The Principled Design of Computer System Safety Analyses

David John Pumfrey

Submitted for the degree of Doctor of Philosophy

University of York

Department of Computer Science

September 1999

For my parents

Abstract

Safety critical computing is a relatively young and rapidly developing technology, which nevertheless is being deployed in applications where a single accident may have extremely severe consequences. The safety record of critical systems presently in service is reasonably good, but increasing expectations of functionality and performance are challenging the capabilities of current design and assessment processes. One specific area where limitations of existing methods are becoming obvious is in the analysis techniques that are used to derive safety requirements and to provide evidence that they have been satisfied. There are significant practical problems in using existing analysis techniques to evaluate computer systems, but few viable new computer-specific methods have been developed.

This thesis proposes and evaluates a set of principles for the design of effective techniques to address novel computer system safety analysis requirements. The principles are based on an appreciation of the technical concepts underlying successful existing system level analysis techniques, and of the practical qualities necessary to make a method industrially acceptable. The principles are applied in the development of two new safety analysis techniques for systems containing computers.

The first new technique developed is Software Hazard Analysis and Resolution in Design (SHARD), a variant of the process industries' HAZOP technique. SHARD provides a structured approach to the identification of potentially hazardous behaviour in software systems. The second technique, Low-level Interaction Safety Analysis (LISA), implements a novel analysis approach based on a concept of system resources. It provides a method for establishing detailed evidence about the safety implications of interactions between software and the hardware upon which it is executed. The thesis describes the evaluation of the techniques through a series of large scale case studies and industrial trials.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor, Professor John McDermid, whose help, encouragement and patience have been invaluable.

This thesis has a large practical element, and I would like to thank British Aerospace for allowing me to observe and participate in so many of their projects, and for the time and resources they have committed to case studies and discussions of the work. Particular thanks must go to John Anderson and Mike Burke, the British Aerospace Dependable Computing Systems Centre (DCSC) research managers at York, for the time and effort they have devoted to finding suitable case studies, arranging contacts, and promoting this work within the company.

Too many engineers within the company have participated in case studies to name everyone individually, but special thanks are due to the co-ordinators of the biggest projects: Chris Harper, formerly of British Aerospace Airbus, Matt Tucknott of British Aerospace (now Matra-BAe) Dynamics, and Dave Ogden and Duncan Rawsthorne of British Aerospace Military Aircraft and Aerostructures (BAe MA&A).

Within the Department at York, I would particularly like to thank John Clark, Tim Kelly and Mark Nicholson for their friendship, support and constructive comments; also Neil Audsley for his contribution to the LISA case study.

Finally, on a personal note, I could not have got this far without the support and encouragement of Abi Robertson, who has helped me keep life in perspective throughout the writing up of this thesis.

Author's Declaration

Some of the material presented in this thesis has previously been published in the following papers:

J. A. McDermid and D. J. Pumfrey, 1994, "A Development of Hazard Analysis to aid Software Design," *COMPASS '94: Proceedings of the Ninth Annual Conference on Computer Assurance , Gaithersburg, MD*, pp. 17-25, IEEE, ISBN 0-7803-1855-2.

J. A. McDermid, M. Nicholson, D. J. Pumfrey, and P. Fenelon, 1995, "Experience with the Application of HAZOP to Computer-Based Systems," *COMPASS '95: Proceedings of the Tenth Annual Conference on Computer Assurance, Gaithersburg, MD*, pp. 37-48, IEEE, ISBN 0-7803-2680-2.

J. A. McDermid and D. J. Pumfrey, 1998, "Safety Analysis of Hardware / Software Interactions in Complex Systems," *Proceedings of the 16th International System Safety Conference, Seattle, Washington*, pp. 232-241, System Safety Society, P.O. Box 70, Unionville, Virginia 22567-0070, USA.

J. A. McDermid and D. J. Pumfrey, 2000, "Assessing the Safety of Integrity Level Partitioning in Software," *Lessons in System Safety: Proceedings of the Eighth Safety-critical Systems Symposium, Southampton*, Ed. F. Redmill and T. Anderson, pp. 134-152, Springer, ISBN 1-85233-249-2.

Some of the examples and diagrams used in Chapter 2 were originally written by the author for use in course notes and exercises for the Safety and Hazard Analysis module of the Department of Computer Science's MSc in Safety Critical Systems Engineering and a related series of industrial courses.

All of the work contained within this thesis represents the original contribution of the author.

Chapter 1

Introduction

1.1 The Sinking of the *Titanic*

Early in the afternoon of 11th April 1912, the R.M.S. *Titanic* left Queenstown (now Cobh), Ireland, her last port of call before her maiden transatlantic voyage. 882½ feet long, 92½ feet wide, over 175 feet from the keel to the top of her four smokestacks and displacing 52,310 tons, she was the largest vessel afloat. On that voyage, she carried 1,316 passengers (far less than her capacity of 2,566) and a crew of 885 (again, less than her maximum complement of 945) – a total of 2,201 people.

Four days later, at 11:40 p.m. on Sunday 14th April, travelling at a speed her navigator estimated to be in excess of 22 knots, the *Titanic* struck an iceberg in the area of the Grand Banks of Newfoundland. It quickly became obvious that the damage was extensive, and at 12:45 am the first of the lifeboats was launched. *Titanic* carried 20 lifeboats, but with capacities of between 40 and 65 there was room for only 1,178 people. In the event, the *Titanic* sank very slowly at first and many people were reluctant to leave the apparent safety of the ship for small open lifeboats, with the result that many of the boats were launched with less than their full complement. By the time the last lifeboat was launched at 2:05 am, there were still 1549 people left on the ship. The first ship to arrive on the scene was *Carpathia*, which picked up the 652 occupants of the lifeboats, and rescued a further 60 people found swimming or clinging to wreckage at the scene.

The accident – dubbed "the greatest maritime disaster in history" – was investigated by a United States Senate Inquiry and later, more thoroughly, by a British Board of Trade Inquiry [60]. Some of the findings of this inquiry make almost unbelievable reading.

The number of lifeboats that the standards of the time required ship to carry were based not on the permitted passenger complement, but on the ship's gross tonnage. Around the turn of the century, there was a spectacular increase in the size of the ships being built, mostly, like *Titanic*, for use on the profitable and prestigious transatlantic routes. Despite this, the tables relating number of boats to tonnage had last been updated in 1894. The tables only gave requirements for ship up to 10,000 tons – less than ¼ of the size of the *Titanic*. In fact, the *Titanic* carried *more*

lifeboats than were required, as vessels with double bottoms and radio equipment, both of which the *Titanic* had, were entitled to claim reductions in the required number of boats.

Ice was a well-known hazard of the north Atlantic routes in spring. Pilots' guides and charts of the time note that there was a high probability of encountering southward drifting ice north of 43° N (although sightings of individual bergs had been reported as far south as 39° N), and west of 45° W (the eastward limit of exceptional sightings being about 38° W). The *Titanic* was following a route known as the Outward Southern Track, which followed a great circle route from Fastnet to 42° N, 47°W, then struck out westwards to pass just south of the Nantucket Shoals and thence via coastal waters to New York. This route would have taken her less than 100 miles south of the normal southern limit of ice, and at least 300 miles north of the most southerly reported sightings.

During the 24 hours leading up to the accident, the *Titanic* was shown to have received at least six separate radio messages from other ships warning of ice. However, there is no evidence that more than three of these were ever passed to the captain or other bridge officers. At that time, radio on board ship was a recent innovation. The operators were all employees of the Marconi Company, and it is clear that their main role was as a service for passengers, rather than an integral part of the navigation and management of the ship. At about 10:20 p.m. on the 14th April, the *Californian*, which had been steaming west on a course only 10 miles north of the *Titanic*, sent a radio message warning that she had stopped for the night after encountering heavy ice. The *Titanic*'s radio operator was busy; Cape Race, the first of the American shore stations, had just come in range, and he was clearing a backlog of messages for the passengers. He curtly told the *Californian* "Shut up. I'm working Cape Race". Taken together, even the three messages known to have been taken to the bridge clearly showed the existence of an extensive field of floating sheet ice, "growlers" (small icebergs) and substantial icebergs extending down to 41°N and from 49° to 51° W – directly ahead of the Titanic.

The only action that Captain Smith is known to have taken in response to the warnings was to tell the lookout to keep a sharp watch for ice. Some of the hatches forward of the bridge and crow's nest were closed to avoid spoiling the lookout's night vision. Incredibly, the lookout was not provided with binoculars, with the result that when he spotted the iceberg dead ahead, it was less than 500 yards from the ship. The time from the lookout's call to the bridge to impact was a mere 37 seconds, during which time the engines were signalled "full astern", and the helm swung hard over. These actions ensured that, rather than hitting head-on, the *Titanic* struck the berg

obliquely, scraping past hard along the starboard bow. The impact was so gentle that many on board were unaware of it, but it was sufficient to open a long narrow tear in the side of the ship.

The *Titanic*'s hull was divided into sixteen compartments, and had been designed so that she would remain afloat if any two compartments were completely flooded. Further, she could survive the flooding of all of the front four compartments. The intention was that the ship should be her own lifeboat – whatever went wrong, she was expected to remain afloat, although her owners (the White Star Line) and builders (Harland and Wolff in Belfast) never actually went so far as to claim she was unsinkable. Openings in the bulkheads dividing the hull compartments could be sealed by automatic watertight doors, and this was done in the first few seconds after the impact. However, the long tear in the side of the ship had penetrated as far as boiler room 6 – the crucial fifth compartment from the bow. Since the compartments were not closed at the top (the bulkheads extended well above the waterline, but the decks above were not strengthened or watertight – see Figure 1) it was now inevitable that the ship would sink as her bow went down and water spilled over into successive compartments.

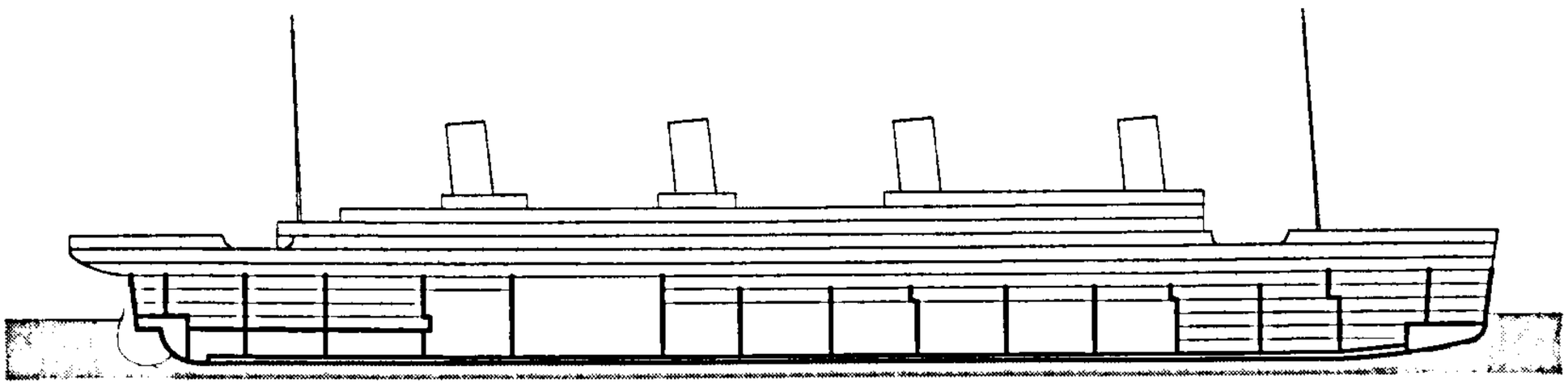


Figure 1 - Division of *Titanic*'s hull into compartments

The Board of Trade Inquiry into the sinking concluded that the cause of the accident was Captain Smith's failure to reduce speed and maintain an adequate lookout when ice was expected. The massive loss of life was blamed on the shortage of lifeboats, and the main recommendation of the inquiry was that all ships should be equipped with sufficient lifeboats for the number of people they carried, regardless of other provisions such as life jackets and flotation aids. Interestingly, the Inquiry conclusions make no reference to the design assumptions of the ship, merely noting that she complied with regulations and had passed the necessary sea trials (a very superficial affair, lasting less than 24 hours). Other factors that seem incredible by modern standards, such as the lack of binoculars, are merely noted as normal for ships of her class.

In 1985, an expedition led by Dr. Robert Ballard [3] located the wreck of the Titanic, lying in two major parts (the ship broke in two upon sinking) at a depth of 12,460 feet. Subsequent expeditions recovered parts of the vessel's structure, and concluded that the long split in the hull caused by the collision was partly a result of poor materials. There was too high a proportion of clinker in the wrought iron rivets joining the ship's steel plates, and the impact had caused a horizontal seam to "unzip" as the rivets failed. The deficiency was relatively minor, and would probably never have been discovered had it not played a part in the sinking. The *Titanic*'s sister ship the *Olympic*, built at the same time and to a virtually identical hull design (her superstructure was slightly smaller), made over 500 Atlantic crossings before being retired from service in 1935. The third ship built to the same pattern, the *Brittanic* was sunk by German mines in 1916.

1.2 The destruction of the Hindenburg

The airship LZ-129 *Hindenburg* was conceived as the ultimate in luxury transatlantic travel. Almost as long as the Titanic, her 7 million cubic feet of hydrogen and helium gases provided 242 tons of gross lift, sufficient to carry 50 passengers and 60 crew in futuristic living quarters built into the keel of the 112 ton airship. The *Hindenburg* was the flagship of the Zeppelin Company, at the time the undisputed world leaders in airship design and construction. She had originally been intended to use only inert helium gas, but the only world's only sources of natural helium were in the United States, which had just passed an act controlling its use and export. The design was altered to use dual gas cells, with an inner cell containing hydrogen surrounded by helium in the outer cell.

The *Hindenburg* was the largest airship the company had built, and was specifically designed for the transatlantic service which began in 1936, making 10 successful return journeys between her German base at Friedrichshaven and Lakehurst Naval Air Station, New Jersey. The first trip of 1937 arrived at Lakehurst on 6th May. The crossing had been delayed by headwinds, and the first attempt at landing had to be abandoned because of the unsettled weather left by passing thunderstorms, but by early evening the winds had dropped, and the *Hindenburg* approached the mooring mast. The landing lines had just been dropped to the ground crew when a fire broke out near the tail of the craft. The flames spread rapidly forward, and the entire airship was destroyed in just 34 seconds. Amazingly, several people survived, either jumping from the blazing wreck or crawling out after it crashed to the ground, but 40 people were killed.

The official enquiry investigated two possibilities. The first was that the fire was the result of an act of sabotage. Relations between the USA and Nazi Germany had deteriorated, and the

Hindenburg was a potent symbol of German national pride. However, there was no evidence to support this theory, and it was soon abandoned in favour of the eventual conclusion, that escaping hydrogen gas had been ignited by lightning.

The hydrogen fire explanation was not challenged publicly until Addison Bain, a former engineer on the Space Shuttle programme, became interested in the accident [36]. Studying photographs, film footage and witnesses accounts, Bain realised that there were many inconsistencies in the evidence. Photographs show the *Hindenburg* well alight, with flame pouring from more than half the envelope, and yet the airship is still horizontal – the hydrogen could not have escaped in large quantities to fuel the blaze, as at this point it was still providing lift. Moreover, all the witnesses agreed that the fire had burnt with yellowish or orange flames, but hydrogen flames are virtually colourless, and blue if visible at all.

Bain suspected that the fire started on the fabric of the airship's outer skin. This was made of a cotton cloth coated with various treatments to make it stiffer, water-resistant and reflective, to avoid excessive solar heating of the lifting gases. The preparations applied to the *Hindenburg* were unique; the ship was expected to fly higher and in worse weather than any previous Zeppelin, and new combinations of chemicals were used to try to meet the tougher requirements. Various fragments of the cloth survived, and chemical analysis revealed the principal constituents of the treatments applied. The fabric had first been coated with a varnish of cellulose acetate and nitrate; the reflective layer consisted mainly of aluminium powder, but also contained iron oxide. Iron oxide and aluminium powder are the reagents in the Thermite reaction used to weld railway lines, and are also components of solid rocket fuel. The cellulose acetate and nitrate (a component of gunpowder) are also highly flammable, and their combustion provides sufficient heat to ignite the metallic fuels.

Laboratory tests proved that, although this is an exceedingly flammable mix, electrical discharge striking the fabric perpendicular to the surface (as lightning would) punched a hole in the cloth without igniting it. However, when the discharge ran along the surface of the cloth, it ignited almost instantaneously. Static electric discharge was a known problem with airships; the Zeppelin design went to great lengths to ensure that all metallic parts of the airship were electrically bonded to the duralmin frame to avoid charges building up. What had been missed in the case of the *Hindenburg* was the potential for the metallic-coated cloth to build up huge charges; the cloth panels were tensioned to the frame using non-conductive cords.

The wet conditions and highly charged atmosphere left by the thunderstorms at Lakehurst on the evening of May 6th meant that the charge that built up on the skin of the airship was higher than ever encountered before. Whilst the craft was airborne, the whole structure was at a similar potential, but as soon as the mooring rope was dropped, the charge on the well-bonded frame flowed away to earth, leaving a huge potential difference between the cloth panels and adjacent metalwork. Bain believes that this was the source of the discharge that ignited the outer covering of the ship. There is no doubt that the hydrogen burned; however, it was not released until the heat of the fire caused the gas cells to rupture.

Subsequent to Bain's investigation, it emerged that the engineers working for the Zeppelin Company had uncovered the real cause of the accident very soon after it happened. Later Zeppelin airships such as the *Graf Zeppelin* built in 1938 were coated with bronze instead of aluminium, and the cords lacing the skin panels were treated to make them conductive. However, the official explanation of lightning igniting escaping hydrogen was convenient for all involved; it avoided the Zeppelin Company appearing liable, and insurance companies could claim an "act of God" and reduce their pay-outs.

1.3 Learning from history

Although both over 60 years ago, the *Titanic* and *Hindenburg* disasters provide a challenging starting point for a consideration of the state of system safety engineering at the end of the 20th century. Both were the biggest, most advanced technology of their day – very much equivalent to the status of the safety critical, computer controlled systems being developed and deployed in a range of industries today. Perhaps the most frightening aspect of these examples is not the serious mistakes obvious with the benefit of hindsight, but the similarities between these and much more recent accidents.

The most striking aspect of the sinking of the *Titanic* (and the subsequent investigation) is the almost unbelievable complacency of so many of the people concerned. The end of the Edwardian era was a time of great confidence in engineering and scientific progress, and the decisions that condemned so many may be seen as a product of this extreme confidence. However, Ballard himself draws parallels between the *Titanic* and the much more recent loss of the space shuttle *Challenger* in January 1986, pointing out the ways in which an overconfidence in technology and an underestimation of the power of the natural environment led to negligence by those in command in both instances.

