The Dangers of Failure Masking in Fault-Tolerant Software: Aspects of a Recent In-Flight Upset Event

C. W. Johnson*, C. M. Holloway†

* Dept. of Computing Science, University of Glasgow, Glasgow, G12 9QQ, johnson@dcs.gla.ac.uk
† NASA Langley Research Center, 100 NASA Road, Hampton VA 23681-2199, U.S.A., c.m.holloway@nasa.gov

Keywords: accident, software, fault-tolerance, maintenance

Abstract

On 1 August 2005, a Boeing Company 777-200 aircraft, operating on an international passenger flight from Australia to Malaysia, was involved in a significant upset event while flying on autopilot. The Australian Transport Safety Bureau's investigation into the event discovered that "an anomaly existed in the component software hierarchy that allowed inputs from a known faulty accelerometer to be processed by the air data inertial reference unit (ADIRU) and used by the primary flight computer, autopilot and other aircraft systems." This anomaly had existed in original ADIRU software, and had not been detected in the testing and certification process for the unit. This paper describes the software aspects of the incident in detail, and suggests possible implications concerning complex, safety-critical, fault-tolerant software.

1 Introduction

Software plays an increasingly significant role in avionics. The rapid growth of this technology provides manufacturers and operators with a degree of flexibility that would not otherwise be possible. The introduction of software systems also enables rapid reconfiguration of key applications during both development and the operational life of an airframe. This flexibility also enables a high degree of fault tolerance. Software systems may detect potential failures and avert mishaps through the use of redundancy. However, the introduction of new technologies often creates new hazards. This paper describes how the development of fault tolerant software can mask previous failures. Redundant systems have enabled aircraft to continue in operation for many months after an initial fault has occurred. If uncorrected, this creates the latent conditions for an accident when the software cannot respond to further system failures.

To illustrate some of the potential dangers associated with fault tolerant software, we use a case study involving an in-flight mishap on a Boeing 777-200. During the climb after take-off the crew observed a LOW AIRSPEED advisory and a slip/skid indication. Their Primary Flight Display then indicated that they were simultaneously approaching a stall and the aircraft's overspeed limit. Such conflicting indications can undermine the crews' confidence in their systems. The aircraft pitched up and systems began to indicate a decrease in airspeed from 270 to 158 knots. Eventually the flight crew regained control of the aircraft, and the aircraft returned to Perth. The flight data recorder indicated unusual acceleration values for all three planes of movement [1]. These values were provided by the Air Data Inertial Reference Unit and affected a range of aircraft systems during both automated and manual flight. This paper describes the way in which software contributed to the potential accident. A further motivation for studying this incident is that near misses provide an opportunity to learn about the causes of potential accidents before there is any loss of life.

The flexibility of avionics software has also increased the complexity of avionics. This presents considerable barriers to accident investigators who must piece together the complex events and contributory factors that led towards an adverse event. For this reason, we use Events and Causal Factors (ECF) diagrams to provide an overview of the case study incident. This approach helps to map key areas of the many pages of text that are used in the official report [3]. ECF diagrams were originally developed by the US Department of Energy. It is important to stress, however, that this is only one of several different notations that might be used to provide a similar overview [4]. The focus here is less on the technique used for the analysis than on the role that 'fault tolerant' software played in the causes of a potential accident.

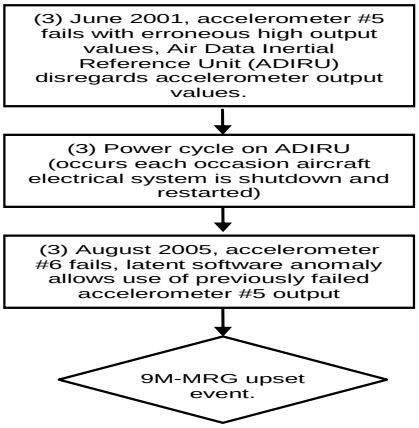

Figure 1: High-Level Overview of the Case Study Incident

2 Overview of the Incident

Figure 1 illustrates the immediate events that contributed to the case study incident. The focus here is not on identifying a cause

for the accident but simply on establishing a more detailed timeline for the sketch that was presented in the introduction. The number that is used to label each box in the ECF diagram refers to the page on which that event is described in the Australian Transport Safety Bureau's official report [1]. This enables readers to use these diagrams to support the more detailed analyses of the textual documentation. The diamond indicates the eventual outcome of the events that lead to it.

