the fail safe design criteria and consider it for systematic faults.

4 Fail Safe Design Criteria Applied to Systematic Faults

The most likely source of many of the fault tolerance design features present in actual aviation systems with respect to treating systematic faults and failures, comes not from the system safety or software assurance standard framework, but instead from the application of the fail safe design criteria. This section examines the fail safe design criteria, and its effect on architecture and systematic faults.

4.1 14 CFR 25.1309 / AC25.1309 Fail Safe Design Criteria

[AC25.1309] defines the fail safe design concept as follows:

“In any system or subsystem, the failure of any single element, component, or connection during any one flight (brake release through ground deceleration to stop) should be assumed, regardless of its probability. Such single failures should not prevent continued safe flight and landing, or significantly reduce the capability of the airplane or the ability of the crew to cope with the resulting failure condition.”

“Subsequent failures during the same flight, whether detected or latent, and combinations thereof, should also be assumed, unless their joint probability with the first failure is shown to be extremely improbable.”

The fail safe design concept implies the application of fault tolerant design approaches including:

- Redundancy or Backup Systems, Monitors
- Isolation of Systems, Components, and Elements
- Designed Failure Effect Limits
- Designed Failure Path
- Fault and Error Tolerance

4.2 Design criteria impact on fault tolerant architecture

Fault tolerance and architecture inherently underpin the application of the fail safe design concept and the aforementioned design approaches. Fault tolerance exists in several different guises depending on the level of system abstraction the system is being examined from. Broadly, though, fault tolerance mechanisms can be classified as one of the following:

- system level fault tolerance – mechanisms usually provided at a system or line replaceable unit (LRU) level to provide tolerance to sub-system faults (noting that the sub-system fault may be caused by a factor external to the system);
- hardware implemented fault tolerance – implementation of system level fault tolerance mechanisms by hardware;
- software implemented fault tolerance – implementation of system fault tolerance mechanisms by software; and
- software fault tolerance – mechanisms provided at software level for containing or mediating software errors, faults and failures.

Table 1 summarises commonly used fault tolerance mechanisms in fault tolerant systems as sourced from the examination of actual aviation systems, as well as [HiM01] and [Ham01].

5 Observations from Examination of Aviation Systems and Fail Safe Design Criteria

This section applies the fail safe design criteria discussed in Section 4 to the actual aviation systems examined in Section 3 for the purposes of eliciting the commonalities in handling of systematic failures by the actual aviation systems. Bringing this into the context of systematic faults such as those identified using [Pum99]'s taxonomy and treated using those approaches articulated in [Wea03] (by absence or detection/handling), it is evident that the fail safe design criteria heavily influences the treatment strategies for systematic faults.

The following sub-sections examine this in specific detail.

System Level Fault Tolerance	Hardware Implemented Fault Tolerance	Software Implemented Fault Tolerance	Software Fault Tolerance
<ul style="list-style-type: none"> • Simplex, no fault tolerance • Simplex, with disengagement features • Dual standby • Self checking pair (single or dual) • Self checking pair with simplex fault down • Triple modular redundancy <ul style="list-style-type: none"> ○ fault down to self checking pair or fault down to simplex 	<ul style="list-style-type: none"> • Redundancy • Dissimilar Hardware • Distinct Hardware • Command / Monitors • Voter Comparators <ul style="list-style-type: none"> ○ Average ○ Middle Value Selection ○ 2/3 Majority Vote • Watchdog Timers 	<ul style="list-style-type: none"> • Error Detection – recognition of the incidence of a fault <ul style="list-style-type: none"> ○ Replication Checks ○ Timing Checks ○ Reversal Check (Analytical Redundancy) ○ Coding Checks ○ Reasonableness Checks ○ Structural Checks ○ Diagnostic Checks • Damage Confinement / Fault Containment – restriction of the scope of effects of a fault • Damage Assessment – diagnosis of the locus of a fault • Error Recovery – restoration of a restartable service • Service Continuation – sustained delivery of system services • Fault Treatment – repair of a fault • Distributed Fault Tolerance 	<ul style="list-style-type: none"> • Multi-version software <ul style="list-style-type: none"> ○ N-version program ○ Cranfield Algorithm for Fault Tolerance (CRAFT) ○ Distinct and Dissimilar software • Recovery Blocks <ul style="list-style-type: none"> ○ Deadline mechanism ○ Dissimilar Backup Software • Exception Handlers <ul style="list-style-type: none"> ○ Hardened Kernels ○ Robust Data Structures and Audit Routines ○ Run Time Assertion • Hybrid Multi-version Software and Recovery Block Techniques <ul style="list-style-type: none"> ○ Tandem ○ Consensus Recovery Block

Table 1: Summary of Fault Tolerance Mechanisms

5.1 No Single Failure Criterion

[AC25.1309] effectively states that no single failure of a software or hardware component should lead to Major, Hazardous or Catastrophic failure condition. So the 25.1309 airworthiness requirement requires that consideration is given all single failures to show that they cannot directly result in the Major, Hazardous or Catastrophic failure conditions.

Therefore it must be assumed that when dealing with the presence of systematic faults, any given absence argument might be invalidated (due to unknown faults, irrespective of how well assured it might be), or that any given detection and handling argument might also be invalidated (due to unknown faults in the detection and handling mechanism, irrespective of how well assured it might be).

5.1.1 Invalidating Absence Arguments

If an absence argument is invalidated then only a detection and handling argument can address the invalidation. This is because once the absence argument is invalidated the failure is now assumed to have occurred in the software and it is no longer absent. While appropriate system architecture might mask that fault at higher levels of abstraction thus making it (or its effects) absent, the system will inevitably employ detection and handling to achieving this masking.

This subsequent detection and handling argument might be made later in the software (provided invalidating the absence argument doesn't also lead to invalidation of its detection and handling in software), at a LRU level (hardware or software implemented) or system architecture level.

5.1.2 Invalidating Detection and Handling Arguments

If a detection and handling argument is invalidated then only a detection and handling argument later in the software (provided invalidating the original detection and handling argument doesn't also lead to invalidation of its detection and handling in software), or at an LRU level (hardware or software implemented) or system architecture level can address the invalidation. Note it only takes either the detection OR the handling argument to at least be invalidated to invalidate the whole detection and handling argument. Detection may be at a different level of abstraction to the handling – although most often handling is at the same or higher level than the detection feature.

5.1.3 Impact on Argument of No Single Failure Criterion

The no single failure criterion therefore places serious constraints on the structure of the argument for Major, Hazardous and Catastrophic failure conditions. Table 2 identifies the effect of these constraints on the number of arguments typically necessary for any given failure modes – and at what level within the system the argument typically addresses the failure mode, as determined from the aviation systems considered in Section 3.

Note that the previous paragraphs have focussed on the Major, Hazardous and Catastrophic failure conditions. Implicitly the no single failure criterion also infers that a single failure can acceptably lead to a Minor or No Safety Effect failure condition. Thus it is also possible to represent these failure conditions in Table 2.