Even the official report of the Board of Trade displays an air of unjustified self-assurance; it discusses the inadequacy of the requirements for lifeboat provision without any apparent criticism other than a mild note that they are outdated. Similarly, it notes that binoculars, floodlights and other aids to watch-keeping are not required for a ship of *Titanic*'s class, without acknowledging that this might be a deficiency in the Board's requirements. The general tone, that a regrettable incident has occurred but the blame lies elsewhere, can be observed in any number of more recent accident reports.

There are also technical questions posed by the *Titanic* disaster. How was the structure of the hull decided? What analysis decided the number of compartments, the height of the bulkheads and the survivability requirements? These may seem unfair questions given the technology of the time, but again there are modern parallels. For example, compare the apparent assumption of the likelihood of head-on collision (evinced by the ability of the ship to survive the flooding of the four forward compartments as compared to only two elsewhere) to the almost identical assumption made by car manufacturers for crash testing of new vehicles until the regulations were strengthened in 1997.

The *Hindenburg* highlights other problems of safety engineering which continue to challenge modern engineers. Perhaps the most obvious of these is the problems associated with re-assessment of safety after design changes. In the *Hindenburg*'s case, this change was two-fold; it was a modification (enlargement) of a previously successful design, and there was revision of the concept when it became clear that sufficient helium would not be available to lift a ship of that size. The serious engineering challenges of constructing such a huge, lightweight structure were solved admirably, but there was no consideration of the impact of the (apparently minor) change in the fabric treatments applied. This is particularly interesting given that static electric build-up was clearly a known problem, and care had been taken to provide discharge paths for the frame and metal components.

Also interesting in this case was the recently revealed cover-up by the Zeppelin Company. Again, there are an alarming number of modern parallels; the aftermath of accidents such as Piper Alpha [20], Bhopal [72] and Seveso (described in [50]) all revealed attempts to hide, or at least play down, evidence of previous accidents, operating problems and inadequate safety management.

These observations could apply to safety engineering and management in any industry. Safety is essentially an *experiential* discipline; accidents are avoided by applying the lessons learned when

things go wrong, and historical data is one of its most important resources. To illustrate this, consider the development of aviation over the last 150 years. Early developments were characterised by a fly-fix-fly approach – perhaps better characterised as fly-hope to survive-fix-fly. The more cautious early pioneers of flight either paid or tried to persuade others to pilot their creations. Sir George Cayley's coachman reputedly resigned with the words "With respect, Sir, I was not hired to fly" after completing a brief and rather uncontrolled flight across a Yorkshire valley in one of Cayley's gliders in 1853 [13]. Accidents, injuries and fatalities were common as engineers struggled to master the basics of flight, and relatively little thought was given to safety.

By contrast, at the end of the 20th century, flying is one of the safest ways to travel. Figures from Boeing for 1996 [6] show that by the end of the year there was a world-wide fleet (excluding former Soviet Union countries) of 12,343 commercial jet aircraft, which provided a combined total of 16.3 million flights. During 1996 there were only 30 accidents involving deaths of passengers or aircrew, resulting in 1,300 fatalities – a rate of 80 deaths per million flights. Every accident is followed by an extensive investigation that tries to establish what went wrong, and whether there is a need to modify aircraft or improve crew training or procedures to prevent a recurrence. The reports from these enquiries are widely disseminated (in the UK, the Air Accident Investigations Branch publishes its reports on the world-wide web [1]) for the benefit of the whole industry.

1.4 Implications for computer system safety

For safety critical computing, the implications are clear. This is a relatively young and rapidly developing technology, which nevertheless is being deployed in many contexts such as nuclear power generation, civil aerospace, rail transportation and others, where a single accident may have extremely severe consequences. Safety critical computing simply cannot afford the sort of learning experiences that aviation technology obtained from the early days of fly-fix-fly experimentation. It is essential that lessons and skills learned in the more traditional engineering disciplines are transferred to the new technology quickly and effectively.

In 1994, MacKenzie published what remains the most comprehensive attempt to date to catalogue computer-related accidental deaths [56], to try to provide answers to fundamental questions about the safety (or otherwise) of computer systems in critical applications. His survey estimates that world wide up to the end of 1992, 1,100 people (± 1,000) died in computer-related incidents. The high uncertainty in this figure represents the difficulty in deciding which incidents

should be attributed to computer related failures. The survey details 34 specific incidents, concluding that physical failures of computer equipment accounted for approximately 4% of the deaths, software failures 3%, and failures in human-computer interaction 92%. Again, these figures are qualified by warnings that the data available is too poor to draw statistically sound conclusions.

MacKenzie concludes that computing is an error-ridden technology that nevertheless has a reasonably good safety record in practice. This should not be a surprising result. Recognition of the possibilities for, and potential consequences of, computer system failures has led to conservatism in design, with redundancy, backup devices and mitigation implemented in alternative technologies to reduce the impact of computer failure.

As computers are deployed in more and more safety critical applications, it is becoming obvious that there are still many significant problems to be solved. Widely publicised incidents such as the collapse of the London Ambulance Service computer-aided despatch system [32] and the destruction of Ariane 5 [55] emphasise the need for improvement in both the management and technical aspects of safety critical systems development and deployment. The diversity of current standards and practices, and disagreement over concepts such as the use of Safety Integrity Levels (SILs) illustrate the absence of common understanding and agreement within the industry on the best route to improved products.

Even the idea of computer safety itself remains controversial. Some researchers and practitioners maintain that safety is strictly a systems issue. In their view, it is the responsibility of the systems engineers to specify the properties required from the computers in order to guarantee safety; the problem for the computer system designers is strictly one of correct implementation of the specifications. Others maintain that there is so much functionality and complexity incorporated into modern computer systems that it is essential to regard safety engineering activities as an integral part of their development. (For a recent example of this debate, see the discussion of the Ariane 5 failure in the archives of the *hise_safety_critical* mailing list [35]).

Although safety critical computing is a large and active research area, most authors concentrate on aspects such as the specification and proof of functional properties, defining architectures for redundancy and fault tolerance, or suitability of design methodologies and programming languages. There is surprisingly little work describing analysis techniques for defining the safety requirements or assessing the actual safety related behaviour of a completed system. This thesis

takes the view that, whether they are considered to be aspects of systems engineering or parts of the computer system development process, these analysis techniques are essential to the production and assurance of computer systems in critical applications. They are also one of the areas in which safety critical computing can benefit directly from other engineering disciplines, by studying and developing the ideas which underpin successful techniques.

1.5 Thesis Proposition

This thesis investigates the following proposition:

> It is possible to establish a set of principles for defining effective computer system safety analysis techniques. These principles enable sound techniques to be developed to satisfy novel analysis requirements.

This proposition is supported:

1. by demonstrating that the principles are based on concepts and methods underlying existing successful system safety analysis techniques;
2. by explaining observed weaknesses in existing computer system safety analysis processes and techniques; and
3. through the development and successful industrial application of two new computer system safety analysis techniques, addressing identified deficiencies in the range of existing analyses.

1.6 Thesis Structure

Chapter 2 introduces basic safety engineering terminology and processes to set the context for a review of selected system safety analysis techniques from a range of industries. The criteria used in selecting a small set of techniques for review from over 100 identified in the published literature are explained, and then the chosen techniques are described, with emphasis on the principles on which they are based and their intended role in the safety process.

Chapter 3 explores some of the concepts identified in the review of terminology and safety analysis techniques in Chapter 2. The concepts of hazards and failures are discussed in some depth, as these are fundamental to all of the analysis techniques studied, and differences in interpretation explain some of the inconsistencies in approaches to safety engineering between different industries and nationalities. The chapter concludes by proposing a new model for classifying the role of techniques in the safety process, and using it to explore the relationship between the different techniques required to build a complete and coherent safety process.

Chapter 4 reviews the published literature on safety analysis for computer systems, including modified versions of HAZOP and fault trees, the use of Petri nets, and Failure Propagation and Transformation Notation (FPTN).

Chapter 5 proposes nine basic principles for the selection or design of analysis techniques for computer systems. These address issues highlighted by observation of, and participation in, industrial projects, as well as some of the specific problems discussed in Chapter 4.

Chapter 6 considers the analysis requirements of computer systems developers, and identifies areas in which existing techniques do not meet these needs. Two of these areas are discussed in some detail, providing the motivation and background for the development of SHARD and LISA described in Chapters 7 and 8.

Chapter 7 describes Software Hazard Analysis and Resolution in Design (SHARD), a new safety analysis techniques which was developed to fulfil one of the unsatisfied analysis requirements identified in Chapter 6. SHARD is a predictive, requirements setting analysis, based on the chemical industries' HAZOP technique. The chapter describes the initial proposal for SHARD and the way in which it was developed and refined through a series of case studies and industrial trials. As well as describing the technical development of the method, the chapter also discusses its practical integration into current industrial working practices, comparing the traditional team based approach of HAZOP with alternative ways working. It then presents a complete method description, incorporating all of the additional rules and guidance derived from the case studies.

Chapter 8 discusses the development of Low-level Interaction Safety Analysis (LISA), the second of the two new safety analysis techniques based on the principles discussed in Chapter 5. LISA is intended to produce evidence of achievement to contribute to the safety argument for a completed computer system design. It explores the interactions of critical software with the underlying hardware by modelling the system as a set of *resources* (physical devices and timed events), and examining how the system resources are used and managed. The chapter also describes the large case study that was used to develop and evaluate LISA, and presents a complete method description.

Chapter 9 evaluates the contribution of the work described in this thesis. The theoretical aspects of the work are discussed in terms of their contribution to the development of SHARD and LISA,

and their value in describing and evaluating other analysis techniques. In assessing the effectiveness of SHARD and LISA, the impracticality of conducting comparative experiments is noted, and alternative ways of assessing their value and maturity are proposed, based around case studies, technology transfer and industrial acceptance.

Chapter 10 concludes by examining how the work described supports the thesis proposition and identifying possible directions for future work, both to strengthen SHARD and LISA, and to extend the theoretical aspects of the thesis.

Chapter 2

Survey of system safety and hazard analysis techniques

This chapter presents a review of a number of hazard and safety analysis techniques. There is a huge range of techniques in use in industrial practise and proposed in research literature, and it would be impossible to review (or even merely describe) all of them in this thesis. Section 2.3 explains the selection of techniques for inclusion in more detail, and the techniques themselves are examined in sections 2.5 to 2.14. However, before reviewing the techniques, it is necessary to define some terminology and to briefly examine the safety lifecycle to provide context for the descriptions of the techniques.

A significant, and frustrating, problem encountered by any system safety researcher or practitioner is the variation in the definition and understanding of key terminology between different industries, different standards within industries and, particularly, between Europe, the USA and other nationalities. Chapter 3 reviews the definitions of some key concepts and the implications of some of the different interpretations.

2.1 Definitions

The primary definitions in this section are taken from Defence Standard 00-56 [78].

Safety: The expectation that a system does not, under defined conditions, lead to a state in which human life or the environment is endangered.

Accident: An unintended event or sequence of events that causes death, injury, environmental or material damage.

Risk: The combination of the frequency, or probability, and the consequence of an accident.

Note that, by these definitions, avoidance of accidents is sufficient to ensure safety, but the concept of an accident also includes material (financial) loss. The majority of safety analysis activities are inherently capable of investigating all risks associated with accidents. Some authorities insist on a "pure" safety process (i.e. limited only to human death or injury), but this does not fundamentally affect the way in which safety engineering activities are carried out.

System: A bounded physical entity that achieves in its domain a defined objective through the interaction of its parts.

Domain: The part of the external world, including users and personnel, that affects, and is affected by the system and may be harmed by an accident caused by it.

Component: A discrete structure, such as an assembly, within the total system considered at a particular level of analysis.

Some additional terms related to the definition of systems are also used in this thesis. A **platform** is the largest engineered artefact (e.g. a complete aircraft, train or chemical process plant), and this term is used for emphasis when discussing properties of the entire system. The term **subsystem** is used to define a group of components within the system which are related by functional objectives. For example, an aircraft fuel subsystem consists of physical components such as tanks, pumps and pipes which are widely distributed, and cannot be considered to be a discrete structure. This thesis also follows common practice in using the term **environment** rather than domain for the external world. The term **operator** is used as a generic term for anyone who interacts with a system in a controlling capacity, thus car drivers and aircraft pilots are operators.

The behaviour of a system, subsystem or component is described in terms of **events, states** and **actions,** where an action implies a deliberate intent by a control system or operator to alter the state of the system.

Hazard: A physical situation, often following from some initiating event, that can lead to an accident.

Fault: An imperfection or deficiency in the system

Failure: The inability of a system or component to fulfil its operational requirements.

The term **flaw** is similar to fault, but implies a deficiency in the original state of the system (i.e arising from specification, design or manufacturing errors) rather than an imperfection arising as the result of a component failure.

2.2 System and safety lifecycles

Many models have been proposed for system lifecycles, and for the engineering, safety and software development processes by which they are created. For the purposes of examining the roles of the analysis techniques surveyed here, a simple "V" model (adapted from McDermid and Rook [59]) is sufficient. The inner "V" in Figure 2 shows the major activities in an idealised system development process; the round-cornered boxes around the outside of the "V" show the safety tasks which are associated with these development activities. Sections 2.2.1 to 2.2.6 briefly describe the purpose and conduct of each of the safety tasks.

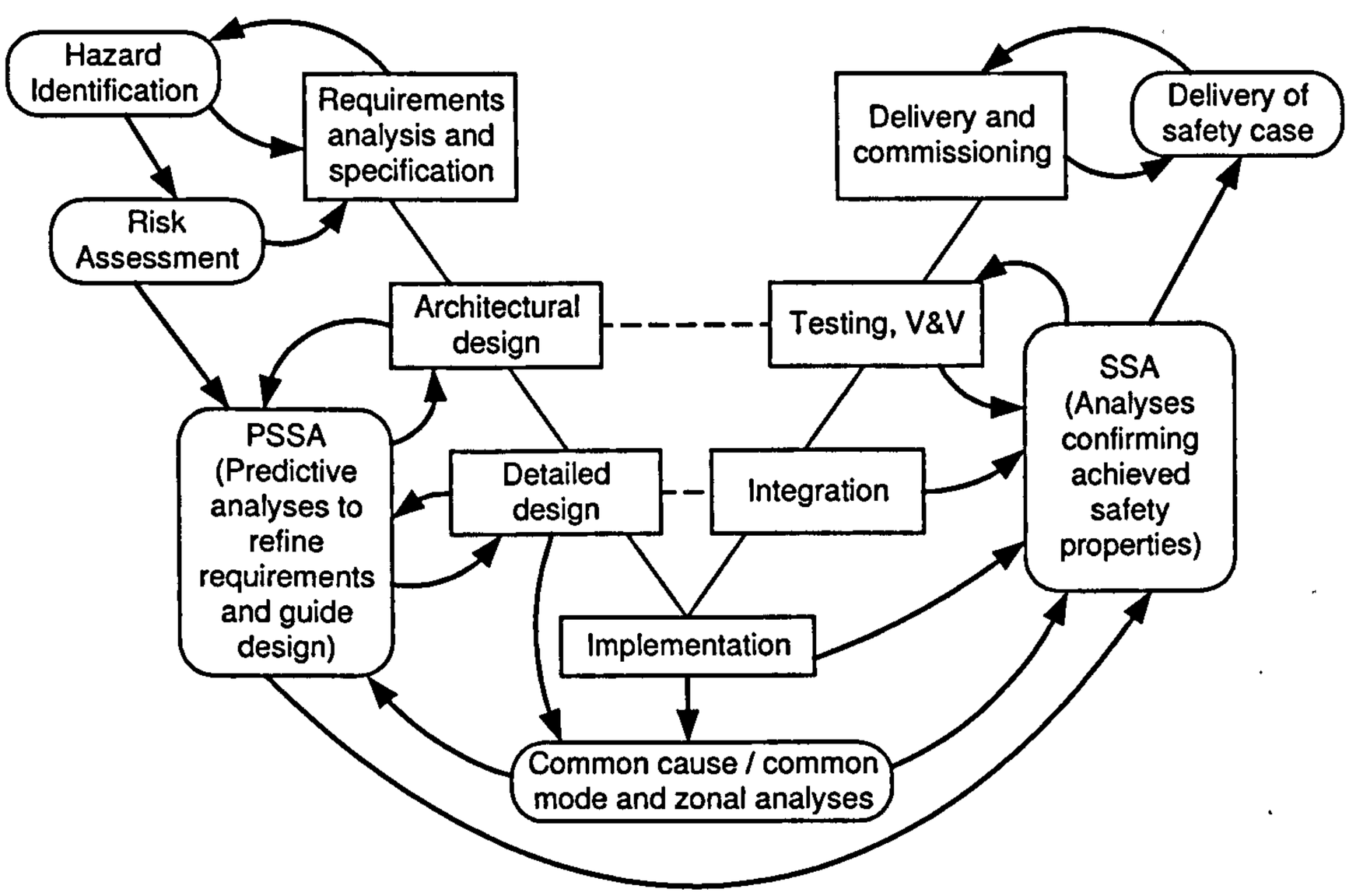


Figure 2 - V lifecycle model showing safety activities

2.2.1 Hazard Identification

Hazard identification is the first step in the safety process, and sets the context for all subsequent activities. Its role is to identify the potential hazards in the proposed system so that they can be managed and controlled. The techniques required in this phase are generally exploratory, asking "what if" type questions.

2.2.2 Risk Assessment

Risk assessment (often called hazard analysis) examines each of the identified hazards to determine how much of a threat they pose, i.e. the severity and likelihood of potential accidents.

This provides the basic information required to decide on the acceptability of design proposals and the steps that are necessary to reduce risks to acceptable levels.

There are numerous approaches to risk assessment. Detailed quantitative analyses may be used where accurate data is available. More often, qualitative assessments are used, categorising severity and probability into broad bands ("catastrophic"... "negligible" and "frequent"... "incredible"). A hazard risk index uses these classifications to determine the acceptability of the risk.

Reduction of risk to acceptable levels is based on two main principles. The first of these is that risks should be reduced *as low as reasonably practicable* (ALARP) [5]. This principle identifies risks in three categories. Low levels of risk are classified as *broadly acceptable*, and generally need no additional reduction. Extremely high risks are *intolerable*, and must be eliminated. In between these extremes is the ALARP region, which defines risks that are tolerable provided that they can be shown to have been reduced so far as is practicable, i.e. to the point where the costs of further reductions would be disproportionate to the improvement gained.

The second important principle is that of precedence ordering of risk reduction measures:

- *eliminate the hazard* – the best option, where possible, completely eliminating the risks
- *reduce the probability* of the hazard arising, or *mitigate the effects* of the hazard
- *provide alerts and warnings* – allowing operator intervention, evacuation etc.
- *establish procedures* – the least preferable option.

The risk assessment activity is the source of many of the initial safety requirements.

2.2.3 Preliminary System Safety Assessment

Preliminary system safety assessment is the term used in ARP 4761 [71] for the safety activities which accompany the design activities in the development process. Safety analysis in this phase of the project has two main roles; firstly to ensure that a proposed design can reasonably be expected to meet its safety requirements, and secondly to refine the safety requirements and help guide the design process. The balance of these two roles will change from an initial emphasis on confirming the acceptability of overall proposals during architectural design to the identification of specific safety requirements to guide detailed design.