As can be seen, the initial events may be traced back more than four years before the mishap. Accelerometer number 5 failed with erroneous high output values. The Air Data Inertial Reference Unit (ADIRU) software was programmed to disregard such output from the accelerometer and, instead, to rely on values from backup systems. The initial failure was, however, masked following a restart of the ADIRU. This created a vulnerability that was compounded by a latent software error – or bug. Following the failure of another accelerometer the ADIRU could revert to accepting input from the accelerometer that had initially failed and been excluded from its computations. This scenario contributed to the failure that is described in the opening sections of the paper.

3 Air Data Inertial Reference Unit Architecture

Figure 2 provides an overview of the architecture that supports the Air Data Inertial Reference Unit (ADIRU) on the B777. A core concept was to exploit redundancy as a means of achieving fault tolerance. The unit was divided into seven fault containment modules or areas. Each of these was physically and electrically isolated from the others. The intention was that the aircraft could have a fault in any of the modules and still remain serviceable. This feature would enable operators to continue flying until the number of fault-free modules fell below a minimum specified by the component manufacturer. This fault tolerance supported lower operating costs, for instance, by reducing the potential disruption to aircraft operations from unscheduled maintenance. In particular, the ADIRU used a fault containment module to analyze data from accelerometers and gyros before passing them through the ARINC 629 units to navigation and flight control systems.

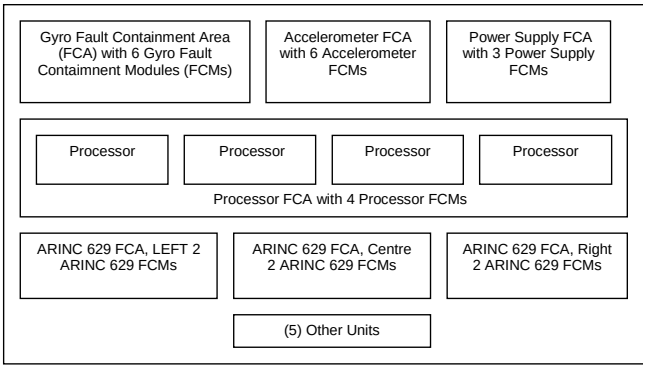

Figure 2: Air Data Inertial Reference Unit (ADIRU) Architecture (ATSB, 2007, p.5)

4 Redundancy & Fault Masking

Figure 3 builds on the previous analysis and illustrates how the architecture and design objectives for the ADIRU software contributed to the case study incident. As in Figure 1, the rectangles represent events leading to the mishap and page numbers in the ATSB report are indicated in parentheses. The ovals denote contributory factors. These provide a short-hand representation for many individual events that together create the preconditions for an incident. For example, many aspects of this mishap can be traced back to the initial aim of reducing operating costs and disruption to aircraft scheduling. This created a condition in which designers developed fault containment areas or modules that enabled operators to defer maintenance until the number of faulty Fault Containment Units exceeded specified tolerances.

