System Theoretic Framework for Assuring Safety and Dependability of Highly Integrated Aero Engine Control Systems

by

Malvern J. Atherton

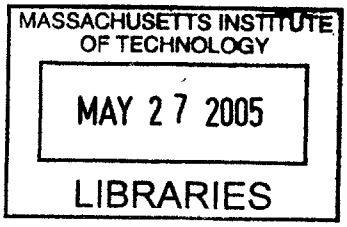
Graduate Diploma in Control and Information Technology,
University of Manchester Institute of Science and Technology, UK, 1990
B.Sc Electrical and Mechanical Engineering, University of Edinburgh, Scotland, 1988

Submitted to the System Design and Management Program
in Partial Fulfillment of the requirements for the Degree of

Master of Science in Engineering and Management

At the

Massachusetts Institute of Technology
May 2005



Signature of Author _____ 20th May 2005

Malvern J. Atherton
System Design and Management Program
May 2005

Certified by _____

Nancy Leveson
Thesis Supervisor
Professor of Aeronautics and Astronautics

BARKER

System Theoretic Framework for Assuring Safety and Dependability of Highly Integrated Aero Engine Control Systems

by

Malvern J. Atherton

Submitted to the System Design and Management Program
in Partial Fulfillment of the requirements for the Degree of

Master of Science in Engineering and Management

ABSTRACT

The development of complex, safety-critical systems for aero-engine control is subject to the, often competing, demands for higher safety and reduced development cost. Although the commercial aerospace industry has a general good safety record, and has placed much emphasis on process improvement within a strong safety culture, there continues to be a large number of design and requirements errors found during development and after entry into service.

The thesis assesses current system safety practice within the aero engine control system industry, including international standards, and reviews the current practice against the research at MIT by Professor Nancy Leveson. The thesis focuses in particular on software safety as this is the area that has proven most challenging and most likely to experience high costs. The particular research topics reviewed are Intent Specifications, the System Theoretic Accident Modeling and Processes (STAMP) technique, and requirements completeness criteria. Research shows that many problems arise from requirements and design errors rather than component failures.

Several example incidents from an engine company are reviewed and these show a pattern of common problems which could have been caught by the use of more comprehensive requirements completeness checks and by the use of Intent Specifications. In particular, assumptions are not currently documented in the specifications but are kept separately, and the need to identify assumptions is not emphasized enough in existing processes.

It is concluded that the existing development process has significant room for improvement in the coordination between the safety assessment and system development processes. In particular, more could be done by the use of requirements completeness checks, software hazard analysis, the adoption of the Intent Specification approach and in the use of the STAMP models.

Acknowledgments

I wish to thank my company and MIT for giving me the opportunity to become a part of the SDM Community and my thesis advisor Nancy Leveson for her guidance. Additionally, there are several people who have provided guidance and support who I would like to acknowledge. At MIT I would like to thank Fernando Cela Diaz, Darrel Quah, Steve Friedenthal, Dan Fry and Karen Marais. At my company I would like to thank Andy Pickard, Derek Achenbach, William Fletcher and Stephen Fisher. Finally, I would like to thank my family and Jackie Duane for their support.

Table Of Contents

List Of Figures

List of Tables

Acronyms

AC	Advisory Circular
AD	Airworthiness Directive
ARINC	Aeronautical Radio Inc (aerospace digital communication standard)
ARP	Aerospace Recommended Practice
ATTCS	Automatic Takeoff Thrust Control System
BIT	Built In Test
BITE	Built In Test Equipment
CCA	Common Cause Analysis
CCDL	Cross Channel Data Link (data bus connecting FADECs)
CCM	Cause Consequence Matrix
CCR	Certification Check Requirements
CMC	Central Maintenance Computer
CMR	Certification Maintenance Requirement
DER	Designated Engineering Representative
DOORS	Dynamic Object Oriented Requirements System
DOT	Department Of Transport
EEC	Electronic Engine Controller
EICAS	Engine Indications and Crew Alerting System
EUROCAE	European Organization for Civil Aviation Equipment
FAA	Federal Aviation Authority
FADEC	Full Authority Digital Engine Control
FAR	Federal Aviation Regulations
FCS	Fault Code Store
FHA	Functional Hazard Assessment
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FNF	Fault Not Found
fpm	Feet per minute
FPMU	Fuel Pump and Metering Unit
FTA	Fault Tree Analysis
GPS	Global Positioning Satellite
HAZOP	Hazard and Operability Analysis
HCI	Human Computer Interface
HP	High Pressure
HSI	Hardware/ Software Interface
I/F	Interface
IFSD	In-Flight Shut Down
IPT	Integrated Product Team

JAA	Joint Airworthiness Authorities
lb	pound mass
LOTC	Loss Of Thrust Control
LP	Low Pressure
LRU	Line Replaceable Unit
LSOV	Latching Shutoff Valve (in the FPMU)
MIT	Massachusetts Institute of Technology
MMV	Main Metering Valve (in the FPMU)
N1	LP shaft speed signal in rpm (i.e. fan speed)
N2	HP shaft speed signal in rpm (i.e. core speed)
NFF	No Fault Found
NTSB	National Transportation Safety Board
NVM	Non Volatile Memory
P2.5	Pressure at engine station 2.5 (HP compressor inlet)
PM	Program Management
PRA	Probabilistic Risk Assessment
PRV	Pressure Raising Valve (in the FPMU)
PSSA	Preliminary System Safety Assessment
QA	Quality Assurance
QA	Quality Assurance
RA	Resolution Advisory (in TCAS)
RBD	Reliability Block Diagram
RSML	Requirements State Machine Language
RTCA	RTCA, Inc. (acronym no longer has an expansion)
SAE	Society of Automotive Engineers
SC	RTCA Special Committee
SE	Systems Engineering
SFTA	Software Fault Tree Analysis
SIRT	Systems Integration and Requirements Task Group (an SAE Committee)
SpecTRM	Specification Tools and Requirements Methodology
SpecTRM-RL	SpecTRM Requirements Language
SQA	Software Quality Assurance
SSA	System Safety Assessment
STAMP	System-Theoretic Accident Modeling and Processes
STPA	STAMP-based hazard analysis
TA	Traffic Advisory (in TCAS)
TBD	To Be Determined/Defined
TCAS	Traffic Collision Avoidance System
V&V	Verification and Validation
V1	Pilot decision speed (for committing to perform a takeoff)
Wf	Fuel Flow Rate (to the engine, in lb/hour)

1 Introduction

The aerospace industry has long been at the forefront of developments in systems engineering and the emerging field of system safety. This research reviews current approaches used for developing and certifying digital control systems for commercial aero engines, focusing on software-based systems in particular. Despite great efforts on process improvements in the industry, development costs continue to be high, particularly when safety-critical software is involved. Much emphasis has been made on improved Program Management (PM) including cost and schedule control and managing program risk. Systems Engineering (SE) concepts have also gained a great deal of importance as a framework for managing complexity, which goes hand in hand with PM.

However, despite these efforts, safety-critical software development projects continue to be hampered by significant levels of rework, leading to projects being late and/or over budget. Additionally, unforeseen problems encountered in service also continue to demand much attention and rework after initial certification. The essential problem is one of system complexity, and when software is involved the complexity problem is greatly multiplied. Problems all too frequently seem to get past the rigorous design, validation and verification processes and make it to integration testing, or worse, into service. Often, anomalies are found that were not predicted by the system safety assessment process, and this raises the question of the validity of the safety case itself.

This thesis takes a broad look at recent research in the Aeronautics/ Astronautics Department at MIT in System and Software Safety (in particular looking at Intent Specifications and the STAMP modeling technique developed by Professor Nancy Leveson) and evaluates the

existing aero engine control system development and certification process in light of this research. The STAMP (System-Theoretic Accident Modeling and Processes) technique provides a new perspective on system safety. In STAMP, the concept of *failures* as the atomic element leading to accidents is replaced by the concept of *constraints*. The traditional role of the numerically based Probabilistic Risk Assessment (PRA) in the certification safety case is questioned in the STAMP analysis, which uses a broader set of considerations to establish the safety of a complex system. Some examples of safety-related design problems found late in the development process (i.e. during aircraft integration testing or operational service) are analyzed to establish whether conceptual weaknesses exist in the system development and system safety processes.

One example was a problem during an asynchronous power-up of two independent FADEC (Full Authority Digital Engine Control) computer channels controlling a turbofan engine. The turbofan engine was in a start sequence but the FADECs inadvertently scheduled the maximum possible fuel flow to the engine, leading to a brief fireball at the back of the engine. Furthermore, the subsequent pilot-commanded shutdown failed to close the fuel shutoff valve. This event occurred during aircraft integration testing and was corrected before certification. However, the software under test had already been through extensive software verification, system requirements validation and system verification processes at the FADEC supplier and engine manufacturer, including engine ground-tests. One factor in the event was that the pilots operated the system in a way not foreseen by the designers and hence not tested in the system verification process. The event was readily reproduced by the engine and FADEC suppliers once the required entry conditions were understood. Analysis of the design revealed about seven requirements errors in the affected part of the software design, only two of which were newly created in that software build.

The fact that so many latent errors had existed in the system and were mostly unknown (because they were masked by other factors) was surprising to the author, but is not uncommon in such systems. This example is typical of many anomalies that are found in software-based safety-critical systems. There is a need to develop more inherent robustness in the design process to reduce the likelihood of such surprises. The safety case for a software-based system depends strongly on how inherently robust the design is to design changes (the problem of *brittleness* of software as it is modified) and how well understood the external environment is (pilot behavior, external inputs and noise etc), in other words, a problem with the requirements. It is hypothesized that if many of the anomalies that occur in service are the result of poor requirements arising from misunderstood interfaces, then how accurate is the safety case if it is based primarily on a PRA approach? The PRA approach does not model requirements errors and assumes accidents are caused by mechanical failures. This is an issue which the STAMP modeling process attempts to address.

Furthermore, if the design and certification process can be made more robust to uncertainty in the environment, then the cost of rework can be reduced. In addition to looking at the system safety process and the STAMP technique, this research will also consider weaknesses in the requirements development process, and two lines of research will be reviewed for applicability to FADEC systems - Intent Specifications and Requirements Completeness Criteria. One of the reasons why design errors occur is because of misunderstandings and/or a lack of documentation of the assumptions and rationale behind the existing design. Thus, when changes are made, these assumptions can be violated and without an understanding of the violated assumptions, the testing process may not be able to catch the problem.

So the issue of system safety is closely related to the issue of requirements and design robustness and the certification process. They are also related to the overall objectives of program management, which is to achieve certification of a safe product that meets the customer's technical, schedule and cost objectives.

The insights gained from the reviews of MIT system and software safety research to the aero engine development and certification process will be considered with regards to the existing commercial aerospace recommended practices and standards (in particular ARP 4761, covering system safety [1] and DO-178B covering the software life-cycle processes [2]).

2 Existing System Development and Certification Process

2.1 Airworthiness Requirements

In the United States, the requirements for safety of highly integrated electronic control systems for aero engines come under FAR part 25.1309 [3], which addresses system safety requirements for airplane systems in general. These requirements are very general, and do not provide specific guidance on how the safety objectives shall be met. In essence the FAR states that the airplane must be designed so that the "occurrence of any failure condition that would prevent the continued safe flight and landing of the airplane is extremely improbable" (refer to APPENDIX A FAR Part 25.1309 – Safety Regulations, section (b)(1)). It also states that warning must be provided to the crew to alert them of unsafe conditions, so that corrective action may be taken. Regarding how to demonstrate compliance, section (d) states that compliance must be demonstrated by analysis, and where necessary, by appropriate testing. The analysis must consider possible modes of failure, the probability of multiple failures and undetected failures, the resulting effects on the airplane considering the stage of flight and operating conditions. The requirements seem quite comprehensive and cover some important and challenging objectives. The ability to consider the probability of multiple failures and undetected failures is quite a challenge as this requires a good understanding of the complexities of the system, including the design weaknesses that may exist. Also, it would only be possible if the external environment was well understood, so that the effect of all noise sources and unexpected inputs (including pilot commands) were understood. However, as a set

of high-level requirements for aircraft safety, these seem to be comprehensive and appropriate. This leaves the challenge of meeting these requirements in practice, and for this, the FAA developed an associated Advisory Circular, AC 25.1309-1A [4].

This AC is a little dated (1988) but does provide guidance for compliance with FAR 25.1309 (b), (c) and (d), which are the pertinent parts addressing safety of highly integrated complex systems such as those with embedded control software. FAR 25.1309 is based on the *fail-safe* concept in which the failure of any single component, or connection, during any one flight should be assumed, *regardless of its probability* [4, section 5 (a) (1), p.2]. This is a very sound principle when considered relative to current thinking on complex systems, where the complexity exceeds the ability of analysis to prove that the system will behave in a specific way under all conditions. In particular, software cannot be fully tested, and therefore complete reliance on probabilistic approaches is impossible. The *Software Myths* discussed in Section 2.2 of [5] provide evidence for this.

The Advisory Circular also requires that a Functional Hazard Assessment (FHA) is performed. It discusses methods such as FTA (Fault Tree Analysis), FMEA (Failure Modes and Effects Analysis), FMECA and RBD (Reliability Block Diagrams). Some emphasis is also given to the issue of latent failures and the potential use of CCRs (Certification Check Requirements) and CMRs (Certification Maintenance Requirements), which can be used to provide additional checks for latent failures if the automated systems are unable to ensure complete detection. So, it does include probabilistic approaches as forms of analysis.

The AC focuses on the system level, and hence does not explicitly cover software. It refers to AC 20-115A (superseded by B [6]), which itself identifies DO-178B [2] as defining a suitable means for demonstrating compliance for the use of software within aircraft systems.

The fail-safe principle leads to the requirement for protection against single-point electrical failures. The aircraft industry is probably ahead of several other safety-critical industries in the soundness of some of its fundamental safety concepts. In particular, the fail-safe principle leads to an avoidance of relying on microprocessor control for some safety-critical engine functions such as the detection and accommodation of an engine shaft over-speed or a shaft break. These systems are designed to cut the fuel flow to an engine that has suffered some form of failure causing the shafts to accelerate to speeds that could lead to blade release or other hazards. The engine turbine casing is not designed to prevent release of turbine blades under these conditions, and hence the condition itself must be prevented by removing the source of energy to the turbine before the critical speeds are reached. The FADEC (digital) systems have all the appropriate sensor and actuator controls to achieve this goal, but instead, the certified systems use analog electrical systems for reasons of speed of response and to provide independence from the software. Indeed the software would be considered to be one of the candidate systems for creating the hazard, though no specific probability analysis is performed to analyze such a possibility. The fail-safe principle works well for functions where mechanical (or analog electrical) interlocks or devices (such as the over-speed shutdown system) can be used, and it is the guiding principle for many system designs. However, for software, it is much harder to demonstrate that a design will fail-safe because the number of states is so large that complete analysis is impossible.

2.2 Aerospace Recommended Practices

Although AC 25.1309-1A provides guidance on concepts in designing for safety and suggests some analytical methods such as FTA, it does not provide a comprehensive guide to the system safety process. For this, ARP 4761 (Guidelines and Methods for Conducting the Safety

Assessment Process on Civil Airborne Systems and Equipment) has been produced by the SAE. This sits alongside ARP 4754 (Certification Considerations for Highly-Integrated Complex Aircraft Systems), which addresses system development and certification in general. The relationship between these standards is shown in Figure 1 below. The figure also shows the life cycle processes for software (DO-178) and hardware (DO-254). DO-254 primarily concerns electronic hardware and is outside the scope of this report.

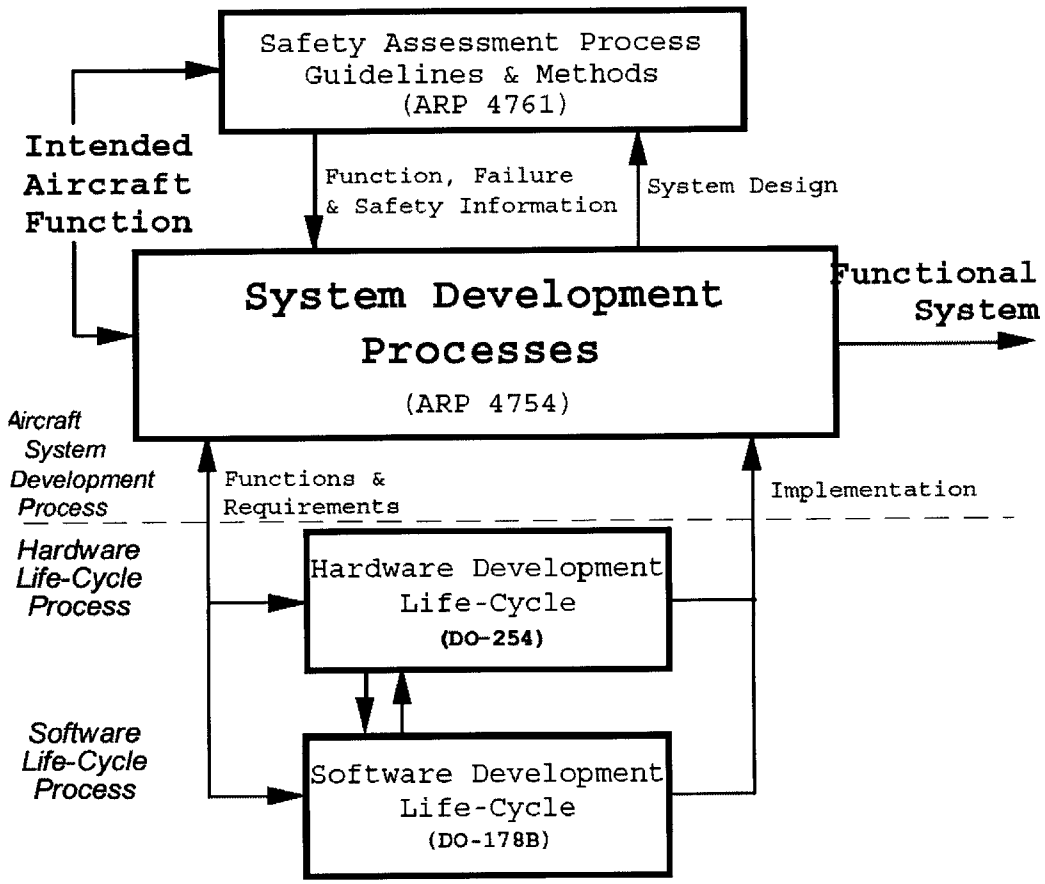


Figure 1 Certification Guidance Documents Covering System, Safety, Software and Hardware for Aerospace Systems, adapted from ED-79/ARP-4754 [7]

2.2.1 ARP 4754

ARP 4754 (ED-79 in Europe) addresses system development in general, and follows much that is accepted current good practice in systems engineering. The document discusses the requirements development and allocation process, including types of requirements, such as

customer requirements, functional requirements, operational requirements etc. It discusses the relationship between architectural choices and partitioning and system safety and lists development assurance levels based on the failure classification for system partitions. These assurance levels (A to E) are similar in concept to those used for software development in DO-178, and they define the set of processes to be followed for a given system. Systems with high development assurance levels require more process steps.

The document also discusses verification and validation processes (V&V), documentation and validation of assumptions, and configuration and change management. The relationship between the system development and safety assurance processes is shown in **Error! Reference source not found.**.

Some research has raised concern about the degree of emphasis placed on development assurance levels as a basis for assuring safety. A lot of ARP 4754 (and DO-178 discussed below) is devoted to determining how to choose a level for a particular system or subsystem, but there is not a great deal of evidence showing that systems developed to a particular level will really achieve the anticipated level of safety [8]. This is discussed further in section 2.2.3 below.

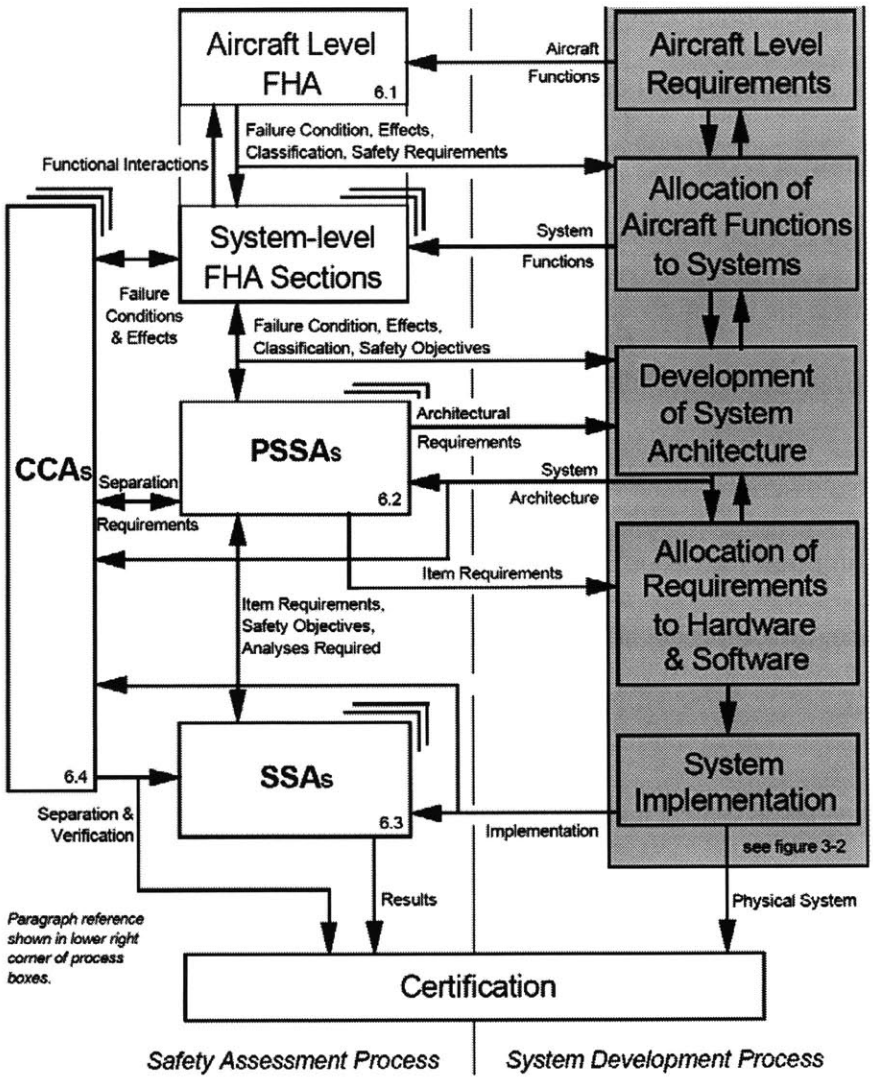


Figure 2 Safety Assessment Process Model from ED-79/ARP 4754 [7]

2.2.2 ARP 4761

The good summary of the ARP 4761 process, from **Y. Papadopoulos and J. A. McDermid is reproduced below [9]:**

" The model is illustrated in **Error! Reference source not found.**. System development is defined as a process of hierarchical system decomposition which is driven by the requirements derived at each stage.

At the first level of this decomposition, the aircraft functional requirements are supplemented with *safety requirements* and allocated to a number of systems. The safety requirements for these systems are established from the results of an aircraft *Functional Hazard Assessment (FHA).*

At the second level of hierarchical decomposition, the potential functional failures of each system are assessed by a system level *FHA,* and a decision is taken on an appropriate system architecture that can meet the system requirements.

Preliminary System Safety Assessment (PSSA) of the architecture follows. The aim of the *PSSA* is to establish the requirements for sub-systems or items of the architecture. The architecture is likely to contain parallel, dissimilar, multiple channel elements. Any assumptions of independence of failure between these elements, must be substantiated. *Common Cause Analysis (CCA)* is therefore appropriate at this stage in order to establish requirements for the physical or functional separation of these systems.

Sub-system requirements are then interpreted to appropriate *development assurance levels* for these sub-systems. When the decomposition process has reached the stage of implementation, these development assurance levels define the techniques (& their rigour) for the specification, development and assessment of hardware and software.