These roles mean that PSSA is necessarily very closely linked with the design activities, and requires analysis techniques that can work predictively from incomplete information. It also requires analysis techniques which are relatively quick and inexpensive to apply. The majority of techniques used during this phase are deductive (investigating the possible causes of a specific condition), starting from the platform level hazards and requirements identified during PHI, and carrying information about acceptable failure modes and rates down from risk assessment into the design.

PSSA is generally recognised as one of the weaker phases of many existing safety processes. The author has collaborated on a paper [21] which investigates some of the issues in effective integration of the safety and design processes.

2.2.4 System Safety Analysis

System safety analysis is the main confirmatory safety activity, producing the evidence that demonstrates that safety requirements have been met. Techniques used in this phase are a mixture of inductive (working forward from known or hypothesised condition to possible outcomes) and deductive methods, which can investigate the detail of a completed design. Quantitative analysis is common, combining figures for actual component failure rates to calculate overall probabilities of platform level hazards.

The methods used also include a range of Particular Risk Assessments – specialised techniques dedicated to the investigation of unusually severe or complex hazards. Many of these techniques use modelling or simulation. Examples include the investigation of fan bursts on aircraft engines, nuclear and radiation safety analyses, and the atmospheric or water-borne dispersal of chemical releases.

2.2.5 Common Cause / Common Mode Analysis

Many safety analysis techniques assume independence between component failures. Common cause and common mode analyses are a range of techniques for identifying non-independence of failures. They are applied throughout the development process in support of both PSSA and SSA. Sources of non-independence investigated include common design flaws, manufacturing, installation and maintenance problems, and common external threats such as fire. They also consider a range of generic failure mechanisms; for mechanical and electrical components these typically factors such include stress, heat, corrosion and vibration. Zonal Hazard Analyses are techniques that consider common failure mechanisms related to physical proximity.

2.2.6 Delivery of safety case

The final safety activity in a development process is the delivery of a safety case – a comprehensive and defensible *argument* that the system is acceptably safe to use in a given context, supported by the necessary *evidence*. Kelly [43] has presented a notation for representing the structure of safety arguments, and has also proposed the concept of safety case patterns – common argument structures for supporting particular claims. The analyses carried out in the PSSA will contribute to the safety case argument, justifying the design approach; the evidence presented in the safety case will consist largely of results from the system safety analysis activities.

2.3 Selection of analysis techniques

The main focus of this work is the identification of concepts and principles from system level safety analysis techniques which can help to inform the development of new and improved computer system safety analyses. The primary interest of this survey, therefore, is *qualitative* analysis techniques, with an emphasis on hazard identification and the relationship between system design and safety properties.

As noted in the introduction to this chapter, the analysis techniques included in this survey were selected in accordance with a set of criteria intended to ensure that the techniques reviewed were as representative of current best practice as possible, and that the most important concepts were captured. These criteria were:

1. The techniques of interest are those which examine the technical, functional and behavioural aspects of a system.

   Management tools, risk assessment and management techniques and human factors assessments are all acknowledged to be vital to the engineering of safe systems; however, they are outside the scope of this survey. In practice, the emphasis on qualitative analysis did not restrict the selection, since the main techniques used quantitatively in the safety process are extensions or variants of techniques that have been included. However, the method descriptions in sections 2.5 to 2.14 concentrate on qualitative aspects.

2. There should be at least one technique representative of each phase of a typical safety critical system lifecycle.

   The overview of the safety lifecycle in section 2.2 shows the diversity of analysis approaches required by the various phases. This criterion was intended to ensure that the concepts most important to each phase would be included in the review.

3. Where possible, techniques selected should be "generic", i.e. they should be applicable to a range of types of system and used in several industries, rather than being technology or industry-sector specific.

   This criterion was intended principally to exclude from consideration the large number of particular risk assessments, that is, the highly specialised techniques which have evolved to analyse high-risk hazards in particular industries (for example, a large number of techniques for assessing hazards associated with handling of nuclear materials, and techniques such as fan burst and tyre shred modelling in the aerospace industry). It also seems reasonable to expect that an analysis technique that is already generic is more likely to be readily adapted to a new domain than one whose application is limited.

4. Both *inductive* and *deductive* techniques should be included.

   These are the two traditional classes of safety analysis; inductive analysis working forward from a known cause to discover its potential effects, and deductive analysis searching for the causes of a specific (known or expected) unwanted event.

5. Where there was a choice of techniques that satisfied the first three criteria (i.e. several candidate generic techniques applicable to a given lifecycle phase), the technique selected should be that in most common industrial use.

   In judging techniques against this criterion, the industries principally considered were the European aerospace, transport and military sectors. These industries all have relatively mature safety cultures and strive to implement best current practice. From a practical viewpoint, these were also the industries whose requirements were most fully understood. British Aerospace supported much of the practical and case study work reported in this thesis, and companies from the other sectors have been involved in system safety courses, providing opportunities to discuss ideas and study examples with practising engineers.

6. Where it was clear that a technique embodies unique concepts, it was included in the set even if it did not meet criteria 1-5.

   Naked Man (section 2.6) was included because it was substantially distinct from the rest of the set. Sneak Circuit Analysis (section 2.11) was included because of its predominance in American system safety processes, although there are very few instances of its application in European industry.

The most comprehensive listing of safety analysis techniques in the literature is to be found in the System Safety Analysis Handbook [73]. The handbook is intended to be a starting point for safety professionals, and aims to provide an overview of a wide range of techniques, together with references to allow readers interested in a particular technique to obtain more detailed

information. It therefore provides an excellent basis for comparing and selecting techniques. In addition to the 101 techniques contained in the 2nd edition of the Handbook, a further nine techniques have been identified in literature surveys.

Table 1 summarises the techniques identified. Of the groups of techniques listed here, the first four (accident investigation, management tools, human factors and particular risk assessments) are outside the scope of this survey. The majority of the risk assessment techniques are quantitative methods for comparing risks, evaluating trade-offs and assessing the effectiveness of risk reduction measures and are, again, outside the scope of this survey. It should be noted that the computer specific techniques identified in group 2 are simply restatements of general safety lifecycle processes with more explicit links to software development activities, and add nothing of value to the principles outlined in section 2.2.

	Technique group		Systems	Computer specific	Total
1	Accident and incident investigation methods		5	0	5
2	Management tools and plans and others		22	3	25
3	Human factors analyses		9	0	9
4	Particular risk assessments		15	0	15
5	Hazard identification		20	1	21
6	Risk assessment		11	0	11
7	Event based analyses	Inductive	9	2	11
8		Deductive	1	1	2
9		Integrated	5	0	5
10	State based analyses		2	0	2
11	Common cause analyses		4	0	4
	Totals		**103**	**7**	**110**

Table 1 - Summary of safety analysis techniques identified

From the remaining groups, the following techniques have been selected for more detailed examination:

- Hazard identification
  - Preliminary Hazard Identification (section 2.5)
  - Naked Man (section 2.6)
- Inductive event based analysis
  - Functional Failure Analysis (section 2.7)
  - Event Trees (section 2.12)

  - Failure Modes and Effects Analysis (section 2.9)
  - Sneak Analysis (section 2.11)
- Deductive event based analysis
  - Fault Trees (section 2.10)
- Integrated event based analysis (combining inductive and deductive approaches)
  - HAZOP (section 2.8)
  - Cause-consequence analysis (section 2.13)
- Common cause analysis
  - Zonal Hazard Analysis (section 2.14)

Petri net analysis, the most widely applied state based technique, is discussed with other computer system analyses in Chapter 4 because, although theoretically of general applicability, most of the documented case studies and industrial applications of this technique have been to computer systems.

2.4 Example system

To illustrate the application of the analysis techniques described in sections 2.5 to 2.14, the vehicle speed sensor subsystem of a theoretical automotive powertrain (i.e. engine and gearbox) control application is used as an example. The main components of the system are illustrated in Figure 3.

The vehicle speed is measured by a variable reluctance sensor that detects the rotation of a toothed wheel on the output shaft of the gearbox (i.e. the rotation of the toothed wheel is directly coupled to the rotation of the vehicle's driven road wheels). Each tooth edge passing the sensor generates a signal with either a positive or negative initial peak, resulting in an output waveform of the shape shown in Figure 4a. The sensor output is taken via the vehicle wiring loom to the powertrain management system module, where it is converted by signal processing electronics into a logic level square wave (Figure 4 b), which is fed to an input pin of the control unit. Within the control unit, the rising and falling edges of the signal are counted, and used to calculate the vehicle's speed, acceleration and distance travelled. These values are provided to the engine management functions, and to the electronic dashboard for display to the driver. The vehicle is also fitted with an electronically controlled automatic gearbox, and the vehicle speed is one of the primary inputs for gear selection.

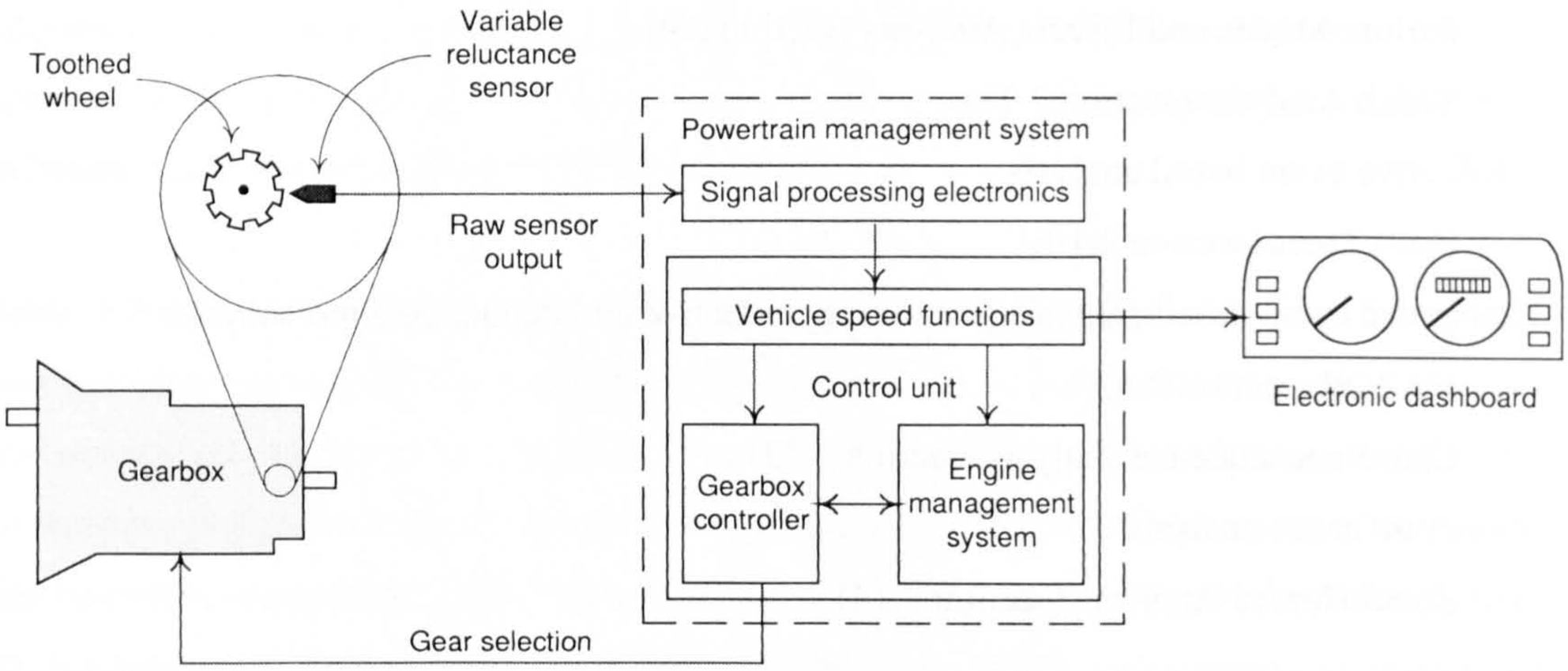
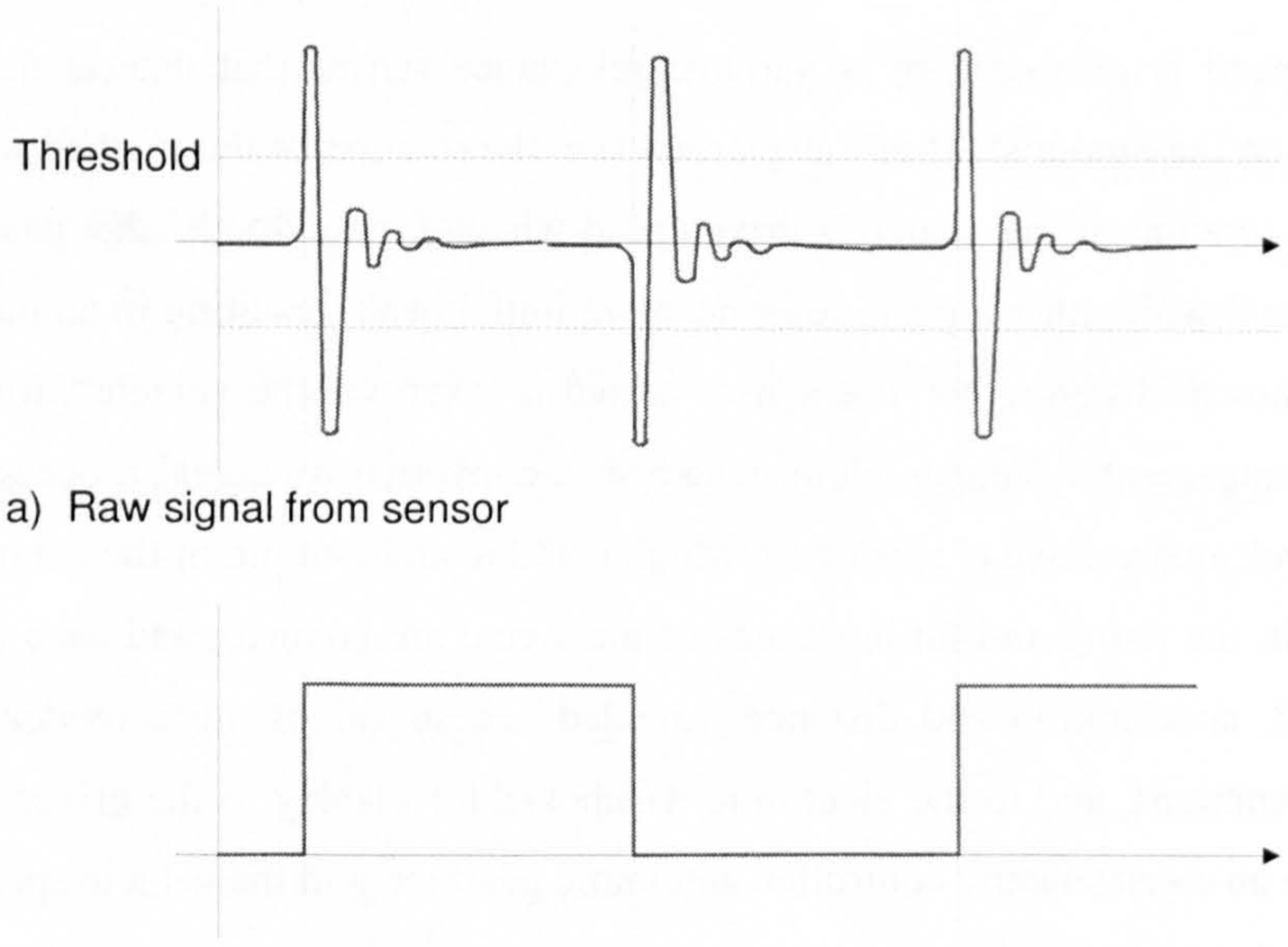


Figure 3 - Main components of vehicle speed sensor subsystem

The vehicle can be considered stopped if the time ΔT between edges exceeds a predetermined limit T_{max}, and the minimum time T_{min} expected between edges is determined by the maximum speed that the vehicle is capable of. The vehicle also has known maximum acceleration and deceleration capabilities; these determine the maximum relative change in the period of the signal. These parameters can be used by the system to test for input faults. For additional fault detection, the speed sensor subsystem can also use values provided by other functions of the powertrain management system. For example, there may be a fault if the indicated vehicle speed is zero, but engine load and speed are high – the energy must be being dissipated somewhere.



a) Raw signal from sensor

b) Square wave output from signal processing electonics

Figure 4 - Vehicle speed sensor waveforms

2.5 Preliminary Hazard Identification (PHI)

Despite its importance in the safety process, preliminary hazard analysis is normally a relatively simple task. It is very rare for an organisation to develop a safety critical system in a domain in which it has no experience. For most projects, therefore, PHI consists of consulting a checklist of known hazards of the application domain, and deciding which are of concern. For example, British Aerospace maintains comprehensive hazard lists documenting the hazards which have been identified (or encountered) with a range of aircraft types. New projects may introduce a small number of novel hazards (such as the additional hazards associated with controlling an aerodynamically unstable aircraft which had to be addressed in the Eurofighter development programme), but there is rarely a need to conduct a hazard identification starting from a blank sheet of paper.

Probably as a result of this, there are few techniques which can truly be described as methodical or systematic processes for identifying hazards. Most texts describe PHI as a process of brainstorming to suggest potential hazards, followed by critical review to identify genuine hazards from the suggestions. Numerous checklists are available to guide the process, although it is worth noting that many checklists actually numerate possible *causes* of hazards, and need to be interpreted in the system context to find the specific hazards.

The most systematic PHI techniques available are those based on the identification of energy sources, and the potential means for an unwanted or uncontrolled release of that energy to "targets" (usually people) that might suffer harm. Energy Trace and Barrier Analysis (ETBA) is the most systematic of such techniques listed in the System Safety Analysis Handbook [73]. This procedure combines checklists of energy sources with general prompts to investigate the intended work of the energy in the system, the way it is controlled, and possible changes in the energy (e.g. transformation into another type of energy).

2.6 Naked Man

Naked Man is a technique worth examining because of its unique emphasis on the *intrinsic* safety of a system, and systematic study of the value of additional safety and protection systems.

The examination starts by considering the most basic possible system – the minimum that is needed to operate, with all additional control and safety functions removed. The hazards associated with this "primordial" system are identified and assessed, and removed or reduced as

far as possible by revisions of the basic concept. The system is then reassessed with each of the proposed safety related control and protection functions added, to determine which provide the most significant (or most cost-effective) improvements.

There is no specific analysis method associated with this process – Naked Man uses event trees (section 2.12) and other analyses as required to assess each variant of the system. However it is, in concept, the only technique to explicitly impose the hierarchy of risk reduction measures introduced in section 2.2.2. The analyst / designer is forced to think about the safety of the system *concept*, and search for ways to make fundamental improvements, rather than simply "patching over" problems with additional safety features.