The left–hand chain of events captures elements of the ECF diagram shown in Figure 1; including the initial failure of accelerometer number 5 in 2001 and the subsequent failure of number 6 in 2005 that triggered the mishap. As can be seen, however, Figure 3 significantly extends the analysis in the previous diagram. For example, it refers to the Secondary Attitude Air Data Reference Unit (SAARU). This extends the concepts of redundancy and of defence in depth by providing an alternate source of attitude, heading and air data to the ADIRU. A comparison was made between the output of the ADIRU and the SAARU during the mishap and this helped to mitigate the impact of the incorrect values from the ADIRU. This illustrates an important point in this paper – the aim is not to undermine the use of fault tolerance and redundancy but instead to demonstrate the particular dangers of software masking component failures both to operational crews and maintenance teams. In this case, it can be argued that the redundant SAARU provided sufficient control for the pilot to avert more serious consequences from the ADIRU faults.

The right side of Figure 3 looks more closely at maintenance issues in the case study incident. The initial failure of accelerometer number 5 in 2001 triggered a maintenance message on the on-board maintenance computer; this was known as an ADIRU MM 34-20010 event. Such messages could be read by maintenance teams using a ground-based terminal but were not directly visible to the crew. Some ADIRU events can, however, be displayed in-flight on the Engine Indication and Crew Alerting System (EICAS). If such an in-flight warning occurs, then the ADIRU must be replaced with a serviceable unit within three days. As can be seen in Figure 3, the crew did not receive such a warning following the 2001 accelerometer failure and so the ADIRU was not replaced. However, the ECF diagram also denotes that different operators implemented different practices and risk mitigation strategies following such MM events. The aircraft manufacturer noted that: "the ADIRU can be dispatched with MM 34-20010 present until such time that the operator deems it prudent to remove the ADIRU to avoid a schedule interruption due to occurrence of the ADIRU Status message. The decision to remove the ADIRU based on the presence of MM 34-20010 only is made by the operators on an economic basis, not a safety basis" [1, p.8].