The software columns in Table 2 both refer to the same configuration item. This is because it is possible to make an initial absence or detection/handling claim within the software and then provide the detection and handling capability at a later point in the functional flow, or architecturally within the software. The second software column should not be interpreted as a separate configuration item, perhaps resident in a monitor for example. This is considered at the LRU level.

Severity	Software%	Software			LRU Level		System Level
Catastrophic , Hazardous / Major	Absence (Primary, Secondary, and Control)	AND	Detection AND Handling*	OR	Detection AND Handling	OR	Detection AND Handling
			Detection*	AND	Handling	OR	Handling
			-	-	Detection	AND	Handling
	Detection AND Handling	AND	Detection AND Handling#	OR	Detection AND Handling	OR	Detection AND Handling
			Detection#	AND	Handling	OR	Handling
			-	-	Detection	AND	Handling
Minor, No Safety Effect	Absence (Primary, Secondary, and Control)	OR	-	-	Detection AND Handling	OR	Detection AND Handling
			-	-	Detection	AND	Handling
	Detection AND Handling	OR	-	-	Detection AND Handling	OR	Detection AND Handling
			-	-	Detection	AND	Handling
% - initiating fault invalidates this argument under no single failure criterion * - provided invalidating the absence argument doesn't also lead to invalidation of its detection and handling in software # - provided invalidating the original detection and handling argument doesn't also lead in invalidation of its detection and handling in software Logical conventions: Logical operators and conditions within the same cell assume parenthesis <i>Italics</i> – evaluate the logical operator prior to normal type face and bold operators – assume parenthesis encapsulates the cells either side of the operator Bold – evaluate the logical operator last							

Table 2: No single failure criterion as inferred from actual aviation systems

To illustrate the intent of Table 2, a simple example will be considered. Consider a system with a catastrophic failure condition. Table 2 infers that there are two approaches the system designer could use to address the no single point of failure criterion. The first is to assure both the absence (Table 2 row 2, column B¹) or direct/immediate detection and handling (Table 2 row 3, column B) of the initiating fault and provide a supplemental detection and handling mechanism at a higher software level (column D), LRU level (column F) or system level (column H). Thus resulting in at least two failures being required to realise the catastrophic failure condition.

However, this is only the first criterion we need to examine, the next section considers further constraints on these identified effects, and will likely further constrain Table 2.

5.2 Combinations of Failure Criterion

[AC25.1309] also effectively states that no additional combinations of failures of software and/or hardware components should lead to Major, Hazardous or Catastrophic failure unless their joint probability is Extremely Improbable.

As Extremely Improbable can never be defensibly argued for any single component, Extremely Improbable is traditionally argued by the following combinations of failure

¹ Row identifiers are positive integer values starting at Row 1 for the Heading row of Table 2. Column identifiers are alphabetic values starting at Column A for the Severity Column.

likelihoods - Extremely Remote AND Remote, Extremely Remote AND Probable, Remote AND Remote, Probable AND Probable AND Probable (using 14CFR25.1309 terminology). All these statements are based on the presumption of independence between elements of the design. It is this criterion that typically leads to at least triple redundancy in hardware systems with catastrophic failure conditions and dual redundancy in most hardware systems with hazardous or major failure conditions (as is apparent in the aviation systems examined in Section 3).

However for software, probability and likelihoods have traditionally had little relevant meaning, because software failures are systematic (refer to Section 2.1). Therefore we need to resolve an equivalent interpretation for the first sentence of this section that doesn't infer probabilities.

One way is to speculate that the joint likelihood of no two combinations of software initiated failures is ever commensurate with extremely improbable. The hypothesis, based on examination of actual systems, is that the burden of demonstrating this level of knowledge of the system or the stochastic model would generally be unattainable. Section 6 provides further discussion on knowledge and uncertainty. Therefore, no two software initiated failures should lead to a catastrophic failure condition. This implies that there is at least sufficiently independent detection and handling of the initiating software failure mode within the software itself, and at the LRU level or system architecture level for catastrophic failure conditions. For major and hazardous failure conditions, two software initiated failures may be tolerable (because the consequences require another event to be contributory to a catastrophic failure condition); provided they are sufficiently independent of each other. The independence is most practically achieved by detecting and handling the faults at a level outside the software. Overall, the hypothesis is reasonable, and broadly comparative to the outcome for probabilistic hardware failure assessments. It is also supported by the examination of the aviation systems discussed in Section 3.

For combinations of three software failure modes, these may be considered extremely improbable, provided there is detection and handling of software failure modes outside of the software in question (i.e. at either the LRU level or system architecture level). With each layer of detection and handling mechanisms, the burden of demonstrating this level of knowledge of the system or the stochastic model is more reasonably attainable. Again the result is broadly comparative to the outcome for hardware failures, and is supported by the examination of the aviation systems discussed in Section 3.

Section 6 examines the effects of layered detection and handling mechanisms on uncertainty in the stochastic model in further detail.

5.2.1 Invalidating the Absence Argument and the Detection/Handling Argument – Catastrophic Only

In this case we are not only going to invalidate the absence argument, but also the detection/handling argument that provides the treatment to the invalidation of the absence argument. This leads to it being necessary to detect and handle the failure mode outside of the software – either by the LRU (e.g. through a monitor) and/or by the system architecture level (e.g. through combinations of redundancy, analogue backup, diverse system components, etc), or both the LRU and system architecture levels. These results are supported by the examination of the aviation systems in Section 3.

5.2.2 Impact on Argument of Combinations of Failure Criterion

The combinations of failures criterion therefore place some further constraints on the structure of our argument for Major, Hazardous and Catastrophic failure conditions.

Table 3 identifies the effect of these constraints on the number of arguments required for any given failure modes – and at what level within the system the argument typically addresses the failure mode, as determined from the aviation systems considered in Section 3.

Severity is the accident effect if the LRU Level and System Level mechanisms were absent, or the software fault was permitted to propagate without intervention at the LRU Level and System Level.