At the final stage, a *System Safety Assessment (SSA)* is conducted to collect, analyse and document evidence that the implementation meets the safety requirements established by the *FHA* and *PSSA.*

In the context of safety assessment, the results from the *CCA* provide the arguments that substantiate assumptions of independence between parallel, dissimilar components. "

In summary, the analyses recommended by ARP 4761 are listed below:

- Functional Hazard Assessment (FHA, at aircraft level and system level)
- Preliminary System Safety Assessment (PSSA)
- System Safety Assessment (SSA)
- Common Cause Analysis (CCA), consisting of
    - Particular Risk Analysis
    - Common Mode Analysis

- Zonal Safety Analysis

The relationships between these are illustrated in Figure 3 below. ARP 4761 provides more detail on common-cause analyses than was available previously and provides a more systematic means to evaluate safety early in the design process and to reduce surprises at the end of the development program [10]. It clarifies methods of calculating failure rates and latent failure exposure times for fault tree analyses.

The methods of analysis proposed above (FHA, PSSA, SSA and CCA) are not universally used by all aerospace companies. In one development site at an aero engine company reviewed, the engine control system had not ever had a CCA performed, and the FHA was used only on more recent projects. Only the PSSA and SSA have been consistently produced since ARP 4761 introduction.

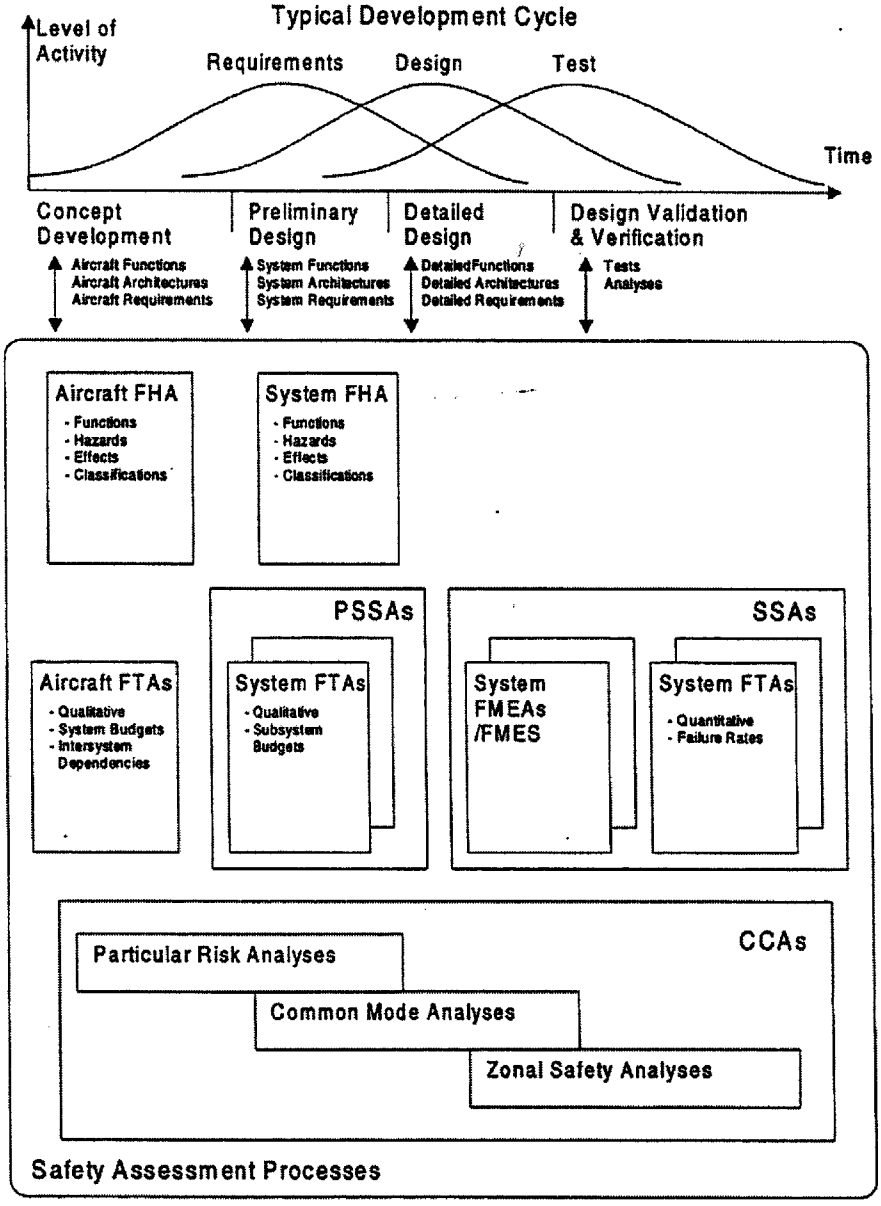


Figure 3 Overview of System Safety Process from ARP 4761 [1]

The basic approach discussed in ARP 4761 is to use the FHA to determine the top-level events that should be analyzed by fault tree analysis. The basic root cause event in FTA is a failure of some component. Therefore, it cannot effectively model design, maintenance or human errors. The CCA should address some of these weaknesses because it addresses violations of redundancy assumptions, such as those due to design errors, latent failures or maintenance errors. However, the CCA analysis seems to be difficult for companies to produce. The

question of how to capture the effect of design errors is very hard because it does not lend itself to traditional probabilistic methods. The approaches discussed in chapter 4 below (STAMP, requirements completeness and Intent specifications) are offered as a possible means to address this.

2.2.3 DO-178B

DO-178B (ED-12B in Europe) provides process guidance specifically for software. Software cannot be said to *fail* in the sense of physical failures, and hence there are no quantitative safety analyses applicable to software. The emphasis is on design and development assurance to minimize the likelihood of design and implementation errors. This assurance is to be achieved through processes for design, review, testing, configuration management, change management and process assurance. The level of activity in these areas that is required on a particular project depends on the safety assurance level required. In a similar way to ARP-4754, four levels are defined, A to D, with A being the highest level. Aero engine control software is all developed to DO-178B level A.

Figure 4 below shows the relationship between DO-178B and the system level safety assessment process. It shows that the feedbacks from the software process include fault containment boundaries, error sources and software requirements and architecture. Fault containment boundaries refer to the boundaries around a partition in a partitioned software system.

Traceability between system and software requirements and design is also fundamental to ensuring that the software meets the system requirements. Appendix A2.2 of ARP-4754 confirms these items as coming up to the system level from the software level. It appears that the assumption is that software process will determine what its fault containment capability is

and provide this to the system level for consideration in the system safety assessment process. In practice however, any known weaknesses in the software will tend to be reviewed and corrected (if known to cause a system effect). The events that cause problems in real systems are generally *unknown* and therefore the information does not flow up to the system level.

An RTCA/EUROCAE special committee is currently considering revisions to DO-178B/ED-12B. One comment in the discussion forum of this committee is that the existing document is fairly prescriptive in terms of the processes to be followed [11]. Other standard documents have made a move towards more "goal-based" approaches. Furthermore, it has been acknowledged that DO-178B does not provide specific guidance on how to achieve software safety, beyond the establishment of a development assurance level to be used for the software development program. It appears that the assumption is that, having established the assurance level, all the emphasis needs to be on ensuring that the requirements that flow down from the system level are faithfully met in the implementation. In other words, safety attributes are essentially handled at system level and flow to the software lifecycle process in the form of requirements. This is in contrast to the approach taken in MIL-STD-882B, for example, which has well defined tasks for "software system safety" (refer to task section 300 [12]). Also, a good summary of the software system safety process of MIL-STD-882B (with comparisons to DO-178B) is provided in sections 2.4 and 2.5 of the US DOD Joint Services Software Safety Committee's Software System Safety Handbook [13]. It is argued in the following chapters that the software lifecycle process can do more to ensure system safety than satisfy the requirements flowing down from the system level. The approach taken in intent specifications (see 4.2 below) aims to achieve this.

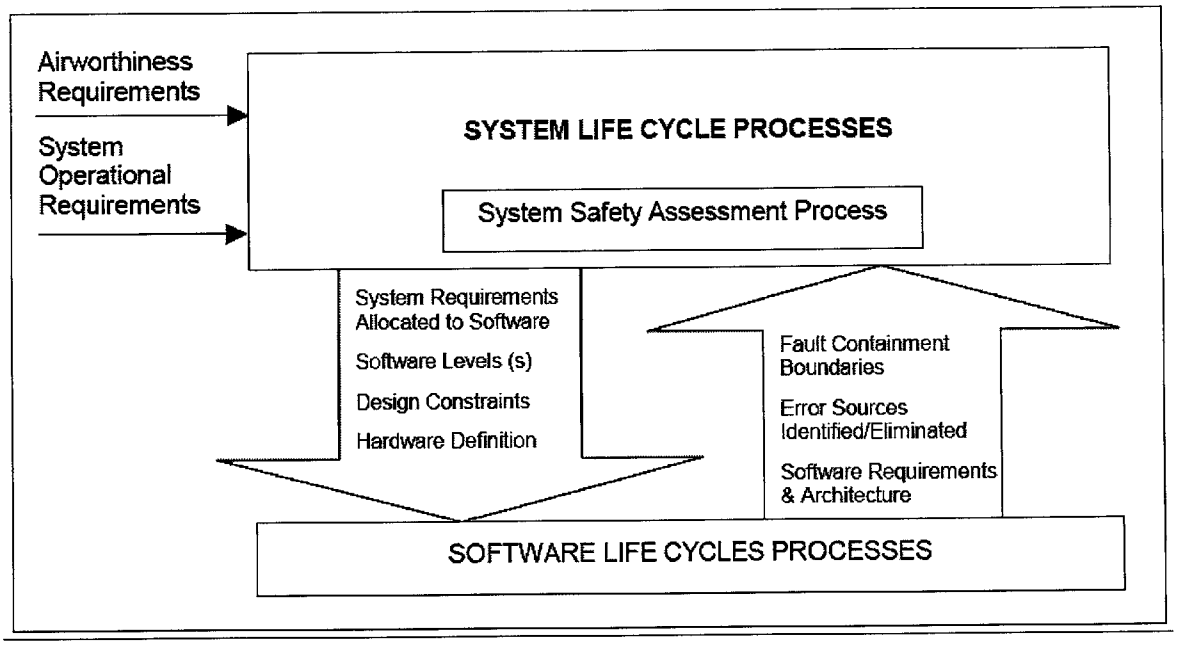


Figure 4 System Safety-Related Information Flow Between System and Software Life Cycle Processes, from DO-178B [2]

2.3 System Safety and Development Processes at one Aero Engine Company

The aerospace industry standards above provide the framework for company specific processes. In one aero engine company studied, the processes for the engine control system and software were strongly influenced by the aerospace standards, such that it is not worth repeating the company processes here.

A review of actual project experience, however, suggests that the system engineering and software engineering processes covered by ARP-4754 and DO-178B have been easier to follow than the safety assessment processes of ARP 4761. In general, the key concepts in ARP-4754 and DO-178B are understood by engineers, such as the importance of traceability, configuration and change management, process and design reviews, system and software validation and verification. The fail-safe concept of AC 25.309-1A is broadly understood and

forms the basis of much architectural thinking. This discipline could be argued to have led to the generally good safety record of digital control systems in the commercial aerospace industry. Some of the practices being adopted in other industries such as SW-CMM have not had a large impact on aero engine control system software because the existing process framework addresses much of what SW-CMM aims to achieve and in a manner that offers greater flexibility to companies to demonstrate compliance to the FARs than does SW-CMM.

Some of the safety assessment processes of ARP-4761 have proven to be difficult to produce, such as the CCA, and the FHA. The CCA should cover design errors in systems and software, but at one particular site at the company reviewed, this analysis has not been performed on any project, so no attempt to try to understand potential redundancy violations has been performed. As the examples in the following chapter will show, several system problems that had a safety impact were caused by design errors and the complex nature of these would have made any attempt at a Common-Cause Analysis very difficult.

3 Analysis of Safety Related Incidents and Issues

In this chapter, examples of design, operator and testing errors in an aero engine control system are reviewed. The control system in question is for a small turbofan engine on a regional jet aircraft. In each case, the review aims to establish to what extent these problems are due to failures to apply the processes discussed in chapter 2 above, or weaknesses in the processes themselves. This review must consider a broad system boundary that includes operator (pilot) and maintenance processes, program pressures etc, rather than just the technical understanding of the event. None of these examples led to an accident, but all required requirements changes (and subsequent software changes) and could have affected system safety. These problems are typical of the types of problems that are found and corrected in the industry on a regular basis. They illustrate many of the common errors discussed in the literature [14-16].

3.1 Engine Safety Hazards Overview

The only catastrophic engine events are uncontained blade release due to shaft over-speed. For this hazard, an automatic over-speed detection system is used to cut fuel off to the engine, thereby removing the energy source driving the acceleration, and hence leading to engine deceleration (probably with compressor surge). The over-speed protection system is microprocessor independent (i.e. uses analog electronic hardware but no digital software), and this form of architecture has been widely used in the industry in recognition of the inherent risks of any software-based system.

However, although the catastrophic events are protected with mechanical systems, there are a significant number of non-catastrophic hazards that do have software mechanisms. The

following two engine hazards (top-level events) receive much attention and analysis, in predicting and monitoring in service event rates.

- Loss of Thrust Control (LOTC)
- In-flight Shut Down (IFSD)

These are broken down into more specific system hazards, such as for different flight phases.

There are several other hazards such as the inability to shutdown an engine, or the failure to restart following an in-flight shutdown. These hazards have significant contributions from software.

3.2 Aero Engine Control System Architecture Overview

Figure 5 below shows the basic elements in the engine control system used for the example events. This architecture was developed during the 1980s and is not representative of new designs in many respects. However, the main systems issues in the events discussed in this chapter are applicable today. There are two identical, but separately housed, FADEC units running identical software (an individual FADEC is often referred to as a "channel"). Both FADEC channels are powered, and read and validate sensor inputs, however, only one channel is in control of the primary fuel flow and compressor variable geometry at any given time. The other FADEC acts as a hot standby.

Sensor input data is shared over an inter-FADEC digital link (CCDL) to allow each FADEC to perform fault-detection of dual-redundant input signals. Each FADEC also performs fault checks on its own output actuator drives and on the data received from the aircraft. The results of these checks are shared across the CCDL. If one of the FADECs detects a fault that prevents it from being able to control the engine, it declares itself "incapable", and this information is

shared across the CCDL and is used in the channel selection logic that determines which channel shall be in control at any given time. If both FADECs are capable, then they both have the ability to drive discrete outputs, such as the command to close the fuel shutoff valve.

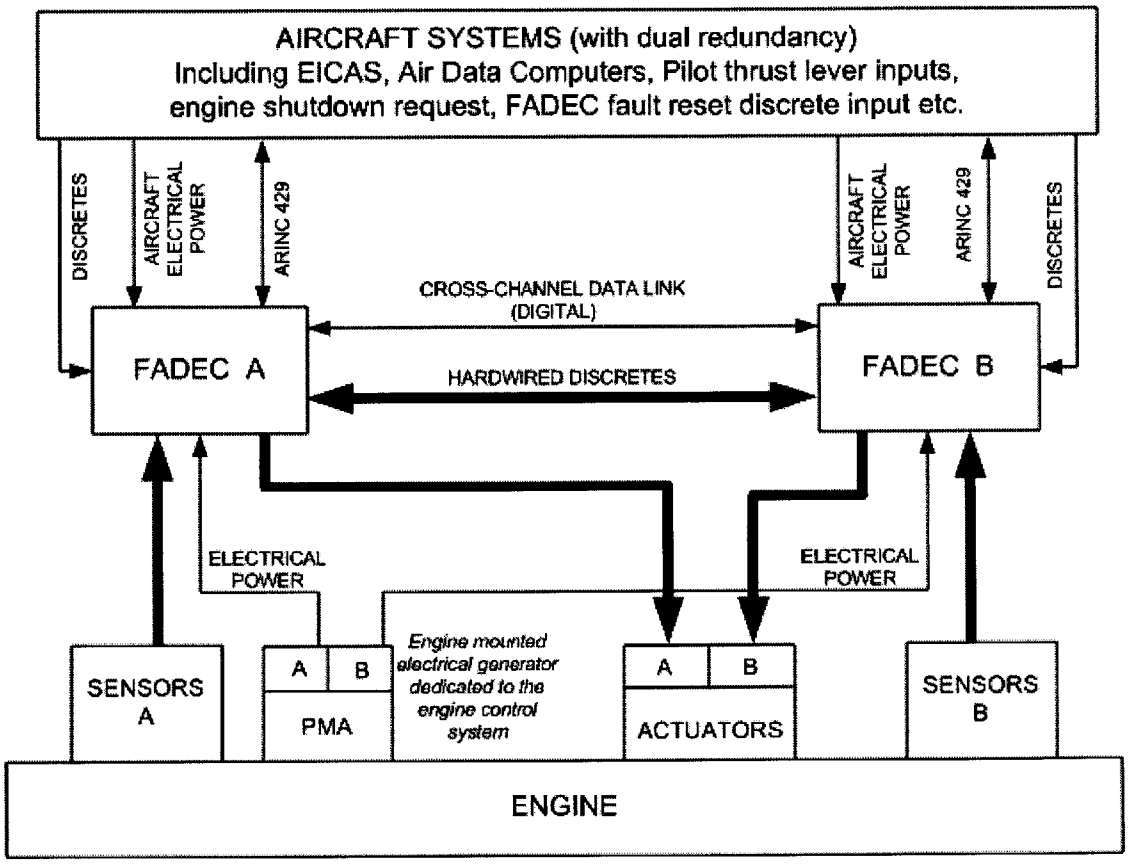


Figure 5 Engine Control System Architecture

The system is designed to provide dual redundancy of the FADEC hardware, external sensor inputs, including aircraft input data and pilot commands, and actuator drives.

Note that many aircraft systems use triple redundancy in which voting is used to select the active system for control. In aero engines, dual redundancy (as shown above) is standard. This does mean that voting schemes cannot be used because only an odd number of systems can generate an obvious winner. Dual redundancy requires logic to detect when a particular input or output system has failed, initially without reference to the other channel. For example, if a particular sensor signal goes outside its normal operating range, it is declared failed. The

alternative channel's signal will then be used. In addition, signals from the two channels can be cross-checked and in the event of a difference exceeding a certain tolerance (accounting for sensor measurement errors and other factors such as transient errors due to communication delays), the signal furthest from an independent model can be declared as failed for the purposes of signal selection. The independent model is typically the expected value of the sensor, based on different sensor signals altogether. For example, a modeled (synthesized) value of the high-pressure shaft speed can be derived from the low pressure shaft speed, taking advantage of the well defined thermodynamic relationships in the engine.

The FADEC typically uses range checks and cross-checks (in combination with model arbitration where appropriate, as discussed above) to detect and isolate faults. It also uses a fault confirmation counter to confirm the existence of a fault (the fault is said to be confirmed if it exists for longer than a defined confirmation time) and provide some protection against transient effects. Once a fault has been confirmed, it is typically latched so that the affected signal cannot be reused even if the original fault condition goes away.

However, the pilot has the ability to reset the latches via a pilot input ("fault reset" button provided on the overhead panel). This reset would only be performed on rare occasions as part of a checklist item or in the event that the pilot was particularly concerned about the engine control.

The FADEC software executes at two iteration rates. A fast rate of 25ms is used for main closed-loop control functions and the slow rate, at 125ms, is used for less time-critical functions. In the following discussion, an "iteration" refers to a 25ms cycle in the fast loop of the software.

3.3 Engine Torched Start Incident

This example is chosen because it illustrates the extent to which latent errors can exist in a system and how a hazardous condition can be created after apparently small changes are made that expose new mechanisms. In this example, a "torched" start (flame observed at the back of the engine during a ground start attempt) occurred in engine ground-tests at the aircraft manufacturer, while testing a new version of FADEC software prior to certification. The design was changed after the incident, prior to certification, and with no further incidents. The FADEC software had already been through the system development process and extensive system verification tests at the engine manufacturer prior to release to the aircraft manufacturer for engine/airframe integration ground-tests (no flight). Several anomalies were observed in this incident, which has parallels with incidents and accidents reported in the industry.

3.3.1 Incident Summary

An engine ground start was performed, but the test procedure included cycling power to both FADEC channels (causing an intentional dual channel reset) and then cycling power to one FADEC channel (causing an intentional single channel reset) prior to moving the engine start switch to *on*. This sequence had been performed in previous ground-tests and was designed to test the robustness of the system to power interrupts. Power interrupts are more likely to occur in the FADECs prior to engine start because the dedicated FADEC power supply (from a permanent magnet alternator) is not available until the engine has spooled up to a certain (low) speed. Prior to this, the FADECs run on aircraft battery electrical power, which is not designed to be guaranteed reliable.

When the engine start was requested, the pilot observed fuel introduction (on the cockpit displayed fuel flow) to the engine without ignition (no indication of a temperature rise), which

was interpreted as a "no-light" start, the procedure for which is to shutdown the engine and perform a "dry motor" to flush out the un-burnt fuel before attempting another start. In the "dry motor" procedure, the engine is spooled up by the starter motor, but fuel and ignition are both set to *off*. The airflow through the engine flushes out any un-burnt fuel that has collected in the combustion chamber. This avoids a potentially large flame that could occur in the subsequent start attempt. However, the dry motor had the opposite effect to that expected, and a large flame was observed at the back of the engine during the dry motor.

Analysis by the engine manufacturer revealed a complex mechanism in which one of the FADECs became locked in a ground-test mode throughout the whole sequence. The other FADEC attempted to perform the engine start, but the mode mismatch between the two FADECs led to abnormal behavior in which the latching shutoff valve (LSOV) in the Fuel Pump and Metering Unit (FPMU) was inadvertently opened too early in the start sequence, thus creating the initial no-light start. Subsequently, when the pilots shutdown the engine in preparation for the dry-motor procedure, this valve was left open. Additionally, the FADEC that was attempting to perform the engine start subsequently declared itself "incapable" and one of the fail-safe actions in this scenario is to set ignition on. Another factor was that the FADEC that was locked in ground-tests was sending out a maximum (electrical) current to the fuel metering valve actuator (this was part of the ground-test sequence), which led to the FPMU delivering the maximum fuel flow rate possible (limited only by the pump speed). All these factors led to a large flame. There was no risk of the engine accelerating to a self-sustaining speed, and the engine actually surged and spooled down without further incident.

3.3.2 Overview of Design and Changes Made in the Development Software

The software under test included two particular changes relevant to the incident mechanism. Figure 6 below shows the relevant events and design errors and features that contributed to the incident. This diagram is similar to a cause-consequence diagram but uses a different symbol set. A labeling scheme is used to identify the events and design features. The two changes in the development software are labeled CN1 and CN2. CN1 was an intentional change that was associated with a new power-up check to determine the hardware configuration standard of an external device. This check was executed only in a ground power-up (that is when the FADEC is powering up with the engine not running and the aircraft on the ground). The check needed to be performed in both FADEC channels and each channel waited for the other channel to annunciate its check results before the FADEC could exit power-up checks mode (also known as self-tests). This design was developed to ensure both FADECs agreed on the results of the test, thus increasing the reliability of the check result. The idea was that the check should be over well before an engine start attempt was made. However, just in case there was a circumstance in which a FADEC was still waiting for test results when a start was requested, an unlock feature had been designed to pull a FADEC out of this mode when entering start mode.

The second software change, CN2, was an unintentional requirements change that arose from a redesign of the fuel scheduling during engine start. Although not explicitly intentional, the change was known before this incident and its impact evaluated and determined to be acceptable. The change meant that when an engine start was requested by the pilot, a FADEC would only transition from ground-test mode to starting mode when N2 exceeded 500rpm.