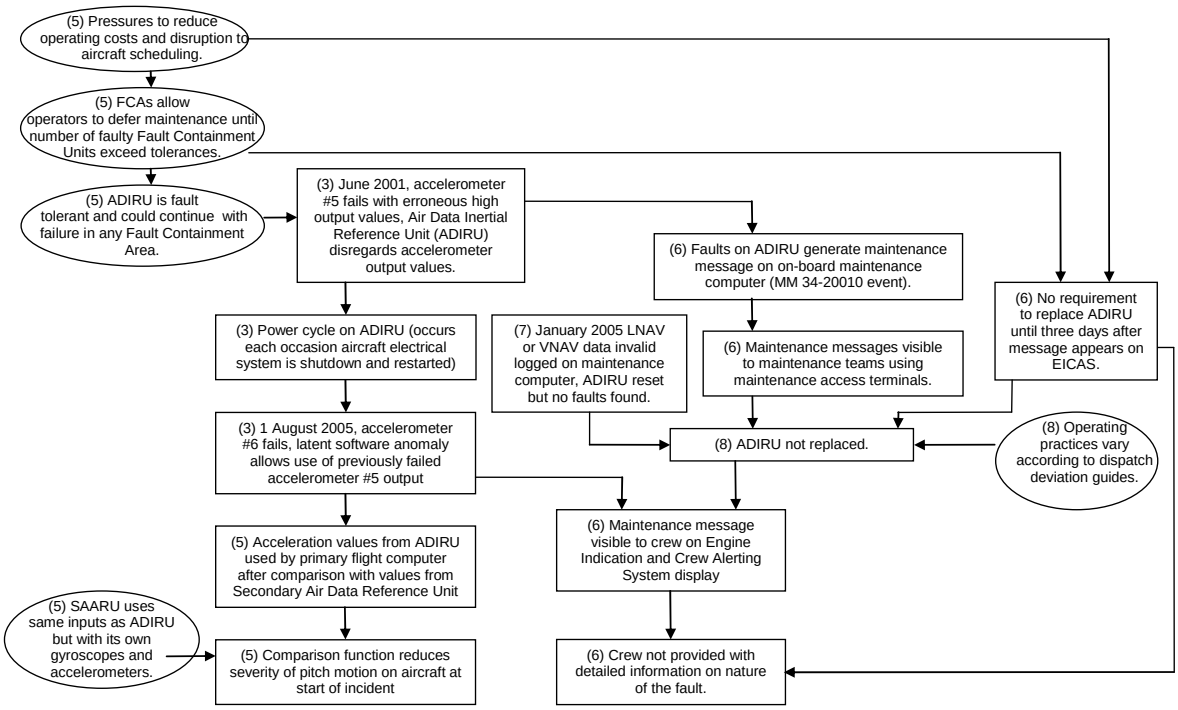

Figure 3: Maintenance Not Required to Replace Fault Tolerant ADIRU

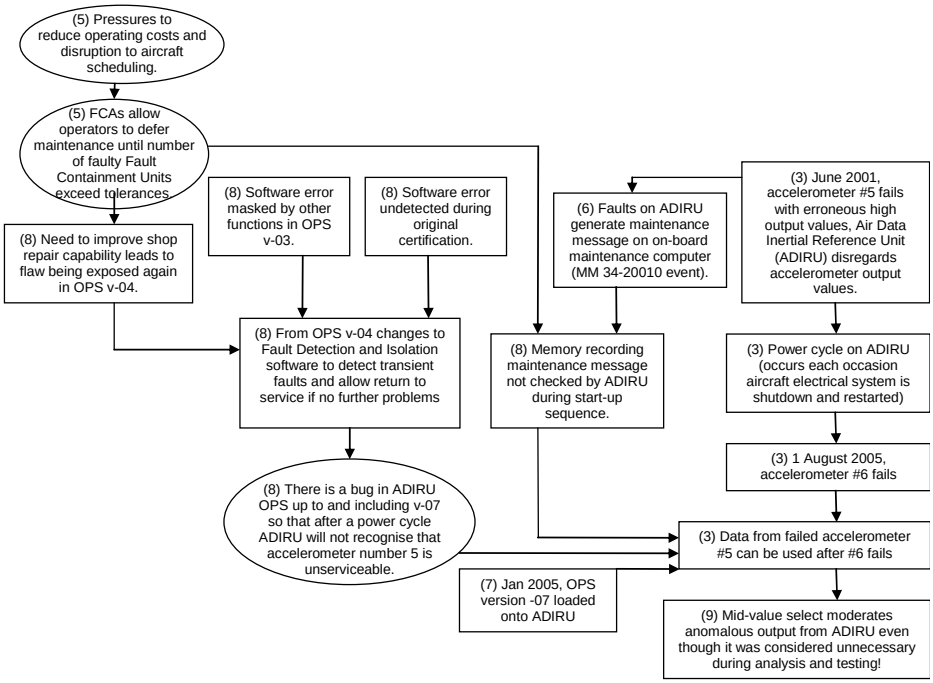

Figure 4: Software Propagates Failure and Mitigates Adverse Effects

5 Software Architecture and Fault Masking

Figure 4 focuses on the software issues that contributed to the in-flight mishap. As before, many of the particular events can be traced back to pressures to reduce operating costs and avoid disruption to aircraft scheduling. This created the Fault Containment Area architecture that enabled maintenance to be deferred. It also led to software requirements for the ADIRU so that the system would check the status of critical components but allow the unit to continue operation if minimum criteria for the availability of FCAs were met. The implementation of this software requirement was flawed in early versions of the OPS –

software. However, up to version v-03 this problem was mitigated by additional checks in other areas of the application. A renewed requirement to improve shop repair capability led to the flaw being exposed again in OPS v-04. The OPS software up to and including v-07, therefore, contained a bug such that after a power cycle the ADIRU would not recognise that accelerometer number 5 was unserviceable.