Severity	Software%	Partitioned Software			LRU Level		System Level
Catastrophic	Absence (Primary, Secondary, and Control)	AND	Detection AND Handling&*	OR	Detection AND Handling&	AND	Detection AND Handling&
			Detection AND Handling&*	AND	Detection AND Handling&	OR	Detection AND Handling&
	Detection AND Handling	AND	Detection AND Handling&#	OR	Detection AND Handling&	AND	Detection AND Handling&
			Detection AND Handling&#	AND	Detection AND Handling&	OR	Detection AND Handling&
Hazardous / Major	Absence (Primary, Secondary, and Control)	AND	Detection AND Handling&*	OR	Detection AND Handling	OR	Detection AND Handling
	Detection AND Handling	AND	Detection&#	AND	Handling	OR	Handling
			-	-	Detection	AND	Handling
Minor, No Safety Effect	Absence (Primary, Secondary, and Control)	OR	-	-	Detection AND Handling	OR	Detection AND Handling
			-	-	Detection	AND	Handling
	Detection AND Handling	OR	-	-	Detection AND Handling	OR	Detection AND Handling
			-	-	Detection	AND	Handling
% - initiating fault invalidates this argument under no single failure criterion & - additional faults may invalidate these arguments under combinations of failure criterion * - provided invalidating the absence argument doesn't also lead to invalidation of its detection and handling in software # - provided invalidating the original detection and handling argument doesn't also lead in invalidation of its detection and handling in software Logical conventions: Logical operators and conditions within the same cell assume parenthesis <i>Italics</i> – evaluate the logical operator prior to normal type face and bold operators – assume parenthesis encapsulates the cells either side of the operator Bold – evaluate the logical operator last							

Table 3: Combinations of failure criterion as inferred from actual aviation systems

5.3 Specific Circumstances for Absence Arguments

Absence arguments (for omission, commission, early, late and value) are never valid for input data (data originating outside the software of the LRU) to software within an LRU – these types of faults should be detected and handled at the input to the

software, as is done by most aviation systems; or by ensuring that the fault propagates to a detectable fault at a higher system level. Detection will usually need to be more extensive than simply checking the valid flag provided with the data from the sensor because this doesn't provide detection of timing or omission related failures, and because the valid flags coverage of credible value failures is often very limited. Typically a combination of range, rate, physical world checks, or comparison to a redundant or diverse source are required.

While the detection and handling of this class of faults may be deferred until later in the system functional flow, this is rarely suitable. For example, in Flight Control Systems, there are minimal benefits to processing control laws based on invalid input data and then attempting to trap the failure at the system's output or control actuator. This is because the vast majority of input data failures are not easily discernible at this point in the system. The only times it might be suitable is if through physical limiting (e.g. mechanical limiting) the Flight Control System's authority is limited to a worse credible failure severity of minor (clearly not applicable to full authority systems).

6 Bounding Uncertainty

A key factor in proposing the layering of detection and handling mechanisms at different levels of abstraction in a system (e.g. software level, partitioned software level, LRU level, system level) is that it permits the uncertainty associated with detecting and then providing a suitable handling response to the fault to be bounded to an amount that is useful for reasoning about knowledge and the safety of the system. This section examines how architecture is used to bound uncertainty.

6.1 Using Architecture to Bound Uncertainty

To examine the effect of architecture on uncertainty, consider a series of cascading faults in a system with detection and handling mechanisms at the software, LRU and system levels.

At the occurrence of the 1st fault at the software level (i.e. Failure of 1st Absence / Detection and Handling Mechanism), the knowledge that there is a valid mechanism is a function of the following:

- the understanding of types of failure that might occur (i.e. to what extent is an appropriate mechanism provided to achieve coverage of all classes of the taxonomy of potential software failure modes, noting that the lower the level faults are examined at, the greater the number of practical classes in the fault taxonomy); and
- the appropriateness of absence / detection and handling mechanisms given the specific known fault that has occurred.

The uncertainty is a function of the following:

- the extent to which the taxonomy of potential software failures modes is incomplete for the specific failures that could occur in the system (i.e. are there sources of failure that haven't been understood?);
- the immediate effect of failure sources that haven't understood been (i.e. is the effect something that has been left unanticipated, even in a generalised sense?); and

- the suitability (or unsuitability) of the extant absence or detection/handling mechanisms for these unknown sources of failure (i.e. is the mechanism going to do something undesirable in the presence of an unknown fault?).

Therefore, for the 1st fault with no detection/handling mechanisms, uncertainty is unbounded and will tend to infinity. Even if a detection/handling mechanism is employed, the ratio of uncertainty to knowledge may still tend to be very large depending on the extent of the fault coverage by the mechanism. This poses problems for failures with severe consequences.

At the occurrence of the 2nd fault, this time at the LRU level (i.e. Failure of the 2nd Detection/Handling Mechanism), the knowledge that there is the valid detection and handling mechanism is a function of the following:

- the extent to which the taxonomy of failures should resolve the failures of the 1st mechanism, which should be finite at this level (the existence of the detection / handling mechanism is explicitly having to detect classes of failure of the 1st mechanism);
- the degree to which it is possible for the 2nd detection / handling mechanism to be activated from the cascading fault condition;
- the appropriateness of the absence / detection and handling mechanism at the LRU given the specific known fault class that has occurred (i.e. is the behaviour of the mechanism valid at this level of abstraction); and
- the coverage of intended coupling paths between software and LRU level mechanisms.

The uncertainty is a function of the following:

- the extent to which the cascading faults don't resolve to the taxonomy of faults handled at this layer;
- the suitability (or unsuitability) of absence or detection/handling mechanisms for unknown sources of failure, and its effects; and
- the extent to which unintended independence violators might be active (but should be limited by the degree of physical partitioning).

At the occurrence of the 3rd fault, this time at the System level (i.e. Failure of the 3rd Detection/Handling Mechanism), the knowledge and uncertainty parallel the observations listed above for the 2nd mechanism, with the following key differences:

- the extent to which the taxonomy of failures at the System level resolves the failures of the 2nd mechanism should be better than at the 2nd level as the number of classes of failures the cascading faults need to resolve to should be decreasing (with ultimate convergence at two failures modes – i.e. loss of the function and malfunction of the function);

Thus it is possible to see that ultimately each additional detection and handling mechanism layer bounds the uncertainty to the extent to which the cascading faults from the lower level resolve to the taxonomy of faults handled at the current layer. The effect of abstraction with each layer is that the taxonomy of faults at that each higher layer will resolve to a smaller complete set, until at the system boundary we are left with just the 'loss of' and 'malfunction' aircraft failure conditions. It is this

ability of the architecture to resolve faults and failures to a smaller set of practically considered failure conditions that bounds uncertainty.

Summarising the effects of bounding uncertainty, as follows:

- With no absence or detection/handling mechanisms, uncertainty is unbounded and will tend to infinity. Therefore this type of architecture should only ever be employed when there is no safety effect.
- With one (1) absence or detection/handling mechanism, uncertainty may still tend to be very large depending on the extent of the fault coverage. Therefore, a system with only one mechanism layer must not have severe failure modes.
- With two (2) layers of mechanisms, uncertainty may be very large, but it is likely much less and will often tend towards a finite value depending on the extent to which the classes of cascading faults resolve to the taxonomy at the second layer. Therefore a system with two mechanism layers is suitable for any system except for those with the most severe failure modes, provided the right mechanisms are employed at each layer of course.
- With three (3) layers of mechanisms, uncertainty may be large, but it is likely much less and will often tend towards a small finite value depending on the extent to which the cascading faults resolve to the taxonomy at the second and third layers. Therefore a system with three mechanism layers is suitable for any system, even those with severe failure modes, provided the right mechanisms are employed at each layer of course.
- Additional mechanisms may bound the uncertainty further, provided they continue to enforce the resolving of fault classes to those analysed and treatable at the subsequent mechanisms layer.