One of the exercises used for teaching on the Department's MSc in Safety Critical Systems Engineering provides a good example of the purpose of this technique. The process plant illustrated in Figure 5 is intended to make a product "Z" by mixing reagents "X", a toxic, viscous liquid which must be heated before it can be pumped around the system, and "Y", which is non-toxic and flows easily at the plant's normal operating temperatures. The reaction is mildly exothermic provided the correct proportion of "X" and "Y" is maintained, but can become explosive if the two are brought together in the wrong proportions. An analysis such as HAZOP (section 2.8) applied to this proposed design rapidly uncovers many flaws; there is insufficient instrumentation, no redundancy of control system or valves, and there are many ways in which dangerous situations can arise.

The Naked Man approach shows that the concept of this proposal is badly flawed – a great many (relatively complex) safety mechanisms must be added before the overall behaviour of the plant is acceptable. A complete redesign along the lines of Figure 6 is preferable. The critical quantities can be measured before the reaction begins, and using gravity to feed the reagents means that there is less reliance on complex components such as pumps and flow meters. Although this revised sketch is far from complete, fewer additional measures are required to make its overall safety acceptable.

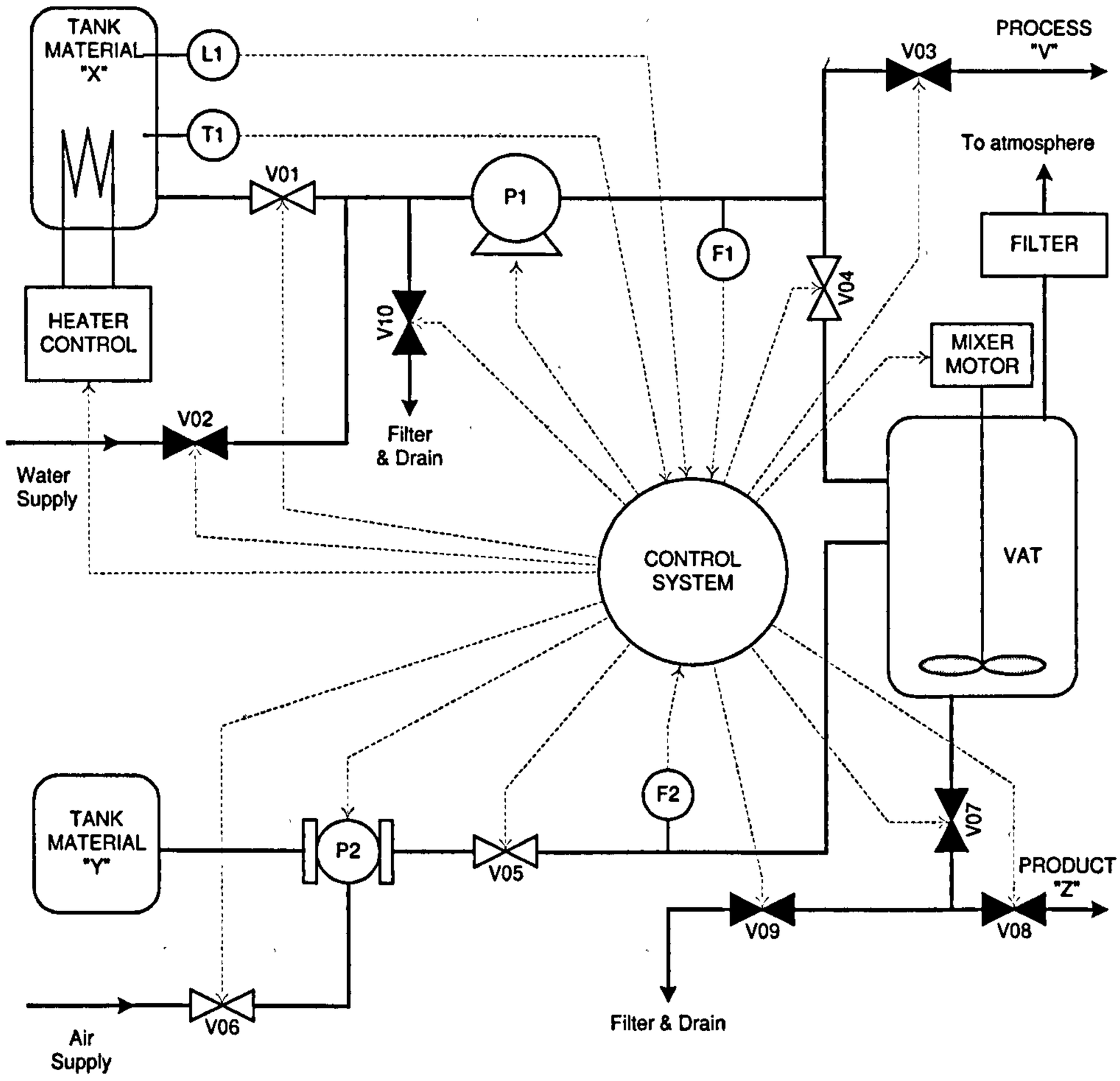


Figure 5 - Unsafe process plant concept

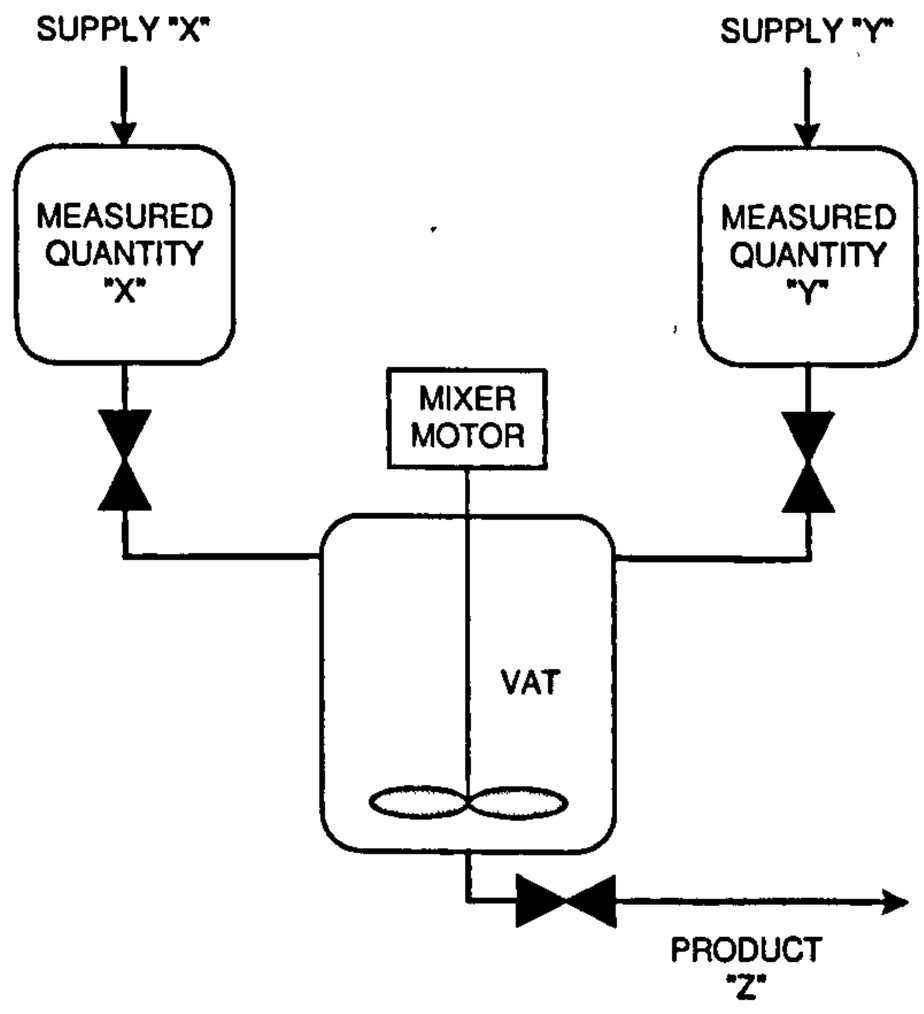


Figure 6 - Safer mixer concept

2.7 Functional Failure Analysis (FFA)

Functional Failure Analysis (FFA) is a predictive safety analysis technique that is very widely applied in the Aerospace industry. Closely related techniques are also common in both the automotive and rail industries but FFA is rarely identified as a distinct technique in the safety literature. The best description identified is that given Appendix A of ARP 4761 [71] (where it is called System Functional Hazard Assessment). The reason for this apparent neglect is that most authors have considered FFA to be a member (or variant) of the Failure Modes and Effects Analysis (FMEA) family of techniques (c.f. section 2.9). Indeed, in the automotive industry, it is usually known as *predictive FMEA*, but the important "predictive" label is frequently dropped, and the technique loses its unique identity.

It is considered separately here because, although a tabular, inductive event analysis like FMEA, FFA's method, and its role in the process are quite distinct. FFA is a *predictive* technique, which should be applied early in the development of a design to help identify and refine safety related requirements. In terms of the "V" safety lifecycle model discussed in section 2.2, this technique is a candidate for use in the hazard identification and preliminary system safety analysis phases.

FFA is structured around the *functions*, or active components, of a system. For each function (identified from a suitable design representation), the analyst considers the effects of three broad hypothetical failures types:

- Function not provided;
- Function provided when not required;
- Function provided incorrectly.

Each of these failure types is intended to act as a prompt, and it may be necessary to consider more than one interpretation of each. For example, the first prompt (function not provided) is easy to interpret for responsive functions (such as brake operation on a vehicle in response to the driver's pedal input). However, in the context of a continuous or periodic function (such as evaluation of a control law in an iterated control loop) there are numerous different cases to consider, such as single transient loss, repeated (possibly periodic) transient losses, or complete (permanent) loss. These different cases may have quite different effects.

Similarly, "function provided when not required" could prompt consideration of spurious events (e.g. uncommanded application of brakes), unwanted repetition of an intended event, excessively rapid iteration etc. Interestingly, there are functions to which this failure type does not apply;

these are continuous functions which must always be available; for example, maintaining minimum separation of aircraft in an air traffic control system. The third failure type (incorrect provision) is an awkward "catch all", which covers a wide range of possible erroneous behaviour, and must be interpreted very carefully in each new context. Possible interpretations include incorrect timing (either start time or duration) of the function, incompleteness, asymmetry or unwanted side effects.

The basic procedure for FFA is:

- Identify functions.
- For each function identified, suggest failure modes, using the three failure types as prompts.
- For each failure mode, consider the effects (at different levels, e.g. function / subsystem / system, if necessary).
- Identify and record any actions necessary to improve the design.

Table 2 shows a fragment of a basic FFA for the vehicle speed sensor example.

Function	**Failure mode** a) not provided b) provided when not required c) provided incorrectly	**Effects**	**Comments and Recommendations**
Vehicle speed sensor	a) No vehicle speed data provided to engine management system	Vehicle speed used in alternative load calculations and crank speed check routines. Effect minimal, and restricted to loss of some error detection capability.	No significant safety effect.
	a) No vehicle speed data provided to gearbox controller	Vehicle speed is primary input to gear selection, including kick-down. Effect is significant reduction in accuracy and timeliness of gear shifts. May impair driveability of vehicle, especially in start-stop traffic.	Potentially hazardous. Driver warning essential; loss of speed display on dashboard is not sufficient. Design must incorporate detection (via comparison with engine speed / load data?) and illumination of transmission warning light.
	b) No meaningful interpretation	N/A	Both controllers sample VS data when needed, therefore data cannot be provided when not required.
	c) Inaccurate vehicle speed data provided to engine management system	Effect minimal, and restricted to loss of some error detection capability.	No significant safety effect.
	c) Inaccurate vehicle speed data provided to electronically-controlled automatic gearbox management system	Effect is unpredictable. May result in unwanted gear shifts or selection of inappropriate gear. May impair driveability of vehicle, especially in start-stop traffic.	Potentially hazardous. Depending on degree of inaccuracy, failure mode may be hard / impossible to detect. Investigate acceptability of failure rates of specified components.

Table 2 - Fragment of FFA output for the vehicle speed sensor example

FFA can be applied at any level of design development; ideally, it should initially be applied at the highest possible level, and the results propagated down as part of the requirements / rationale for more detailed design.

There are a number of more developed variants of this technique. For example, the ARP 4761 method description requires the analyst to consider, in addition to the above, the flight phase in which the failure occurs, environmental factors (such as severe weather), loss of warnings to the flight crew, and abnormal or emergency conditions (e.g. loss of hydraulic power) which might contribute to the effects of the failure. The failure effects are also classified (qualitatively) by severity and a target probability, selected by severity of effects from a predetermined table, is assigned. These additional steps make the technique more complex, introducing issues of coincident and common cause / common mode failures, but also serve to emphasise the role of the technique as an aid to design development and identifying safety requirements.

Another innovation of the process suggested in the ARP, more evident from the contiguous example in Appendix L than from the method description in Appendix A, is the application of FFA to function hierarchy trees, such as the example shown in Figure 7. One of the potential advantages of this approach is that this sort of hierarchy could potentially be devised very early in the development process, allowing safety analysis to be started much earlier.

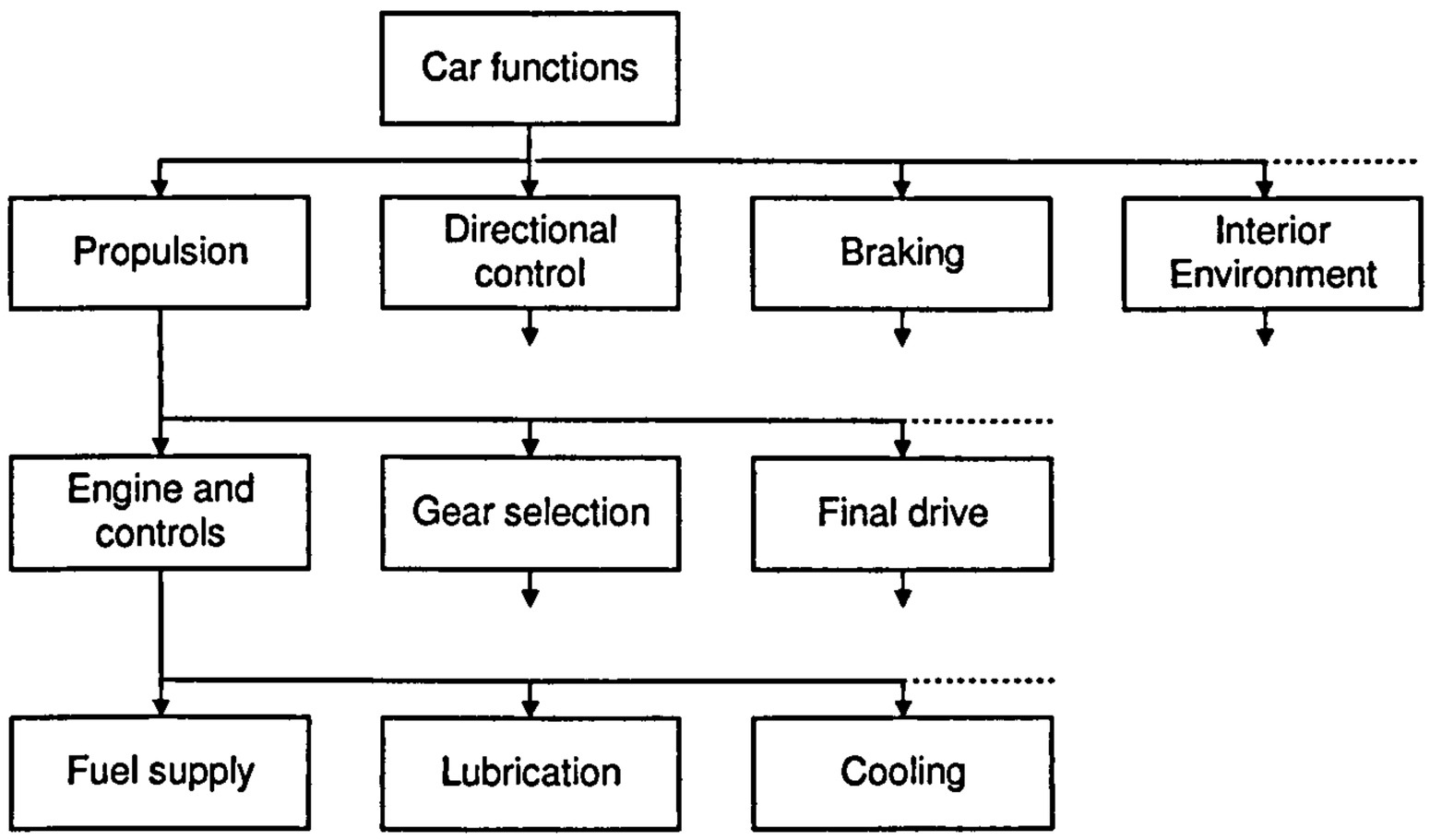


Figure 7 - Partial function hierarchy tree for a car

2.8 HAZard and Operability Studies (HAZOP)

Hazard and Operability Studies (HAZOP) is a predictive safety analysis technique which was developed within ICI in the mid 1960s specifically to help assess and refine the design of new process plant. It is now widely used in the chemical (especially petrochemical) industry, and has also been widely applied to nuclear and food-processing applications. The Chemical Industries' Association guidelines [17] describe the fundamentals of the technique, and Kletz [44] provides a short tutorial introduction. There are many publications (mostly in process industry journals such as *Hydrocarbon Processing* and *Chemical Engineering*) describing experiences applying the technique and suggesting improvements; practical examples can also be found in a number of accident reports from the Health and Safety Executive (e.g. [34]).

HAZOP is described as a technique of *imaginative anticipation* of hazards and operating problems. It is unique among the techniques reviewed here in that it is strictly a team analysis activity, and guidelines on its application typically devote as much space to describing the composition of the study team, and to appropriate working practices, as they do to describing the steps of the method.

The HAZOP team normally consists of about six people, representing a number of interest groups. For a standard HAZOP of a new plant design, these include:

- The *plant design engineers* whose primary role in the meetings is to explain their design. Where changes are considered necessary, the designers' role may also include advising the meeting about cost / benefit trade-offs of various options; they will also eventually be responsible for producing revised designs to satisfy the meeting's recommendations.
- *Commissioning or installation engineers* are expected to help resolve practical issues relating to the siting, structure and construction of the plant. With experience of the problems of startup and initial operation of new plant, they also have a key role in suggesting possible problems in the proposed control and safety systems.
- *Operators and maintenance engineers* may have either a questioning or explanatory role in the meetings, exploring whether the proposed design meets operating requirements, and what provision has been made for the safe management of unusual circumstances such as shutdown and maintenance.
- One or more *independent experts*, who have broad knowledge of process chemistry and engineering but who have not previously been closely involved with the project. Their main role in the HAZOP meetings is to suggest and explore possible deviations from the intended

behaviour of the plant, and to apply their experience to identify and suggest alternatives to any undesirable features of the design.