Figure 4 also denotes how a maintenance message was generated following the 2001 fault on accelerometer number 5. However, the fault status was not checked by the ADIRU following a power up for the reasons presented previously. This combination of events led to use of data from the failed accelerometer when a further fault was detected in 2005 with accelerometer number 6. Again, however, this incident reveals how the 'defence in depth' techniques of modern avionics helped to mitigate the potential consequences of this bug. The ADIRU software had been designed to pass on mid-range values if components provided data that varied between two implausible extremes. It is important to stress that designers may underestimate the value of the defences they create. The bottom event in Figure 4 records that this additional safety feature was considered to be superfluous during testing and analysis of the ADIRU.

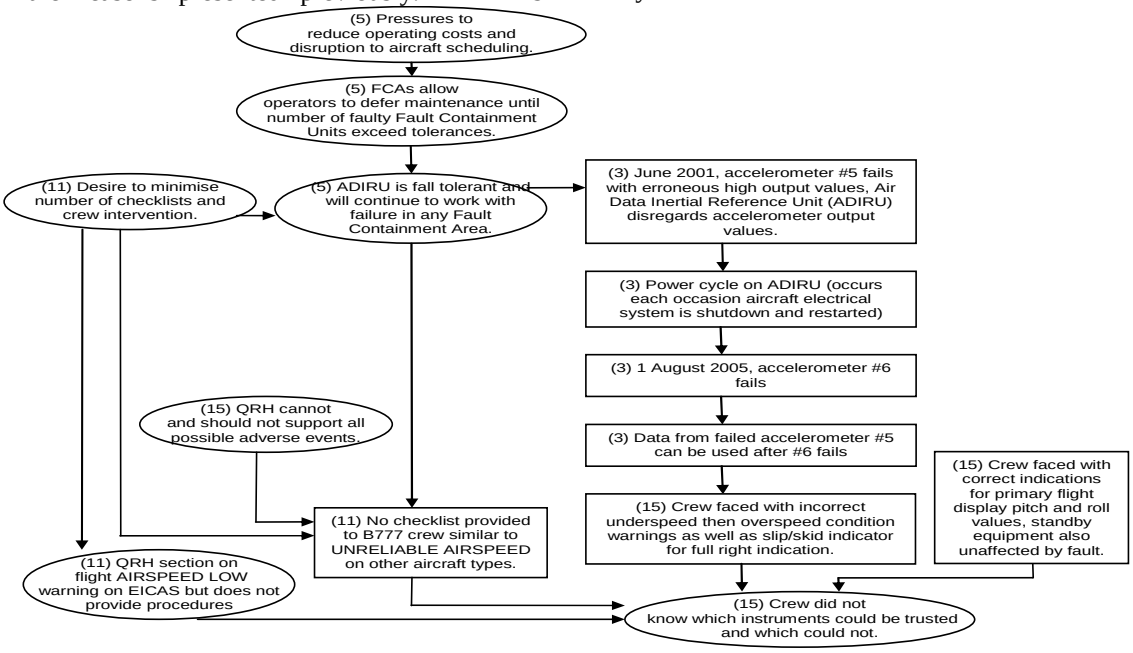

Figure 5: Fault-Masking Undermines Crew Interaction

6 Human Factors Issues

Figure 5 illustrates how the impact of software bugs propagates beyond the embedded systems of modern avionics to affect crew interaction. The ECF diagram illustrates how the silent masking of redundant failures undermined the ability of human operators to diagnose and respond to the problems that confronted them. Page 15 of the ATSB report describes how the crew was faced with incorrect underspeed warnings, followed by overspeed warnings, as well as a full right slip/skid indication following the failure of accelerometer number 6. At the same time, correct indications were presented for the pitch and roll values on the primary flight display. Standby equipment was also unaffected by the fault. The crew were, therefore, unsure which of the instruments to trust. Their uncertainty was exacerbated by design decisions that stemmed from the underlying philosophy of masking redundant failures during continued operations, mentioned in previous sections. Figure 5 records that previous aircraft types had provided checklists for UNRELIABLE AIRSPEED in the crews' Quick Reference Handbooks (QRH). However, the redundancy implemented by the B777 software systems implied that the crew should never be faced with such an indication. The system was designed to ensure that unreliable input from key components should automatically trigger a transition to redundant systems without any intervention from the crew. It was reasonable, therefore, to argue that this section should be dropped from the QRH. Crews often complain when documentation is unnecessarily verbose or difficult to navigate. It was possible for multiple failures to trigger a NAV AIR DATA SYS message on the EICAS. The QRH then referred the crew to an unreliable airspeed table. However, this particular warning was not triggered during the case study incident. The key point here is that the problems facing the crew were exacerbated by the decision to mask information that designers did not consider would be needed by the crew. However, the circumstances of this mishap reiterate the importance of providing some degree of visibility to the crew when it is difficult or impossible to guarantee the resilience of fault tolerant systems [6].

Figure 6 continues the analysis of crew interaction in the aftermath of the 2005 auto throttle failure. As mentioned, the pilot was faced with a situation that designers had not considered to be possible. The auto throttle system remained active and the underspeed/overspeed warnings suggested that the malfunction may have been related to these functions. In consequence, the pilot attempted to disconnect the autothrottle by pressing the thrust lever disconnect switch and pushing the autothrottle

engage switch to toggle it off. However, these attempts were ineffective because the crew failed to switch the autothrottle arm switch from ARMED to OFF. In consequence, the autothrottle continued to increase thrust in response to the low-speed data that was erroneously being supplied from the ADIRU and the fault accelerometer. The interaction difficulties faced by the aircrew when attempting to disarm the autothrottle are typical of the ways in which the stress imposed on a human operator by an initial failure can trigger further mistakes. These errors serve to compound the problems created by an initial mishap. Figure 6 illustrates how the crew now not only had to understand the reasons for the erroneous and inconsistent readings from their instrumentation, they now had to find an explanation for their failure to disarm the auto throttle. Without additional support from documentation, such as the QRH, it can be difficult for any crew to avoid a gradual loss of situational awareness through apparent interactions between such compound failures.