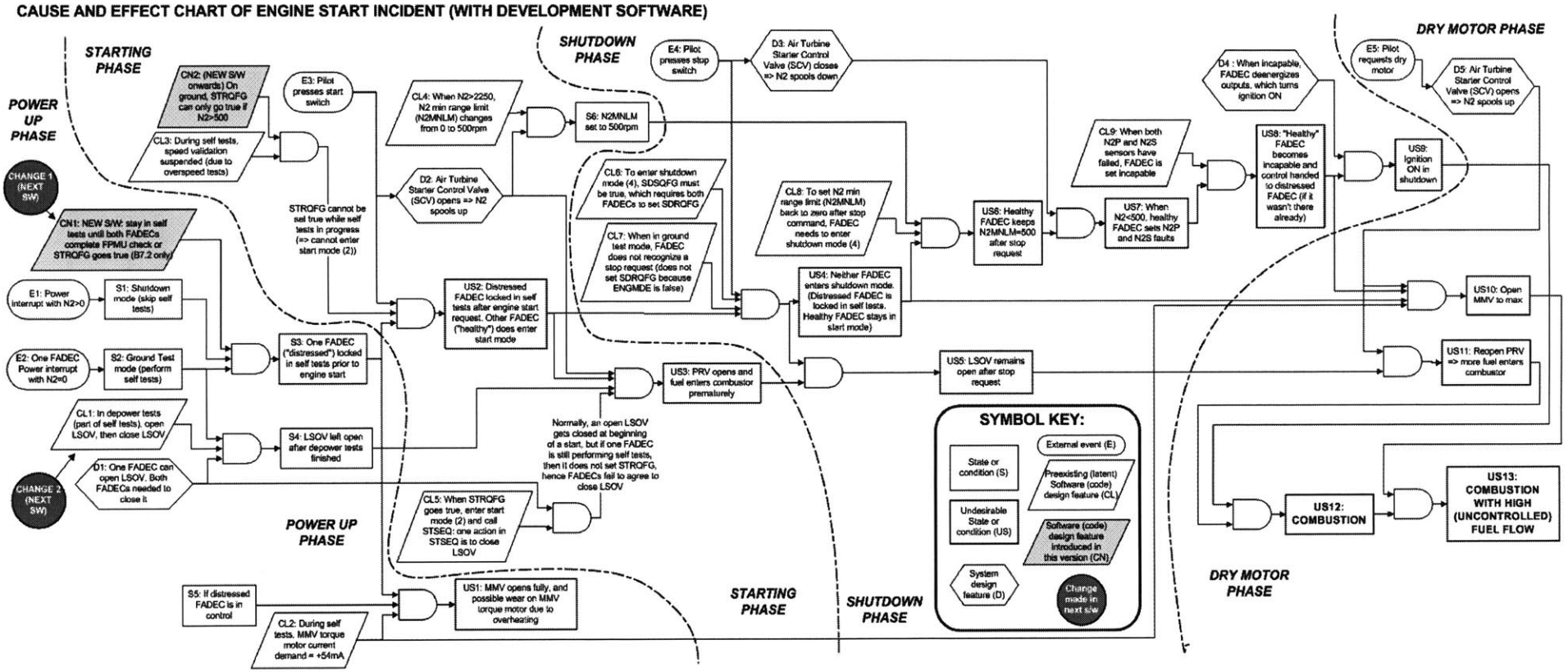


Figure 6 Cause and Effect Analysis of Engine Startup Anomaly

Note that the pilot start request switch is directly hardwired to the Air Turbine Starter (ATS) valve, independent of the FADECs. Thus, N2 would be expected to climb above 500rpm soon after the start request, and this 500rpm threshold would therefore have no functional impact on the start sequence. However, an existing feature of the FADEC ground power-up sequence is that the N2 signal is not validated while executing self-tests. This is because one of the self-tests is to exercise the over-speed shutdown system by creating a simulated high N2 speed signal to check that the microprocessor independent over-speed shutdown system operates correctly. Therefore, while self-tests are in progress, the N2 signal internal to the FADEC is set to zero to avoid creating confusion with the FADEC logic while this simulated high value is being generated in hardware. This test is only executed in a ground power-up with zero initial shaft speed, so the forcing of this signal to zero is not normally a problem.

Prior to the incident, the engine had just been shut down, and when the dual power reset was performed, N2 had not yet dropped to zero. This meant that the FADECs powered up in shutdown mode rather than in ground-test mode. This is not in itself a problem, as a subsequent start request would allow both FADECs to enter start mode. However, in this incident, a single channel power interrupt was also performed just prior to a start request. This second interrupt occurred when N2 had dropped to zero, so this FADEC did power-up in ground-test mode. This FADEC therefore performed self-tests in which the N2 speed signal was forced to zero. It also performed the new hardware configuration test described above and waited for a corresponding test result from the other FADEC. However, because the other FADEC had powered up in shutdown mode, it did not perform this particular configuration test. Consequently, the FADEC in ground-test mode got locked up, waiting for a test result from the other FADEC that never came. However, the unlock feature in that check should have been able to pull the FADEC out of that test when the start was requested. However, the CN2

change meant that this FADEC could only enter start mode when N2 exceeded 500 rpm, which was not possible because the self-tests were still in progress in which this signal was forced to zero. Hence this FADEC remained locked in ground-test mode after the start request was received.

When the pilot pressed the start switch, the FADEC that was in shutdown mode did enter start mode. One of the first tasks performed in start mode is to close the FPMU latching shutoff valve. This may seem like an unusual action considering that this valve should be closed to begin with. In fact, the valve would be closed, but under some circumstances the latching solenoid could get into the wrong state during power-up tests. One possible cause of this solenoid being in the wrong state was a result of the system design of the LSOV. To ensure that a single FADEC could not inadvertently cause an engine shutdown, agreement between FADECs must be obtained for the LSOV to close. This is a good fail-safe design when the engine is running. It ensures that no single electrical failure could lead to an engine in-flight shutdown (IFSD). Part of the ground power-up tests involved energizing the LSOV open and then closed. This was to test the continuity of the LSOV circuits. However, a feature that was never appreciated in the system design was that in the event of an asynchronous ground power-up (that is, the two FADECs powering up more than 2 seconds apart in time), the LSOV would be left in the open state following the power-up tests (because the two FADECs would pass through the "close LSOV" task at different times and hence fail to get agreement on this action. Despite this feature not being appreciated/understood, it never had any functional effect, because the first task that was performed when an engine start is commanded was to go ahead and close the LSOV. This would typically be done synchronously and thus would ensure LSOV closure. The requirements that specified that the LSOV be closed on entry to start mode may well have been designed to address the problem of a potentially open LSOV from ground-

tests, but no rationale was documented about these requirements, and the memory of why this feature existed had been lost over time.

Therefore, when the pilot requested the start, only one FADEC entered start mode and the other "distressed" FADEC was waiting for a test result that would never come. Consequently the backup action of closing the LSOV on entry to start mode did not work, because agreement between FADECs was not achieved and hence the start was commenced with an LSOV in the open state. The LSOV had been left open due to the asynchronous power-up. When the engine started to spool up, the fuel pump pressurized the fuel system enabling the valves in the system to start moving. The LSOV is connected to the PRV (pressure raising valve), which is the last fuel valve before the engine combustion chamber. Thus when servo pressure was available, fuel could start to flow into the combustor at a much lower engine speed than normal for fuel introduction. In a normal start, the PRV would only be opened at a speed of at least 25% HP shaft speed.

Several other errors and features in the design compounded the problem of premature fuel introduction. Firstly, the FADEC in control happened to be the one locked in ground-test mode. It was still in "self-test" mode and one action in this mode is to set the main metering valve torque motor current to its maximum value. This is not normally a concern in self-test mode when the engine should be static and shutdown. However, in this incident, with the engine spooling up and the PRV open, the main metering valve was also allowed to become fully open. The fuel flow rate would therefore be the maximum that the fuel pump could deliver, without regulation by the metering valve.

Even this scenario of premature fuel introduction would not necessarily have been a problem. The pilot observed fuel introduction without light-off and hence initiated a shutdown to be

followed by a dry motor, as discussed in 3.3.1 above. This course of action by the pilot is in accordance with standard pilot procedures for a "no-light" engine start. However, further complications arose in this shutdown. First, the FADEC locked in ground-test mode was unable to recognize the shutdown request. The fail-safe feature in which both FADECs must agree before executing a shutdown is a feature of hardware design (interlocks) and software (mode transition logic). Thus neither FADEC was able to enter shutdown mode. This created a further complication for the FADEC that was in start mode. In that FADEC, when N2 exceeded 2250 rpm (which was achieved in the aborted start), the N2 fault detected logic changed the minimum range check limit for the N2 signal to 500 rpm. This is designed to improve fault-detection for a running engine. When the engine is running, the N2 speed would always be well above 500 rpm. This range check limit would revert to 0 only when shutdown mode is entered, which never happened, hence this FADEC detected a perceived dual channel N2 speed signal fault when the engine spooled down. Total loss of the N2 signal is very serious and the affected FADEC declared itself incapable. One of the actions performed when a FADEC goes incapable is to default its ignition system to ON (a fail-safe action).

It is interesting to reflect on the N2 fault-detection logic here. If the engine had been running (at or above idle speed) and the FADECs had failed to agree to perform an engine shutdown, then the engine would never have spooled down. The minimum range check limit could have stayed at 500 rpm without any problems. However, in an aborted start sequence, the only thing keeping the engine shafts spinning is the torque from the starter motor, which is not under FADEC control. Therefore, the normally simple logical connection between FADEC shutdown agreement and actual engine spool-down does not apply during an aborted start, and this system feature was completely missed in the requirements. Essentially, there was an implicit

assumption in the requirements that there was a causal link between FADEC shutdown agreement and engine spool-down. No such assumption was ever documented however.

Consequently, we now have an engine spooling down with one FADEC incapable and the other locked in ground-test mode. The pilot then initiated a dry motor, which involved selecting the start switch to ON with ignition set to OFF. When the FADECs perform a start with ignition off, they will not open the LSOV, and hence no fuel will be admitted to the engine. However, in this incident, neither FADEC was able to enter start mode. Furthermore, the LSOV was still left open from the mechanism described above and the FADEC that was incapable had set ignition on as a fail-safe action, overriding the ignition off command from the pilot. The FADEC in ground-test was still in control and had opened the main metering valve fully open. Consequently all the conditions were set for combustion in the dry motor. A high fuel flow rate in combination with low air speed may have led to a large flame visible at the back of the engine, but was not particularly hazardous, as an engine is unlikely to sustain this condition or accelerate. It is much more likely to stall and run down.

3.3.3 Analysis and Lessons Learned

In this incident there were nine latent design features of which about five could be argued to be requirements errors or could create conditions that were not adequately handled in the overall design. There were also two newly introduced problems. In chapter 15 of Safeware [5], a comprehensive list of requirements completeness criteria are presented. These criteria apply to state-based requirements and the relationships between state-based logic and the external environment. Many of the issues observed in this example could have been caught using these requirements completeness criteria. See Table 1 (in APPENDIX B Requirements

Completeness Criteria With Example Systems) for details. The following summarizes the process weaknesses.

1. Off-normal conditions not adequately tested or not tested in combination :
    a. Power interrupts during a shutdown sequence
    b. Asynchronous power interrupts
    c. Start request shortly after a power interrupt (while self-tests still in progress)
2. No completeness criteria assessed against state-based requirements.
3. Lack of coordination between hazard analysis and software design process. That is, potentially hazardous states not identified in the software design.
4. Lack of documentation of assumptions. For example, the original design of power-up tests assumed the two channels would go through this mode synchronously. There was actually a two second tolerance on this, implemented as a software design assumption but was not flowed back up the system hierarchy to create a system constraint.
5. Lack of documentation of design rationale. For example, the rationale for closing the LSOV on entry to start mode seemed in retrospect to be valuable as a means of addressing the possibility of an LSOV left open after an asynchronous power-up, but this interpretation was inferred only after analysis of the incident.

Design weaknesses are summarized below :

1. Design did not provide for the use of defaulted data. For example, when the N2 signal was defaulted to zero, the downstream software used it without regard to its validity. The completeness criteria in Table 1 below identify the need to specify timing bounds on the use of input data.

2. Lack of detection of mode mismatch between the two FADEC channels. The analysis revealed the significant problems that could result when two channels are allowed to operate in different modes for a prolonged period. Analysis of the design should have recognized the risks of this situation and should have logged a mode mismatch if it lasted for a "significant" period (perhaps over 20 seconds, which would allow time for a FADEC to recover from a power interrupt).

3. Controller conflict. In this case the pilot start/stop switch is directly connected to the Air Turbine Starter (ATS) control valve, and hence it is outside the control of the FADEC. Thus, a conflict of competing controllers could result. The aircraft/engine system architecture should have highlighted this conflict and led to requirements in which special attention was given to potential command conflicts. In this example, the software did not consider the possibility that an engine could spool up when a FADEC was in ground-test, because the FADEC in that mode would not have commanded a start. Similarly, the software did not handle the possibility that an engine could spool-down without the FADECs agreeing to command a shutdown. Hence the minimum N2 range check limit would remain above zero, which subsequently caused one FADEC to go incapable on the grounds that the N2 signal had dropped well below what would be regarded as a minimum possible value for a running engine.

3.4 Pressure Sensor Redundancy Violation

This example was chosen because it is a typical example of a redundancy assumption violation, and it concerns a dual redundant pressure signal that measures air pressure at the inlet to the high-pressure compressor. The pressure signal is known as "P25" (station 2.5 in the engine, following the international station numbering scheme for jet engines). This pressure is used by

the FADEC to limit the maximum and minimum fuel flow to the engine, to help avoid and recover from surge and to help avoid combustor lean blowout. Small errors in the signal have no effect on engine control because the signal is not used as part of the primary thrust control loop. However, large errors can lead to a loss of thrust control (LOTC) or combustor flameout (causing an in-flight shutdown, IFSD).

3.4.1 Event Summary

In this event, a loose connector for FADEC channel A led to intermittent open circuit faults on a number of sensor inputs carried by this particular connector. The open circuit conditions occurred particularly when vibration levels increased such as when the engine power level was changed by the pilot. These open circuit faults occurred over the course of a flight lasting about one hour. All the affected open circuits were successfully detected and accommodated except for the P25 signal. The P25 logic initially declared that the channel B P25 signal had failed, which forced the controlling FADEC to use the intermittently erroneous channel A signal. This led to uncommanded thrust perturbations (an LOTC hazard).

The pilot subsequently performed a fault reset, which cleared the condition, and the FADEC A then detected its own signal as having failed. The thrust oscillations were reported, and investigated by the engine manufacturer, making use of fault recording data inside the FADEC and in the aircraft Central Maintenance Computer. The aircraft completed its flight with no further anomalies.

3.4.2 Review of Design

The FADEC input conditioning circuitry is designed to ensure that open circuit faults lead to an input signal voltage that quickly goes out of the normal operating range (ideally within one

iteration), enabling easy detection of the fault. However, this is not always the case and drifts of a signal towards an out-of-range condition can occur.

During such drift scenarios, erroneous in-range sensor readings are presented to the fault-detection software. In the case of the P25 fault-detection software, a synthesized model of the P25 signal based on fan speed (and considering external air pressure and temperature) was used to arbitrate between the channel A and B readings once a cross-check tolerance is exceeded. Provided the synthesis model is sufficiently accurate compared to the cross-check tolerance, the model is able to reject the failing signal without difficulty, and ensure that the healthy signal is selected for engine control, with no noticeable effects on control. However, the more inaccurate the synthesis model is, then the higher the probability that the model will "pick" the wrong signal. The software was designed to pick the signal closest to the model for up to 10 iterations (250ms) after which a fault confirmation counter would latch the signal furthest from the model out and prevent use of that signal for the rest of the flight (or until the pilot performs a "fault reset" or a power interrupt occurs).

This design had a number of weaknesses. First, the accuracy of the synthesis model was not well understood; it was found that model errors could be large in comparison to the cross-check tolerance at some conditions, leading the model to potentially prefer the failing signal in the event of a cross-check failure. Secondly, when the signal does go out of range, the cross-check and model arbitration logic was allowed to continue operating with the "last good" value of the signal before it went out of range. At this point, it should have been clear that the out-of-range signal had failed, but the range check logic did not pass back the out-of-range fault flag to the cross-check logic until 250ms later (when its own fault confirmation counter had latched). Thus the cross-check logic initially had no knowledge that the signal it was using was

a "last good" value and hence should be treated with suspicion. If the "last good" value happened to be closest to the model, the cross-check would then latch out the healthy signal after 250ms. The range check fault counter would subsequently latch out the failed signal, thus leading to both channels being declared failed.

This design had to be substantially overhauled. The use of two separate fault counters (one for out-of-range failure and one for cross-check failure) that could get into a race condition has been avoided in the new design and the synthesis model has been optimized to improve its accuracy. Furthermore, the detailed implementation of the cross-check has been changed to improve its robustness, along with a significant lengthening of the (single) fault confirmation counter.

3.4.3 System Design Lessons

In the example application, the system requirements essentially stated that the system "shall detect and accommodate single channel electrical faults", such as the one in this example. Some aircraft manufacturers have expressed the fault-detection and annunciation requirements with a quantitative goal, such as "at least 99% of single channel faults". The FAA guidelines call for all single electrical faults to be accommodated.

In this example, the design had already been through a design change about two years previously, to address earlier weaknesses in the design. The earlier changes had greatly reduced the frequency of these events but had not eliminated them. Schedule pressures in the first change had curtailed the amount of analysis that could be performed and hence a tactical solution that addressed the known problems was performed, without conducting a widespread review aimed at finding other potential weaknesses.

One contributing factor was a lack of awareness of best practice in fault-detection design, and a second was a lack of time planned for a thorough analysis. A full analysis of fault-detection requirements and designs such as this takes a lot of time and effort. A full analysis requires an understanding of the failure modes of the sensor and associated systems (including harnesses, connectors, FADEC input-conditioning circuitry etc.), an understanding of the consequences of erroneous signals on the engine control at various operating conditions and flight phases, and an understanding of the sensitivity of any models used as part of the detection scheme, including both steady-state and transient accuracy. Furthermore, the ideal design process would have started with a system architecture that considered the best design of FADEC input conditioning circuitry to support easier fault-detection. All too often the hardware is designed with assumptions about the control functions and may not use the best techniques available.

3.4.4 System Safety Lessons

The first issue from a safety perspective was that this problem was caused by a loose connector, which is regarded as a maintenance error rather than a random physical fault. Second, the fault tree assumed that a single channel open circuit would have been detected and accommodated and therefore would not have led to a LOTC event.

The safety case relied heavily on the FTA due to the lack of a Common-Cause Analysis (CCA), and the FTA is designed to match the system "as specified", and not "as built". It is impossible to develop a safety case that realistically accounts for design errors that are not actually known. The weaknesses of the design were actually known before this LOTC event, but an earlier, simpler design change was assumed to have addressed the service problems. In practice, the earlier design change did greatly reduce the probability of these events.

When a complete redesign was implemented after this event, the safety analysis was reviewed to determine what changes would be needed. No changes were required, because the design change simply makes the system more compliant with the original system requirements. It seemed paradoxical that a safety-related design change would have no impact on the safety case. The designers of the new fault-detection software were confident that the new design would now finally address any weaknesses in the fault-detection logic, and this was consistent with all the analysis that was performed, as well as system testing.

If there were any weaknesses in the new design, then these would have been unknown. Indeed, had there been known weaknesses, these would simply have been addressed in the design. Therefore, the lack of awareness of where redundancy violations may occur makes it unlikely that these will ever be addressed in the existing safety analyses. In other words, design errors cannot be modeled in a system safety analysis.

Furthermore, it is the design errors (and maintenance errors etc.) that are likely to have the greatest impact on the safety of the system. This is because, in the absence of design, maintenance or operator errors, any random failure that does occur will lead to top-level hazards as predicted by the safety analyses, which will thus be at rates that can be managed. This is particularly true in systems with a lot of redundancy because (assuming the redundancy always works), the system will be less sensitive to increases in the failure rate of random base events.

3.4.5 Process Lessons

The conclusion from the above is that the quantitative approaches to safety analysis are not helpful in determining the real safety of a complex system, where design errors are likely to play a dominant role.

The qualitative analyses have proven difficult for the industry to perform and as a result, the actual safety level of complex aero engine control systems are hard to define. Clearly more guidance is needed on how a safety case can be developed that considers potential design, maintenance or operator errors, and redundancy assumptions should be backed up with analyses aimed at establishing more thoroughly the weaknesses in designs.

3.5 Engine Thrust Shortfall Warning Anomalies

This example involves a higher level function with a pilot interface. The aircraft application involved was a twin engine regional jet, in which reduced thrust takeoffs are permitted with an ATTCS (Automatic Takeoff Thrust Control System). An ATTCS is an FAA-allowed system in which the engine takeoff power can be set up to 10% below that required to meet single engine-out climb performance. The system must be capable of automatically detecting an engine failure and automatically raising the thrust on the good engine by 10%, so as to achieve the required climb performance. Additionally, if the engine failure is detected before the pilot decision speed, V1, then the pilot must abort the takeoff. Thus a message is provided in the cockpit, known as a "LOW N1" warning (a reference to the engine fan speed, N1, which is the engine thrust setting parameter). This warning is displayed on the EICAS. EICAS messages are categorized into three priority levels: warning (red), caution (amber) and advisory (cyan), with warning being the highest priority.

The engine failure detection and thrust bumping logic was implemented in the FADEC, and a signal is sent to the aircraft EICAS system for displaying the "LOW N1" warning. Suppression of the warning above V1 was a feature of the aircraft EICAS logic.

High-level requirements for the engine failure detection logic were therefore provided by the aircraft manufacturer, but the design of the detection logic was the responsibility of the engine

manufacturer. Clearly because this system involves the engine and aircraft interface, with the pilot in the loop, close coordination of the interface was needed. The aircraft manufacturer performed integration tests specially designed to look for false positives (nuisance messages) and failures to detect real thrust shortfalls. This system was regarded as safety-critical and received a significant amount of design and test attention at both companies (aircraft and engine).

3.5.1 Flow down of high-level requirements

When an engine loses power during takeoff, the aircraft acceleration will be slowed, so a longer field length will be required, assuming the takeoff is not aborted. The speed of response of the ATTCS system will determine how quickly the good engine power is bumped 10% after the bad engine loses power. This will affect takeoff field performance calculations, which are published in the airplane flight manual. The field performance calculations were based on early tests of the engine in which engine failures were simulated by the pilot pulling the fire-handle. This is a device the pilot uses to shut down an engine in the case of a fire emergency, and this cuts fuel to the engine independently of the FADEC. The FADEC will therefore not know why engine power has been lost and should thus trigger an ATTCS event. Note that a normal engine shutdown is commanded by the pilot using a run/cut switch and this signal is routed via the FADEC. The FADEC inhibits the ATTCS logic when performing a normal engine shutdown.

The aircraft requirements specified that the ATTCS system must be sensitive enough to detect steady-state thrust losses as small as 3%. The time to detect this was not specified, but the initial FADEC design allowed 12 seconds to confirm such a power loss. Larger losses had to be detected sooner, though no specific aircraft requirements were defined. A detection time of

0.5 seconds was used in the FADEC software requirements for rapid power losses (logic based on rapid HP shaft (N2) deceleration).

The aircraft takeoff performance calculations were based on the results of ATTCS response times from testing of engines with early FADEC software. One key feature of the calculations was an assumption that the engine thrust would not drop more than 40% below nominal takeoff power before the ATTCS system triggered.

The engine manufacturer did not include this function in its own safety analysis because it is an aircraft function and does not contribute to any of the top-level engine hazards. The probability of an engine failure is covered in the engine safety analysis, but not the operation of the ATTCS. The aircraft manufacturer performed the safety analysis of this function, and this was based on the above assumptions about the ATTCS responsiveness.

3.5.2 ATTCS design

The logic implemented involved two basic approaches for detecting a power loss. One based on a rapid deceleration in the HP shaft ("N2DOT" logic) and the other based on a prolonged shortfall (12 seconds) between the requested fan speed, N1R, and the actual speed, N1. Both forms of logic were only active when the throttle was near the takeoff throttle setting. The 12 second period for the second type of logic ("slow" logic) was designed to avoid false warnings in a normal acceleration to high power, when N1 would naturally be below N1R until the engine had had time to accelerate to full power. Both FADEC channels executed the logic, but in the initial EICAS design, only the warning from the controlling FADEC was used to set the cockpit message.