Therefore, the bounding of uncertainty provides conceptually a compelling case for structuring specific layers of absence and detection/handling for treating systematic faults. Combining this principle with the observations from aviation system permits architectural assurance requirements to be inferred. Section 7 examines this approach.

7 Assurance of Architecture

Based on commonalities in the treatment of systematic faults identified in the examination of actual aviation systems (Section 3), the application of the fail safe design criteria (Sections 4), the commonalities of the fail safe design criteria in the actual systems for systematic failures (Section 5), and examining how these factors contribute to bounding the uncertainty of the effects of systematic failures on a system (Section 6), this section proposes a framework for quantifying the assurance of safety architecture in aviation systems.

7.1 ASAL concept

This paper proposes a framework based around the concept of an Architectural Safety Assurance Level (ASAL). Note that the ASAL described in this paper is not related to any of the architectural related assurance level concepts being proposed by the [ARP4754] committee currently undertaking revision of the standard. The ASAL provides direct quantification (and benchmarks) of the extent to which the system's architecture is tolerant to systematic faults. The degree of fault tolerance is

directly associable with the normal Aircraft Failure Condition Severities defined by standards such as [ARP4754]. Four ASAL levels are proposed and presented in Table 4.

Failure Condition Severity ¹	Architectural Safety Assurance Level (ASAL)	Systematic Fault Tolerance
Catastrophic	ASAL3	At least three (3) diverse ² systematic faults are necessary for the aircraft failure condition to be realised
Hazardous / Major	ASAL2	At least two (2) diverse ² systematic faults are necessary for the aircraft failure condition to be realised
Minor	ASAL1	At least one (1) systematic fault is necessary for the aircraft failure condition to be realised
No Safety Effect	ASAL0	Systematic fault tolerance is not required, however the designer may choose to incorporate fault tolerance to provide assurance of system availability and reliability
<p>1. The worst credible failure condition severity of <u>loss of</u> and <u>malfunction of</u> the aircraft function with which the system and its software is associated.</p> <p>2. For a systematic fault to be diverse of another systematic fault, it must be shown that the activation of one fault does not automatically lead to the activation of another systematic fault. In practice this is achieved by ensuring that the faults must occur in independent components and/or at differing layers of abstraction (e.g. software, LRU, system) where the correct functioning of the subsequent detection and handling mechanisms can be shown to be independent of the initiating fault condition or the detectable class of fault at the next layer is distinct of the initiating class of fault.</p>		

Table 4: Architectural Safety Assurance Level

7.2 Absence and Detection/Handling Mechanism Requirements

A key factor in providing for diverse systematic faults identified in Table 4 is providing detection and handling mechanisms at differing layers of abstraction with a system. This allows the independence of the functioning of the detection and handling mechanism to be achieved physically in the system design, and it also bounds the uncertainty of fault coverage of these mechanisms.

Using the taxonomy of layers of detection and handling mechanisms identified in Section 5, the proposed framework uses the following levels of detection and handling mechanisms:

- Software – at the typical software component level (and software component in question), and includes software fault tolerant features and software implemented fault tolerance features.
- Line Replaceable Unit (LRU) – at the typical avionics box level within an aircraft, and includes fault tolerant features such as:
 - command/monitors (note additional software in the monitor is considered at the LRU level, although the software safety argument for that monitor software CSCI would also consider its effects at the software component level),
 - voting planes,

- output wraparounds (although the feedback is usually hardware implemented, the comparison is usually software implemented),
- hardware BIT, etc.
- System Architecture Level – at the typical system architecture level within an aircraft and may include redundancy, analogue backup, diverse system components, etc. Note that redundant components running the same software configuration only provides protection against hardware related failures, or failures of independent input sensors. It provides no protection against systematic failures of the software. The emphasis here is on system level architectural features that provide protections against systematic software failures by detection and handling of faults.

Relating the levels of detection and handling mechanisms to the ASAL concept defined in Section 7.1 provides a framework as defined in Table 5.

ASAL	1st Absence/Detection and Handling Mechanism	2nd Detection/Handling Mechanism	3rd Detection/Handling Mechanism
ASAL3	Software Level	Partitioned Software Level [#] or LRU Level [*]	LRU Level [*] or System Level
ASAL2	Software Level	Partitioned Software Level [#] or LRU Level or System Level	Not Required
ASAL1	Software Level OR LRU Level OR System Level	Not Required	Not Required
# must be independent of the initiating failure and the 1st Absence / Detection and Handling mechanism (i.e. through a partitioning mechanism)			
* must be independent of the proceeding detection/handling mechanism			

Table 5: ASAL Architecturally Layered Fault Tolerance

7.3 Benefits of ASAL concept

The ASAL concept provides the following perceived benefits to assurance frameworks:

- The ASAL concept explicitly integrates requirements for architectural treatments to systematic faults into the traditional assurance approach, and is compatible with the existing safety analysis of [ARP4754] and other similar standards.
- The ASAL concept provides a multidimensional (better than binary) perspective on the absence and detection/handling of systematic faults commensurate with the worst credible failure condition.
- The ASAL concept quantifies (in the product context) the degree of fault tolerance within a system and its software for each system's contribution to aircraft level failure conditions. Therefore, the ASAL as a level inherently has a product meaning.
- The ASAL concept is simple, and therefore doesn't burden assurance frameworks with complex, non-objective prescriptions.

- The ASAL concept doesn't prescribe specific architectures, and is therefore, inherently flexible. It instead focuses on the treatment of systematic faults by the architecture.
- The ASAL concept encourages fault tolerance architectures for the systems whose functions most need fault tolerance (i.e. those with the most severe hazards or failure conditions)
- The ASAL concept is analytically compatible with observations of systematic fault tolerance management in actual aviation systems.

7.4 Limitations of ASAL concept

The ASAL concept introduces or highlights the following potential limitations:

- The explicit integration of the ASALs with software assurance standard (e.g. RTCA/DO-178B) objectives hasn't yet been clarified.
- The ASAL concept sets no benchmarks for the level of evidence required to demonstrate that numbers of diverse systematic faults do not contribute to identified failure modes. The ASAL concept does not address 'how much is enough?' for software evidence.
- The ASAL concept relies on bounding uncertainty, of which a fundamental factor is the extent to which faults at one layer of abstraction resolve to a detectable set at the next layer of abstraction. However, the ASAL concept doesn't provide an explicit measure of the specific contextual claims about detecting and handling systematic faults as they propagate to high levels of system abstraction, and thus support inferences about the suitability of the proposed detection and handling capabilities of the system architecture.

Section 8 of this paper provides insight into several approaches being explored as part of this research work that are intended to address these limitations. Empirical evaluation of the ASAL concept is proposed in later phases of this research.

7.5 Additional factors

The following paragraphs provide insight into a number of factors relevant to ASAL.