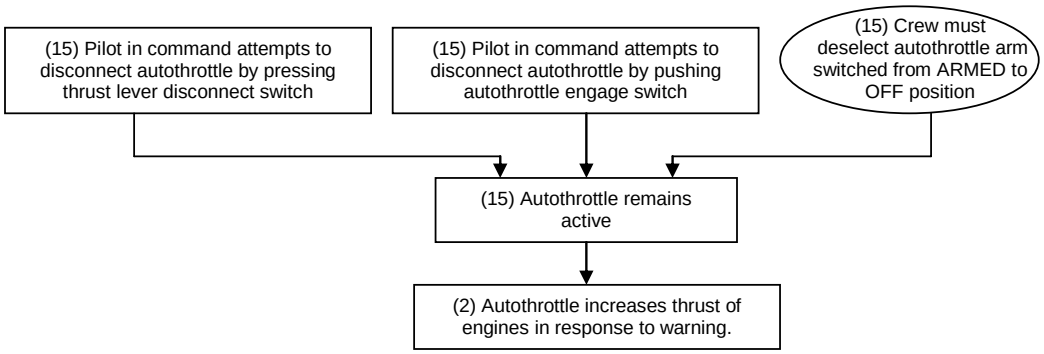

Figure 6: Auto Throttle Uncertainty Further Undermines Situation Awareness

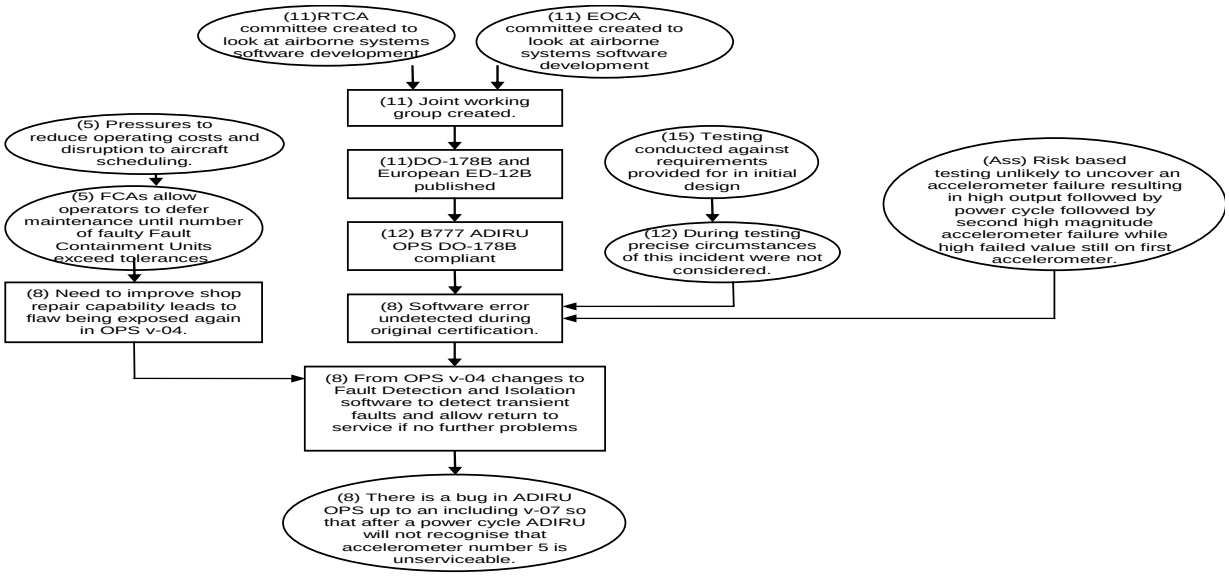

Figure 7: Process-Based Certification and the Selection of Test Cases

7 Certification Issues

Figure 7 illustrates some of the certification-related issues that were raised by the case study incident. It notes that DO-178B [7] and the European ED-12B equivalent were derived from joint working groups within the RTCA and the predecessor of the European Organisation for Civil Aviation Equipment. It also shows that the ADIRU OPS software was DO-178B compliant. Even so, testing failed to consider the precise circumstances of the failure that was revealed in this incident. The diagram also records an assumption that risk-based testing was unlikely to uncover an accelerometer failure resulting in high output, followed by power cycle, followed by a second high magnitude accelerometer failure while the high failed value was still on first accelerometer. Although the consequences of such a scenario were potentially severe, such a contingency might arguably have been considered too unlikely to be considered in detail during system testing. It is, however, difficult for the ATSB and other investigatory organisations to confirm such analyses without access to detailed design process information that is likely to be commercially sensitive.

This incident reiterates the importance of considering the impact of multiple component faults in the requirements that are used to guide the development of fault tolerant software, especially where fault-masking techniques shield operational personnel