3.5.3 Problems experienced

The early logic was found to be too sensitive in some situations, which resulted in false warnings when the pilot advanced the throttle in a certain way at the start of the takeoff. These led to some unnecessary aborted takeoffs at low aircraft speed. This was an inconvenience but not a safety hazard. Before new FADEC software could be implemented in the fleet, advice was provided to operators about how to advance the throttle to reduce the risk of these problems. New FADEC software was introduced in which some of the tolerances in the N2DOT branch of the logic were opened up to avoid these problems.

At about the same time as these improvements were being introduced, however, another change was being implemented in the EICAS display logic that allowed cockpit warnings to be based on data from either FADEC, rather than just the one in control. This change was independent of the new ATTCS FADEC logic. It transpired that this exposed a previously hidden problem with the logic, which led to false warnings at high aircraft speed. The warnings occurred close to V1, which led to high speed aborted takeoffs that were much more stressful than low speed aborts. In all cases there were no actual engine failures. Analysis revealed that the standby FADEC was generating false warnings. The reason was that under some conditions the standby FADEC could end up in a different thrust mode from the controlling FADEC, and thus believe that the requested fan speed, N1R, was higher than the controlling FADEC. Because the controlling FADEC was the one actually controlling the engine, the real fan speed achieved, N1, would match the lower demand of that FADEC. The standby would thus complain that the engine was not at the required power level, and this used the 12 second slow logic, hence leading to warnings 12 seconds after the start of the takeoff. This was typically close to V1 speed. Further requirements changes were implemented to ensure that the standby

FADEC would always be in the same thrust mode as the controlling FADEC, thus correcting this problem.

However, when this new logic was undergoing aircraft integration tests, it was found that some fire-handle-induced shutdowns were undetected by the FADECs. This caused some confusion because the logic changes at that time did not affect the sensitivity of the logic, and this problem had not been observed in previous tests. It transpired that all the previous tests had been conducted on prototype aircraft, but the new tests were on a production aircraft. The prototype aircraft was wired slightly differently than production aircraft with respect to the fire-handle. On the prototypes, the fire-handle-shutdown signal closed fuel valves in the wing roots and also sent a signal directly to the engine FPMU (fuel pump and metering unit) to close the valve there. On production aircraft, the signal only went to the fuel valves in the wing roots. The difference meant that on production aircraft, there was an additional 7 gallons of fuel in the lines from the wing roots to the engines, and hence this would continue to burn for a few seconds after the shutdown command, thus leading to a much slower shutdown. Instead of a rapid deceleration of HP shaft speed, these tests showed a gradual deceleration, which was then dubbed a "splutter out". It was just slow enough to evade the N2DOT detection logic. The FADEC did detect the problem using the "slow" logic after 12 seconds, but this was deemed to be too slow for this type of shutdown.

The error required a more significant redesign of the logic, and it was decided to start by carefully reviewing all the aircraft and engine system requirements before proceeding. This exercise revealed that several assumptions about the ATTCS that had been central to the takeoff field performance calculations had never been communicated to the engine company as requirements. These included the assumption that the ATTCS would trigger before thrust had

dropped 40% below takeoff power. The design of new logic to meet this requirement, while avoiding the possible reintroduction of false trips, was quite a challenge. The new design was constrained by the need to develop a system that was compatible with the existing airplane flight manual published takeoff field performance charts. Ultimately new system requirements were agreed between the aircraft and engine companies that allowed for a significant improvement in responsiveness without new false warnings.

3.5.4 Lessons Learned

- Assumptions built into important aircraft takeoff field performance calculations had not been flowed down as requirements to the engine manufacturer.
- Aircraft-level safety analyses were based on the same assumptions about the operation of the ATTCS.
- Several aircraft used for airframe/engine integration tests had configuration differences from production aircraft, which affected test results.
- Changes unconnected with the ATTCS (in this case to the EICAS) had side effects that exposed a latent ATTCS error in the standby FADEC. This problem arises from the coupling between complex systems.

3.6 Field Maintenance Issues

The cases described above illustrate how easy it is for problems to arise when system complexity is high. Numerically based system safety analyses are not accurate when the likelihood of requirements and design errors are high in comparison to the likelihood of random mechanical failures. A further example of the problems caused, in part, by system

complexity is the widespread problem of No Fault Found (NFF) removals in the aerospace industry.

The engine FADEC systems and other digital systems on the aircraft all have a significant amount of BITE capability. The faults detected by all these systems are typically recorded in a central maintenance computer (CMC) or similar system on the aircraft. Each BITE capable system usually has its own internal non-volatile memory (NVM) for recording fault and other diagnostic data that can be analyzed by the unit manufacturer when returned for maintenance.

When faults are detected and annunciated to the aircraft, field maintenance engineers will attempt to diagnose the problems with the aid of a Fault Isolation Manual (FIM). The FIM provides detailed instructions on how to isolate the cause of the problem and thus which LRU or LRUs to replace on the aircraft. Replaced LRUs are then sent to their manufacturers for testing. However, a high proportion have no fault found (NFF), in some cases over 50% [17]. The problem of NFF removals is proving to be a rapidly growing problem in the industry. The causes of NFF removals have been classified into three groups:

- Troubleshooting Procedures: The aircraft operator may have certain maintenance policies that exacerbate the NFF rate, such as a policy of tending to remove the digital devices first, as they are usually easiest to remove. The FIM is intended to provide an effective sequence for diagnostic activity, but this sequence may be time consuming and the pressure to return the aircraft into service to meet flight schedules could lead the maintenance engineer to short circuit the FIM and replace more parts than needed. In this NFF category, the part should not have been removed had the FIM been correctly followed.

- Fault Isolation Manual: Even when the FIM is carefully followed, NFF removals could still occur. The fault messages are designed to isolate the problem as closely as possible to a specific LRU, but isolation is not always possible. For example, when a connector is loose, several systems may appear to have "failed". It is not obvious whether the diagnosis should start with the devices attached to the connector, or with the harness, or the connector itself. Sometimes similar fault messages may appear in the event of a power supply loss. The diagnostic sequence in the FIM is designed to identify which mechanism is most likely to have led to the observed fault message or messages, but the reliability of this diagnostic activity depends on how well understood the system is under the various failure scenarios. For example, intermittent faults can be almost impossible to detect, and the mere act of breaking into a system to perform diagnosis may remove the problem. In this NFF category, the recorded faults should not have indicated that the part be replaced, but errors in the FIM led to inappropriate removal.

- System Design: For example, the design of the fault-detection software may have detection limits that are too sensitive, leading to false trips or the design may not be very good at handling certain off-nominal scenarios such as power interrupts, communication interruptions between devices etc. In this NFF category, the fault messages did incriminate the removed part, but the fault messages were themselves erroneous, arising from design errors in the fault-detection logic.

Figure 7 below shows the relative frequency of NFF causes in one study and breaks down the three categories above into subcategories. In the system design category, only a very small proportion is labeled "software error", with the major contributors in this category being poor system understanding and inappropriate limits for the detection logic.

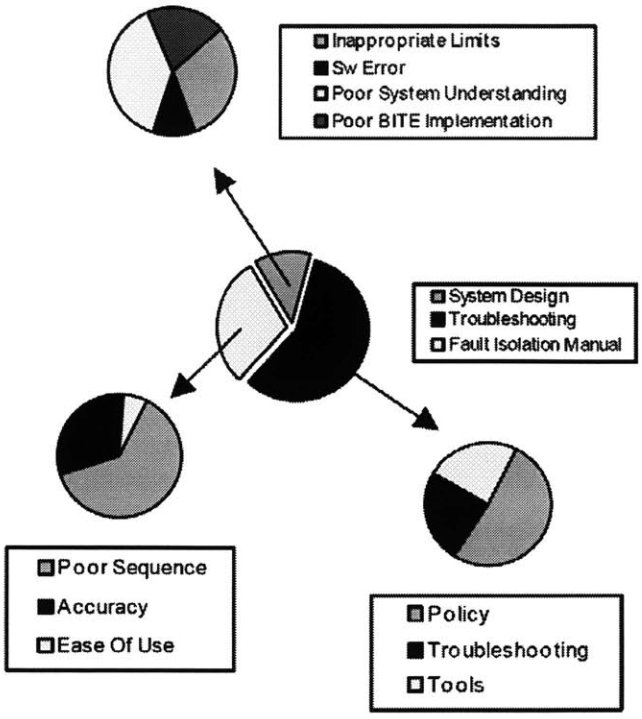


Figure 7 Fault Isolation Failure Root Cause [17]

The following quote from [17] suggests that actual part failures are a much smaller problem than the errors in the design of fault-detection systems, or errors in the associated fault isolation manuals or troubleshooting procedures.

> *"It can be seen that improving the system to reduce the number of product removals with no apparent failure will have far more effect on the overall reliability than a corresponding improvement in the product failure rate".*

The design of fault-detection and isolation systems is not trivial, and the robustness of such designs depends on how well the complete system is understood including off-nominal operating conditions, signal noise, timing issues in communication between devices etc. Quantitative system safety analyses cannot model these types of problems. In the author's experience, the setting of limits on fault-detection and isolation schemes can be extremely

challenging. One school of thought is to set limits as wide as possible while still ensuring that all real faults are detected. This strategy aims to minimize the risk of nuisance faults. This approach requires a very good understanding of the range of observed behavior of real faults. Another school suggests that limits should be set based on the largest size of signal error that can be tolerated in the system before operational performance is affected, which requires a good understanding of the effect of signal errors on performance, under various operating conditions.

Figure 8 below shows how a lack of understanding of the range of operation of a system under normal and faulty conditions can lead to nuisance faults or fault-detection "holes". Interactive complexity makes it very hard to accurately identify where the boundaries of observed normal and faulty behavior will lie. Assumptions are frequently made and these are often not well documented.

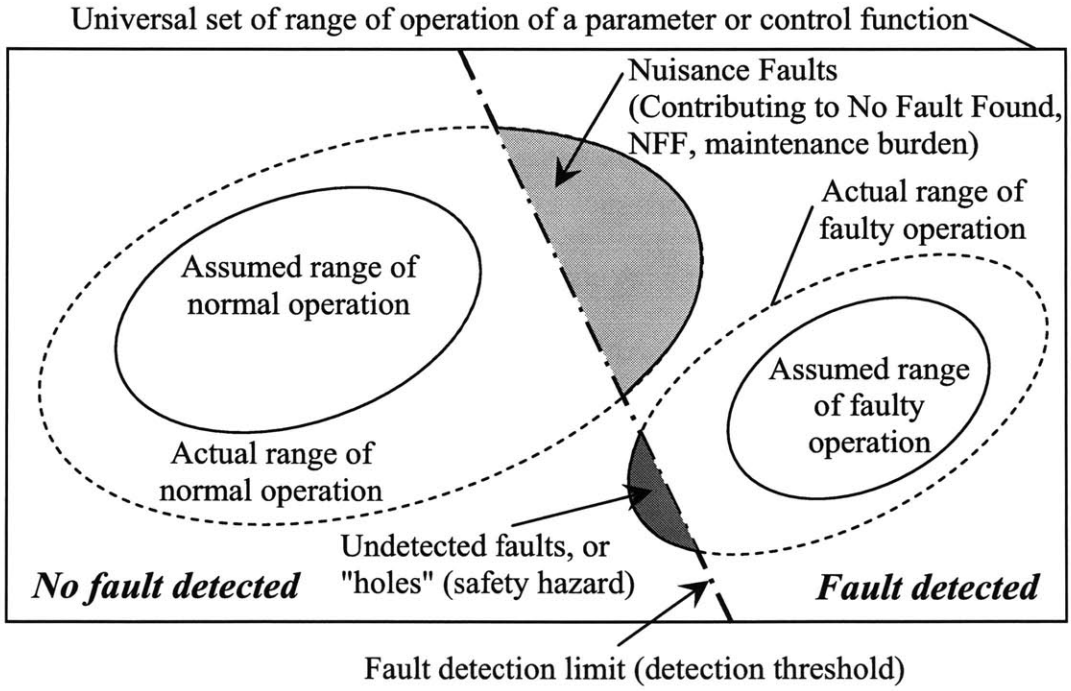


Figure 8 Potential problems with the design of fault-detection limits for BITE software

4 Overview of MIT System Safety Research

Given the great dependence of modern society on highly complex integrated systems, such as aircraft, electric power generating plants, medical technology, telecommunications etc, it is not surprising to find a significant amount of research attempting to improve the safety and dependability of these systems. Traditional system safety research has been built on the concept of reliability in which accidents are treated as a sequence of events (event chain) that lead from some causal event to the accident [18]. This model works well for accidents in which component failure plays a major role. Such models can make use of component reliability data, such as failure rates, and analysis tools such as FTA.

However, investigations into several aerospace accidents is revealing a much richer set of scenarios, in which a broader context of the system needs be studied to fully understand the accident, for example, operator and maintenance actions, and policies, design errors, and financial and schedule pressures as well. Systems that include control software also do not lend themselves to traditional analysis methods such as FTA (although attempts have been made in the past such as software fault trees). Accidents involving software often have requirements and design errors as contributing factors. Although this has been acknowledged for some time, effective tools and methods for analyzing and preventing such accidents have proven elusive.

This thesis will review the developments in the Aeronautics and Astronautics Department at MIT, under Professor Nancy Leveson. In particular, the development of software requirements completeness criteria, Intent specifications and the new STAMP model (System Theoretic Accident Modeling and Processes). Some other research will also be considered such as robust design approaches in systems engineering. However, the system safety research by Professor

Leveson comes closest to addressing the specific needs of aero engine control system safety and aircraft safety in general. Much of the system safety research at MIT has also been directed at problems at NASA, where the challenges are often greater than in the commercial aircraft industry [16, 19, 20].

4.1 System Theoretic Accident Modeling and Processes

Because traditional accident models treat accidents as a sequence of events that lead from a causal event through to the accident, engineers will then tend to design the system to eliminate, minimize or control these events. It is common to focus on the most obvious cause, or the component that can most readily be blamed and redesigned. This can lead to a narrow focus that overlooks the organizational structures responsible for building and deploying a safety-critical system. Many accidents are a result of a number of small and seemingly harmless decisions over a long period of time. It is not helpful then to single out any single component in such accidents, but better to understand the full context of the accident including the contributing factors in the technical system and the organizational context [18].

In the STAMP approach, the system and surrounding organizations are viewed as interacting control loops. Hazardous states are the result of inadequate or missing control actions by systems or organizations. The concept of a "failure" is replaced by the concept of a "constraint". The STAMP approach has its roots in systems theory, which is founded on two pairs of ideas: (1) *emergence and hierarchy*, and (2) *communication and control* [21].

4.1.1 Emergence and Hierarchy

A key feature that distinguishes a "system" from a "component" is the hierarchical structure of systems into layers, in which the higher layers have more complexity than the lower layers.

Properties can emerge in higher layers that have no meaning in the lower layers. Safety itself can be regarded as an emergent property of a system. For example, it is not possible to establish the safety of sensor. The context in which the sensor operates, such as the interfaces with the FADEC, control functions etc, are all needed to establish the safety of the system using this sensor. It is possible to describe the reliability of the sensor however, and this is identified in the FMEA. This is the key distinction between safety and reliability. They are not the same thing [21].

In the system-theoretic approach to safety, emergent safety properties are controlled by a set of safety constraints that specify those relationships among system variables or components that constitute the non-hazardous or safe system states. For example, the jet engine must not accelerate or decelerate inconsistently from pilot commands. These constraints will typically be related to the hazards identified in the hazard analysis. Accidents result from interactions between system components that violate the safety constraints.

4.1.2 Communication and Control

Systems incorporating feedback and control can be regulated to achieve desired functional behavior. The communication required, flows between levels in the hierarchy and the higher levels impose constraints upon the behavior of the lower levels. Figure 9 shows the basic control loop structure of a system.

To be effective, any controller must meet the following conditions [22]:

- Goal Condition: The controller must have a goal or goals (e.g., to maintain the set point).
- Action Condition: The controller must be able to affect the state of the system.

- Model Condition: The controller must be (or contain) a model of the system
- Observability Condition: The controller must be able to ascertain the state of the system.

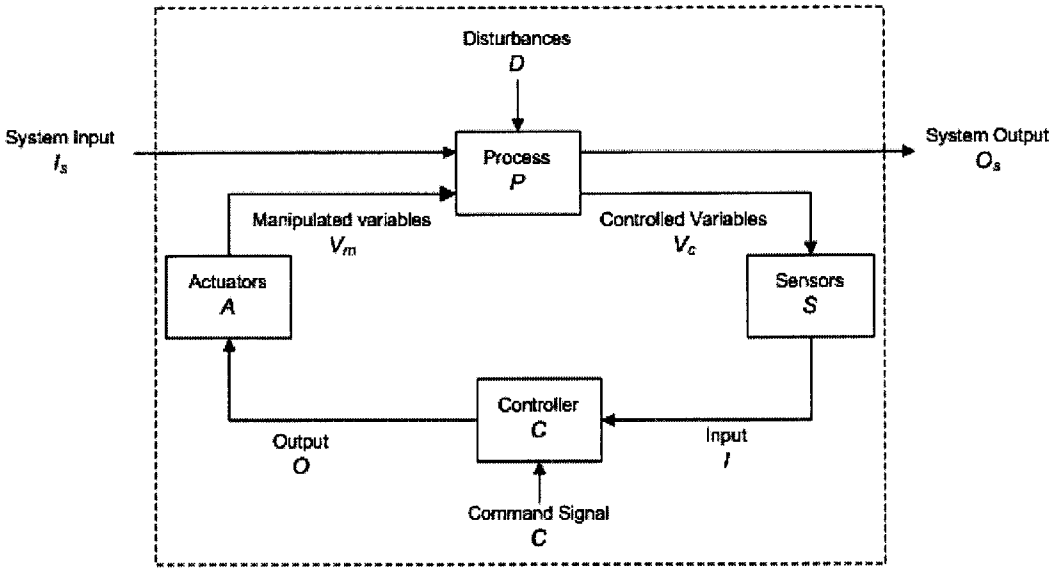


Figure 9 System Context modeled as a closed-loop system

The model condition is not always appreciated, but essentially amounts to saying the controller must have some understanding of the nature of the process and the current process state. This knowledge is often effectively embodied in the design of the control logic. The importance of the model condition is discussed further in 4.2.

4.1.3 Systems Theoretic Model of Accidents

In the STAMP approach, accidents or hazardous conditions are viewed as arising when component failures, external disturbances, and/or dysfunctional interactions among system components are not handled adequately by the control system. That is, when safety constraints that should be enforced by the system control structure are violated. A STAMP model will typically include a diagram of the safety control structure that is broad enough to include the full socio-technical context of the system. This can then be broken down and parts analyzed in more detail as required. The diagram illustrates the relevant communication and control actions.

The following example, Figure 10, shows the safety control structure for a generic socio-technical system. A key feature in Figure 10 is the representation of control and feedback paths, which should typically be in pairs. Lack of one of these paths can reveal flaws in the safety structure. The control structure should be expanded to encompass as much of the broader system as needed to cover all safety constraints.

For example, an understanding of the Space Shuttle Challenger and Columbia accidents cannot be developed from the component failures alone, but requires the organizational relationships, including the role of the safety organization and pressures to meet the launch window. These influences can be represented in the control structure diagram.

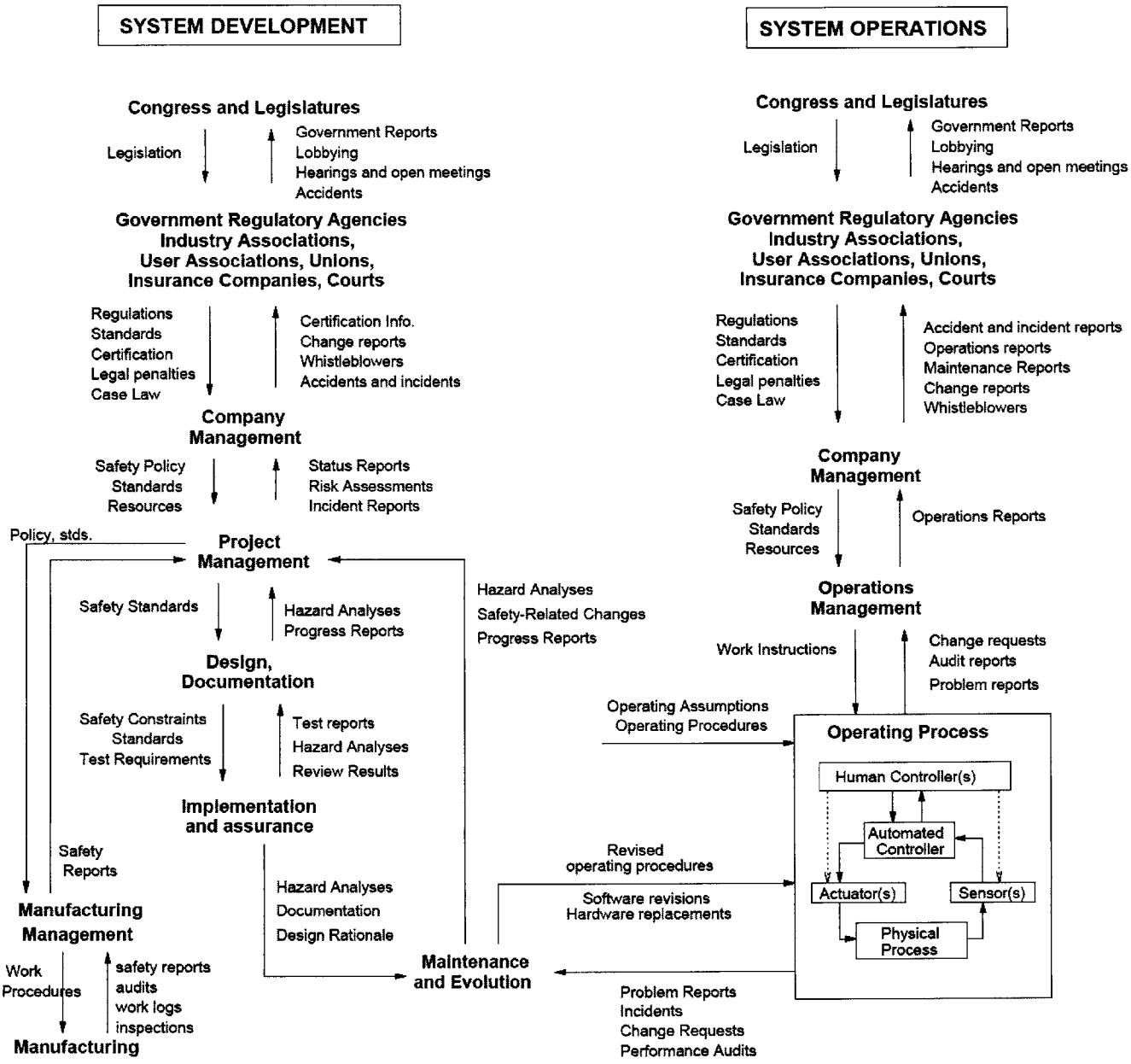


Figure 10 General Form of Control Structure for a Socio-Technical System [22]

4.1.4 STAMP Model Examples

STAMP models have been created at MIT for diverse systems related accidents including:

- Water contamination (e-coli) incident, Walkerton, Ontario, Canada, May 2000 [21]
- Friendly fire shooting of two US Army Black Hawk helicopters by the US Air Force in the Iraq No-Fly Zone in 1994 [22].
- Failure to achieve a geostationary orbit of a Milstar-3 satellite due to a trajectory problem of the Titan IV B-32, April 1996 [23]
- Loss of the Ariane-5 on its maiden flight, June 1996 [22].

In the Titan IV-B and Ariane 5 losses, the control software was a major contributor. The nature of the design errors in these losses has much in common with some of the design errors identified for aero engine control systems in chapter 3 above. In these cases, there was no component failure and the software performed in accordance with the software requirements. However, the system failed to impose constraints on behavior so as to prevent a hazardous event. In the case of the Ariane 5 loss, the reuse of inertial navigation software from the Ariane 4 was a major factor. This software was not designed for the larger loads experienced on the heavier Ariane 5 and led to a data overflow that led to two redundant systems both shutting down. Several assumptions in the earlier design had not been adequately documented. The fact that this software had performed well on the Ariane 4 had led to a perception that it was error free. This mindset arises from a view of component errors (reliability) rather than system errors (system safety).