7.5.1 Conceptual and Mechanistic Independence

Conceptual and mechanistic independence have been suggested by [Wea03] as playing an important factor in assurance of arguments constructed around [Pum99]'s software failure taxonomy. However, how does conceptual and mechanistic independence relate to the ASAL concept defined in this paper? The definitions within the ASAL concept specify several diverse faults. This implies that there is conceptual independence between the initiating software, the LRU level and system level detection and handling mechanisms (where relevant). Systems sharing common software and/or hardware may be prone to common mode failure conditions and are not considered to be diverse. Unless mechanistic independence delivers conceptually different architectures during the design process, it does not play a role in the ASAL concept directly. Mechanistic independence will be considered in the work described at Section 8.

7.5.2 On-demand versus Continuous-demand Systems

The ASAL concept was largely derived in the context of actual aviation systems that are inherently continuous demand systems, although specific functions provided by individual safety functions may be deemed as on-demand. Therefore, does the ASAL concept apply for on-demand systems versus continuous demand systems?

On-demand systems (usually used for protection systems) are usually associated with an availability requirement (therefore continuous demand) on a related aviation system associated with the protection mechanism. Therefore in most cases there is little practical difference between an on-demand system and continuous demand system with respect to the ASAL concept.

8 Assurance of Architecture and the Relationship to Software Failure Claims/Arguments and Evidence Sufficiency

This paper has proposed an assurance framework that provides a direct measure of the degree of the system's fault tolerance against systematic faults and failures, and thus infers the system's suitability for use in the presence of aircraft level failure conditions of differing severities. However, the extent of discussion in this paper has only addressed the architectural effects of layered detection and handling mechanisms on bounding uncertainty of systematic faults. This paper has not addressed which specific detection and handling mechanisms are most appropriate in each context, and how claims to that effect might be assured. Furthermore, this paper hasn't set any benchmarks for the provision of evidence in this regard.

To address these questions, further papers are being developed which propose a Claims (CSAL) and Evidence (ESAL) Safety Assurance Level concept that is compatible with the ASAL concept identified in this paper. The core idea behind claims assurance is to ensure that any assurance levels used for articulating claims assurance in the context of the ASAL have a specific product safety focus (i.e. each and every assurance level has a product meaning, not just a top-down process interpretation). For evidence assurance, the core idea is to provide a framework that is explicit in a product sense of the 'tolerability of limitations' in satisfying the objectives articulated in the framework.

9 Summary

This paper has identified limitations with the current standards' frameworks with respect to architecture and systematic faults and failures. Treatments of systematic faults in a number of actual aviation systems have been examined, including civil and military Automatic Flight Control Systems (AFCS) and Flight Management Systems (FMS). The results of this examination have been contrasted with the fail safe design criteria underpinning the 1309 airworthiness requirements for civil aircraft certification, and commonalities in the treatment of systematic failures identified.

Using the identified commonalities, and examining how these factors contribute to bounding the uncertainty of the effects of systematic failures on a system, a framework has been proposed for quantifying the assurance of safety architecture in aviation systems. The assurance framework provides a direct measure of the degree of the system's fault tolerance against systematic faults and failures, and thus infers the system's suitability for use in the presence of aircraft level failure conditions of differing severities.

Further papers are being developed which propose a Claims (CSAL) and Evidence (ESAL) Safety Assurance Level concept that is compatible with the ASAL concept identified in this paper.

10 References

The following documents, papers and publications are referenced throughout this paper. A number of these documents are not available in the public domain for propriety or confidentiality reasons. Readers wishing to seek further information should direct their queries to the author of this paper, or the relevant standards body.

- [A330] Airbus A330 Flight Deck and Systems Briefing for Pilots, STL 472.755/92 issue 4, March 1999.
- [AAP7211.031-1] Australian Air Publication, "Flight Manual C-130J-30", Royal Australian Air Force, 29 Nov 2005.
- [AC25.1309] Federal Aviation Administration (FAA) Advisory Circular AC25.1309-1A, "System Design and Analysis", 21 Jun 1988.
- [ALR04] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, Vol 1, No. 1, Jan-Mar 2004.
- [ARP4754] SAE International, "Aerospace Recommended Practice 4754 – Certification Considerations for Highly Integrated or Complex Aircraft Systems", November 1996.
- [ARP4761] SAE International, "Aerospace Recommended Practice 4761 – Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment", December 1996.
- [BMO95] H. Buus, R. McLees, M. Orgun, E. Pasztor, L. Schultz, "777 Flight Controls Validation Process, IEEE 0-7803-3050-1/95, 1995.
- [BrT93] D. Briere, P. Traverse, "AIRBUS A320/A330/A340 Electrical Flight Controls – A Family of Fault Tolerant Systems", IEEE 0731-3071/93, 1993.
- [DeJ88] M.K. DeJonge, "Time Controlled Navigation and Guidance for 737 Aircraft", IEEE CH2596-5/88/0000-0546, 1988.
- [DO178B] RTCA Inc., "RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.: RTCA Inc., 1992.
- [DO248B] RTCA Inc., "RTCA/DO-248B: Final Report for Clarification of DO-178B – Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.:RTCA Inc., 2001.
- [DrH92] K. Driscoll, K. Hoyme, "The Airplane Information Management System: An Integrated Real-time Flight-deck Control System", IEEE 1052-8725/92, 1992.
- [GiS99] M. Girard, P. Sharpe, "F/A-18A Testing of Flight Control System Reversion to Mechanical Backup", IEEE, 1999.
- [Ham01] R. Hammett, "Design by Extrapolation – An Evaluation of Fault Tolerant Avionics", The Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts, 2001.