The study meetings are chaired by the HAZOP *leader*, who has overall responsibility for the conduct of the analysis. This responsibility normally extends to the selection of team members and practical administration of organising the study meetings; it also includes ensuring that the necessary follow-up work, such as answering queries or ensuring that recommendations are implemented after the meeting. The discussion and conclusions of the meeting are documented by the *recorder* or secretary. The person selected for this role must have sufficient understanding of process chemistry that they can follow and accurately minute the discussion, but different texts disagree about whether the recorder should be expected to take an active part in the debate.

Whilst HAZOP can be applied to an existing system, either for retrospective hazard identification or when modifications necessitate new safety assessments, it is really intended for use during design of a new plant. As such, the most important output of the analysis is really the set of recommendations made for amendments and improvements to the design. If the team finds that they do not have sufficient information to reach a decision in the initial meeting, it is acceptable to note queries for resolution and adjourn the meeting.

HAZOP is normally applied as soon as the process line diagram (also known as the piping and instrumentation, or P&I diagram) has been produced. For large, complex plants, where HAZOP of the complete line diagram is expected to be lengthy, HAZOP of the flow sheet (the order of processes and reactions, the first step in designing a plant to manufacture a particular product) is advocated. This can help to identify the most critical reactions, dangerous intermediate products etc. For example, the methyl isocyanate (MIC) which escaped in large quantities from the Union Carbide plant at Bhopal in India in an accident on 3rd December 1984, killing over 1750 people, was an intermediate product in the production of pesticides and polyurethane. These products can be manufactured from the same raw materials without the intermediate MIC production if the order of reactions is altered. Even where dangerous reagents or intermediate products are unavoidable, plants can be designed to minimise the inventory if the dangers are properly understood at a sufficiently early stage.

Unlike the majority of safety analysis techniques which concentrate on *component* failure modes, HAZOP analysis usually starts by considering the behaviour of the *flows* between components, studying *physical properties* such as the pressure, temperature and flow rates of materials in the pipelines within the plant. A set of *guide words* is used to prompt consideration of *deviations*

from the intended behaviour of these flows. These guide words provide the structure of the analysis and can help to ensure complete coverage of the possible failure modes. The set of guide words used in the process industry is now fairly stable, generally accepted and well understood, though there are still some minor variations. The standard guide words for process plant analysis (with example interpretations) are shown in Table 3.

Guide Word	Deviation	Example Interpretation
NO or NONE	No part of the intention is achieved	No forward flow when there should be.
MORE	Quantitative increase in a physical property (rate or total quantity)	Higher pressure, flow rate, temperature... Quantity of material is too large.
LESS	Quantitative decrease in a physical property (rate or total quantity)	Lower pressure, flow rate, temperature... Quantity of material is too small.
MORE THAN or AS WELL AS	All intentions achieved, but with additional effects (qualitative increase)	Impurities in flow (air, water, oil...) Chemicals present in more than one phase (vapour, solid)
PART OF	Only some of the intention is achieved (qualitative decrease)	One or more components of mixture missing, or ratio of components is incorrect
OTHER THAN	A result other than the intention is achieved	Any state other than normal operation, e.g. startup, shutdown, maintenance...
REVERSE	The exact opposite of the intention is achieved	Reverse flow.

Table 3 - Process HAZOP guide words

Compared to Functional Failure Analysis (section 2.7), it can be seen that HAZOP has more guide words than FFA has hypothetical failure types, but there is a strong correspondence between the two. "None" is analogous to FFA's "function not provided", whilst "function provided when not required" is related to interpretations of "More" and "As well as". The larger number of guide words in HAZOP provide more guidance than FFA but, like FFA, HAZOP provides a "catch-all" ("Other than") which requires careful interpretation to identify situations not prompted by the rest of the guide words.

For each deviation suggested, the team search for possible causes, such as inappropriate design, or component failures, and consider what the consequences for the plant would be if the deviation were to occur. The method may therefore be considered to have both inductive and deductive characteristics. The scope of the analysis explicitly extends to system operation, maintenance and testing, and includes "unusual" operating circumstances such as startup and shutdown. Although

analysis starts with single deviations, the team is expected to consider additional causal factors when investigating their potential effects.

The essential steps of a HAZOP analysis are:

1. Select a flow in a pipeline.
2. Identify important physical attributes of the flow, such as *pressure*, *temperature*, *flow rate*, *chemical composition* etc.
3. Consider the deviations prompted by applying each guide word to each property for this line section.
4. Determine the possible causes of each of these deviations.
5. Investigate the expected outcome (effect on the plant) of each deviation, taking into consideration operating conditions and other causal factors where necessary, and examining the contribution of protection mechanisms and other mitigation already included in the design.
6. Decide which deviations are safety problems (i.e. those with both plausible causes and hazardous effects).
7. For deviations which are not safety problems, record a justification (i.e. explain why the design is acceptable as proposed.)
8. Consider changes to the plant that will remove, or reduce the probability or severity of, hazardous deviations.
9. Determine whether the cost of the proposed changes is justified.
10. Agree actions and responsibilities.
11. Repeat steps 1 to 10 for all other lines in the plant.
12. Follow up to ensure necessary actions have been taken.

The results of the HAZOP study are presented in tabular form. Figure 8 shows a small segment of a process line diagram, adapted from Kletz [44], and a fragment of analysis output for this example, also from Kletz, is given in Table 4. Two styles of recording HAZOP are identified in the literature. *Recording by exception* lists only hazardous deviations. This style of recording is no longer considered appropriate for the majority of studies, and has largely been supplanted by *complete recording*, which includes every deviation discussed, with a justification of acceptability as described in step 7 above for deviations that are not considered to be safety problems. This provides a complete, traceable record of the team's conclusions.

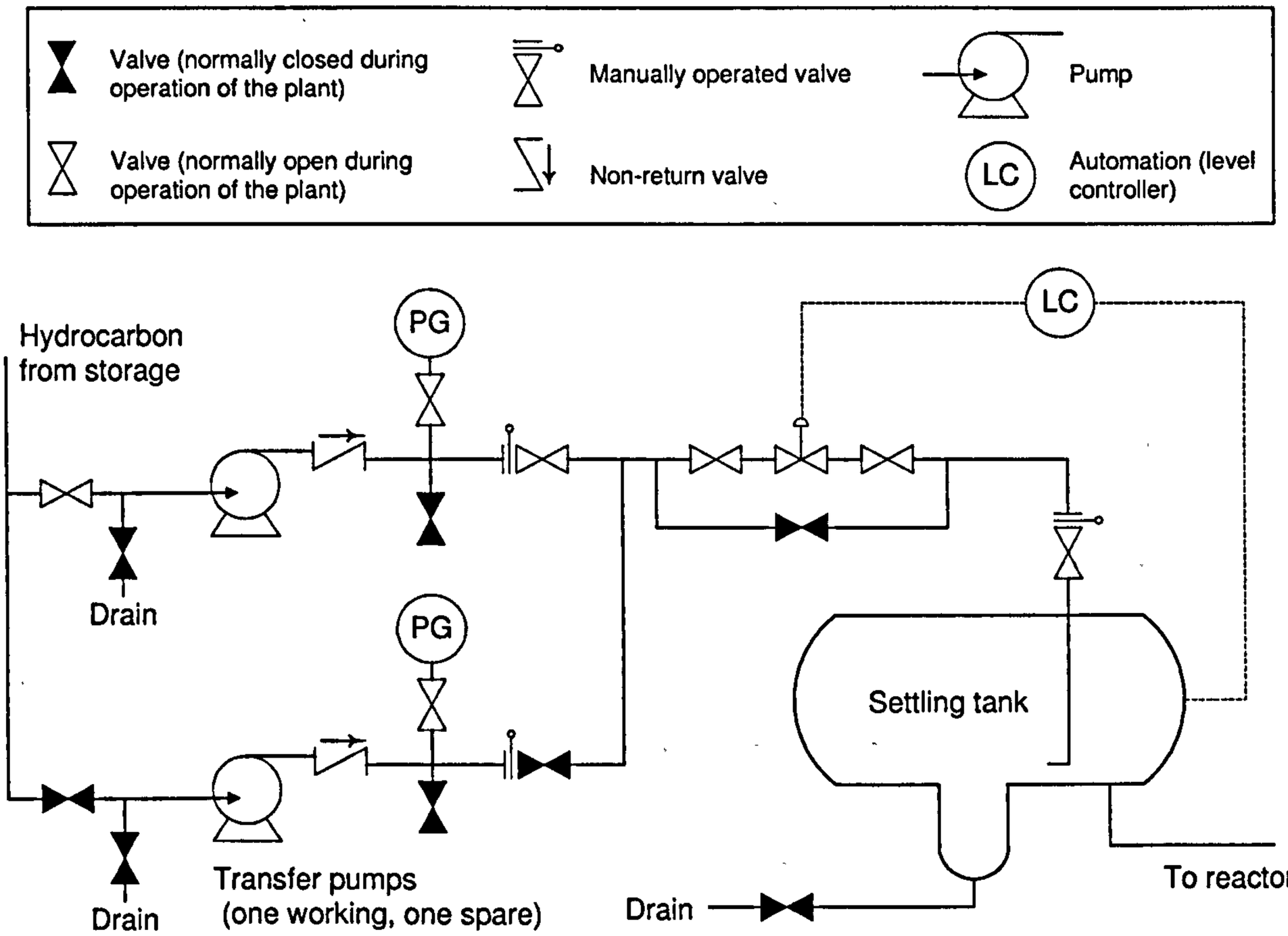


Figure 8 - Fragment of Piping and Instrumentation (P&I) diagram, adapted from Kletz

Guide Word	Deviation	Possible Causes	Consequences	Action Required
NONE	No flow	No hydrocarbon available from storage	Loss of feed to reactor. Polymer formed in heat exchanger	1) Ensure good communication with storage area 2) Install low level alarm on settling tank
		Transfer pump fails (motor fault, loss of power, impeller corroded etc.)	As above	Covered by 2)
MORE	More flow	Level control valve fails to open, or LCV bypassed in error	Settling tank overfills	3) Install high level alarm 4) Check size of overflow 5) Establish locking-off procedure for LCV bypass when not in use
	More pressure	Isolation valve or LCV closed when pump running	Line subjected to full pump pressure	6) Install kickback on pumps
	More temperature	High intermediate storage temperature	Higher pressure in transfer line and settling tank	7) Install warning of high temperature at intermediate storage

Table 4 - Fragment of HAZOP output

Like FFA, HAZOP does alone does not provide detailed evidence of the safety properties of a system; rather, it provides input to the design process, and a structure which can subsequently be used as the framework of a safety argument.

Although HAZOP has historically been used almost exclusively in the chemical processing and related industries, its concepts are very general, and it can readily be adapted for other applications. In response to the wider application of HAZOP, a number of new standards have been published or are under discussion, such as the IEC Guide for Hazard and Operability Studies [37]. Proposals for the application of HAZOP to computer systems are discussed in section 4.2, and Chapter 7 describes further work modifying and extending this technique.

There are also a number of tools such as HAZOPEX and STARS [45] which use expert systems to produce automated HAZOPs directly from a machine-readable design representation.

2.9 Failure Modes and Effects Analysis

Failure Modes and Effects Analysis (FMEA) is the most widely used name for a whole family of related inductive safety analyses. Villemeur [80] cites papers dealing with FMEA applications in the aerospace, nuclear, chemical and automobile industries, and Dhillon's comprehensive bibliography [22] lists around 50 different procedures or standards for applying FMEA. Apart from the inductive working, the major common feature of these techniques is that they start from *known* failure modes of individual components (unlike FFA or *predictive FMEA* discussed in section 2.7, for which the starting point is a set of *hypothetical* failure modes, often at a relatively high level).

Unfortunately, the huge range of analyses with quite different procedures and intent which are all known as FMEA mean that there is little common understanding between different industries and nationalities. Leveson has claimed that FMEA is essentially a reliability analysis that is inappropriate for a use in a safety process – a reasonable assertion for some variants of FMEA, such as the purely quantitative technique described in the introduction to the U.S. Nuclear Regulatory Commission Fault Tree Handbook [69].

There are, however, a number of variants of the technique that are appropriate for use in a safety process; Villemeur's examples are clearly safety rather than reliability analyses. Again, the Appendices to ARP 4761 provide some of the best published descriptions and examples.

The most important feature of an FMEA carried out as part of a safety process is that it is a confirmatory analysis, providing evidence that a system meets requirements that were set at an earlier stage in the process. The related failure modes, effects and criticality analysis (FMECA) is an extension of FMEA which makes provision for a formal categorisation of the severity, probability and resultant risk classification of each failure based on its contribution to system level hazards.

The results of the analysis are simply presented in tabular form; there is no special notation or standard format for FMEA in general use. This is largely due to the diversity of systems in many different industries to which it is applied, and can be seen as a strength of the technique, because it can be tailored to suit individual needs. There are, however, common concepts that will be expressed in some way in every (safety) FMEA. These include identification of:

- the component or subsystem under consideration
- the *known* failure modes of this component or subsystem
- the effects of each failure mode.

Additional columns that may be presented in the table as required include:

- contributing factors, such as system state, which may alter the effects of the failure
- means of detecting and mitigating the failure
- comments and / or recommendations

If a FMECA is being carried out, there must also be columns for the criticality information, i.e.\ severity, probability and risk class. There will normally also be a column containing a justification of the risk category assigned.

The precise method for carrying out an FMEA or FMECA varies from one application to another. Generally, there are five major steps

1. Definition of the system, its components and operating states
2. Identifying the component failure modes
3. Identifying the possible effects of each component failure
4. Investigating other factors, such as detection and protection
5. Forming conclusions and making recommendations

Table 5 contains a fragment of an FMEA for the vehicle speed sensor example.

Component	Failure Mode	Subsystem Effects	Vehicle Effects	Comments
Variable Reluctance Sensor	No signal	Vehicle speed will always be calculated as zero	1) No speed indication 2) Mileometer not incremented 3) Electronic gearbox control may select too low gear, possibly resulting in wheel lockup or transmission damage	Effect 3) requires simultaneous failure of engine load calculation and mechanical interlocks on gearbox
Variable Reluctance Sensor	Noisy (too many edges)	Calculated vehicle speed will be too high. If edges arrive at higher rate than spec., they will be lost.	4) Indicated speed greater than actual 5) Mileometer over-reads 6) Electronic gearbox control may select too high gear, possible resulting in stall	Effect 6) is hard to detect via engine load calculation, unless noise is extreme.
Variable Reluctance Sensor	Intermittent	Calculated vehicle speed will be too low	7) Speed indicated lower than actual 8) Mileometer under-reads 9) as 3)	See above

Table 5 - Fragment of FMEA for the vehicle speed sensor example

2.10 Fault Trees

Fault Tree Analysis was developed jointly by Bell Laboratories and the United States Air Force in the early 1960s to investigate the conditions that might lead to inadvertent launch of a Minuteman missile. It has become by far the most commonly used deductive safety analysis technique, and has been applied in many diverse fields.

A fault tree represents graphically the combinations of events and conditions that contribute to the occurrence of a single undesirable event, called the *top event*. Despite the diversity of application, the symbols representing events, conditions and logical operators have remained remarkably standard, and most of the safety critical systems community still regards the U.S. Nuclear Regulatory Commission Fault Tree Handbook [69] as the definition of standard fault trees. The few significant deviations from the conventions defined in the handbook have remained thankfully obscure and largely unadopted. The standard fault tree event and gate symbols are shown in Figure 9. This is not an exhaustive list; Villemeur's book [80] defines

additional logic gates (e.g. an *m* out of *n* combination gate) and a family of numerical gates, consisting of *quantification*, *summation* and *comparison* gates.

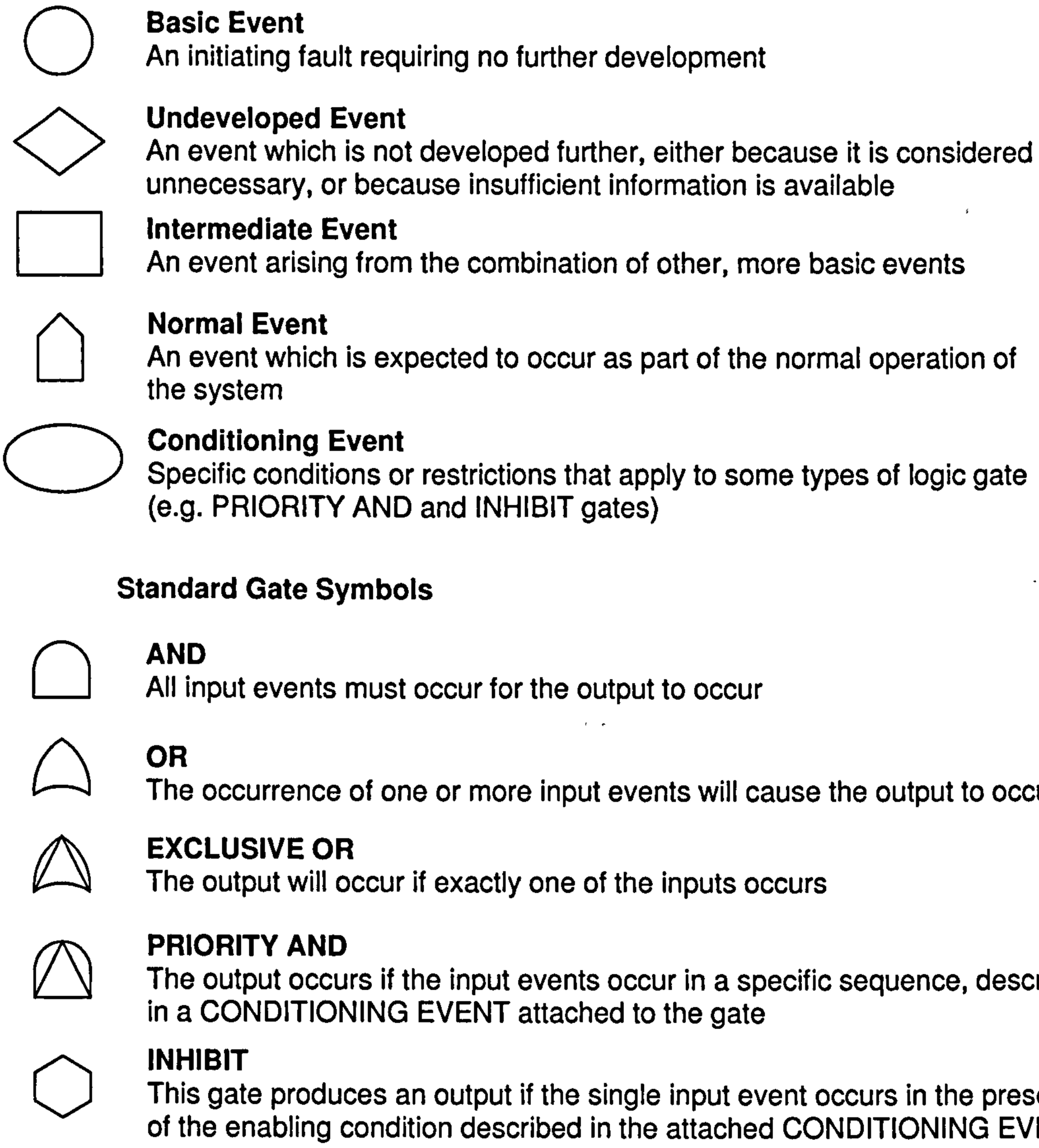


Figure 9 - Standard Fault Tree notation

The method for constructing a fault tree is generally expressed not as a rigid algorithm, but rather as a set of rules that the analyst should follow. The analysis begins by selecting the undesirable event that will form the top event of the tree. The following rules are then applied:

Identification of *immediate cause*:

The immediate, necessary and sufficient causes of the top event are found. It is important to ensure that the true immediate events are identified. For example, in the vehicle speed sensor system, the immediate cause of "no input data to vehicle speed functions" would be "no output from signal processing electronics"; the temptation to jump directly to the obvious cause (i.e. sensor failure) must be avoided.