The basic concepts in these models, particularly in the Titan-IVB and Ariane 5 losses are very applicable to aero engine control systems and aerospace systems in general. Figure 11 shows

the control structure for the development process of the Titan-IVB/ Milstar-3 accident. The controller boxes are expanded with information on the safety constraints that the particular level imposes on the level below, the mental model flaws that exist in the individuals within that organizational unit and the control flaws at that level. Other types of information that could be used is "inadequate control actions", "coordination problems" etc [21].

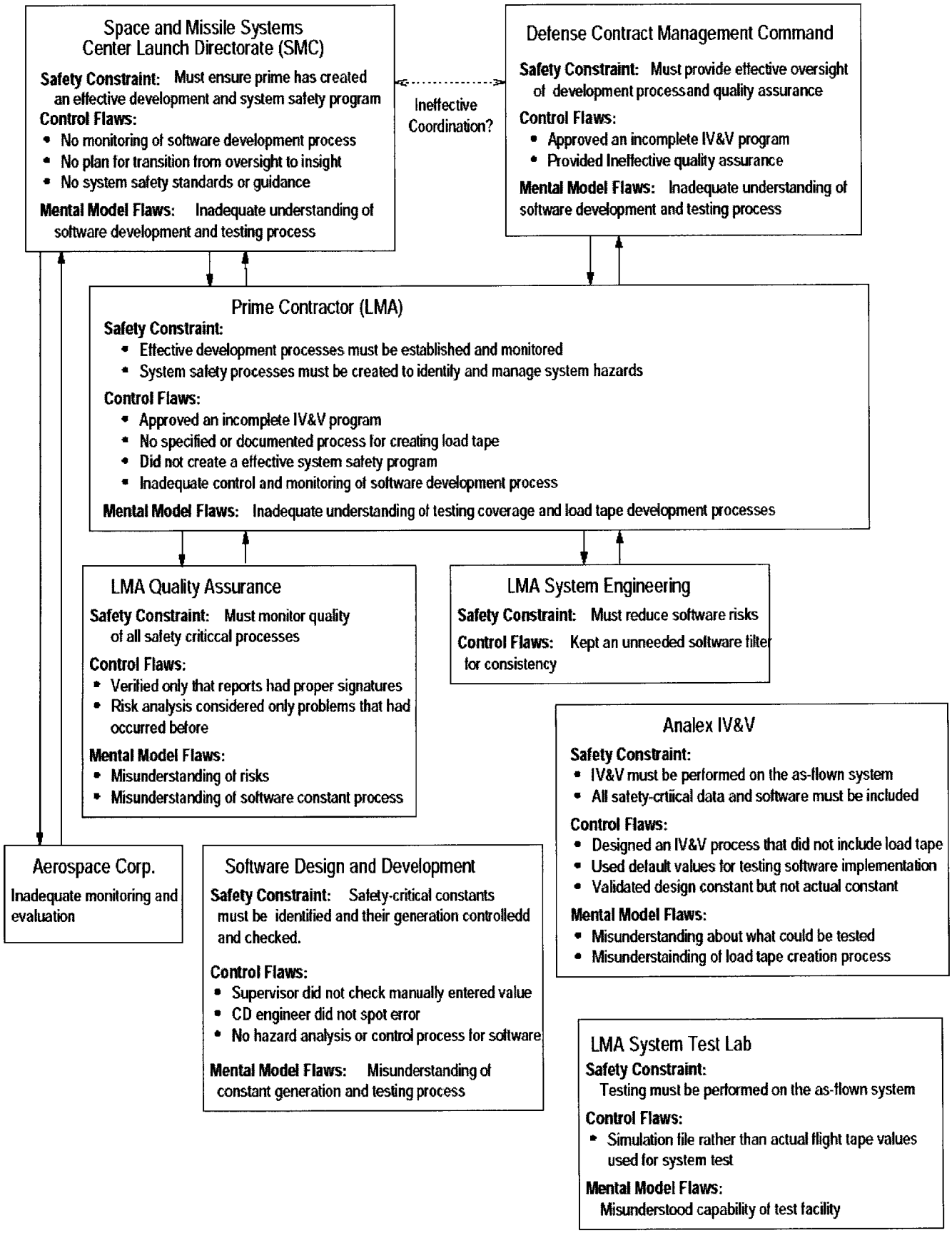


Figure 11 Control Structure for the Development Process of the Titan IV B/ Milstar-3 Satellite Accident [22]

4.1.5 STAMP-Based Hazard Analysis

A STAMP-based hazard analysis (STPA) involves the following steps [18]:

1. Identify the system hazards.
2. Identify system-level safety-related requirements and constraints.
3. Define the basic system control structure.
4. Identify inadequate control actions that could lead to a hazardous system state.
5. Determine the ways the system safety constraints could be violated and attempt to eliminate them, or if that is not possible, to prevent or control them in the system or component design.
    a. Create the process models for the system components.
    b. For each of the inadequate control actions identified, the parts of the control loop within which the controller is embedded are examined to determine if they could cause or contribute to the inadequate control action (or lack of a necessary control action).

The first two steps are standard in existing system safety processes; however, steps 3 to 5 are unique to the STPA approach [18].

4.1.6 System Dynamics

In STAMP, the broad socio-technical system is treated as a control/ communication problem. This is similar to the approach used in the field of system dynamics, and system dynamics models can be used as part of a STAMP analysis. System Dynamics was developed at the Sloan School of Management at MIT (formerly the School of Industrial Management) in the 1950s and 1960s by Jay Forrester. In system dynamics, the principles of dynamic systems and

control theory are applied to socio-technical systems. Some of the early application areas investigated included supply chain management, marketing, project management, quality and so on. It has been used to gain insights into the structure and dynamics of complex systems, which helps in developing policies (control actions) that are more effective than when relying on individual manager's mental model.

System dynamics models have value in helping to communicate each individual's mental model about how a socio-technical system behaves, and what effect interventions may have. The models can be executed to simulate behavior over time, and the effect of stimuli can be investigated, which can help design more effective intervention policies. Some of the first applications that Jay Forrester investigated concerned oscillatory behavior of production and employment levels in manufacturing firms [24] and [25]. System dynamics models were able to show how communication delays between the market forecasting process, production planning, supply chain, employee recruitment process and so on, created a naturally unstable system.

More recently, Repenning has studied the tendency of companies to enter firefighting modes of behavior in managing large engineering design projects [26], such as in the aerospace industry. The value of this research is in showing how an individual's mental model may not lead to the best behavior (management decision making). For example, in a project that is running behind schedule (creating schedule pressure), there is a natural tendency to increase overtime or hire more people. The expectation is that the overtime will allow more work to be accomplished per unit of elapsed time. Similarly the additional employees on the project will increase the work accomplishment rate. However, a good system dynamics model will include factors such as fatigue, which rises with increasing overtime, particularly if the overtime is allowed to persist

for more than a couple weeks (a factor that can also be modeled). New employees also require training and hence are ineffective initially, and may lead to a temporary reduction in productivity while they demand training and support from experienced employees. All these factors can be modeled, thereby providing greater insights into the real behavior to be expected from such policy decisions.

In the Walkerton STAMP analysis (listed in 4.1.4 above), a system dynamics model was developed to model the risk of e-coli contamination of a public water supply. Some of the variables in the model included "operator confidence", "municipality oversight", "quality of training" and "pressure to cut budget".

Currently research at MIT is being done to develop a system dynamics model of the safety of the space shuttle. This will include such variables as the launch rate, fraction of funds allocated to safety, quality system oversight capacity, incident learning, staffing levels etc.

A criticism of system dynamics models is that they can be constructed to support whatever explanation of the world the creator wants, particularly because the tuning of the model parameters is usually done by trial and error to match up with past data. However, good system dynamics models should be developed after extensive interviews with actors in the system, in which individual mental models are documented, and causal relationships identified. Each person in the system will see the relationships differently, but by looking at all the models and trying to break the system down to the fundamental causal relationships, useful models can be developed. The models are useful even if they are not tuned to real data and hence executed. The first stage in creating a model is to construct a "causal loop diagram" which shows the variables and states in the system. These diagrams can be useful in communicating concepts between managers.

4.1.6.1 Some System Dynamics Concepts

Figure 12 below shows a causal loop diagram for project schedule pressure. The '+' and '-' arrows indicate the sense of the causal relationship between variables. For example, when work remaining increases, then schedule pressure also increases, so a '+' sign is shown in the diagram. There are four loops in the diagram denoted as either balancing (B1 and B2) or reinforcing (R1 and R2). Balancing loops are analogous to negative feedback control loops in control theory, and reinforcing loops are analogous to positive feedback loops in control theory.

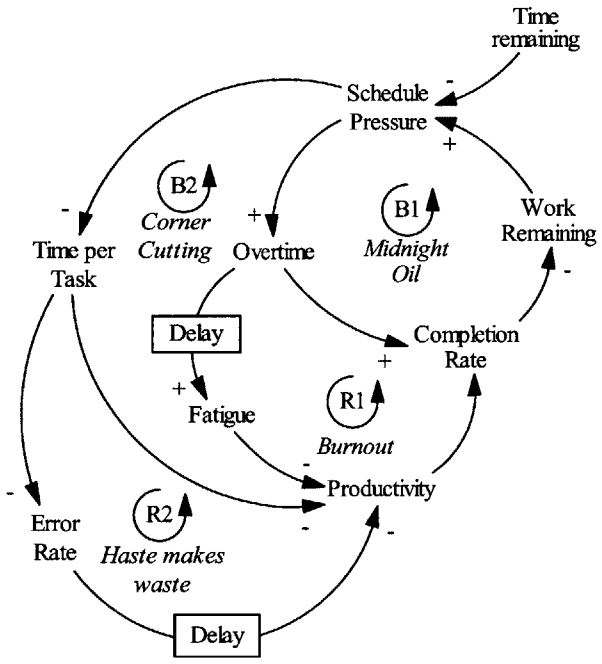


Figure 12 System Dynamics Causal Loop Diagram Example for Project Schedule Pressure

Often actions are taken based on an understanding of a constructive balancing loop. For example, if schedule pressure increases, overtime may be increased. This increases the completion rate and hence reduces the amount of work remaining, thus counteracting the schedule pressure increase. However, many systems also have delayed loops that may be reinforcing and harmful. If the action is used too strongly, or for too long, the other loops may

start to have a more powerful effect. For example, increased overtime may increase fatigue, which reduces productivity and hence reduces the completion rate.

Research in system dynamics has revealed the importance of an individual's mental model in understanding socio-technical systems. Individual's are unable to keep in their heads all the loops, and hence tend to choose actions based on what changes have the most immediately observable response, rather than what may ultimately lead to the greatest benefit. This behavior aims to maximize short-term gain. When stress levels increase, the tendency to focus on the short term will increase.

4.2 Software Requirements Completeness and Intent Specifications

Analysis of apparently dissimilar aerospace accidents involving software reveals some systemic patterns [14]. Traditional accident analysis has tended to look for a chain of events and a root cause. The analyses have also tended to look for mechanical failures, software coding errors, or operator errors. It is not common to look for requirements errors or at the software and system development processes or at the safety culture. Common problems include misunderstandings about the software requirements, lack of documentation of design rationale or design assumptions, and problems with reuse.

This section discusses some key aspects of software requirements that cause problems, and uses these as a basis for developing requirements completeness criteria. Finally, intent specifications put the elements together into a structure that glues the organizational, system, safety, and software design and requirements into a coordinated whole.

4.2.1 Semantic Distance

Requirements specifications exist in order to communicate the required behavior from a system component down to the next level. In going down through the system hierarchy, requirements and architecture/ design are developed together in increasing detail down the system. Requirements are developed based on the specifier's mental model of how the component will function in its environment. However, it is very hard to capture all of the thinking that takes place in the specifier, such as the specifier's mental model.

The semantic distance between the model in the specifier's mind and that implied in the requirements specification needs to be minimized to ensure communication is not lost between the specifier and the reader.

4.2.2 Black Box Requirements

It has been found that black box requirements *(i.e. what vs. how)* are preferred over white box requirements because they ensure that all specified functional behavior is observable at the system boundary. This approach also provides implementers as much freedom as possible to make implementation decisions [27]. It has been found that when designers think in black box terms, their requirements can be more easily comprehended because they avoid creating internal hierarchy and reduce the tendency to create hidden features and interactions .

4.2.3 Mental Models

Section 4.1.6.1 above shows how system dynamics can be used to help understand how one's mental model of a process may differ from the real process. The differences between the mental models of individuals involved in the design process (specifiers and specification users) and the real system are important factors in creating requirements errors (see Figure 13).

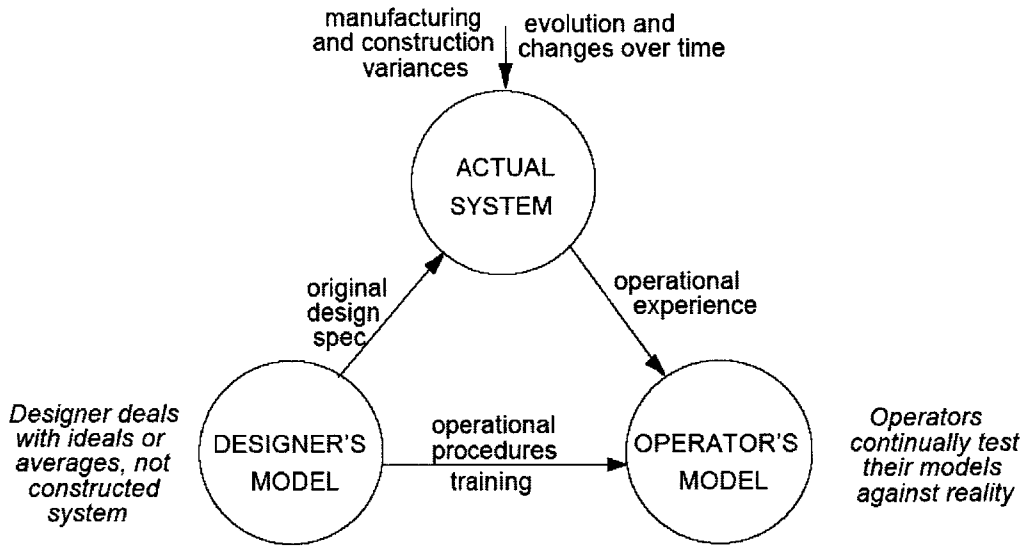


Figure 13 Mental Models of the System [28]

Figure 14 below illustrates how the different mental models create semantic distance between the original user's intentions and the final implementation.

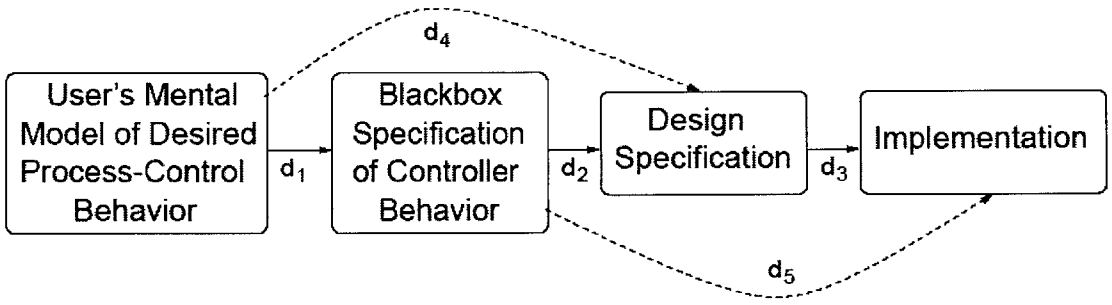


Figure 14 Semantic Distance between Models used in System Specification [29]

4.2.4 Feedback

The system control loop view in Figure 15 shows the models in the controller and the human operator and the feedbacks that allow these models to be updated to reflect the current state of the controlled process. Many accidents have been caused by designs that omit feedback. For example, in the Three Mile Island nuclear power station accident in 1979, an indicator light on the operator display panel was wired to indicate the presence of power to open a relief valve,

not its actual position [22]. A position sensor would have cost more and added complexity. It is not always possible or practical to obtain feedback to confirm the system response to every output, but the following requirements completeness criteria insist that this be checked, and the rationale for any output without feedback must be documented and the safety implications understood.

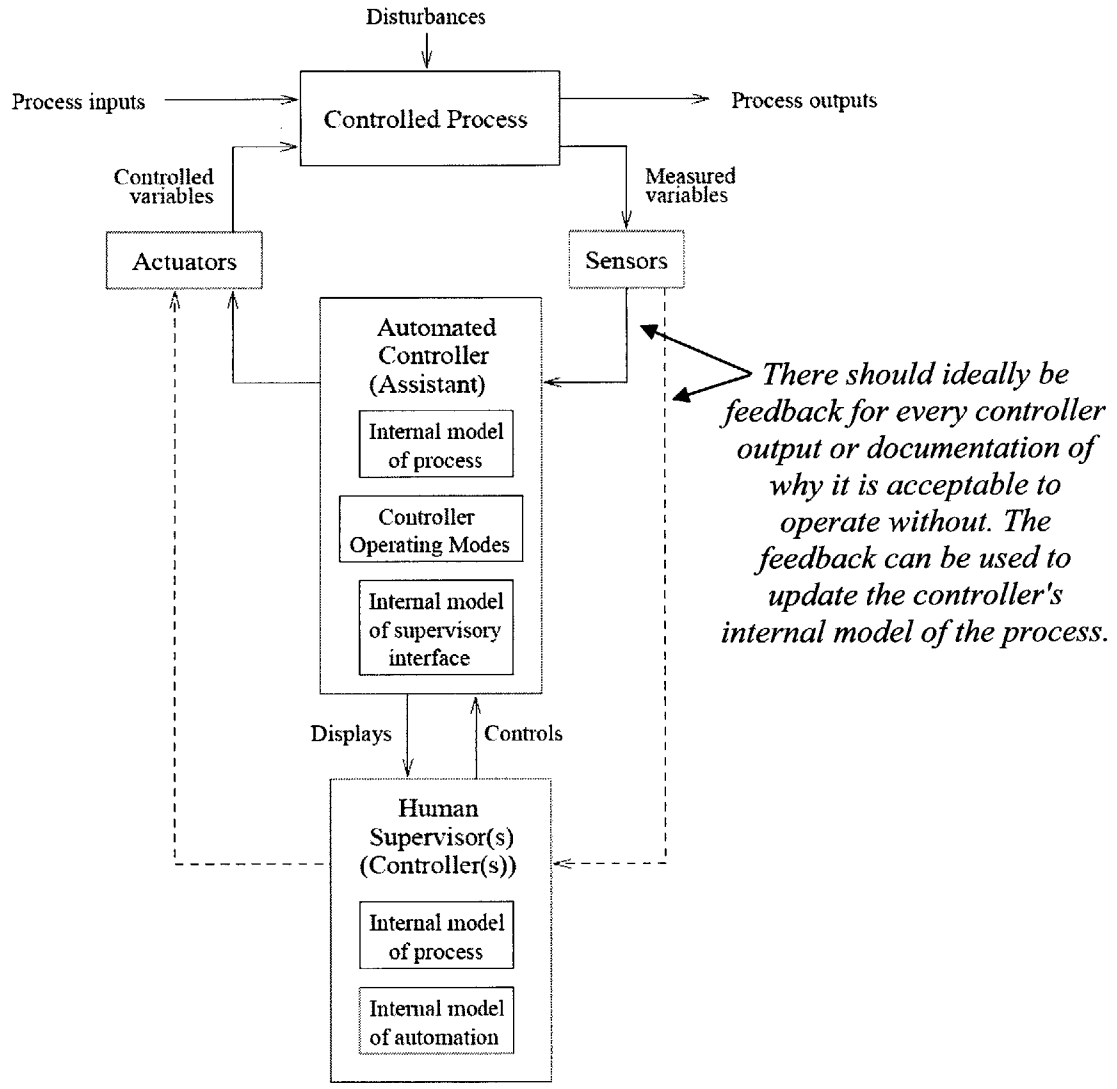


Figure 15 System Control Loop separating Automated and Human Controllers [29]

4.2.5 Rationale and Assumptions

The following quote from [16], a paper on the analysis of causation of aerospace accidents, describes the observations from several accident investigations:

> *Software-related accidents almost always are due to misunderstandings about what the software should do. Almost all the accident reports studied refer to poor specification practices. The Ariane accident, for example, notes that inadequate specification practices and the structure of the documentation obscured the ability to review the critical design decisions and their underlying rationale. Complete and understandable specifications are not only necessary for development, but they are critical for operations and the handoff between developers, maintainers, and operators. Good specifications that include requirements tracing and design rationale are critical for long-lived systems.*

Intent specifications aim to increase the profile of assumptions and rationale and the SpecTRM-RL [30] toolset has been designed to promote the documentation of assumptions and rationale. Within the industry, there has been a tendency to focus mainly on the requirements themselves. Rationale and assumptions are to be documented as part of the design process, but not appear in the final specification. In Intent specifications, and in SpecTRM-RL, rationale and assumptions are viewed as integral to the requirements themselves. By keeping the extra information with the requirements, then the possibility that an assumption may be violated can be seen by a specification user, thus alerting the specifier to a potential problem. Rationale that is documented separately from the specification is much less likely to be seen.

Assumptions and rationale are particularly important when the software, or software requirements, are to be reused. The assumptions may not hold in the new environment. For example, when an air traffic control system that had worked well in the United States for many

years was reused in the UK, it was discovered that the zero longitude condition was not handled in the software. The system effectively folded the UK air space in half, around the Greenwich Meridian [5]. Violation of assumptions in reused software was also a major factor in the Ariane-5 loss [22].

4.2.6 Identification of Hazardous States in Software

The Functional Hazard Assessment (FHA, see 2.2.2), identifies system hazards. The system hazards should be traced to the system/software interface, so that the software states that could contribute to these hazards are known and can be considered in the software requirements development process. For example, the failure to arm the ATTCS system in all FADECs during a takeoff (see section 3.5) is a hazard that can be traced into the software. In the reviewed system, the FADECs were able to recognize this hazardous state and it was handled by triggering the ATTCS, thus providing the extra thrust whether needed or not. This is a fail-safe design, but led to nuisance cockpit warnings and aborted takeoffs. However, this does meet one requirement completeness criterion which is that "every hazardous state must have a path to a safe state" (see completeness criterion 54 in Table 1, Appendix B).

It is worth noting that DO-178B, being process-based, does not prescribe any specific methods for software hazard analysis, in contrast with MIL-STD-882B (task section 300). DO-178B does make some recommendations about good practices for verification, which includes some checks similar to the requirements completeness criteria. For example, the following extract from DO-178B section 6.4.3 part (a):

Typical errors revealed by this testing method include:

- *Incorrect interrupt handling.*
- *Failure to satisfy execution time requirements.*
- *Incorrect software response to hardware transients or hardware failures, for example, start-up sequencing, transient input loads and input power transients.*

4.2.7 Requirements Completeness Checks

Leveson and Jaffe worked on software requirements completeness criteria for safety-critical systems in the late 1980s [27, 29]. This research was built partly on formal methods, and made use of the Requirements State Machine Language (RSML).

The completeness criteria are listed in Table 1 in APPENDIX B Requirements Completeness Criteria With Example Systems. They are based on analyses of typical accidents and design errors. The criteria work for black box specifications of state-based systems. The criteria are based on a control model of the system, as shown in Figure 9 above. Of particular interest is the attention paid to the following areas:

- System startup
- Timing behavior
- Determinism of state transitions
- Environmental assumptions

The example problems discussed in chapter 3 fall into several of the above categories.

DO-178B [31] states the following (in its Appendix D):

> *Research into software development programs suggests that as many as half of the software errors actually result from missed or incorrect requirements or the incorrect interpretation of requirements.*

McDermid summarizes the ways a system can fail as follows [32]:

> *A system may generally fail in one of two ways; (1) as a result of a component failure or (2) as a result of unintended functioning when all components are behaving to specification. Each of these may be caused by either (1) a fault intrinsic to the component or system or (2) by an external disturbance.*

These support the arguments in Leveson's research on the importance of requirements validation, and the documentation of assumptions and rationale.