- [HiM01] E.F. Hitt, D. Mulcare, "Fault-Tolerant Avionics" Chapter 28 in "Digital Avionics Handbook", Edited by Cary R. Spitzer, CRC Press, 2001.
- [Hor94] R. Hornish, "777 Autopilot Flight Director System", IEEE 0-7803-2425-0/94, 1994.
- [JTM07] D. Jackson, M. Thomas, L. Millet, Editors, "Software for Dependable Systems: Sufficient Evidence?", Committee of Certifiably Dependable Software Systems, National Research Council, National Academy of Sciences, USA, 2007.
- [KSQ92] B.W. Kowal, C.J. Scherz, R. Quinlivan, "C-17 Flight Control System Overview", IEEE AES Magazine, pp24-31, July 1992
- [Mar08] P. Marks, "Flight of the Software Bugs", in New Scientist, pp26-26, 9 Feb 2008.
- [McD01] J.A. McDermid, "Software Safety: Where's the Evidence?", Department of Computer Science, University of York, 2001.
- [McD07] J.A. McDermid, "Risk, Uncertainty, Software and Professional Ethics", 20 August 2007.
- [McK06] J. McDermid, T. Kelly, "Software in Safety Critical Systems: Achievement and Prediction", Nuclear Future, Volume 03, No. 03, 2006.
- [McP01] J. McDermid, D. Pumfrey, "Software Safety: Why is there no Consensus?", Department of Computer Science, University of York, 2001.
- [NTS06] National Transportation Safety Board, "Safety Report on the Treatment of Safety-Critical Systems in Transport Airplanes", Safety Report NTSB/SR-06/02, Washington, D.C., USA, 2006.
- [PoK92] D.J. Pop, R.L. Kahler, "C-17 Flight Control Systems Software Design", IEEE 0-7803-0820-4/92, 1992.
- [Pum99] D.J. Pumfrey, "The Principled Design of Computer System Safety Analyses", PhD Thesis, Department of Computer Science, University of York, 1999.
- [Rei08] D.W. Reinhardt, "Considerations in the Preference for and Application of RTCA/DO-178B in the Australian Military Avionics Context", presented at the 13th Australia Workshop on Safety Critical Systems and Software, Aug 2008.
- [Sch95] J. Schruer, "B737 Flight Management Computer Flight Plan Trajectory Computation and Analysis", in Proceedings of the American Control Conference, June 1996.
- [Spi01] C. Spitzer, "The Avionics Handbook", AvioniCon, Inc. Williamsburg, Virginia, CRC Press, 2001.
- [Ucz95] J.S. Uczekaj, "Reusable Avionics Software – Evolution of the Flight Management System, IEEE 0-7803-3050-1/95, 1995.
- [Wea03] R.A. Weaver, "The Safety of Software – Constructing and Assuring Arguments", PhD Thesis, Department of Computer Science, University of York, 2003.
- [Wei92] P. Weindorf, "The C-17 Multifunction Display – A Building Block for Avionic Systems", IEEE AES Magazine, pp32-39, July 1992.

- [Wit95] B. Witwer, "Systems Integration of the 777 Airplane Information Management Systems (AIMS): A Honeywell Perspective, IEEE 0-7803-3050-1/95, 1995.
- [Yea96] Y.C. Yea, "Triple-Triple Redundant 777 Primary Flight Computer", IEEE 0-7803-3196-6/96, 1996.
- [Yea01] Y.C. Yea, "Safety Critical Avionics for the 777 Primary Flight Controls System", IEEE 0-7803-7034-1/01, 2001.

ANNEX A – EXAMINATION OF AVIATION SYSTEMS

FLIGHT CONTROL SYSTEMS

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-17A
Primary Computers	<p>Three (3) Primary Flight Computers (PFC) – digital</p> <ul style="list-style-type: none"> process Normal, Secondary and Direct laws actuation commands transmitted to ACE execution of automated functions such as yaw damper system monitoring, crew annunciation, and onboard maintenance capabilities <p>Four (4) Actuator Control Electronics (ACE) – analog</p> <ul style="list-style-type: none"> interface with the pilot control transducers and to control the Primary Flight Control System actuation with analog servo loops 	<p>Three (3) Flight Control Primary Computers (FCPC) - digital</p> <ul style="list-style-type: none"> process Normal, Alternate and Direct Laws one FCPC is selected as Master: it processes the orders and outputs them to the other computers which will execute them on their servo loops Master checks that its orders are fulfilled by comparing them with feedback received; self monitoring of the master can detect a malfunction and cascade control to the next computer each FCPC can control up to eight (8) servo loops and provide complete aircraft control under normal laws. 	<p>Quad redundant digital flight control system incorporating two (2) flight control computers with two independent channels per computer process:</p> <ul style="list-style-type: none"> control stick, rudder pedal and trim commands pitot static, rate gyro, accelerometer, AOA probe and fight control surface position feedback signals, and send commands to each flight control surface actuator. 	<p>Quadruplex set of digital flight control computers</p> <p>Four channel synchronous operation</p> <p>All output hardware, signals, and feedback are monitored and compared to ensure failure detection and channel output voting.</p> <p>Dedicated cross-channel data link used between channels</p> <p>Input signal voting</p> <p>Actuator loop voting</p>
Secondary Computers	<ul style="list-style-type: none"> ACEs convert the transducer position into a digital value and then transmit that value to the PFCs ACEs then convert PFC commands into analog commands for each individual actuator flight control surface servo loops are distributed among the four ACEs. 	<p>Two (2) Flight Control Secondary Computers (FCSC) - digital</p> <ul style="list-style-type: none"> Are able to process direct laws Either secondary can be the master in the case of loss of all FCPC Each FCSC can control up to 10 servo loops and can provide complete aircraft control 		

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-17A
Additional Control Computers	<p>Other systems:</p> <ul style="list-style-type: none"> Flap Electronics Unit (FSEU) Slat Unit Proximity Switch Electronics Unit (PSEU) Engine Data Interface Unit (EDIU) <p>Airplane Information Management System (AIMS) Data Conversion Gateway (DCG) maintains separation between the critical flight controls busses and the essential systems busses.</p>	<p>High life devices are commanded by two Slat/Flap Control Computers</p> <p>Two (2) Flight Control Data Concentrators (FCDC) acquire the outputs from the various computers to be sent to the ECAM and Flight Data Interface Unit to provide isolation of the flight control computers from other systems.</p>	No additional flight computers	Two dual digital Spoiler Control/Electronic Flap Computers
Computer Architecture	<p>Each PFC has three identical computing "lanes"</p> <ul style="list-style-type: none"> a voting plane scheme is used by the PFCs on themselves. single computing lane within a PFC channel is declared as the "master" lane. all three lanes simultaneously computing the same control laws. the outputs of all three lanes are compared against each other. any failure of a lane that will cause an erroneous output from that lane will cause that lane to be "failed" by the other two lanes. Command Lane, Standby Lane, Monitor Lane. 	<p>Command/ Monitor computer architecture for both the FCPC and FCSC.</p> <ul style="list-style-type: none"> Monitor channel monitors for health of the command channel and control surface runaway Specific variables are permanently compared in the two channels. sensor inaccuracy, rigging tolerances, computer asynchronisation are taken into account errors which are not detectable (within the signal and timing thresholds) are assessed in respect to their handling quality and structural loads effect in the event of a divergence between command and monitor solutions, the affected computer is disengaged and the next highest priority computer takes over 	Two independent channels per computer processor	<p>FCC and SCEFC each use 3 MIL-STD-1750A CPUs.</p> <ul style="list-style-type: none"> In the FCC one processor serves as an I/O processor, and the other two perform control law computations. In the SFEFC one processor serves as an I/O processor and the other two are configured as a self checking pair. <p>The AFCS control panel is implemented with four MIL-STD-1760A CPU configured as two self checking pairs.</p>