Classification of events:

Each contributing event identified should be classified as either a *basic event*, in which case no further decomposition is necessary, a *system defect* or a *component defect*. A system defect is a failure involving more than one component, or a component failure together with a normally occurring event. In this case, the immediate causes of this event must be sought.

Analysis of component defects:

Component defects are divided into *primary*, *secondary* and *command* failures. Primary failures are basic events; secondary and command failures will generally require further decomposition. Figure 10 shows a decomposition of component failure causes suggested by Villemeur.

These three rules are applied iteratively until the tree consists entirely of basic events, or until the desired level of detail has been reached.

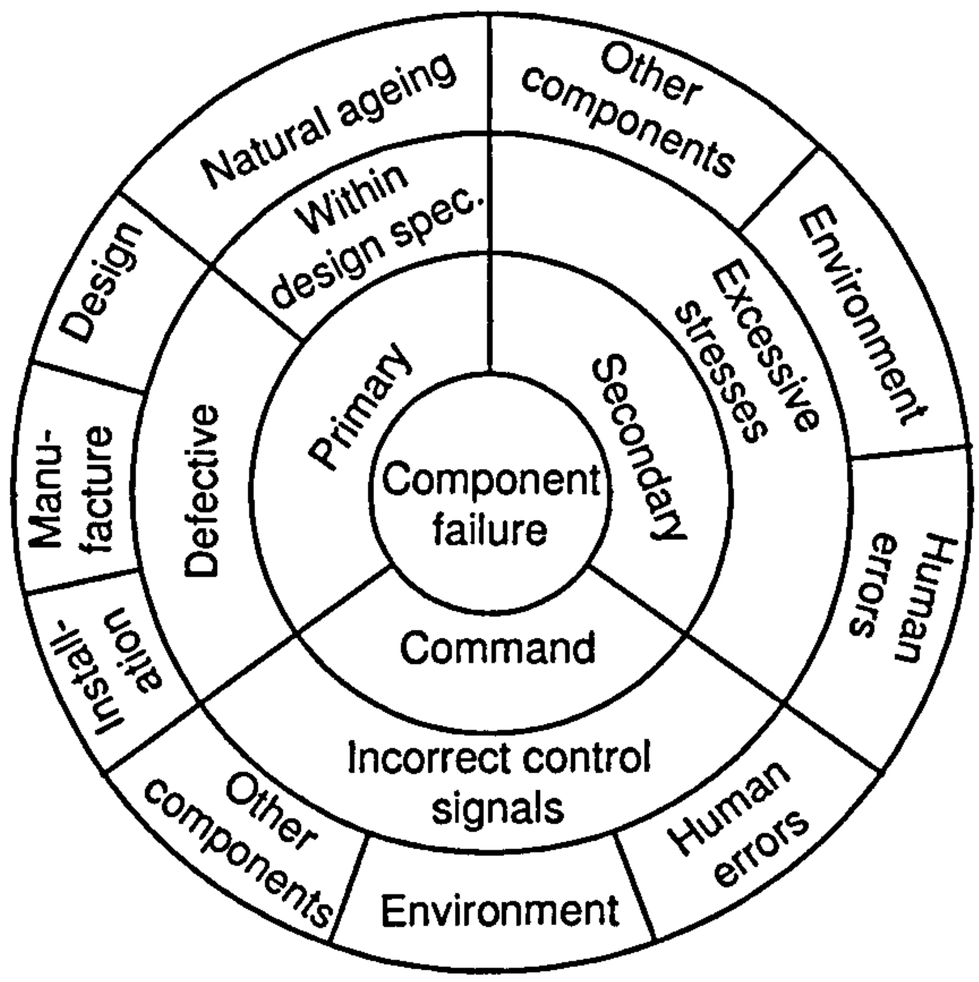


Figure 10 - Villemeur's decomposition of component failure causes

Further guidelines for the construction of correct fault trees include:

- To ensure a methodical analysis, all the inputs to a gate should be defined before any one of them is examined in detail.
- The output of one logic gate should never directly form an input to another gate. There must always be an identified intermediate event between logic gates.
- The statements entered in the event boxes should always be a complete description of *what* the fault is and when it occurs
- Miracles are not permitted. If a component failure is sufficient to cause the top event if all other components in the system behave normally, but other coincident failures could mask the critical failure, then these coincident failures must be assumed not to arise.
- Causes always chronologically precede consequences. This sounds obvious, but can be difficult to apply in the analysis of closed loop control systems

Figure 11 shows a fault tree for the failure event "No vehicle speed data supplied to gearbox controller" in the vehicle speed sensor example, constructed following the rules given above. Note that, as there is no redundancy in this system, the tree consists entirely of "OR" gates.

One of the most important features of the method is its suitability for use in calculating probabilities. The first step is always the reduction of the fault tree to *minimal cut-set form.* Minimal cut-sets are the smallest sets of events which are sufficient to cause the top event. The speed sensor example tree is already minimal – no event appears more than once in the tree. Figure 12 shows the reduction of a small example tree to its minimal form.

The derivation of minimal cut sets is followed by annotating each basic event with its probability. Calculation of intermediate event probabilities then progresses up the tree until the probability of the top event can be calculated.

Fault trees can also be used in a predictive role in the PSSA activity. In this role, the tree is constructed as normal, working down from the top event. However, probabilistic safety requirements are propagated down the tree, starting with the target figure for the top event and deciding how to allocate this between the events at each successive level of the tree. This can also help to determine architectural requirements; for example, an allocated failure rate of 10^{-7} per hour for an electronic system would imply a need to use redundant components.

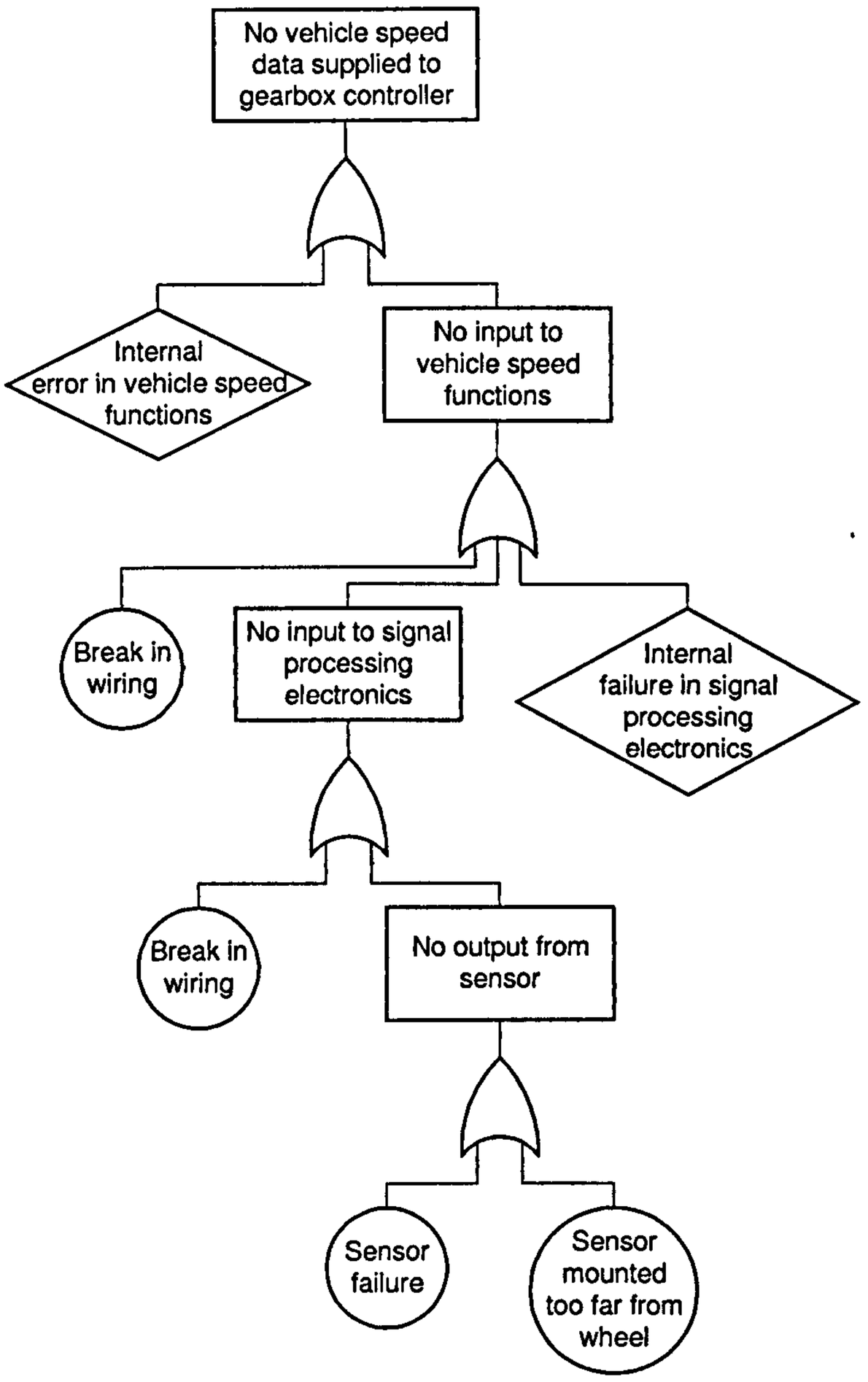


Figure 11 - Fault tree for "No vehicle speed data supplied to gearbox controller"

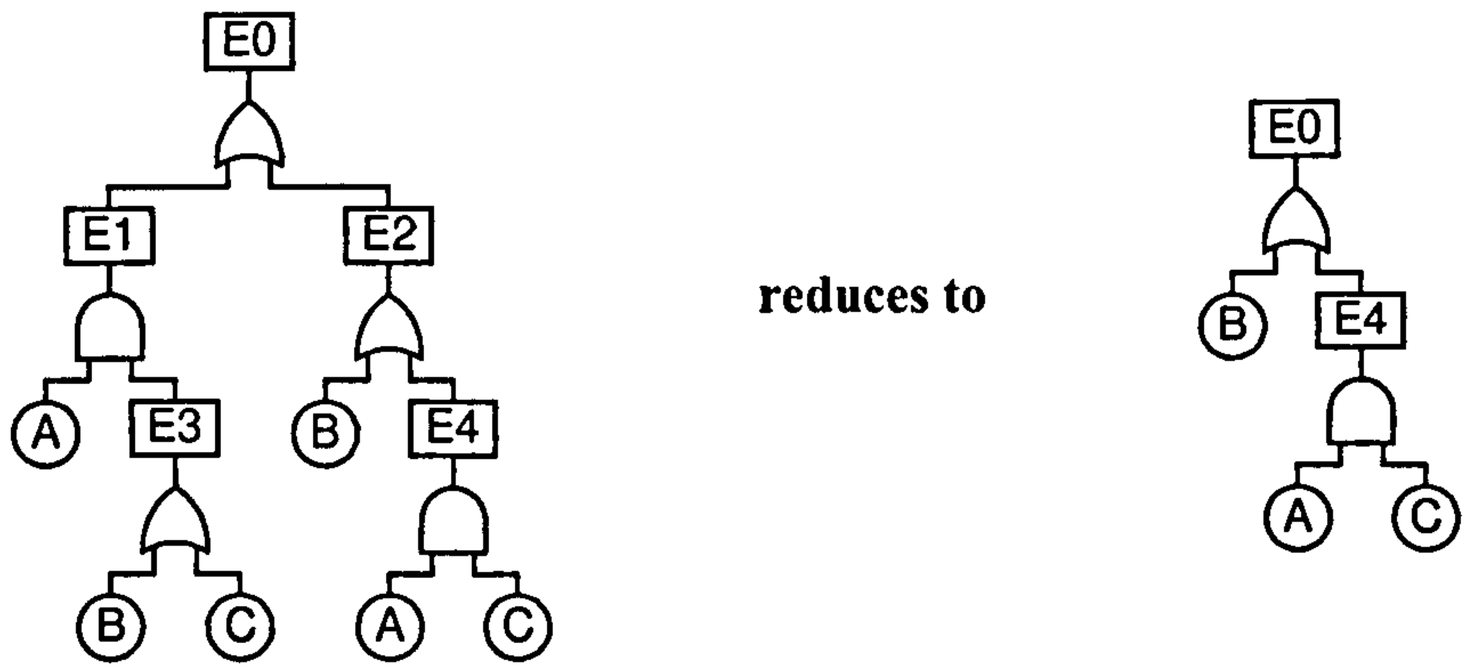


Figure 12 - Simple example of minimal cut-sets

2.11 Sneak Analysis

Sneak analysis originated with an accident involving a Mercury Redstone rocket in 1961. The rocket was connected to the control centre by an umbilical cable attached by tail plugs. When the rocket was launched, the tail plugs disconnected in an order that briefly established a circuit through the abort coil, with the result that the rocket crashed back onto the launch pad.

Rankin [66] and other Boeing engineers developed sneak *circuit* analysis to systematise the search for similar design flaws in other electrical and electronic systems. Their method was very successful within the Boeing Company, which trained large numbers of people in the use of the technique, and applied it not only to its own products but also used it on external contracts for NASA and some nuclear and military systems projects. Variants, usually known as sneak *path* analysis, were also developed for use on hydraulic and pneumatic systems.

The original principle of sneak circuit analysis was the identification of *unintended* circuits, but the technique was rapidly expanded to identify a much wider range of unintended situations. O'Connor [63] identifies five types of sneak conditions for electrical circuits:

1. *Sneak paths* – current flows along an unexpected route
2. *Sneak opens* – current does not flow along an intended route
3. *Sneak timing* – Current flows at an incorrect time, or does not flow at the correct time
4. *Sneak indications* – False or ambiguous indications
5. *Sneak labels* – False, ambiguous or incomplete labels on controls or indicators.

The identification of sneak paths is based around a systematic redrawing of the circuit diagram to make current flow patterns obvious. The principal objectives of the redrawing are to divide the circuit into partitions, where all components that are joined by a possible current path are grouped into the same partition, and to re-express the circuit within each partition such that its structure matches one of a set of standard patterns, or topograms. Note that partitions can always be established at a power supply or earth point. Any circuit can be represented by combinations of the patterns, which are shown in Figure 13. The L_i and S_i in the patterns represent loads and generalised switches, where a generalised switch is any component such as a contact, plug, relay etc. which can control the flow of current.

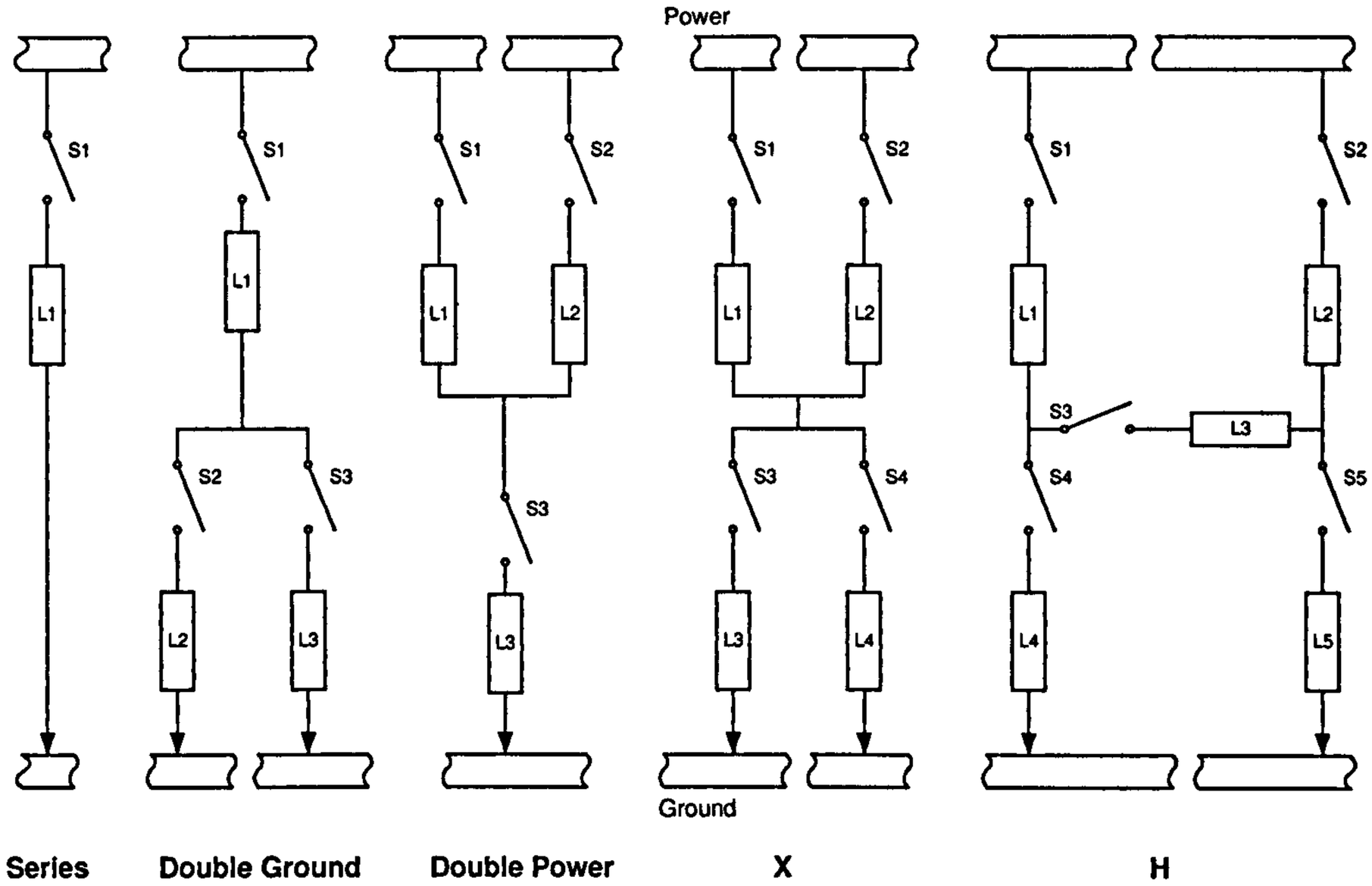


Figure 13 - Sneak circuit patterns

A series of sneak "clues" is then used to identify potential problems in these circuits. For example, clues for the double ground circuit include:

- *Design problems*, e.g. L3 desired but not L1
- *Switching / state problems*, e.g. S1 open and L2 desired
- *Load incompatabilities*, e.g. current flows in L2 bypassing L3, or current flows in L2 and L3 overload L1
- *Unwanted flows*, e.g. reverse flow through L3 to L2 (possible only if the output of L3 passes to another part of the circuit rather than to ground)
- *Interface problems*, e.g. label on S2 does not reflect the function of L2, or indication on L2 does not reflect the function of L2

The relationship between these clues and the general sneak conditions identified above is obvious. There are complete, systematic lists of clues for each of the circuit patterns, and guidance on identifying additional interactions (e.g. where the coil of a relay is load in one partition of the circuit and the contacts are a switch in a different partition).