from additional complexity between maintenance cycles. This might seem like a very specific issue. However, the incident reveals important lessons for engineers, which highlights the more general need for them to read accident and incident reports [2]. It was fortunate that in the case study incident, the crew were able to land their aircraft. However, unless software engineers are made aware of the circumstances surrounding this and similar incidents, there is a danger that future avionics software will continue to suffer from the flaws of previous systems.

This incident illustrates the strengths and the weaknesses of current certification standards for commercial systems containing aviation software. The ADIRU software had been tested and developed to the standard required at the time of certification. DO-178B, in common with most recent software standards across the safety-critical industries, focuses on development processes rather than the certification of particular lines of code. In particular, it contains requirements to ensure the testing of delivered code against the original safety requirements identified for that software. The level of testing is partly determined by the risks that are to be mitigated by the functionality of the code. However, the ADIRU testing "was limited to the original specification and requirements of the component" [1, p. 16]. These did not consider the particular scenario that arose during the incident. This has important implications for process based standards such as DO-178B. In previous generations of product based specifications, the failure of a particular system was symptomatic of potential flaws in that particular component. However, in process based certification the failure of a system implies that there may be wider flaws in the systems developed under that process. It is for this reason that the manufacturer responded to the incident not simply by correcting for the particular combination of accelerometer failures that characterized this mishap, but by also reviewing the handling of a wide range of potential multiple hardware failures.