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-17A
Dissimilarity	<p>Dissimilarity between the PFC and ACE.</p> <ul style="list-style-type: none"> PFCs are identical digital computers ACE are identical analog devices <p>Dissimilarity between the Air Data and Inertial Reference Unit (ADIRU) and Standby Attitude and Air Data Reference Unit (SAARU)</p>	<p>Dissimilarity between FCPC and FCSC digital computer designs</p> <ul style="list-style-type: none"> different processor architectures and manufacturers different software between FCPC and FCSC and between command and monitor channels in each FCPC and FCSC <p>No dissimilarity between the Air Data and Inertial Reference Unit (ADIRU)</p>	No dissimilarity between flight control computers.	No dissimilarity between flight control computers.
Latent Failure Detection	Built in Test	Self-test and peripheral tests	Built in Test	Built in Test
Reconfiguration	<p>The outputs from all three PFC channels are compared.</p> <ul style="list-style-type: none"> Each PFC compares its output for each particular actuator, and with the same command that was calculated by the other two PFC channels. Each PFC channel does a mid-value select on the three commands, and that value is output to the ACEs. 	<p>When the active computer interrupts its operation, one of the standby computers almost instantly changes to active mode with no or limited jerk on the control surfaces.</p>	<p>Three (3) modes of operation</p> <ul style="list-style-type: none"> Control Augmentation System (CAS) – full digital capability including adaptive flight controls and stability augmentation. Direct Electrical Link (DEL) – provided in the event of primary CAS failure, no longer process input from failed rate gyros and/or accelerometers. Mechanical (MECH) – three or more channel failures, pitch roll sensor failures, failure of both servo-valves in one actuator, hydraulic starvation 	<p>All FCS critical inputs, processing and outputs are quad redundant (fail op, fail op, fail passive).</p> <p>The FCCs and SCEFCs operate as a frame synchronous set. In the event of loss of synchronisation, the computers will attempt to re-synchronise.</p> <p>A sensor selection algorithm derives a selected value for each signal as a function of the sensor failure states.</p> <ul style="list-style-type: none"> Average of middle two values (four valid signals) Midvalue of three signals Average of two signals

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-17A
Servos	<p>Actuators arrangements are as follows:</p> <ul style="list-style-type: none"> Elevators, ailerons, and flaperons are controlled by two actuators per surface, the rudder is controlled by three. Each spoiler panel is powered by a single actuator. The horizontal stabilizer is positioned by two parallel hydraulic motors driving the stabilizer jackscrew. <p>The actuation powering the elevators, ailerons, flaperons, and rudder have several operational modes: Active, Bypassed, Damped, Blocked.</p>	<p>Actuators arrangements are as follows:</p> <ul style="list-style-type: none"> Elevators, ailerons are controlled by two actuators per surface, the rudder is controlled by three. Each spoiler panel is powered by a single actuator. The horizontal stabiliser is positioned by two actuators. <p>Servo-jacks can operate in one of three control modes depending upon computer status and type of control surface: active, damping, centering. Normally one servo is active and one is damped on each control surface</p>	<p>Dual servo values in each actuator fed by both flight control computers and two independent hydraulic sources.</p> <p>Aileron and twin rudders are differentially scheduled.</p> <p>Trailing Edge Flaps, Leading Edge Flaps and Stabs are scheduled both collectively and</p>	<p>All four FCCs are connected to each actuator. Outputs from each FCC are summed at the Electro hydraulic Servo Values providing a voting node.</p> <p>Output signal management software function in each FCC compares local channel actuator data with cross channel data to detect, identify and remove local faults</p>
Envelope protection limiting /	<p>Computers provide the following protections:</p> <ul style="list-style-type: none"> Bank angle protection Turn compensation Stall and overspeed protection Pitch control and stability augmentation Thrust asymmetry compensation 	<p>Computers will prevent excessive manoeuvres and exceedance of the safe flight envelope.</p> <ul style="list-style-type: none"> Excessive load factors Overspeed Stall Extreme pitch angle Extreme bank angle 	<p>Conventional envelope protections not provided in a fighter jet</p>	<p>The following protections are provided:</p> <ul style="list-style-type: none"> Angel of attack limiting system Deep stall avoidance All engine out control Safe go-around

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-17A
Sensors	<p>Dual redundant air data and inertial systems:</p> <ul style="list-style-type: none"> Air Data and Inertial Reference Unit (ADIRU) Standby Attitude and Air Data Reference Unit (SAARU) Autopilot Flight Director Computers (AFDC) <p>All critical interfaces into the Primary Flight Control System use multiple inputs which are compared by a voting plane.</p>	<p>Triple redundant air data and inertial:</p> <ul style="list-style-type: none"> Three air data and inertial reference units (ADIRUs) Accelerometers and rate gyros 	<p>The following sensors are used by the Flight Control Computers:</p> <ul style="list-style-type: none"> Pitot Static Rate Gyro Accelerometer, AOA probe And flight control surface position transducers 	<p>Quadruplex sensors including:</p> <ul style="list-style-type: none"> Stick and Peal Force Sensors Stick Position Surface Position Air data and stabiliser sensors Air Data Computers Inertial Reference Unit <p>6 AOA sensors</p> <p>Remaining sensors are dual.</p> <p>Inputs are voted, monitored, selected and sent to each processing channel before use in output signal processing.</p> <p>Digital inputs have validity bits</p>
Mechanical Backup	<p>Two spoiler panels and alternate stabiliser pitch trim are mechanically controlled</p>	<p>Mechanical backup: rudder and trimmable horizontal stabiliser – no artificial stabilisation required</p>	<p>Mechanical backup/linkage to the horizontal stabilators.</p>	<p>Backup mechanical system provides control of the ailerons, elevators, rudders and stabilizer surfaces.</p>

Table 6: Overview of design features supporting detection and handling of systematic faults in Flight Control Systems

FLIGHT MANAGEMENT SYSTEMS, NAVIGATION

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-130J
Flight Management Computers	<p>Dual integrated cabinets which provide the processing and the I/O hardware and software required to perform the following functions:</p> <ul style="list-style-type: none"> • Flight Management • Display • Central Maintenance • Airplane Condition Monitoring • Communication Management (including flight deck communication) • Data Conversion Gateway (ARINC 429/629 Conversion) <p>The applications hosted on AIMS are listed below, along with the number of redundant copies of each application per shipset in parentheses:</p> <ul style="list-style-type: none"> • Displays (4) • Flight Management/Thrust Management (2) • Central Maintenance (2) • Data Communication Management (2) • Flight Deck Communication (2) • Airplane Condition Monitoring (1) • Digital Flight Data Acquisition (2) • Data Conversion Gateway (4) 	<p>Two computers Flight Management Guidance Computer (FMGC)</p> <ul style="list-style-type: none"> • Flight management for navigation, performance prediction and optimisation, navigation radio tuning and information display management • Flight guidance for autopilot commands, flight director and thrust commands – two types of guidance <ul style="list-style-type: none"> ◦ Managed – lateral and vertical flight plan data ◦ Selected – guidance targets selected on the glareshield Flight Control Unit • Flight envelope and speed computation 	<p>Two AYK-14 Mission Computers (MC)</p> <p>One MC will be the active Bus Controller on AVMUX 1 – 6 and the other MC will be the an RT.</p>	<p>Communication / Navigation / Identification – Management System (CNI-MS) consists of</p> <ul style="list-style-type: none"> • 2 Mission Computers (MC) – control the information exchanged between airplane systems via MIL-STD-1553 databases. One MC will be the active Bus Controller on nominated databases and the other will be the Backup Bus Controller for those same databases • 2 Bus Interface Units (BIU) – if both MCs fail the BIU assume the bus controller functions for the applicable databases • 2 CNI System Processors(CNI-SP) – contain the operational logic that permit crew control and functioning of the communication, navigation and identification equipment.