4.2.8 Intent Specifications

Intent specifications aim to capture the full context of information in the system and software design process into a hyper-linked environment. Figure 16 shows the structure of an intent specification. This approach has been effectively used for the TCAS-II specification, developed for the FAA [33], as well as several other examples [20, 34]. The objective is to integrate the formal and the informal parts of the specification, and to support varying levels in the system hierarchy.

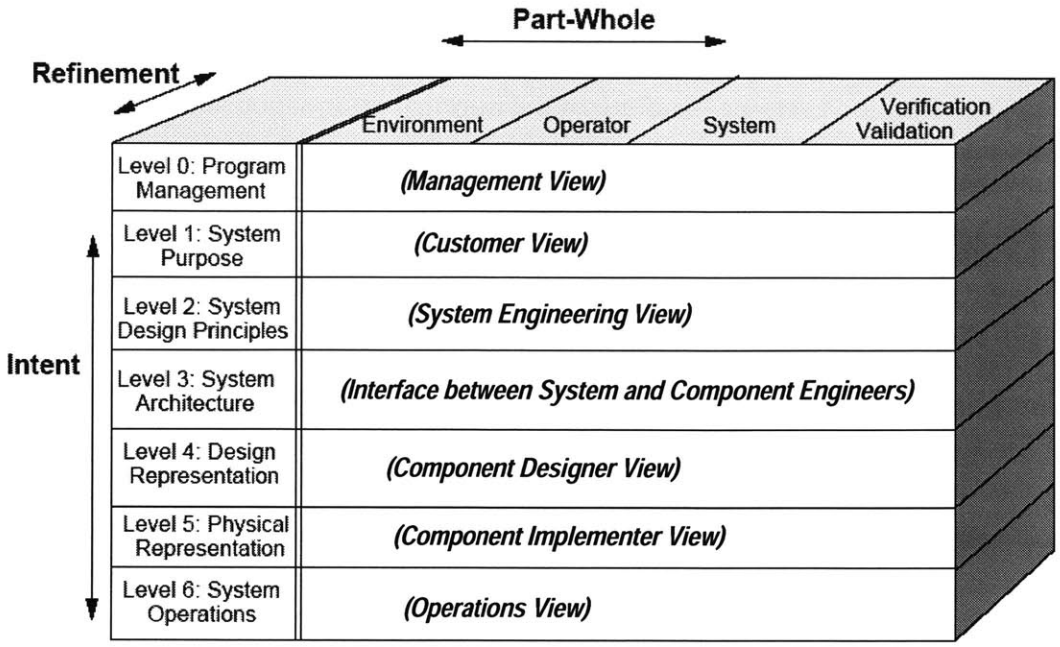


Figure 16 The structure of an Intent Specification [22]

	Environment	Operator	System and components	V&V
Level 0 Prog. Mgmt.	Project management plans, status information, safety plan, etc.			
Level 1 System Purpose	Assumptions Constraints	Responsibilities Requirements I/F requirements	System goals, high-level requirements, design constraints, limitations	Preliminary Hazard Analysis, Reviews
Level 2 System Principles	External interfaces	Task analyses Task allocation Controls, displays	Logic principles, control laws, functional decomposition and allocation	Validation plan and results, System Hazard Analysis
Level 3 Blackbox Models	Environment models	Operator Task models HCI models	Blackbox functional models Interface specifications	Analysis plans and results, Subsystem Hazard Analysis
Level 4 Design Rep.		HCI design	Software and hardware design specs	Test plans and results
Level 5 Physical Rep.		GUI design, physical controls design	Software code, hardware assembly instructions	Test plans and results
Level 6 Operations	Audit procedures	Operator manuals Maintenance Training materials	Error reports, change requests, etc.	Performance monitoring and audits

Figure 17 The structure of an Intent Specification showing typical content [22]

Traditional specifications have placed the informal information such as design rationale and assumptions in design records, but not part of the published and configuration controlled artifact. In intent specifications, the information is kept together. Conceptually, much of the

information in Intent Specifications could be put into a commercial requirements management tool such as DOORS, however, the handling of the informal information such as assumptions and rationale may not be well handled in environments such as DOORS. In the case of DOORS, the fundamental unit of information is a requirement, and each requirement has a unique identifier to which can be added a prefix, such as "SRS" (software requirements specification). However, no other prefixes are allowed in a given specification.

In the commercial tool SpecTRM [30], there is greater flexibility in prefixing information fields, as shown in Figure 18 below. Note the mixing of different types of information in one specification, including requirements (R), safety constraints (SC) and limitations (L), each of which has associated assumptions. In a traditional specification, the safety constraint shown in Figure 18 below would probably be labeled as a requirement, and the limitation would be documented separately as part of the design rationale, perhaps in the change paperwork, or in a design report. However, the explicit labeling of safety constraints helps encourage system safety thinking and traceability to the functional hazard analysis (FHA).

R1. *Provide collision avoidance protection for any two aircraft closing horizontally at any rate up to 1,200 knots and vertically up to 10,000 feet per minute.*

Assumption:

This requirement is derived from the assumption that commercial aircraft can operate up to 600 knots and 5,000 fpm during vertical climb or controlled descent (and, therefore, two planes can close horizontally up to 1,200 knots and vertically up to 10,000 fpm).

SC5.1. *The pilot of a TCAS-equipped aircraft must have the option to switch to the Traffic-Advisory-Only mode, where TAs are displayed but display of resolution advisories is inhibited.*

Assumption.

This feature will be used during final approach to parallel runways, when two aircraft are projected to come close to each other and TCAS would call for an evasive maneuver.

L4. *TCAS is dependent on the accuracy of the threat aircraft's reported altitude. Separation assurance may be degraded by errors in intruder pressure altitude as reported by the transponder of the intruder aircraft.*

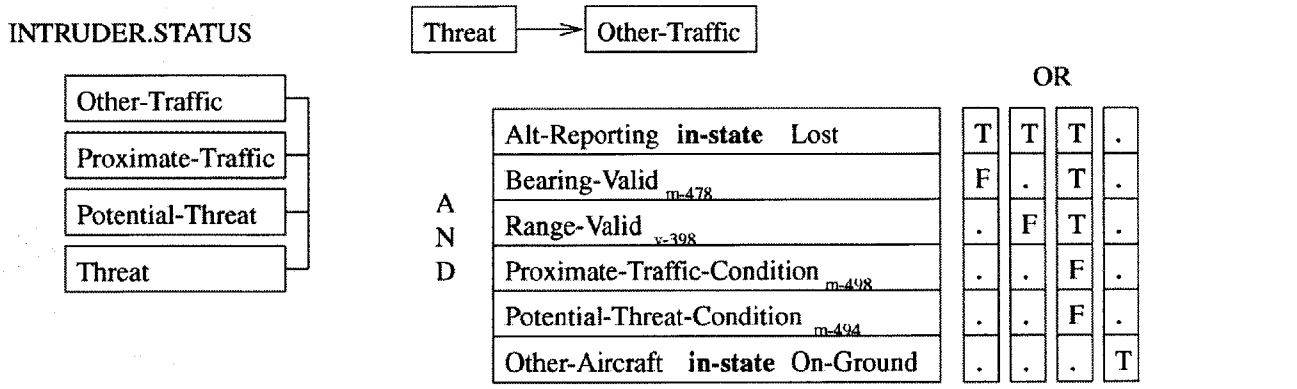
Assumption.

This limitation holds for existing airspace, where many aircraft use pressure altimeters rather than GPS. As more aircraft install GPS systems with greater accuracy than current pressure altimeters, this limitation will be reduced or eliminated.

Figure 18 Extracts from TCAS II Intent Specifications [33]

4.2.9 SpecTRM

The research into requirements completeness criteria led to the creation of SpecTRM-RL, a requirements language available as part of the commercially available SpecTRM package [30]. SpecTRM-RL provides an intuitive (human-centric) language for specifying state transition logic using "AND/OR" tables and does not require a graphical state transition diagram. Automated completeness checks are provided and a requirements animation facility. SpecTRM supports intent specifications and allows the black box specification to be executed. The specification of inputs and outputs also provides fields for encouraging the documentation of requirements and assumptions associated with timing behavior, fault-detection and accommodation etc.



Description: A threat is reclassified as other traffic if its altitude reporting has been lost (⩓PR13) and either the bearing or range inputs are invalid; if its altitude reporting has been lost and both the range and bearing are valid but neither the proximate nor potential threat classification criteria are satisfied; or the aircraft is on the ground (⩓PR12).

Mapping to Level 2: ⩓PR23, ⩓PR29

Mapping to Level 4: ⩔Section 7.1, Traffic-Advisory

Figure 19 Part of a SpecTRM-RL blackbox level specification, for TCAS [33]

Part of a SpecTRM-RL blackbox behavior level description of the criteria for downgrading the status of an intruder (into our protected volume) from being labeled a threat to being considered simply as other traffic. Intruders can be classified in decreasing order of importance as a threat, a potential threat, proximate traffic, and other traffic. In the example, the criterion for taking the transition from state *Threat* to state *Other Traffic* is represented by an AND/OR table, which evaluates to TRUE if any of its columns evaluates to TRUE. A column is TRUE if all of its rows that have a "T" are TRUE and all of its rows with an "F" are FALSE. Rows containing a dot represent "don't care" conditions. The subscripts denote the type of expression (e.g., *v* for input variable, *m* for macro, *t* for table, and *f* for function) as well as the page in the document on which the expression is defined. A macro is simply an AND/OR table used to implement an abstraction that simplifies another table.

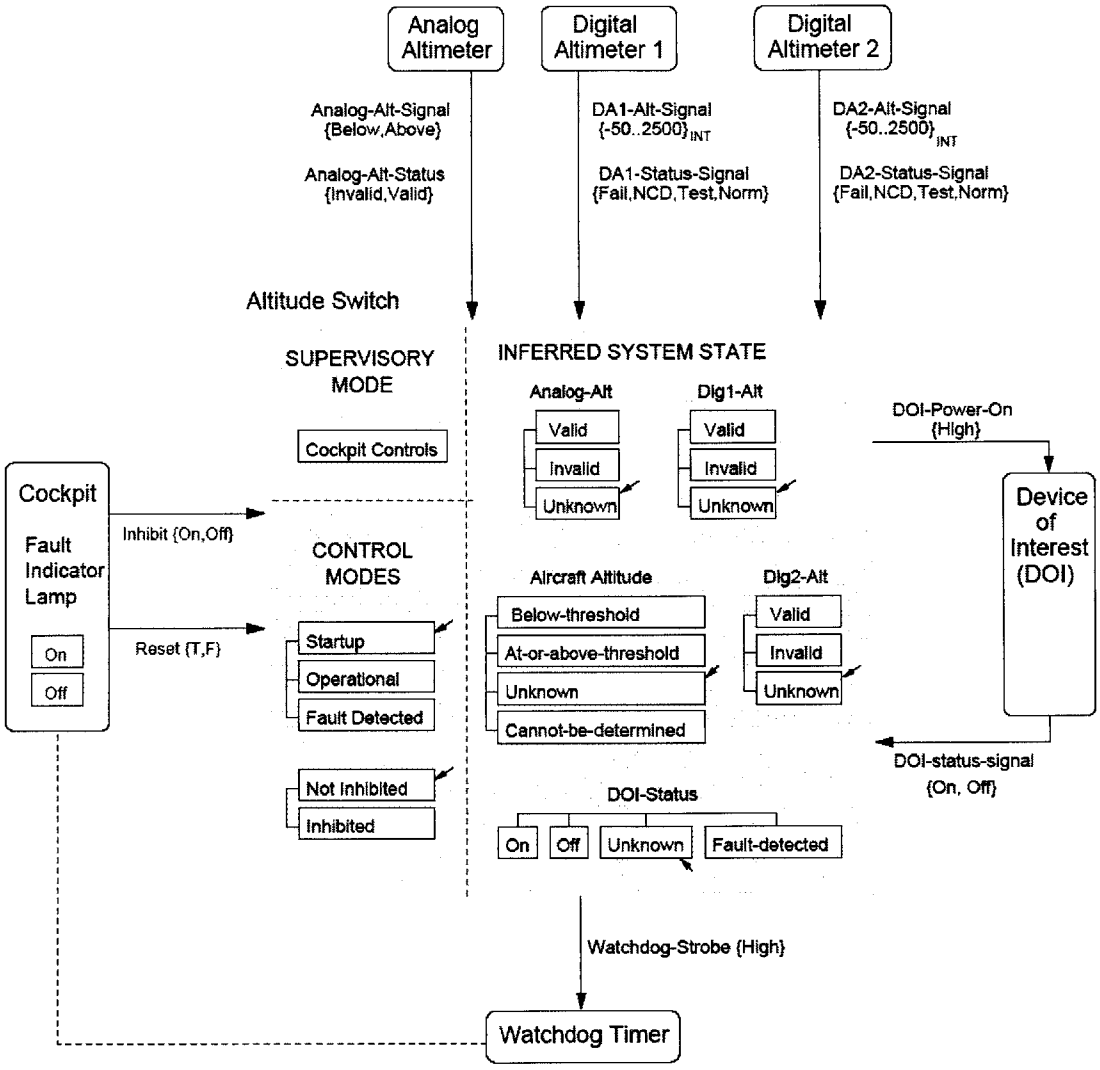


Figure 20 SpecTRM Visualization Display for an Altitude Switch [34]

The visualization display shown in Figure 20 above illustrates some concepts discussed in 4.2.2, 4.2.3, 4.2.4 and 4.2.7. Firstly, this level of specification is blackbox, so the information can be shown at a high enough level to illustrate all the behavior of interest in one diagram. Secondly the term "inferred system state" indicates the internal model of the process (the controller's "mental" model of the system). Thirdly, the illustration of the external devices helps clarify the output and feedback relationships in the system. Lastly, the inputs are shown with their valid ranges or as enumerated types as appropriate.

Output Command

DOI-Power-On

Destination: DOI

Acceptable Values: {high}

Initiation Delay: 0 milliseconds

Completion Deadline: 50 milliseconds

Exception-Handling: (What to do if cannot issue command within deadline time)

Feedback Information:

Variables: DOI-status-signal

Values: high (on)

Relationship: Should be on if ASW sent signal to turn on

Min. time (latency): 2 seconds

Max. time: 4 seconds

Exception Handling: DOI-Status changed to Fault-Detected

Reversed By: Turned off by some other component or components. Do not know which ones.

Comments: I am assuming that if we do not know if the DOI is on, it is better to turn it on again, i.e., that the reason for the restriction is simply hysteresis and not possible damage to the device.

This product in the family will turn on the DOE only when the aircraft descends below the threshold altitude. Only this page needs to change for a product in the family that is triggered by rising above the threshold.

References:

Figure 21 SpecTRM-RL output variable specification example for an altitude switch [34]

The specification of the output command in Figure 21 shows the additional data recommended by the requirements completeness criteria (see 4.2.7 and Table 1). In traditional specifications, much of this data would be defined using separate requirements, such as the requirements for reading feedback and performing fault-detection (referred to above as exception handling). The extra fields provided for input and output variables could probably be prompted using well designed checklists, which is standard practice in the industry.

5 Applicability of MIT System Safety Research to Aero Engine Control System Development

In chapter 4, system safety research at MIT was reviewed. Chapter 5 looks at which of these techniques may be applicable to the design, development and certification of aero engine control systems and to commercial aircraft systems in general.

5.1 System Theoretic Accident Modeling and Processes

The STAMP approach currently lends itself best to problems at the organizational, socio-technical level, although work is underway to try to extend the STAMP framework to help in designing for safety. Figure 22 below shows a high-level STAMP control structure diagram for the aero engine development and certification process. This does not attempt to cover every controller and every communication path, but, the principle control and feedback paths are shown.

Figure 23 expands on the safety assessment and system development processes. This diagram is based on the process model for the interaction between the safety assessment and system development processes, as defined in ARP 4754 [31]. These two views may help in identifying control and feedback failures, particularly in the lower level systems. The control structure diagram can also be expanded down to the physical system level to show the control and feedback signals in the actual system architecture.

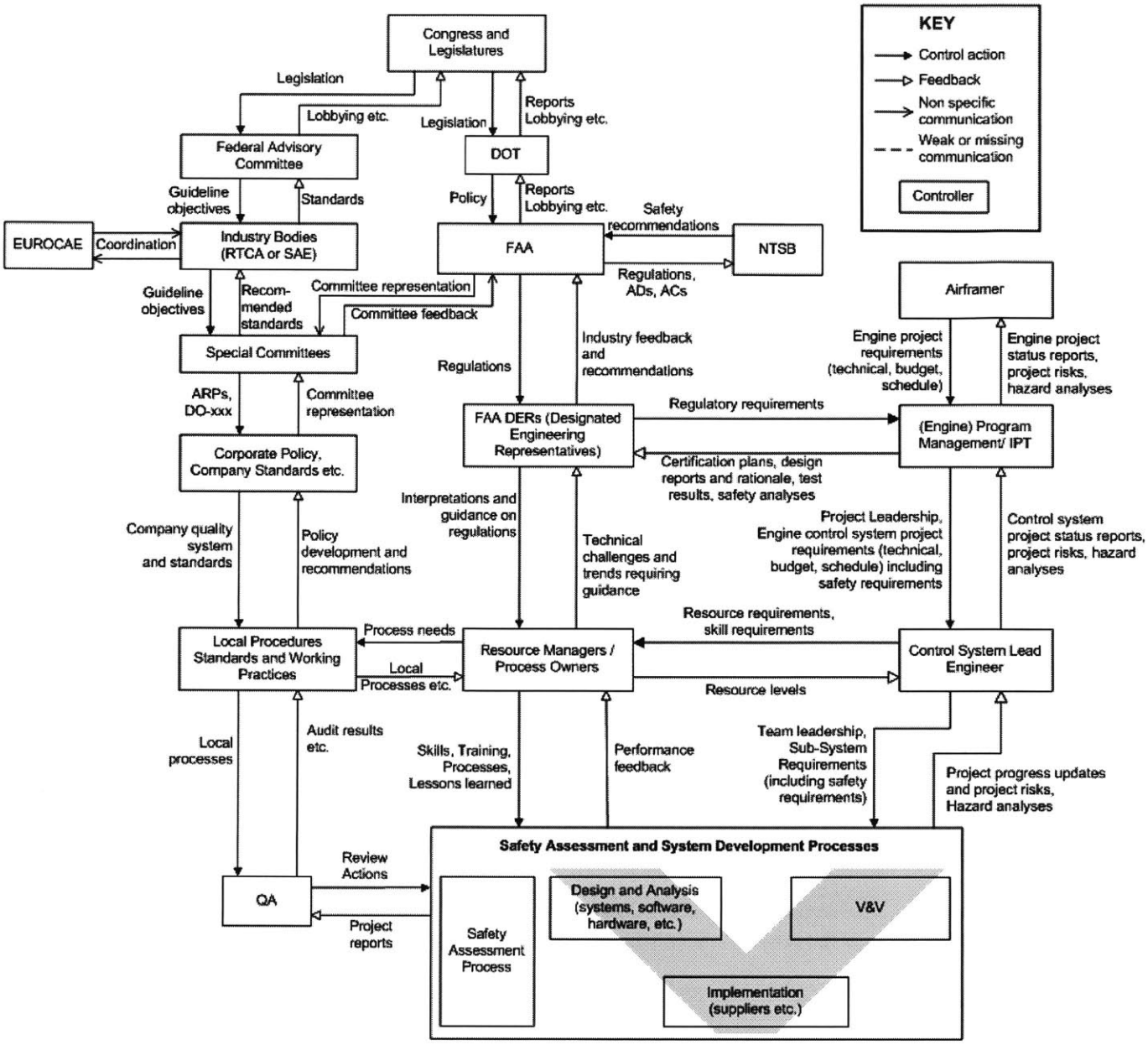


Figure 22 High-level Control Structure for Aero Engine Control System Development Process

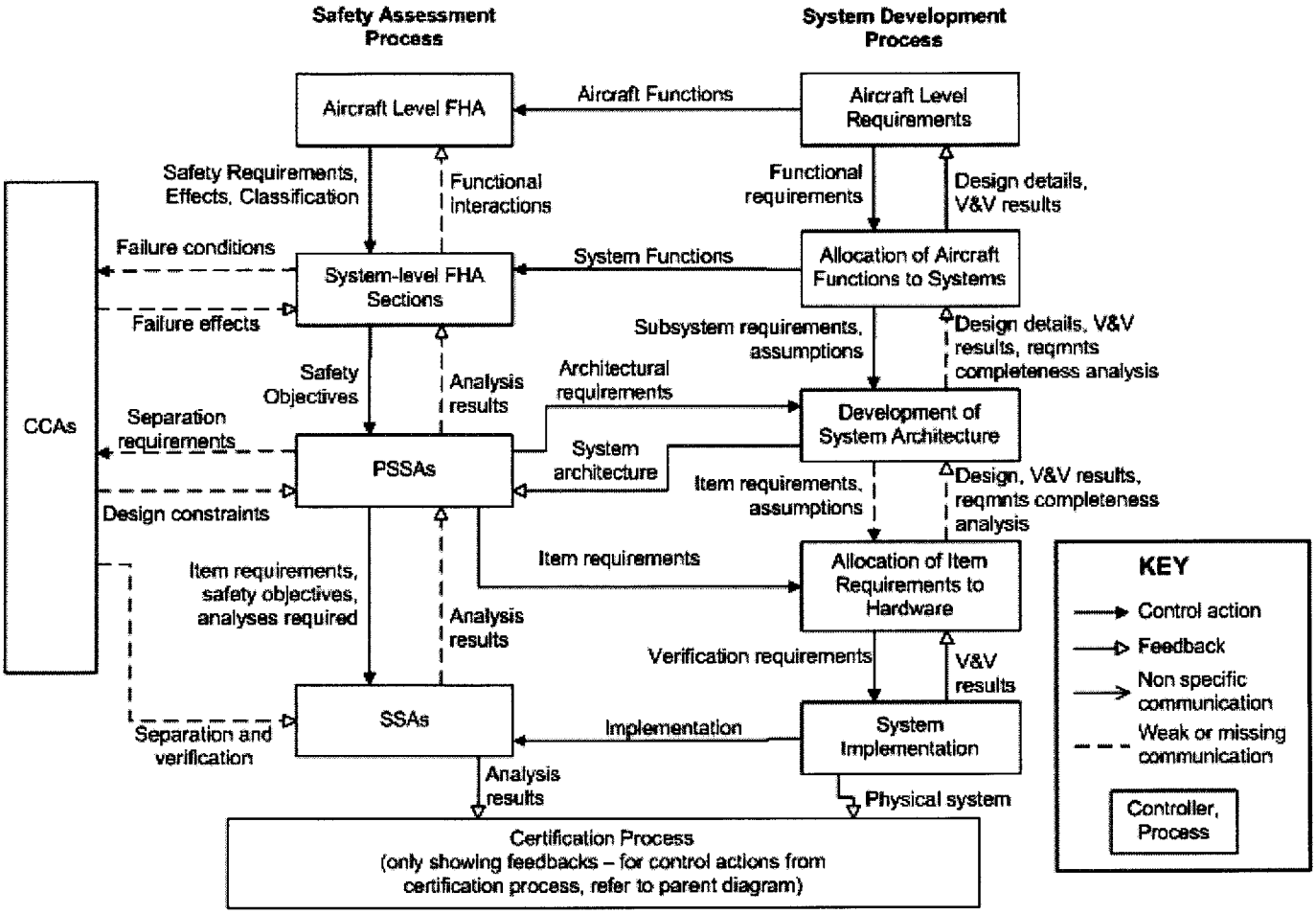


Figure 23 Control Structure for Safety Assessment and System Development Processes (based on ARP 4754)

At each level, the key safety requirements and constraints can be identified along with a description of mental model flaws, flawed assumptions and coordination issues. Refer to Figure 11 in 4.1.4 for an example that was created for the Titan-IVB/ Milstar-3 satellite accident. No attempt has been made to create such a diagram for aero engine control system development, however, some weak communication lines are shown in Figure 23.

In the case of the engine thrust shortfall anomaly (section 3.5), poor flow down of requirements from the aircraft manufacturer was a factor, caused partly by flawed assumptions about the timing of the engine control system response to an engine failure.