8 Conclusions and Further Work

This paper has described how fault tolerant software can automatically redirect input from failed components to redundant resources, providing important benefits. In particular, fault-tolerance supports the continued provision of critical services and can be used to extend the interval between maintenance procedures. Fault masking implies that information about the reconfiguration of redundant resources may be hidden from the crew and from maintenance teams during the interval between these procedures. This offers further benefits. Fault masking will reduce the workload of the flight crew, by reducing the number of checklists that must be consulted in abnormal situations. Maintenance teams can focus on those areas that require immediate attention. By postponing other items until major service intervals they can reduce the time pressures that often complicate unscheduled maintenance procedures.

However, the benefits of fault masking in redundant systems also create new hazards. When service teams fail to identify underlying failures during major maintenance procedures, aircraft may continue to be operated without the additional assurance provided by redundancy. The potential consequences of this hazard are compounded by the effects of fault masking for operational staff. They will not be aware of the underlying faults that affect their application. When subsequent failures finally compromise the use of redundant systems, flight crews will be faced with unanticipated problems. The very fact that the initial fault has been masked implies that any secondary failure will also not be covered in training and documentation that usually support their interaction.

This paper is part of wider initiatives that are intended to assess the impact of what can be called 'degraded modes of operation' on safety-critical applications. Degraded modes of operation refer to situations in which efforts are made to maintain levels of service in the presence of component failures. In other industries, including Air Traffic Management, these can include ad hoc workarounds as staff learn to avoid the constraints that might otherwise be placed on them by faulty equipment [5]. The case study described in this paper extends our analysis to consider the impact that fault tolerant software can have upon complex applications during degraded modes of operation. By masking previous failures, maintenance staff and operators may not recognize the gradual erosion of the safety mechanisms that are intended to support normal operation.

References

[1] Australian Transport Safety Bureau. In-Flight Upset Event 240Km North-West of Perth, WA, Boeing Company 777-2000, 9M-MRG. Aviation Occurrence Report 200503722, Canberra, Australia, 2007.

[2] C. M. Holloway and C.W. Johnson. "Why System Safety Professionals Should Read Accident Reports." In T. Kelly (ed.), *The First IET International Conference on System Safety*, Institute of Engineering and Technology, Savoy Place, London, 325-331, 6-8th June 2006, ISBN 0-86341-646-2, 2006.

[3] C.W. Johnson. *Failure in Safety-Critical Systems: A Handbook of Accident and Incident Reporting*, University of Glasgow Press, Glasgow, Scotland, ISBN 0-85261-784-4, 2003.

[4] C.W. Johnson and C.M. Holloway. "A Survey of Causation in Mishap Logics." *Reliability Engineering and Systems Safety* (80) 3:271-291, 2003.

[5] C.W. Johnson and C. Shea. "The Contribution of Degraded Modes to Accidents in the US, UK and Australian Rail Industries." *Procs of 2007 Int. Systems Safety Society Conf.*, Baltimore, USA.

[6] D.A. Norman. "The 'Problem' with Automation: Inappropriate Feedback and Interaction Not 'Overautomation'." In D.E. Broadbent, J. Reason and A. Baddeley (eds.) *Human Factors in Hazardous Situations*, pages 137-145, Clarendon Press, Oxford, UK, 1990.

[7] RTCA Inc.. "Software Considerations in Airborne Systems and Equipment Certification." RTCA/DO-178B, Washington, D.C. 1992.