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-130J
Control	<p>The other flight deck hardware elements that make up the AIMS system are</p> <ul style="list-style-type: none"> • Six flat panel display units • Three control and display units (left, centre and right) • Two EFIS display control panels • Display select panel • Cursor control devices • Display remote light sensors 	<p>Three Multipurpose Control and Display Units (MCDU) (only two at a time) provide:</p> <ul style="list-style-type: none"> • flight plan definition and display • data insertion (speeds, weights, cruise level, etc) • selection of specific functions <p>One Flight Control Unit on the glareshield provides manual entry of:</p> <ul style="list-style-type: none"> • speed • heading • altitude • vertical speed <p>Two thrust levers linked to the FMGCs and FADECs provide autothrust or manual thrust control</p>	<p>Left and Right Digital Display Indicators (DDIs)</p> <p>Up Front Controller (UFC)</p> <p>Digital Map Computer (DMC)</p>	<ul style="list-style-type: none"> • 3 CNI Management Units (CNI-MU) – primary crew interface to the CNI-MS. • 1 Communications / Navigation / Breaker Panel (CNBP) • 2 Avionics Management Units (AMU) • 2 Heads Up Display (HUD)
Display		<p>Two Primary Flight Displays (PFD) and two Navigation Displays (NDs) provide visual interface with flight management and guidance related data.</p> <p>PFD:</p> <ul style="list-style-type: none"> • FMGC guidance targets • Armed and active modes • System engagement targets <p>ND:</p> <ul style="list-style-type: none"> • Flight plan presentation • Aircraft position and flight path • Navigation items including radio aids and wind) 	<p>Left and Right DDIs</p> <p>UFC</p> <p>DMC</p> <p>Heads Up Display (HUD)</p>	

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-130J
Computer Architecture	<p>Dual cabinets each contain four core processor modules (CPMs) and four input / output modules (IOMs), with space reserved in the cabinet to add one CPM and two IOMs to accommodate future growth. The shared platform resources provided by AIMS are</p> <ul style="list-style-type: none"> • Common processor and mechanical housing, • Common input/output ports, power supply, and mechanical housing, • Common backplane bus (SAFEbus™) to move data between CPMs and between CPMs and IOMs, • Common operating system and built-in test (BIT) and utility software. <p>Applications are integrated on common CPMs. The IOMs transmit data from the CPMs to other systems on the airplane, and receive data from these other systems for use by the CPM applications. A high-speed backplane bus, called SAFEbus™, provides a 60-Mbit/s data pipe between any of the CPMs and IOMs in a cabinet.</p> <p>Communication between AIMS cabinets is through four ARINC 629 serial buses.</p>	<p>Two computers Flight Management Guidance Computer (FMGC)</p> <p>FMGC are identical single channel computers</p> <p>MCDU are identical single channel computers</p>	<p>Two AYK-14 Mission Computers</p> <p>AYK-14 MCs are identical single channel computers.</p> <p>Other AYK-14 modules include a core memory and MIL-STD - 1553A/B, Tactical Data System, RS-232, and discrete Input/Output (I/O).</p>	<p>MC are identical single channel computers</p> <p>CNI-SP are identical single channel computers</p> <p>BIU are identical single channel computers</p> <p>MC, CNI-SP and BIU are all different computer architectures</p>
Dissimilarity	No dissimilarity between AIMS cabinets	<p>No dissimilarity between FMGCs</p> <p>No dissimilarity between MCDUs</p>	No dissimilarity between MCs	<p>No dissimilarity between MCs</p> <p>No dissimilarity between CNI-SPs</p> <p>No dissimilarity between BIUs</p>
Latent Failure Detection	Built in Test	Built in Test	Built in Test	<p>MC BIT</p> <p>CNI-SP PBIT and IBIT</p>

	Boeing 777	A330 / KC-30A	F/A-18A/B	C-130J
Reconfiguration	<p>Hardware fault detection and isolation is achieved via a lock-step design of the CPMs, IOMs, and the SAFEbus™. Each machine cycle on the CPMs and IOMs is performed in lock-step by two separate processing channels, and comparison hardware ensures that each channel is performing identically. If a miscompare occurs, the system will attempt retries where possible before invoking the fault handling</p> <p>and logging software in the operation system. The SAFEbus™ has four redundant data channels that are compared in real time to detect and isolate bus faults.</p>	<p>Selected guidance has priority over managed guidance mode.</p> <p>Normal mode, dual mode, single mode</p>	<p>One MC will be the active Bus Controller on AVMUX 1 – 6 and the other MC will be the an RT.</p> <p>If the MC BC fails, the other MC assumed control of all buses.</p>	<p>One MC is capable of performing the functions of both MCs with no reduction in capability</p> <p>If one MC fails, the other MC assumes control of all seven buses with no loss of system integration performance.</p> <p>BIU provides backup in the event of dual MC failures.</p> <p>Each CNI-SP calculates its own solutions independently, and compares the results with the other CNI-SP. Either CNI-SP can perform all the functions alone, should the other CNI-SP fail. The CNI-SP operates in one of three modes: dual, single active/inactive and independent.</p>
Sensors	<p>Redundant Inertial Navigation Systems / Global Positioning Systems</p> <p>Radio Navigation (VOR, ILS, ADF, DME)</p>	<p>Each FMGC tunes its own side except when in single operation</p> <ul style="list-style-type: none"> • One VOR • One ILS • One ADF • 5 DMEs <p>3 Inertial Reference Systems</p> <p>FMGC position is a blend of IRS and radio position</p> <p>Uses GPIRS position in priority mode</p>	<p>Comm #1 and Comm #2 (UHF, VHF, HF)</p> <p>EGI – INS / GPS</p> <p>VOR, ILS, TACAN , DME, ADF</p> <p>Combined Interrogator Transponder (CIT)</p>	<p>The CNI-MS controls the following equipment:</p> <ul style="list-style-type: none"> • 2 UHF radios • 2 VHF radios • 2 HF radios • 2 Embedded GPS/INS (EGI) • 2 VOR/ILS/MB radios • 2 TACAN radios • 2 ADF radios • 2 IFF transponders
Backup	<p>Stand-by navigation instruments</p> <p>Communications can be independently tuned</p>	<p>Stand-by navigation instruments</p> <p>Communications can be independently tuned</p>	<p>Stand-by navigation instruments</p> <p>Communications can be independently tuned</p>	<p>Stand-by navigation instruments</p> <p>CNBP can independent tune radios</p> <p>Bus Interface Unit (BIU) provides backup in the event of MC failures</p>

Table 7: Overview of design features supporting detection and handling of systematic faults in Flight Management Systems