In the case of the safety assessment process, weak communications are shown in the common cause analysis and in several of the feedback paths up the system hierarchy. This is based on the system discussed in chapter 3. The feedback paths on the design side are indicated as weak because of a lack of adequate documentation of assumptions, rationale and inadequate requirements completeness analysis in the examples studied. Wilkinson also discusses difficulties in the System Level FHA process as well [35].

Much of the information shown in the above diagrams is available in several sources within the company site studied, and in industry guidelines. It appears that the value of the STAMP framework is in the concepts behind the analysis. In STAMP, the emphasis is on viewing the levels in the socio-technical system as system levels which impose constraints on the behavior of the level below, and require feedback to monitor status. Each level has a mental model of the process they are trying to control underneath them, and their model can only be updated based on feedback received. Delayed or missing feedback will therefore lead to disconnects in the mental models and consequently leads to inappropriate control actions.

This thinking process is very similar to that needed to construct system dynamics models of socio-technical systems. It would seem useful therefore to try to develop a system dynamics model alongside the STAMP control structure, and the two views could support each other. This approach is similar to work being done in looking at the NASA safety culture.

In the specific case of aero engine control systems however, it could be argued that the high-level control structure depicted in Figure 22 is working quite well. The weakness noted in the thrust shortfall example are not typical of other problems reviewed. However, it does appear that the weaknesses in the coordination of the safety assessment process are more fundamental, though great progress has been made in recent years in the company studied.

Research by Hawkins and McDermid discusses the concept of safety contracts in object oriented safety-critical systems [36]. These safety contracts appear to have some similarities to the safety constraints in STAMP models.

5.1.1 Relationship between Safety Assessment Process and System Development Process

A basic problem with the safety analysis, which was discussed in the introduction, is that the FTA approach only models component failures and does not handle requirements errors such as those discussed in chapter 3. The failure rate figures for top-level events such as IFSD and LOTC based on the FTA do not account for the majority of actual event scenarios, in which requirements errors are a major factor. Lindsay and McDermid state that *"design error dominates over physical failure for software and other logically complex elements"* [37].

It could be argued that the types of events most likely to lead to real safety problems are very unlikely to be in the FTA. The real problem scenarios arise from misunderstandings and violated assumptions, including redundancy assumptions. Because these are unknown, clearly they cannot be in the FTA. Indeed, if they were known, they would have been addressed in the design already, and hence would not need to be in the FTA. It is not normally acceptable to discover a requirements problem with safety impact and leave it unresolved for any length of time, particularly when it involves software, which tends to go through post-EIS updates about once every year or two.

This leads to a paradox which can be observed between the system/software development process and safety assessment process. The safety analysis is often chasing the tail of the software development process and never catches it. Consider the example of the P25 sensor fault-detection errors in section 3.4. When the problem was understood to the extent that the

issue could be discussed with the safety assessment team, it was already planned to correct the design. The FTA is built assuming the redundancy works and hence was an inaccurate model of the old software design. There would be no point updating the FTA, however, because the new software would then be under development, which would bring the new design back into line with the assumptions in the FTA. The FTA is still very important in modeling the response of the system to mechanical failures, assuming the system behaves as intended (according to the system requirements). Clearly other processes are needed to ensure dependability of a system liable to have requirements errors.

There have been attempts to build software fault trees [38-40], however this technique can only be as good as the software hazard identification (the top-level events for an SFTA), and may then only provide the same results as a static analysis. Any hazardous paths found should simply be eliminated, so there would be no need to quantify any problems found [5].

The lack of a documented common-cause analysis on the system investigated could be viewed as a weakness, but a more fundamental problem appears to be the tracing of system hazards down to the software interface, and then designing the software in such a way as to maximize the robustness to uncertainty. This is discussed more in the next section. When hazardous software states are identified and handled, it may then be possible to produce a common-cause analysis using this information.

The following quote from the US DOD Software System Safety Handbook [13] makes this point quite well:

> *Although the software engineer may implement a combination of fault avoidance, fault removal, and/or fault tolerance techniques in the design, code, or test of software, they usually fail to tie the fault or error potential to a specific system hazard or failure mode.*

While these efforts most likely increase the overall reliability of the software, many fail to verify that the safety requirements of the system have been implemented to an acceptable level.

Clearly, system developers need to be more aware of the hazards that are relevant to the functions they are working on, and be able to trace these down into the software, with documented assumptions and run-time tests to check these assumptions as appropriate.

5.2 Intent Specifications

The conclusions in 5.1.1 above on system safety suggest that more emphasis should be placed on assuring design robustness and less on quantitative analyses. This is not to suggest that FTAs and Markov analyses are not valuable, but their applicability is restricted to understanding the system response to random (mechanical) failure mechanisms, assuming the system functions as specified.

The broad class of safety problems arising from requirements errors requires a different approach. Furthermore, this class of problems appears to account for the majority of scenarios that could cause accidents. It appears that the concepts behind Intent Specifications and the software requirements completeness criteria demands a paradigm shift from current practice.

In current system development processes, "requirements" are the central elements in the process. Traceability is used to ensure coverage of requirements at the lower levels in the system hierarchy. Validation and verification are used to ensure that the requirements are correct and have been implemented correctly. Other processes, such as configuration management, are used to ensure that consistency is achieved between requirements, tests and safety analyses in a typical environment of evolutionary or iterative development. The existing

process step that is designed to capture assumptions and rationale and ensure requirements are complete is the requirements validation process. In current practice, the requirements validation process is making increasing use of requirements animation and simulation, and this helps to identify some of the unanticipated behavior that requirements completeness checks aim to avoid. Furthermore, stage gate reviews, quality audits and certification audits and reviews also provide opportunities to iron out misunderstandings, validate assumptions, capture rationale, and review how well the system will meet the safety requirements.

The objective of the above (current) activities is to ensure that the requirements are correct at the specification stage and that additional information such as assumptions and rationale are documented somewhere (not usually in the published specification). Once the requirements are deemed to be correct, then they can be implemented and verified. If there is found to be a need to update the requirements at some later time, then the previously documented assumptions and rationale can be consulted by the designers/ specifiers to help in the redesign effort.

However, there is a feedback missing in this process, i.e. the ongoing validation of assumptions and continuing validity of the rationale, as seen by all specification readers, not just the designers. In Intent Specifications, the assumptions and rationale (and other information such as safety constraints, limitations etc) are elevated in importance to a level alongside the requirements. In other words, the requirements cannot be allowed to exist without carrying this accompanying information with them. The accompanying information is seen as essential to provide meaning to the requirements (the requirements are naked without this accompanying information).

If one views this other information as needing to live with the requirements continuously, then a different perspective of the ideal development process emerges. Figure 24 shows how intent specifications carry more information content than current best practice.

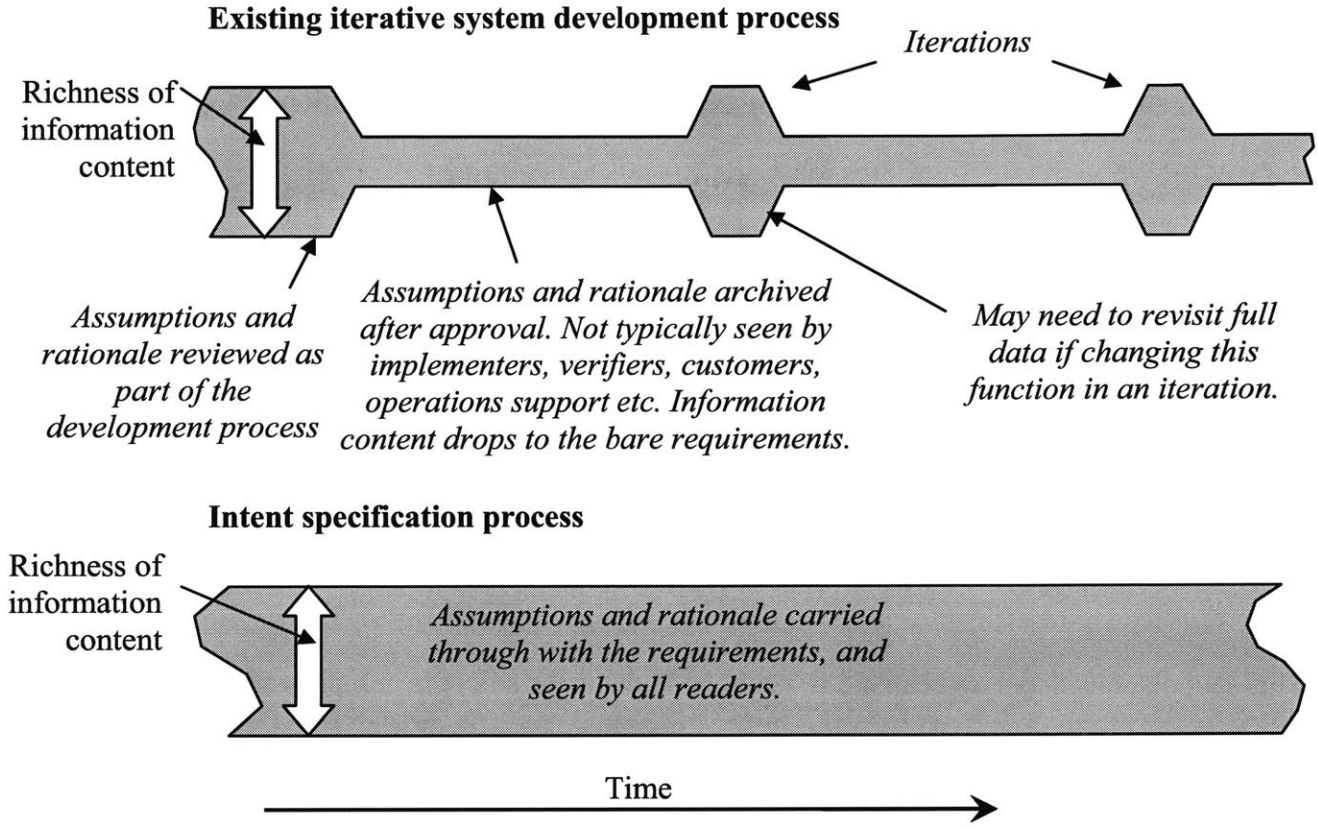


Figure 24 Information Richness - Intent specifications vs. Current practice

The reason for not carrying the full "intent" of the specification in the traditional process is that specifications have traditionally been viewed as a document that defines the task list for an implementer rather than a document that describes a mental model of the systems engineer. The research has revealed the importance of mental models (see Figure 13) in the various actors in the development process including the actual system controller. In the traditional approach, the aim is to get the specification "correct" at each level of the system hierarchy before passing it down to the next stage. But if the specification is right, what would be the

need for the extra information referred to above, if it isn't needed by the lower level to ensure compliance to the requirements? In a sense, Intent specifications have the same initial objective (to ensure the specification is "correct") but the difference is that intent specifications recognize that "correctness" cannot always be guaranteed in the initial development process, and it takes more account of the life cycle development of the system in which the environment may change, use cases may change etc. Only by keeping the assumptions and rationale together can the validity of the system in its changing environment be confirmed.

It is also worth noting that DO-178B, and more particularly ARP-4754 (for example, refer to Figure 5 of ARP 4754 [7]), do place a lot of emphasis on *assumptions*, but this emphasis does not seem to lead to the same degree of attention in actual system development.

5.3 Requirements Completeness and Controller Internal Models

This section considers how realistic it would be to make the software comply with some of the completeness pre-requisites, such as the existence of an internal controller model and the objective of obtaining feedback for every output. These are some of the more challenging objectives that were not satisfied in the examples reviewed in chapter 3. It is recognized in the requirements completeness table in 'APPENDIX B Requirements Completeness Criteria With Example Systems' that the failure to have an internal controller model and the lack of use of feedback were major factors in these incidents.

In the existing aerospace development process, requirements completeness comes under the 'requirements validation' process step. The activities of this step are reviewed in several design review meetings (for each functional change), and in various stage gate reviews including the preliminary design review (PDR) and critical design review (CDR). A certification audit,

known as the validation audit, is also performed to review the outcome of the other reviews. However, the existing process does not apply requirements completeness checks to the depth recommended in the research (see Table 1, in APPENDIX B Requirements Completeness Criteria With Example Systems).

Commercial tools such as SpecTRM [30] include automated completeness checks. Other tools such as SCADE [41] also claim to have completeness checks (refer to the 'Design verifier' part of SCADE).

5.3.1 Feedback

Consider for example, the engine torched start incident in 3.3. The command to open or close the LSOV had no feedback from the final valve, and therefore the software was unable to confirm the appropriate state of the valve. However, even in situations lacking feedback, the software could still incorporate a model to indicate the expected state-based on past actuator outputs. The design reviewed had no explicit model in the FADEC. In the actual system, a FADEC does receive data from the opposite FADEC channel indicating the output status of the LSOV commands, and so a simple model of the expected state of the valve could have been developed. This would have helped identify the inappropriate states that the software entered into as part of this incident.

5.3.2 Hazardous Software States and Internal Models

In the system reviewed in chapter 3, the loss of sensor signals, or disagreements between FADEC states are generally recognized and have associated fault-detection and accommodation software. This leads to transitions to less hazardous states, as required by the requirements completeness criteria. Aerospace systems have a huge amount of BITE (built in

test equipment) which detects faults, logs the faults internally, transmits them to aircraft systems such as a central maintenance computer etc. However, these designs are not explicitly developed in response to the hazard analyses. Rather, the redundancy objectives have driven the need for fault-detection. Thus there may be several potentially hazardous software states that are not being identified by the existing design process.

It is suggested here that the identification of hazardous software states can be enhanced using the documented assumptions and by ensuring that the software models the anticipated state of all output devices. Some design assumptions can be checked in software.

For example, in the redesign of the P25 fault-detection logic discussed in 3.4, there was a feature in which the validated signal value would be set to the lowest of the two channels under certain conditions (actually when the synthesis model had failed in combination with a cross-check failure between channels). This design was chosen on the *assumption* that the typical failure mode of the sensor was low (this is consistent with analysis of sensor failure data). It would be possible to test this assumption in software automatically by detecting cross-check failures in which the failed signal was high and logging this event in NVM (that is, logging this as an assertion). Currently, there is no precedent for adding this type of design feature because it does not directly contribute to meeting the higher level functional requirements.

The NASA software safety standard refers to this feature as a "trap" and provides the following definition of it [42]:

> *Trap. Software feature that monitors program execution and critical signals to provide additional checks over and above normal program logic. Traps provide protection against undetected software errors, hardware faults, and unexpected hazardous conditions.*

With regard to the possible use of internal process models, consider the torched start example in section 3.3. The software controlled the LSOV but did not incorporate any explicit model to anticipate the state of the external device. Its state was implicitly inferred from the condition of the engine. For example, while the engine is running, the valve must obviously be open (if it was closed there would be no fuel going to the engine). However, in startup, particularly before servo pressure is available to move the PRV, it is not possible to infer the state of the LSOV from externally observable inputs. However, this does not mean that no model can exist. All control commands to the LSOV are actually visible to each FADEC, therefore the FADEC could predict its state from the sequence of commands to the LSOV. As part of the power-up sequence, the FADECs perform ground-tests including the opening and closing of the LSOV. Only one FADEC is needed to open the LSOV, but agreement between FADECs is needed to close it. In the case of an asynchronous power-up, the two FADECs will fail to agree on their closing commands. However, the FADECs do share their command statuses across the CCDL, and hence the failure to synchronize commands could have been identified. The predicted LSOV state could then have been left at "open", which should have been identified as a hazardous state, and logged as an assertion in NVM. In the actual design, there was code to close the LSOV on entry to start mode. The rationale for this was undocumented, but a system that logged an assertion that the "LSOV was believed to be left open after power-up checks" state would have provided evidence to validate the need for such a requirement. In the actual redesign following the incident discussed in 3.3, the power-up test to open the LSOV was deleted, thus eliminating the potential for this hazard altogether. However, this was not necessarily the most elegant solution as the system was now unable to detect a failure to open the valve until an actual commanded engine start, which although safe, may have delayed any necessary maintenance action to address the problem.

The discussion above is not to suggest that every documented assumption must be tested in software or that there must be a process model for every output. However, the existing design process focuses on the flow-down of requirements and on ensuring traceability between levels, but does not allow for additional requirements that are associated with checking assumptions and identifying hazardous states. Where the requirements specifier may have thought to add such requirements then they will be implemented and verified, but there is no review process that comprehensively and proactively establishes how assumptions will be validated and how hazardous software states will be identified and eliminated.

Therefore, there appears to be an opportunity to enhance the system and software safety process by improved coordination with the system development process, using the requirements completeness criteria listed in Table 1, the concepts behind Intent Specifications, and developing assertions in software to monitor these processes.

In a discussion with Nancy Leveson on what to do when the completeness criteria detects a missed transition criteria from a software state, the author suggested that if the external environment was such that the transition could never occur, then perhaps the missed criteria may not be needed. Nancy's recommendation was to add the missing criteria to the software requirements anyway even if the system design suggested it should not be needed. For example, the entry conditions for the FADEC *ground-test* mode included the requirement that N2=0; however, the exit conditions included only the condition that the start switch was pressed. The requirements completeness criteria would identify the lack of an N2 exit condition as an omission (an incomplete specification). However, the assumption about the system was that the only way that N2 could go above zero is if the engine had started, which would require a start command from the pilot, and hence there should be no need for an N2 exit condition. This was

an undocumented assumption. In the actual incident, the FADEC was unable to recognize the start switch because of an inadvertent (and separate) design change, which meant that the start switch could only be recognized when N2>500 rpm. However, during ground-test mode, N2 validation had been temporarily halted (leaving a fixed N2 value of zero), so this meant the start switch could not be recognized and hence the assumption upon which the ground-test exit requirements were based did not hold. This illustrates the problem with hidden assumptions that may hold in initial versions of software, but can fall apart with apparently small design changes later on.

Software is said to become more "brittle" as it is progressively developed. This is the phenomenon that as changes are progressively made, a lot of (often inadequately documented) assumptions and design rationale start to be violated. Essentially the cohesiveness that the original architecture had starts to break down, and the interactive complexity rises rapidly.

This suggests that the overhead of explicitly testing assumptions in software, and including apparently redundant state transition paths (with logged assertions) may be worthwhile for future-proofing the software, i.e. reducing the brittleness and improving the robustness to changes in the environment.

5.3.3 Data Timing Issues

There were several cases in the incidents reviewed where external input parameters were used when they had stale or defaulted data. One of the implications of the requirements completeness criteria is that every external input signal should have an associated status variable that indicates the following types of information:

- Data age
- Use status (failed, defaulted, synthesized, data borrowed from other FADEC, etc.)
    - this is information for downstream processes on how and whether the signal should be used
- Maintenance Status (latched failed, not latched failed)

The completeness objectives associated with specifying the time bounds of each signal (high and low) can be captured in the fault-detection logic. The goal of the age indication is to capture the case of using a last good value of the signal (which was an issue in the P25 logic – refer to section 3.4).

Some of the above signals do exist in the current design; however, there is no process for enforcing downstream functions to use the status information. The requirements completeness criteria would regard as incomplete any function that does not explicitly define how the function should operate for each state of the input. In the case of the torched start incident, the software used an N2 signal that was being defaulted to zero. Had the requirements for entering start mode considered such a scenario, the designer would have been prompted to develop an exception where start mode could be entered without meeting the N2>500 criterion, in the case

where N2 was defaulted. The software could also detect a start switch being pressed while the N2 signal was defaulted and log an assertion in NVM.

5.4 Other Recommendations

The following are additional recommendations from the MIT system safety research that are not specifically discussed elsewhere in this chapter.

- Software safety must be part of the system safety plan (software hazards)
- The software needs to specify the off-nominal behavior as much as the nominal (e.g. asynchronous power-ups, unusual pilot commands such as a start soon after an interrupt)
- The specifications must include what the software must not do as well as what it must do
- Assertions and run-time checking must be used (several of the recommendations in 5.3 suggested the logging of information on unexpected states in NVM, particularly in connection with the violation of assumptions or the entering of hazardous software states (5.3.2))
- Robustness testing shall be performed (testing outside the specifically defined requirements, to cover range of possible input variable values including noise and timing skews)

5.5 Organizational Learning

When organizations learn from mistakes, they typically engage initially in "single loop" learning, as shown in Figure 25 below. In this mode, the organization applies well known techniques to the problem. For example, organizations typically address apparent process failings by creating more processes to plug the perceived gaps. However, there is a danger that

this process just creates a bow wave of process change as it attempts to plug more and more holes found.

It is hypothesized here that some of the recommendations discussed above require a change in the mental model of system development, which involves "double loop" learning. In the company studied, it has often been said that the engineering was working well, and that program management was the area that needed most improvement. Better processes would help address the deficiency, it was believed. What this thesis finds is that there are weaknesses in the engineering process, caused by a lack of application of systems theory to the system safety and system development processes. This suggests that there is an opportunity for some fundamental rethinking about the engineering process – that is, the creation of a new mental model about how to design and develop safe complex highly integrated systems.

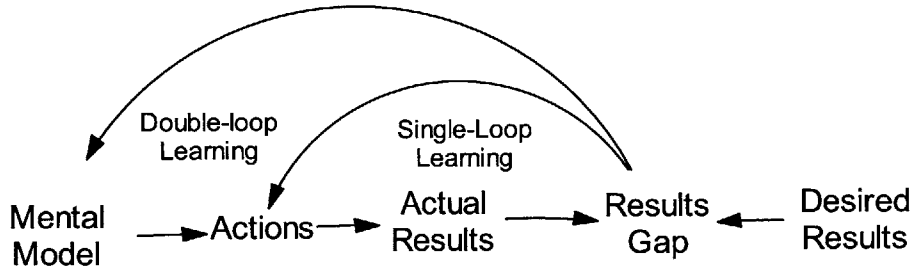


Figure 25 Single and Double Loop Organizational Learning (from [43])

Figure 26 below (adapted by Carroll [44] from Rasmussen's model [45]) shows some of the boundaries which affect how an organization balances the competing needs of safety, economic competitiveness and workload. Individuals naturally explore the *space* available between the boundaries, and rely on indicators to help establish where the boundaries are. These indicators are provided by the various checks and balances that an organization creates, such as reviews, audits, professional judgment etc. The safety culture of an organization

depends on the strength of the safety culture boundary relative to the pressures imposed by the other boundaries.

It is argued in this thesis that the safety culture in the aero engine industry is generally healthy, but the existing safety assessment processes do not handle the complexity issues created by the use of software well, and there is considerable room for improvement. A greater emphasis on software hazard analysis, requirements completeness checks and requirements modeling should create a more effective safety assessment process, may reduce the likelihood and cost of rework, and facilitate greater requirements reuse.