Sneak *path* analysis (described by Taylor in [76]) generalises the principles of sneak circuit analysis to produce a technique that in many respects can be considered a generalisation of HAZOP. The power supply and earth of the electrical analysis are replaced by a general "source"

and "target". Potential paths from source to target are identified with the aid of sneak clues, such as:

- Unintended path from source to target
- Reverse flow along path
- Excess flow along path
- Intended path created at the wrong time
- Intended path can be cut off at the wrong time
- Intended flow can be diverted
- Two paths merge, or allow unintended interaction
- Person or object can move into flow path.

This effectively combines the investigation of unintended behaviour in expected paths (analogous to the identification of flow deviations in HAZOP) with a systematic search for completely unintended flows which could have similarly hazardous outcomes. The examples given in Taylor's book include conditions where sumps or drains form unexpected routes for chemical reagents to combine.

Taylor generalises the identification of sources and targets even further, such that they include:

Sources	**Targets**
• Energy	• Humans
• Poisons	• Fuels
• Corrosives	• Other chemical reagents
• Sharp edges	
• Chemical reagents	
• Ignition sources	
• Moving objects	

This generalisation appears to cast sneak analysis in very much the same role as preliminary hazard identification methods based on analysis of energy containment. References have also been found to software and human factors sneak analyses.

Frustratingly, this technique is subject to commercial restrictions, and there are very few publications that provide more than incidental detail. Those that do exist imply (as the observations above demonstrate) that sneak analysis has grown from its original focused method and objective to encompass a range of related analyses and has become, in effect, a safety process in itself.

2.12 Event Trees

Event trees are simply branching trees, drawn across the page from an *initiating event* on the left to the possible outcomes on the right. The centre of the page is divided into columns, with each column representing a possible subsequent event or action, such as the functioning of a protection mechanism. Each of these events can either occur or fail to occur; these cases are represented by a branching of the tree in the appropriate column. They can be used as a basis for calculating the probabilities of the possible outcomes if the probabilities of the intermediate events are known.

The construction of event trees is, essentially, an inductive process, proceeding from an assumed initiating event to a set of possible outcomes. The essential chart construction steps are:

1. Select an initiating event
2. Identify protection mechanisms which may have an effect after this initiating event
3. Determine the order in which the protection mechanisms should function
4. Construct the tree, and describe the interpretation of each of the possible outcomes.

Figure 14 shows an example of an event tree which might be constructed for the vehicle speed sensor subsystem. Note that the fault detection and handling mechanisms in the control software are represented as the first two columns of alternative circumstances.

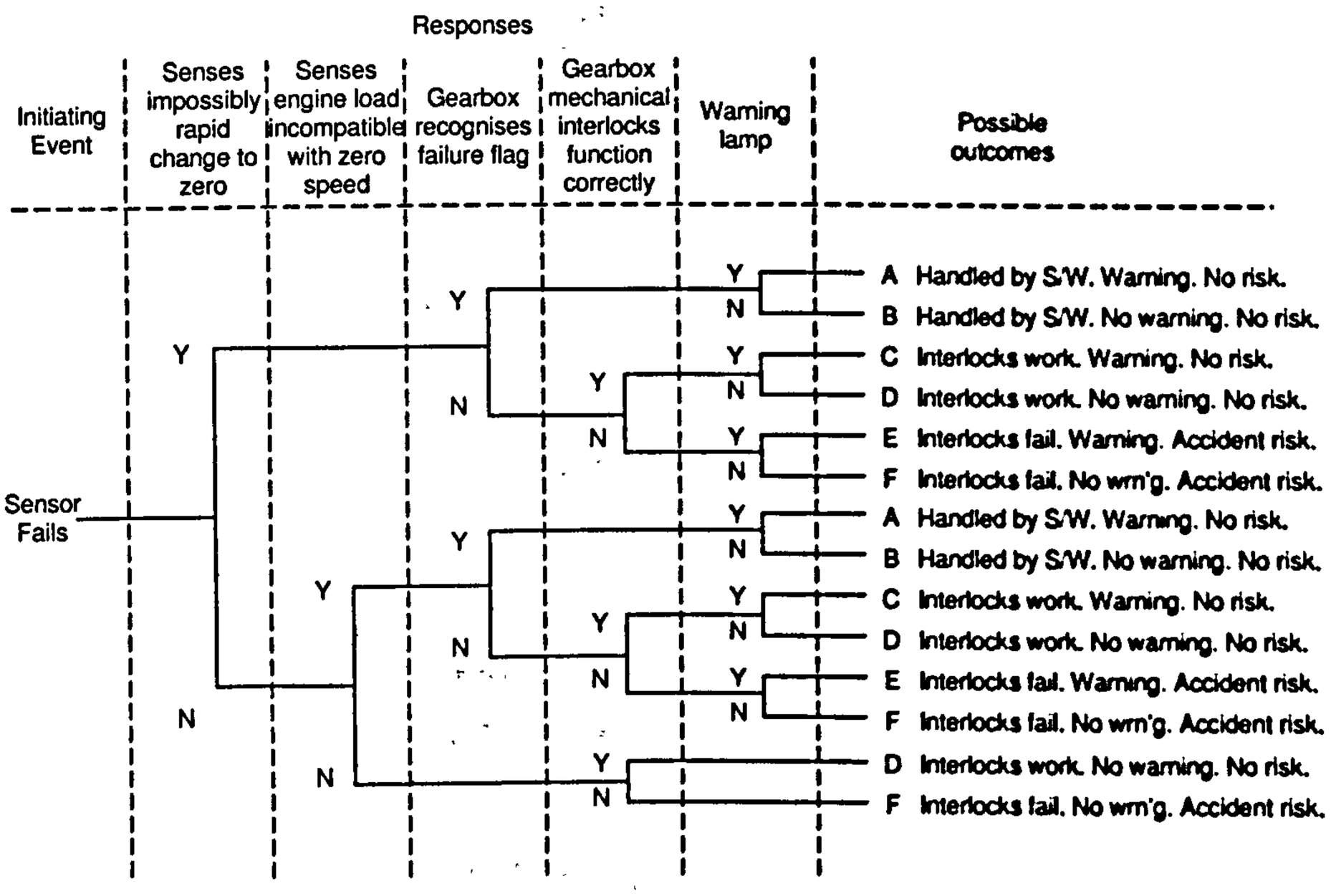
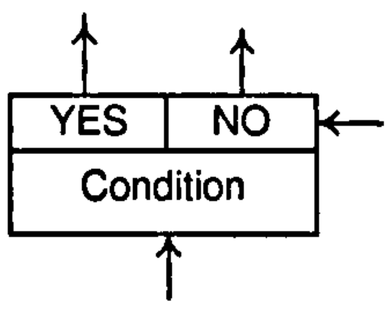


Figure 14 - Sample event tree for the vehicle speed sensor example

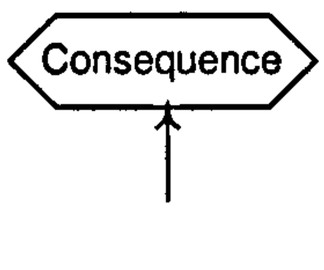
2.13 Cause-Consequence Analysis

Cause consequence diagrams were developed by researchers at the Danish national research laboratory (RISØ) in the early 1970s. Despite the apparent advantages of the method, it does not seem to have found widespread favour, and the only documented examples of its application are in the nuclear industry in Scandinavia [61] and the USA [11].

The method is perhaps the most comprehensive of the diagrammatic techniques surveyed, combining fault tree style causal analysis with a consequence analysis whose method, although not notation, is similar to that of event trees. The notations used in the literature for this method vary somewhat; all versions use standard fault tree notation for the causal parts of the analysis, and Figure 15 shows the notation most commonly used for representing the consequence parts.

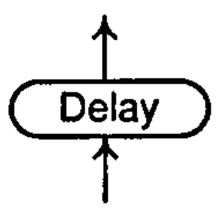


Decision Box
The output of this box is YES if the condition is met, NO otherwise. The condition is usually of the form "protective action X is taken?". The arrow entering NO is the output of a fault tree which shows what could prevent this action being taken



Consequence Event Descriptor
The terminal symbols of the diagram. Unlike fault tree top events, there may be many, each describing a possible outcome

Descriptor Tag
Used to label internal flows to show what event or state they represent



Delay
Introduces delay into an internal flow

Figure 15 - Symbols used for the consequence part of a Cause Consequence Diagram

The essential steps of a cause-consequence analysis are:

1. Identify one critical event, known as the *initiating event*
2. Identify the effects of this event on the system and its environment.
3. Determine whether *protective actions* should be triggered by the changes in the system or environment. These form the decision boxes of the consequence chart.
4. Consider the effects of these actions; note that an action which should be triggered may fail, or an action may be triggered which is actually undesirable in the context of the event which has actually occurred.
5. Display the results of steps 2 to 4 using the consequence notation.
6. Identify potential causes of the initiating event, and record the results in fault-tree notation.
7. Identify the potential causes of failure of the protective mechanisms; these are displayed as fault-trees whose top event is the "NO"' branch of a decision box.

Strict application of this procedure will result in a cause-consequence diagram which has only one initiating event; however, Burdick and Fussell [11] comment that one of the advantages of the method is that it can be worked either forward from events or backward from consequences, thus giving a good way of tracing and depicting combinations of events, causes and consequences. Probability calculations using a cause-consequence diagram are also possible; Nielsen et. al. [61] present a detailed worked example.

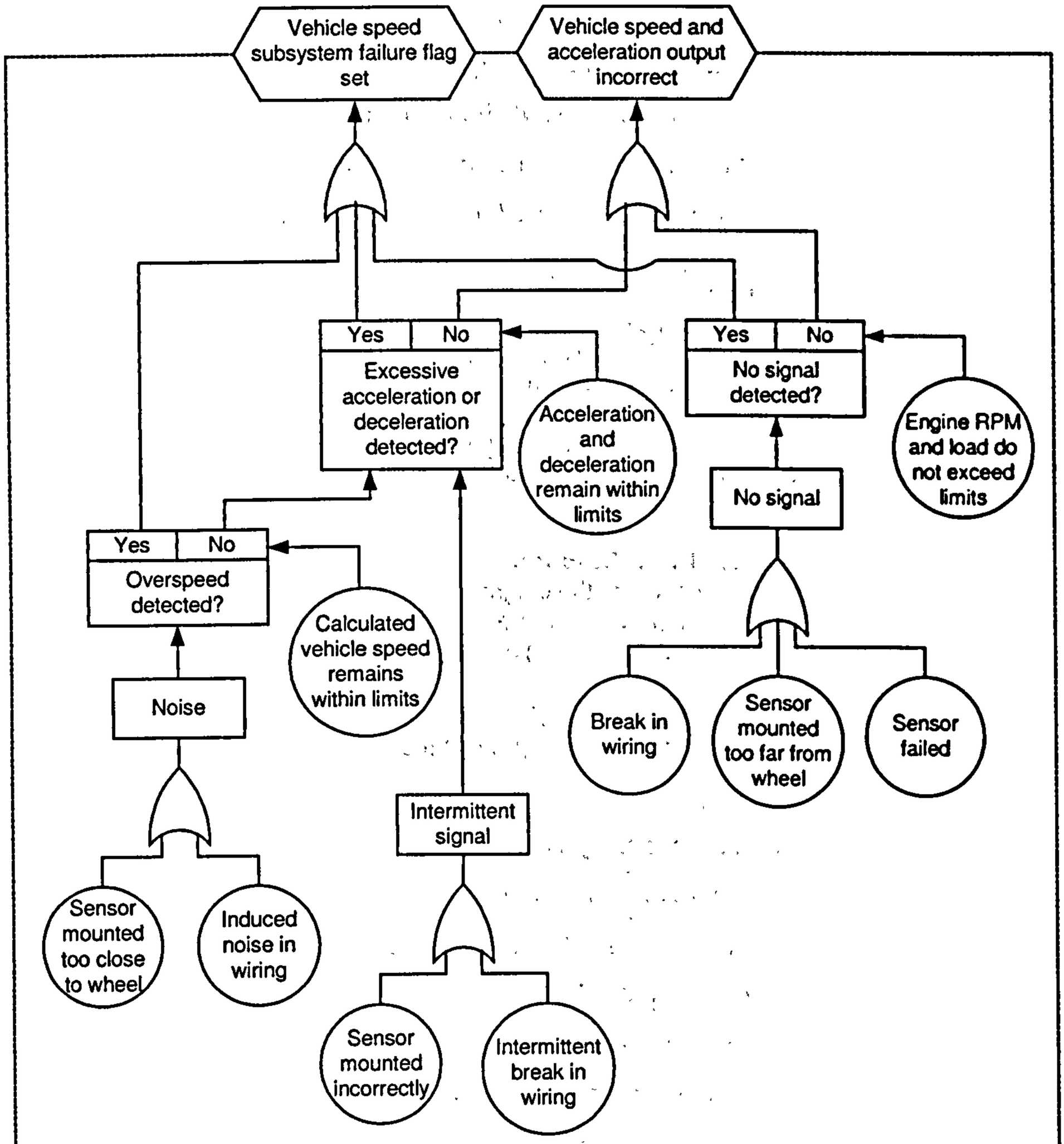


Figure 16 - Cause Consequence Diagram for the vehicle speed sensor example

Probably the greatest strengths of this method are its ability to represent multiple failure conditions and many possible outcomes, and the direct representation of fault detection

mechanisms and error handlers as decision boxes. This makes it much easier to comprehend a complex set of related failures and error handlers. The notation also naturally represents sequential and time-related events so, for example, multiple detection mechanisms can be represented in the order in which they are actually applied. Figure 16 shows an example cause-consequence chart for the speed sensor subsystem, which has been constructed to illustrate the interactions between different protection mechanisms that can be represented in the notation.

2.14 Zonal Hazard Analysis (ZHA)

Zonal Hazard Analysis is a technique that explores the ways in which physical proximity of components can result in common failure. Its primary purpose is to identify cases of non-independence between systems or technologies that had been assumed to be independent. For example, consider an aircraft that has hydraulically actuated control surfaces on the wings, and electrical sensors that report the position of the control surfaces to the flight control system. A hydraulic leak that occurs in close proximity to the sensor, or to its wiring, may cause a simultaneous loss of both control over the surface and information about the current position of the surface. This is potentially much more hazardous than simple loss of control over the surface, since without positional information the flight control system will be unable to take automatic action to trim the aircraft with compensatory movements of surfaces on the other wing.

There are many detailed procedures for carrying our ZHA on different types of system, but the essential steps are:

- Determine the zones.

  Zones reflect physical containment, for example the space between bulkheads in aircraft, or the interior of an equipment bay. The exterior of the platform is usually also treated as a zone.
- Identify the equipment in each zone, including services (cables and pipes) that pass through it.
- Consider causal factors. Typical examples of causal factors include:
  - clearance from moving parts
  - thermal heating & cooling, vibration, ionising & non-ionising radiation
  - existence of sharp edges and corners
  - stresses on pipes and cables
- Produce design recommendations to eliminate any problems found

2.15 Conclusions

This chapter has defined basic system safety terminology, outlined the safety activities in a typical development process, and briefly described the most important features of a range of analysis techniques. This review prompts a number of general observations about analysis techniques:

- It appears that the safety analysis techniques which are most successful (i.e. those which are widely applied across a range of industries) are those which have very well-defined methods, e.g. HAZOP and fault trees.
- Considering the list of techniques presented in the System Safety Analysis Handbook, it is obvious that many of them are extremely similar. "Families" of closely related technique, sharing very similar methods and notations can be identified. This appears to be due to subtle customisations of a small number of successful analysis techniques to fit specific requirements in different industries or organisations. This leads to the conclusion that an essential property of analysis techniques is that they must fit the way that people and organisations work, down to a very fine grain of detail.
- Even amongst the techniques surveyed here, which were deliberately selected to be as diverse as possible, a number of common concepts emerge. For example, almost every method includes somewhere a set of "prompts" to guide and structure analysts thinking.

These common themes provide strong clues to the basic elements of a successful analysis technique.

Chapter 3
A framework of concepts

This chapter explores the system safety definitions, processes and analysis techniques described in Chapter 2. It identifies a number of core concepts which underlie traditional system safety analysis, and which form a "vocabulary" for the design of new techniques to address novel analysis requirements. It also proposes a new model for classifying safety analysis techniques.

3.1 Hazards

Since hazard identification is the first technical activity in a system safety programme, and is the driver for all subsequent activities, it is essential that it be done well. Unfortunately, experience of participating in and reviewing hazard identification activities in industrial projects, and of teaching hazard identification as part of courses in system safety analysis, shows that many practising safety engineers struggle to understand or explain what a hazard is. This section explores the concept of a *hazard* in some detail, both to examine why it is problematic, and to show how it is key to identifying and understanding many of the properties of safety analysis techniques.

As Chapter 1 observed, even where accidents have occurred, it is common for the circumstances to be so complex that investigators cannot identify causes and event sequences with any certainty. The complexity and uniqueness of every accident means that it is impossible for the developers of a new system to envisage all the potential accidents it may cause or become involved in during its life. For example, it is easy to suggest potential failures of a car's brake system but, depending on the speed the car is travelling when the failure occurs, the location, traffic conditions etc., the same failure could result in any outcome from delayed travel to a written-off vehicle, or even death. Instead, most safety analysis is based around the identification and management of *hazards* – conditions that can be summarised crudely as "accidents waiting to happen".

One of the most common problems observed in hazard identification is the difficulty engineers experience in trying to separate the hazard from the potential accident. For example, in an exercise involving a level crossing, the majority of the participants initially identified "collision of train and road vehicle" as a hazard. This is an accident – the harm has been done. The hazards

are those situations that could lead to a train or a road vehicle entering, or remaining in, the crossing when they should not.

The British Standard [10] definition of a hazard is:

> *"A situation that could occur during the lifetime of a product, system or plant that has the potential for human injury, damage to property, damage to the environment or economic loss".*

There are many situations that can readily be identified as hazardous by this definition. For example oil spilled on a staircase, or loose cables trailing across a floor are obvious "slip" and "trip" hazards.