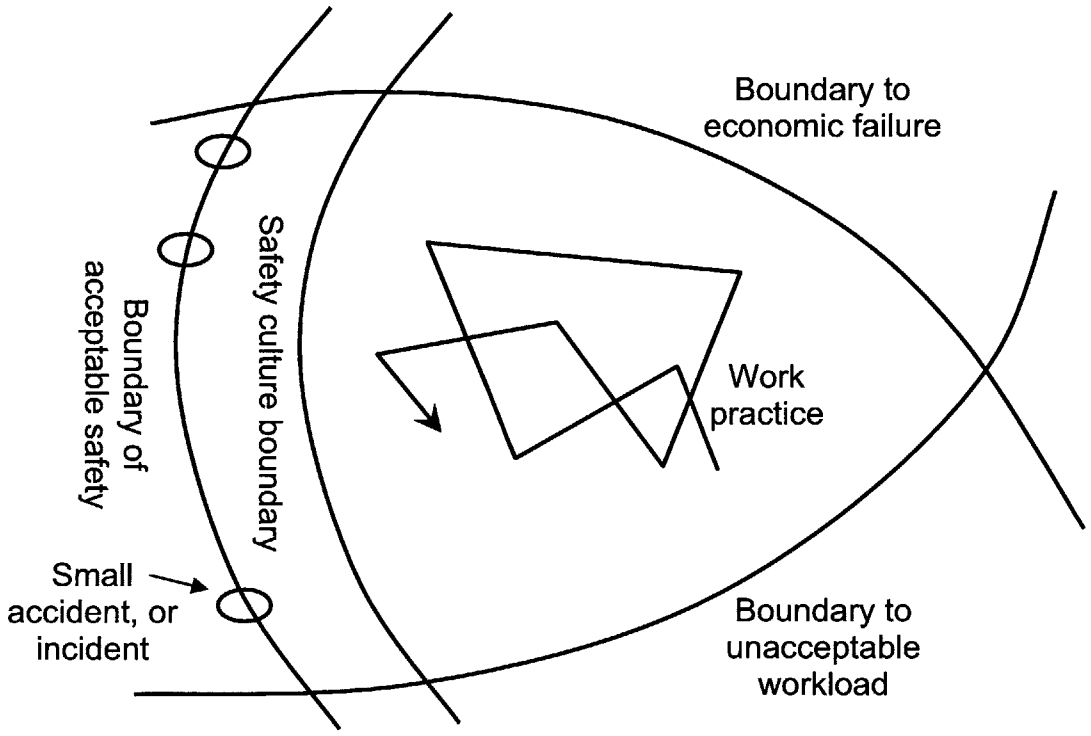


Figure 26 Human Behavior Constraints ([44], adapted from [45])

5.6 Aerospace Recommended Practices

The conclusions drawn above have some implications for the ARP guidelines. First, ARP 4761 implies that much of the safety analysis will make use of techniques such as FTA or Markov

models. The issues raised here about the significance of requirements errors in the system safety process needs to be reflected in the ARP at some point. Thus more emphasis on the documentation of safety constraints, assumptions, rationale and mental models and how these are flowed down through the system and into software may be needed. Emphasis on this area may displace some of the emphasis on the quantitative approach.

Currently, the ARP delegates software safety to DO-178B, which focuses on ensuring that the software performs its required functions. DO-178 may need to be modified to better recognize the importance of identifying hazardous software states and the techniques for testing for requirements completeness. DO-178 may also need to consider the importance of internal process models in the software and the concept of continuously ensuring that the perceived system state matches the actual system state. Previously such considerations have always been regarded as being application-specific design decisions and outside the scope of DO-178. Both of the system development ARPs (4761 and 4754) and DO-178 can be said to be "process centric". That is, they avoid any guidance on specific design techniques or approaches but provide process guidance that could be applied to a broad range of system environments.

However, what the research at MIT suggests is that there are some fundamental design issues that are common to safety-critical digital-control systems, and these should be the focus of some of the objectives and recommendations in these guides. For example, the role of assumptions, rationale, internal models, feedback etc.

6 Conclusions

The complexity of highly integrated aero engine control systems exceeds the capability of designers to understand or test all possible modes of operation. Furthermore, the system complexity increases significantly when embedded software is used in the system. The existing development process focuses on requirements. More specifically, the existing process addresses the development, configuration, allocation, implementation, validation and verification of requirements. The documentation of the rationale and assumptions of requirements are currently addressed as part of the requirements validation process, and will typically be documented separately from the requirements themselves. The lack of emphasis on (and documentation of) rationale and assumptions has contributed to a significant number of incidents during development and service operation.

Intent specifications have been developed at MIT and provide a new framework and discipline for the development of requirements, including assumptions, rationale, safety constraints, operator and maintenance requirements etc. Intent specifications allow for a richer set of information, i.e. going beyond the pure requirements themselves. This aims to ensure that violations of assumptions can be identified earlier in the program, thus reducing development and rework costs.

The existing safety assessment process for aero engine control systems makes significant use of quantitative methods such as fault tree analysis (FTA). The FTA approach addresses random mechanical failures, but no design or requirements errors. It has been found, however, that the majority of safety issues involving software are not caused by random mechanical failures but

by requirements errors. The FTA is designed to match the requirements, which represent the intended operation of the system. However, when software requirements errors or code errors are discovered, the requirements and code will be corrected, to match the original intent. Therefore, the FTA will often not need to change for such software updates. This reduces the usefulness of the FTA approach in predicting the rate of the top-level hazards.

In general, it has been found that the aerospace recommended practices for system development (ARP 4754) and software development (DO-178B) are better understood and followed than the safety assessment process (ARP 4761). At one site in an aero engine company, the common cause analysis recommended by ARP 4761 had not been generated on any program.

The software lifecycle process (DO-178B) primarily addresses system safety by the selection of a development assurance level (A to D) for the software. Once this choice has been made, system safety is assumed to be achieved by following the software lifecycle processes defined in DO-178B, but not via any specific software hazard analysis methods. This is in contrast to MIL-STD 882B which places significant emphasis on software hazard analysis. The system development approach recommended in Intent Specifications (and the commercial tool SpecTRM) allows for the specification of safety constraints and the identification of hazardous states in the software.

Analysis of several software-related incidents that occurred during development or operational service of an aero engine control system revealed requirements and design errors that were similar to accidents that have been documented in the literature. A comprehensive set of requirements completeness criteria were developed at MIT and these were found to cover a significant proportion of the errors analyzed.

Some key concepts in the completeness criteria include feedback, internal models, input parameter fault status and timing issues, and range testing etc. The SpecTRM commercial tool automates the checking of black-box software requirements against the completeness criteria, though a checklist approach could be used instead of the automated checks.

The development approach recommended in Intent Specifications requires a paradigm shift from existing practice. In particular, a greater diversity of information types should be documented in the specifications, including assumptions, rationale and safety constraints. The software lifecycle process should also be better integrated with the system safety process, and these changes should ultimately be reflected in the international aerospace recommended practices.

REFERENCES

1. SAE, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, in *ARP 4761*. 1996, Society of Automotive Engineers.

2. RTCA, *DO-178B, Software Considerations in Airborne Systems and Equipment Certification*. 1992.

3. FAA, *FAR Part 25.1309 'Equipment, Systems, and Installations'*, US Department of Transportation, Federal Aviation Authority.

4. FAA, *Advisory Circular AC 25.1309-1A 'System Design and Analysis'*. 1988, US Department of Transportation, Federal Aviation Administration.

5. Leveson, N., *SafeWare: system safety and computers*. 1995, Reading, Mass.: Addison-Wesley. xvii, 680 p.

6. FAA, *Advisory Circular AC 20.115B 'Radio Technical Commission for Aeronautics, Inc. Document RTCA/DO-178B'*. 1993, US Department of Transportation, Federal Aviation Administration.

7. EUROCAE, *Certification Considerations for Highly-Integrated Complex Aircraft Systems*, in *ED-79/ARP 4754*. 1996, EUROCAE.

8. McDermid, J.A., *Software safety: where's the evidence?* in *Sixth Australian workshop on Safety critical systems and software - Volume 3*. 2001, Australian Computer Society, Inc.: Brisbane, Australia.

9. Papadopoulos, Y. and J. A. McDermid, *The potential for a generic approach to certification of safety critical systems in the transportation sector*. Reliability Engineering & System Safety, 1999. **63**(1): p. 47.

10. Treacy, J., *2003 FAA National Software Conference - Overview of SAE ARP 4761*. 2003, FAA.

11. *Discussion Forum for the RTCA Special Committee/EUROCAE Working Group: SC-205/ WG-71 'Software Considerations in Aeronautical Systems'*. 2005.

12. DOD, *MIL-STD-882B: System Safety Program Requirements*, D.o. Defense, Editor. 1984.

13. Alberico, D., et al., *Software System Safety Handbook*, DOD, Editor. 1999, Joint Services Software Safety Committee.

14. Leveson, N., *The Role of Software in Spacecraft Accidents*. AIAA Journal of Spacecraft and Rockets.

15. Rodriguez, M., et al., *Identifying Mode Confusion Potential in Software Design*. IEEE, 2000.

16. Weiss, K.A., et al. *An analysis of causation in aerospace accidents*. in *Digital Avionics Systems, 2001*. 2001. Daytona Beach, FL, USA: IEEE.

17. James, I., et al. *Investigating No Fault Found in the Aerospace Industry*. in *IEEE Reliability and Maintainability Symposium*. 2003.

18. Howard, J. and K. Kelley, *A Notation Supporting a Systems-Theoretic Hazard Analysis Technique*, Safeware Engineering Corporation: Seattle, Washington, USA.

19. Leveson, N., et al. *Effectively Addressing NASA's Organizational and Safety Culture: Insights from Systems Safety and Engineering Systems*. in *Engineering Systems Division Symposium, March 2004*. 2004. Cambridge: Engineering Systems Division, MIT.

20. Leveson, N.G., *Shuttle Thermal Tile Processing Example Intent Specification*. 2002, Massachusetts Institute of Technology.

21. Leveson, N., et al., *A Systems Theoretic Approach to Safety Engineering*. 2004, Massachusetts Institute of Technology: Cambridge, MA.

22. Leveson, N.G., *A New Approach To System Safety Engineering (unpublished book draft)*. 2002, Massachusetts Institute of Technology.

23. Leveson, N.G., *A systems-theoretic approach to safety in software-intensive systems*. Dependable and Secure Computing, IEEE Transactions on, 2004. **1**(1): p. 66.

24. Forrester, J.W., *Industrial dynamics*. 1961, [Cambridge, Mass.]: M.I.T. Press. 464 p.

25. Forrester, J.W., *System Dynamics and the Lessons of 35 Years*, in *The Systemic Basis of Policy Making in the 1990s*, K.B.D. Greene, Editor. 1991.

26. Repenning, N.P., P. Goncalves, and L.J. Black, *Past the tipping point: The persistence of fire fighting in product development*. California Management Review, 2001. **43**(4): p. 44-63.

27. Leveson, N.G., *Completeness in formal specification language design for process-control systems*, in *Proceedings of the third workshop on Formal methods in software practice*. 2000, ACM Press: Portland, Oregon, United States.

28. Leveson, N., *Lecture Notes for MIT Course 16.683J/ ESD.683J: System Safety*. 2005, Massachusetts Institute of Technology.

29. Leveson, N.G., M.P.E. Heimdahl, and J.D. Reese. *Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future*. in *ESEC/FSE'99: 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 1999. Toulouse, France: Springer-Verlag GmbH.

30. Leveson, N., *Safeware Engineering Corporation - SpecTRM (commercial software application)*, http://www.safeware-eng.com/.

31. SAE, *Certification Considerations for Highly-Integrated Complex Aircraft Systems*, in *ARP 4754*. 1996, Society of Automotive Engineers.

32. Murdoch, J., J.A. McDermid, and P.J. Wilkinson. *Failure Modes and Effects Analysis (FMEA) and Systematic Design.* in *19th International System Safety Conference.* 2001. Hunstfield, Alabama: ISSC.

33. Leveson, N.G., *Intent specifications: an approach to building human-centered specifications.* Software Engineering, IEEE Transactions on, 2000. **26**(1): p. 15.

34. Leveson, N.G., *Sample Intent Specification: Altitude Switch.* 1999, Massachusetts Institute of Technology: Cambridge.

35. Wilkinson, P.J. and T.P. Kelly. *Functional hazard analysis for highly integrated aerospace systems.* 1998. London, UK: IEE.

36. Hawkins, R., J. McDermid, and I. Bate. *Developing Safety Contracts for OO Systems.* in *21st International System Safety Conference.* 2003. Ottawa, Ontario, Canada: ISSC.

37. Lindsay, P.A., J.A. McDermid, and D.J. Tombs, *Deriving quantified safety requirements in complex systems.*

38. Gerogiannis, V.C., I. Caragiannis, and M.A. Tsoukarellas, *A General Framework for Applying Safety Analysis to Safety Critical Real-Time Applications Using Fault Trees.* IEEE, 1997.

39. Leveson, N., S.S. Cha, and T.J. Shimeall, *Safety Verification of Ada Programs Using Software Fault Trees.* IEEE, 1991.

40. Leveson, N.G. and P.R. Harvey, *Software fault tree analysis.* Journal of Systems and Software, 1983. **3**(2): p. 173.

41. *Esterel Technologies Inc., SCADE Suite (commercial software application),* http://www.esterel-technologies.com/products/scade-suite/overview.html: Toulouse, France.

42. NASA, *Technical Standard: NASA-STD-8719.13A 'Software Safety'.* 1997.

43. Carroll, J.S., *Organizational Learning Activities in High-hazard Industries: The Logics Underlying Self-Analysis.* J Management Studies, 1998. **35**(6): p. 699-717.

44. Carroll, J.S., *Class Notes for MIT System Safety Course (16.863J): Talk by Prof. John Carroll (Operations and Management, Sloan School).* April 2005, MIT.

45. Rasmussen, J., *The role of error in organizing behaviour.* Qual Saf Health Care, 2003. **12**(5): p. 377-383.

46. Heimdahl, M.P.E. and N.G. Leveson, *Completeness and Consistency in Heirarchical State-Based Requirements.* IEEE Transactions on Software Engineering. **22, Issue: 6**: p. 363-377.

APPENDIX A FAR Part 25.1309 – Safety Regulations

§ 25.1309 Equipment, systems, and installations.

(a) The equipment, systems, and installations whose functioning is required by this subchapter, must be designed to ensure that they perform their intended functions under any foreseeable operating condition.

(b) The airplane systems and associated components, considered separately and in relation to other systems, must be designed so that—

(1) The occurrence of any failure condition which would prevent the continued safe flight and landing of the airplane is extremely improbable, and

(2) The occurrence of any other failure conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions is improbable.

(c) Warning information must be provided to alert the crew to unsafe system operating conditions, and to enable them to take appropriate corrective action. Systems, controls, and associated monitoring and warning means must be designed to minimize crew errors which could create additional hazards.

(d) Compliance with the requirements of paragraph (b) of this section must be shown by analysis, and where necessary, by appropriate ground, flight, or simulator tests. The analysis must consider—

(1) Possible modes of failure, including malfunctions and damage from external sources.

(2) The probability of multiple failures and undetected failures.

(3) The resulting effects on the airplane and occupants, considering the stage of flight and operating conditions, and

(4) The crew warning cues, corrective action required, and the capability of detecting faults.

(e) Each installation whose functioning is required by this subchapter, and that requires a power supply, is an "essential load" on the power supply. The power sources and the system must be able to supply the following power loads in probable operating combinations and for probable durations:

(1) Loads connected to the system with the system functioning normally.

(2) Essential loads, after failure of any one prime mover, power converter, or energy storage device.

(3) Essential loads after failure of—

(i) Any one engine on two-engine airplanes; and

(ii) Any two engines on three- or more- engine airplanes.

(4) Essential loads for which an alternate source of power is required by this chapter, after any failure or malfunction in any one power supply system, distribution system, or other utilization system.

(f) In determining compliance with paragraphs (e)(2) and (3) of this section, the power loads may be assumed to be reduced under a monitoring procedure consistent with safety in the kinds of operation authorized. Loads not required in controlled flight need not be considered for the two-engine-inoperative condition on airplanes with three or more engines.

(g) In showing compliance with paragraphs (a) and (b) of this section with regard to the electrical system and equipment design and installation, critical environmental conditions must be considered. For electrical generation, distribution, and utilization equipment required by or used in complying with this chapter, except equipment covered by Technical Standard Orders containing environmental test procedures, the ability to provide continuous, safe service under foreseeable environmental conditions may be shown by environmental tests, design analysis, or reference to previous comparable service experience on other aircraft.

{Amdt. 25–23, 35 FR 5679, Apr. 8, 1970, as amended by Amdt. 25–38, 41 FR 55467, Dec. 20, 1976; Amdt. 25–41, 42 FR 36970, July 18, 1977}

APPENDIX B Requirements Completeness Criteria With Example Systems

Table 1 Requirements Completeness Criteria with Examples (from [5, 27, 46])

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
Human Computer Interface Criteria						
1	Displays	What events cause this item to be displayed?			Engine thrust loss during takeoff	
2		Can and should this item ever be updated once it is displayed?			After 4 seconds, stops flashing	
3		Events that should cause the data display item to disappear			Cleared after V1 or when no longer in takeoff mode.	
State Completeness						
4		The s/w and system must start in a safe state				
5		The internal software state (assumed prior to receiving external data) must be updated to reflect the actual state, when data is received from outside	The FADECs initially powered up in shutdown mode, and probably should have entered ground-test mode after N2 had dropped to zero.			
6		All system and local variables must be initialized upon startup, including clocks				
7		The behavior of the software with respect to inputs received before startup, after shutdown or when temporarily off-line must be specified or analyzed to show it can be ignored	The software had no provision for recognizing an engine start request that may have occurred when off-line			
8		Maximum time that the computer waits before first input must be specified	This should have been specified for the hardware standard test (see Figure 6 item CN1).			

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
9		Paths from fail-safe states must be specified. The time in a safe, but reduced function state must be specified	Inadequate exit transitions from ground-test mode.	Once a fault had been detected in one channel, it was latched and only pilot action (pressing fault reset) could unlatch the fault. This is in contrast to some FADEC systems which allow self clearing after a period of time with no fault indications.		
10		Interlock failures should result in the halting of hazardous functions	The fact that both FADECs were in differing modes when performing the self-tests involving the opening and closing of the LSOV (which should have been recognized as potentially leading to a hazardous state) should have been recognized as an interlock failure. The shutdown agreement interlock would only work in self-tests if both FADECs were in the same mode.			
11		There must be a response for an input in any state, including indeterminate states	There was no response specified for a non-zero N2 value while in ground-test mode.			
Input and Output Variable Completeness						
13		All information from inputs must be used somewhere in the specification				
14		Legal output values that are never specified should be checked for specification completeness (e.g. specifying when a valve should be opened, but omitting the specification of closure)				

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
Trigger Event Completeness						
15	Robustness Criteria	Every state must have a behavior defined for every input	This is not generally ensured in the system analyzed.			
16		The logical OR of the conditions on every transition must form a tautology	The entry condition to ground-test mode included the condition N2=0, but did not specify what to do if N2>0.			
17		Every state must have a software behavior (transition) defined in case there is no input for a given period of time (a timeout)	The N2 signal was forcibly set to zero while self-tests were in progress, well after the actual N2 started to rise. Thus the software had no provision for how to handle stale data.	The P25 signal communicated between channels would be set to "last good value" when it had gone out of range in one channel. This made the data out of date when read by the other channel, but no accounting for this was made.		
18	Nondeterminism	The behavior of the state machine should be deterministic (only one exit transition possible at any given time)				
19	Essential Value Assumptions	All incoming values should be checked and a response specified in the event of an out-of-range or unexpected value				
20	Timing Intervals	All inputs must be fully bounded in time, and the proper behavior specified in case the limits are violated or an expected input does not arrive	If a start request is received while a FADEC is going through power-up checks, its entry to start mode would be delayed until power-up checks were complete. The start request would at that time be stale, but no time stamping of momentary inputs was used in this design, so no accounting of this was made.			

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
21		A trigger involving the nonexistence of an input must be fully bounded in time	There was a feature in the software to abort the hardware check in the event of a lack of input from the other channel, but it was invoked by entry to start mode and not by a time bound.			
22	Capacity or Load	A minimum and maximum load assumption must be specified for every interrupt-signaled event whose arrival rate is not dominated (limited) by another type of event				
23		A minimal arrival rate check by the s/w should be required for each physically distinct communication path. Software should have the ability to query its environment over a given communication path				
24		The response to excessive inputs (violations or load assumptions) must be specified				
25		If the desired response to an overload condition is performance degradation, the specified degradation should be graceful and operators should be informed				
26		If function shedding or reconfiguration is used, a hysteresis delay and other checks must be included in the conditions required to return to normal processing load				
Output Specification Completeness						
27		Safety-critical outputs should be checked for reasonableness and for hazardous values and timing	The high output current to the MMV during self-test mode should have been recognized as potentially hazardous if allowed to continue in a state when the engine was running.			
28	Environmental capacity considerations	For the largest interval in which both input and output loads are assumed and specified, the absorption rate of the output environment must be equal or exceed the input arrival rate				
29		Contingency action must be specified when the output absorption rate limit will be exceeded				

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
30		Update timing requirements or other solutions to potential overload problems, such as operator event queues, need to be specified				
31		Automatic update and deletion requirements for information in the human-computer interface must be specified				
32		The required disposition for obsolete queue events must include specification of what to do when the event is currently being displayed and when it is not				
33	Data age	All inputs used in the specifying output events must be properly limited in the time they can be used (data age). Output commands that may not be able to be exceeded immediately must be limited in the time they are valid				
34		Revocation of a partially completed action sequence may require (1) specification of multiple times and conditions under which varying automatic cancellation or postponement actions are taken without operator confirmation, and (2) specification of operator warnings to be issued in the event of such revocation				
35	Latency	A latency factor must be included when an output is triggered by an interval of time without a specified input and the upper bound on the interval is not a simple, observable event.				
36		Contingency action may need to be specified to handle events that occur within the latency period				
37		A latency factor must be specified for changeable human-computer interface data displays used for critical decision making. Appropriate contingency action must be specified for data affecting the display that arrives within the latency period.				

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
38		A hysteresis delay action must be specified for human-computer interface data to allow time for meaningful human interpretation. Requirements may also be needed that state what to do if data should have been changed during the hysteresis period.				
Output to Trigger Event Relationships						
39		Basic feedback loops, as defined by the process control function, must be included in the software requirements. That is, there should be an input that the software can use to detect the effect of any output on the process. The requirements must include appropriate checks on these inputs in order to detect internal or external failures or errors.	Although there was no direct means to verify the opening and closing of the LSOV, had a fuel flow meter signal been available to the FADEC, it may have been possible to perform partial confirmation of the action. Further, a shutdown request would be expected to lead to a drop in N2, and the observation of a drop in N2 without a shutdown request could have been checked during starting (was in engine running mode only).			
40		Every output to which a detectable input is expected must have associated with it: (1) a requirements to handle the normal response, and (2) requirements to handle a response that is missing, too late, too early, or has an unexpected value	See above			
41		Spontaneous receipt of a non spontaneous input must be detected and responded to as an abnormal condition	The software checks for a turbine temperature rise as an indication that the engine has lit, but there was no provision to look for such as event prior to the software commanding an opening of the fuel valve. Thus there was no ability to recognize this spontaneous input.			
42		Stability requirements must be specified when the process is potentially unstable.				

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
Specifications of Transitions Between States						
43	Reachability of States	All specified states must be reachable from the initial state				DO-178B also requires that there be no "dead code" in the software.
44	Recurrent behavior	Desired recurrent behavior must be part of at least one cycle. Required sequences of events must be implemented in and limited by the specified transitions				
45		States should not inhibit the production of later required outputs				
46	Reversibility	Output commands should usually be reversible				
47		If x is to be reversed by y, there must be a path between the state where x is issued and where y is issued.				
48	Preemption	Preemption requirements must be specified for any multistep transactions in conjunction with all other possible control activations	The pilot start request which occurred during one FADEC's power-up sequence preempted the ground-tests. Requirements were not robust in handling this.			
49	Path robustness	Soft and hard failure modes should be eliminated for all hazard-reducing outputs. Hazard-increasing outputs should have both soft and hard failure modes				
50		Multiple paths should be provided for state changes that maintain or enhance safety. Multiple inputs or triggers should be required for paths from safe to hazardous states.				

No.	Category	Criteria	Examples Reviewed			
			Torched Start	P25 Fault Detection	Engine Failure Warning	Other
Constraint Analysis						
51		Transactions must satisfy software system safety requirements and constraints	Safety constraints should have highlighted the risks of several ground-tests, such as the opening of the LSOV and the max current output to the MMV. However, no explicit analysis connected the safety analyses to these design features.			
52		Reachable hazardous states should be eliminated or, if that is not possible (they need to achieve the goals of the system), their frequency and duration reduced.				
53		There must be no paths to unplanned hazardous states				
54		Every hazardous state must have a path to a safe state. All paths from the hazardous state must lead to safe states. Time in the hazardous state must be minimized, and contingency action may be necessary to reduce the risk while in the hazardous state.			The software detected a failure of all FADECs to arm the ATTCS in a takeoff as a hazard. This led to an immediate transition to a safer state which was to trigger the ATTCS.	
55		If a safe state cannot be reached from a hazardous state, all paths from the state must lead to a minimum risk state. At least one such path must exist.			A pilot warning was provided when the software detected a hazardous state, so that even if the ATTCS failed to increase thrust, pilot action could be taken.	