Unfortunately, in complex technical systems the concept of a hazard can often seem somewhat artificial, and this is the problem that the engineers involved in the level crossing exercise experienced. The fundamental problem is that there is effectively a chain of causality from root causes to accidents, and it is often not clear how far along the chain of events a situation arises that should properly be described as a hazard.

The classification of conditions as hazards is very often as much a decision about effective management as it is about technical safety aspects. Hazards are selected because they are situations which can be described clearly and concisely, where responsibility can be allocated, and which, taken together, break the safety engineering activities in a project into a reasonable number of manageable size tasks.

Even in existing systems or situations, the identification of hazards can be difficult, involving complex decisions based on many (often subtle and subjective) factors. For the safety engineer working on a new design the problem is worse, because the requirement is to *anticipate* (and hence avoid or mitigate) hazards. This distinction between *concrete* (real, certain) and *projected* (suggested or expected) situations is encountered at all levels of safety engineering, and is responsible for many instances of confusion or poor practice when the status of information is misunderstood. The distinction between concrete and projected conditions is explored further in section 3.4, which discusses the classification of safety analysis techniques.

It is impossible to summarise in a concise definition all of the knowledge and decision criteria required in determining whether a (concrete or projected) situation should be treated as a hazard.

However, examining some of the shortcomings of a number of definitions highlights some of the key concepts.

3.1.1 Endogenous and exogenous hazards

Leveson [50] makes a distinction between hazards arising from causes within a system (endogenous), and those arising from external threats (exogenous). Most of the discussion in sections 3.1.2 to 3.1.5 concentrates on characteristics of the hazards themselves, and on the way they are transformed into accidents, and these are unaffected by the source of the hazard. However, the distinction between endogenous and exogenous hazards becomes very important when examining the relationship between hazards and failures (section 3.2), and in the application of the hierarchy of risk reduction described in section 2.2.2.

3.1.2 Hazards as risk thresholds

The main problem with the British Standard definition given above is that, taken literally, it is too general. "A situation that *could* occur… that has the *potential* for (harm)" can be interpreted such that it includes almost any activity. For example, travelling in a car is clearly a situation that has the potential for harm, but most people in the developed world would not consider it to be a particularly hazardous activity – that is, the risk is considered acceptable (although there are many factors influencing perception and acceptance of risk which will not be explored here; for discussions of these factors see, for example [41, 62]). What is *implicit* in the definition is that hazards are *unintended* or *unwanted* situations in which the potential for harm exceeds some sort of *threshold* of acceptability. This may occur as a result of external influences (such as severe weather), technical failures, or operator errors. Thus most people would consider that skidding a car is hazardous; it is (usually) unwanted, and the potential for harm is much greater than normal driving.

In terms of the ALARP principle described in section 2.2.2, it is clear that the "broadly acceptable" region extends down to contain activities from which the risk is so low that they do not need to be considered as hazards at all. The hazard is the event or circumstance that causes the level of risk to cross the threshold from always acceptable to potentially unacceptable. The concept of thresholds of risk is important but problematical, as it relies on (predominantly subjective) assessments of acceptable risk, and it can be hard to determine precisely what caused the notional threshold to be exceeded.

3.1.3 Hazards as decision points in event sequences

Comparing the British Standard definition with that given by Leveson [50]

> *"A hazard is a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event)"*

highlights another key issue. The British Standard definition considers situations that have the *potential* to cause an accident; Leveson defines a hazard as a situation that *will inevitably* lead to an accident. The difference is that Leveson's definition is taken in the context of specific contributing factors in the system environment, and this prompts consideration of how a hazard is identified in terms of its development into an accident.

Figure 17 shows a hazard, arising from a set of root causes, which may develop into one of two possible accidents depending upon the subsequent sequence of events, where the sequence of events actually followed is determined by one or more contributing factors (the environmental conditions in Leveson's definition). Alternatively, a safe state may result if another sequence of events follows the emergence of the hazard. It is important to note that a hazard on its own is *not* sufficient to make an accident inevitable.

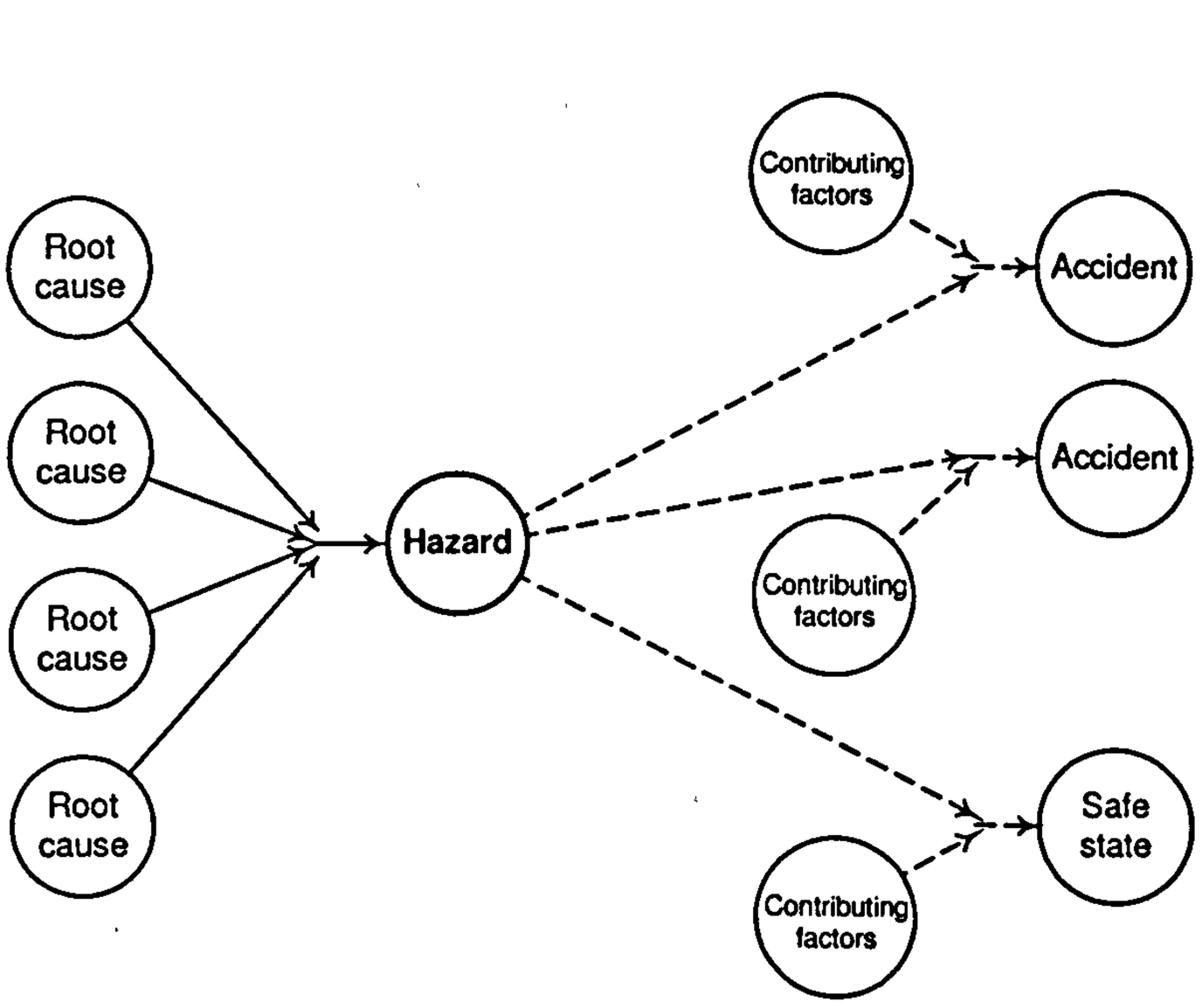


Figure 17 - Relationship of hazard to causes and possible outcomes

In the most general interpretation of Figure 17, the contributing factors, and hence the sequences of events following the hazard may be either normal (expected or intentional) or abnormal

(unexpected or unintentional) events or actions. The selection of any particular sequence may be either deliberate (i.e. due to explicit action on the part of the system or an operator), or simply a matter of luck. In these event sequence terms, a hazard is often identified as *the last system state from which it is possible to make transitions to either a safe state or an unsafe state.* In other words, a characteristic of a hazard is that it is the last decision point before an accident.

This model can be developed further by considering what types of events or conditions combine with the hazard to result in an accident. Taking the simple example hazard of oil spilled on a staircase used in section 3.1, it is clear that this situation can develop into an accident without further abnormal events taking place. All that is required is for someone to use the staircase without looking where they are placing their feet. This is a further important characteristic of hazards in any sort of system; they are usually conditions that can develop into an accident through a sequence of normal events and actions. A situation may properly be described as a hazard if, once it has arisen, an accident can take place with no further failures or abnormal conditions as contributing factors. This characteristic is reflected in the rule of the excluded miracle in fault tree analysis, which states that, once a failure has occurred, the analyst must assume normal behaviour from every other part of the system if that would lead to the top event of the fault tree.

There are circumstances in which it is necessary to treat a condition as a hazard even when further failures or abnormal events are required before an accident can happen. These are cases in which the sequence of events is completely outside the control of the system, or is completely unpredictable. Such cases include natural phenomena (e.g. weather), boundaries of influence (e.g. interfaces to other systems whose properties are not fully known), or cases where the eventual use of a system cannot be predicted. A good example of the last of these cases is found in the automotive industry, where huge numbers of vehicles are sold into a wide range of markets, and to drivers of widely differing experience and competence. It is therefore common to assume for safety analysis that vehicles will be used in ways which exceed their intended capabilities, and also that the driver will react to abnormal circumstances in the worst possible way.

Further important properties can be identified by considering the sequences of events that lead from the hazard to a safe state. These are paths where the accident potential posed by the hazard is mitigated by the contributing factors. Extending the oil spill example further, the hazard can be removed entirely if someone cleans up the spill. Alternatively, the risk of the hazard developing into an accident could be reduced by placing warning signs or barriers around the spill (c.f. the

discussion of the precedence of risk reduction actions in section 2.2.2). Note that the mitigation here may be either a normal or abnormal action, depending on who finds the oil. If there is a janitor working in the area, cleaning up oil spills might be considered a normal part of his job. For other staff, cleaning up is obviously a sensible and desirable action, but which may not be a normal part of their duties.

Putting all of the event sequence concepts together, the definition of a hazard could be expanded to "a condition that, unless mitigated, may develop into an accident through a sequence of normal (or uncontrollable external) events and actions". As with the definition in terms of risk thresholds, this is still inadequate, but captures some important concepts.

3.1.4 Hazards as failures of energy containment

An alternative definition of an accident is "an unplanned or uncontrolled release of energy". It can readily be seen that this definition actually describes the physical process of an accident, unlike the more standard definitions, which describe the accident outcome (harm). Provided that the understanding of "energy" is sufficiently broad, for example by including toxic reactions as a release of chemical energy, there are very few situations and types of system that cannot be regarded as sources of energy that must be controlled and contained. There are a number of hazard identification techniques (see section 2.5) which, in a related way, look for hazards in terms of failures of containment of energy.

Returning to the simple oil spill example that has been used throughout this section, it is fairly easy to express the probable accident scenario in terms of energies. The energy that will be released in an accident will be the potential energy possessed by a person standing upright on the staircase, which will be converted to kinetic energy in a fall. However, it is difficult to see the oil spill itself in terms of the definition that a hazard is a failure of energy containment, since there is no mechanism for containment as such. If a hazard is defined more broadly as "the creation of an unintended path for the release of energy or toxic materials", this now encompasses the oil spill. Note the similarities between this definition, and the explicit modelling of unintended paths in sneak analysis (section 2.11).

There are many situations in which it is relatively simple to identify potentially hazardous energies, and the mechanisms that are used to control them. These include many of the most common application areas of safety critical control systems, such as process and manufacturing plants, nuclear power, military and transport systems. In these cases, the energy release model

can rapidly and systematically identify many hazards. This model is particularly good at identifying hazards associated with systems or mechanisms with explicit safety roles.

3.1.5 Summary of hazard characteristics

Whilst the above discussion has not attempted to identify a single "best" or universally correct and acceptable definition of a hazard, it has introduced a number of important concepts. Key characteristics of hazards include:

- they encode historical knowledge by identifying situations which have been significant causal or contributory factors in past accidents
- they represent decisions about the threshold between acceptable and potentially unacceptable levels of risk
- they may be the result of internal (endogenous) or external (exogenous) causes
- there is no fixed rule for determining whether a particular condition should be treated as a hazard, but pointers for good selection include:
  - conditions which can be mitigated, but from which an accident can arise through a sequence of normal events or actions
  - conditions representing a path for the unintended release of energy or toxic materials
- for any particular development project, the identification of an appropriate set of hazards is essential for effective management.

3.2 Faults and failures

Understanding the concepts of faults and failures, and their relationship to hazards, is important to effective safety analysis for the very practical reason that, whilst many safety analysts have problems with the somewhat abstract concepts of hazard definition, faults and failures are generally seen as very concrete conditions, which can be identified readily, and with certainty. A great many analysis techniques are structured around the identification of failures, their causes and effects, and section 3.3 builds up a "dictionary" of related concepts.

The terms fault and failure have, historically, been used somewhat interchangeably. However, the supporting notes to the definition in BS-4778 emphasise that a failure is an event, whereas a fault is a state. Thus a component which is faulty may cause a failure when it is used (the incorrect state results in an incorrect action or event); similarly, a failure may result in a fault (the incorrect action or event leaves the component or system in an incorrect state). This convention will be adhered to in this thesis.

Far more important than the distinction between event and state, however, is the understanding of what represents the "correct" behaviour that faults and failures are deviations from. Failures can be defined as inability to perform *original*, *designed*, *specified* or *intended* behaviour, and the scope of each of these is quite different. The distinction is vitally important in defining many of the technical and managerial aspects of system safety and, particularly, of software safety engineering. Unfortunately, all of these definitions have been used, and the debate as to which is the most appropriate still continues.

If a system is considered to have failed only when it is unable to perform as *original*, then the only failures included in the definition are those which have arisen since the system was commissioned. This is the reliability engineering view of failure, where the focus of interest is normally the rate of occurrence of *new* failures. This is a relatively narrow definition of failure, which is not appropriate for safety engineering. A system may be reliable (i.e. experience no or few failures which impair its original behaviour) and yet unsafe because it was specified or designed incorrectly, and is therefore reliably performing unsafe actions.

Defining failure with respect to *designed* behaviour includes faults introduced into the system through incorrect manufacturing or commissioning, and definition with respect to *specified* behaviour broadens the scope still further to include errors in the design process. However, for safety engineering, the most appropriate definition of failure is with respect to *intended* behaviour; the broadest possible scope. It is important to note that this is not a universally accepted view. In particular, many American researchers and practitioners contend that failure is always defined with respect to specification.

The authors of the U.S. Nuclear Regulatory Commission Fault Tree Handbook [69] contend that it is possible to have a fault without a failure. One of the examples they cite in support of this view is the unexpected opening of a lifting bridge. The bridge itself works perfectly (there is no mechanical failure); the fault (unintentional opening of the bridge) is the result of an incorrect action by the bridge operator. However, taking a broader view, it can be argued that the operator is as much a component of the bridge *system* as the mechanical parts; the *system* ends up in a faulty state as a result of a failure by the operator. Referring back to the primary – secondary – command models of faults and failures described in section 2.10 it seems that, in this example, a vital command "component" is being excluded by not considering the operator to be part of the system.

A second example cited in the same book is the case of General Beauregard's messengers in an early battle in the American civil war. The General sent a rider with a message to one of his officers. Some time later, the state of the battle having changed, he sent a second rider with an amended message. Later still, a third rider was sent with yet another revision to the message. Unfortunately, the riders arrived in the wrong order, and the officer took actions that were inappropriate. There was clearly a fault in this command system, but the authors of the Handbook (in a view supported by Leveson [50]) contend that there was no failure, as each component (i.e. each rider) did what he was intended to.

This argument is also inconsistent with the broadest view of failure described above. In this case, the General himself is both a component of the system (he issued the commands), and its designer (he devised the way they were to be communicated). The General's intent with his second and third message was to amend earlier instructions in a particular way. The failure in the *system* was the General's failure either to realise that the ordering of his messages was important, or to anticipate the possibility that one or more of his messengers might be delayed. Had he done so, and made the whole status clear in the second and third messages, the system could have worked and correct actions been taken despite the delayed messages.

Unfortunately, although it may seem somewhat abstract and trivial, the understanding of fault and failure is vitally important, because it fundamentally affects the perception of the relationship between failures, hazards and safety and, consequently, the definition of what safety engineering involves.

If failure is defined with respect to intent, and it is considered that the intent of safety critical systems engineering is always to produce accident-free systems, then accidents resulting from endogenous hazards (i.e. those hazards which arise from causes within the system itself) must always be the result of a failure. Accidents may still arise from exogenous hazards (causes external to the system) without there having been a failure. Although this appears to be a logical conclusion from the definitions, and is adopted in this thesis, it remains contentious. Leveson, for example, maintains that this is essentially a reliability approach, and safety involves broader concepts.

3.2.1 Systematic Failure

Considering the discussion above, it can be seen that it is actually the treatment of systematic failures (those arising from mistakes in the specification, design or implementation of a system) which are contentious.

Very few of the safety analysis techniques described in sections 2.5 to 2.14 actually make any structured attempt to identify systematic failures, although many are capable of incorporating such failures when required. For example, in the fault tree handbook, chapter VIII develops an example fault tree for rupture of a pressure tank. In the completed tree, there is one (undeveloped) event ("timer does not time out due to improper installation or setting") which is clearly a description of a systematic failure. However, there are many other places in this tree where there are fairly obvious systematic failure contributions that could potentially have been included. For example, one event is "excess pressure not sensed by pressure actuated switch". A methodical search for systematic causes should have identified as a contribution to this the event "pressure switch does not open due to improper installation or calibration" – an analogous failure to that included for the timer.

Whilst this is an issue of style and completeness for safety analysis at the level of multi-technology systems, it becomes vitally important for computer systems and software, where safety is dominated by consideration of systematic failure.

3.3 A dictionary of concepts in analysis techniques

This section briefly summarises, in alphabetical order, other properties and behaviours of systems that are identified and investigated by the analysis techniques reviewed in Chapter 2. These concepts form a basic "vocabulary" for the review of existing proposals for computer system safety analysis, and for the design of new analysis techniques. Note that probability, severity and risk have already been defined and briefly discussed in Chapter 2.

Cause

A condition that precedes, and contributes to the occurrence of, the (failure) condition which is being investigated. Models of causality are closely related to the definition of failure, and vary considerably, not only between techniques, but also in different authors' application of the techniques. This is most noticeable in the identification of *root causes* (fundamental causes of a failure). If failures are defined with respect to original behaviour, then the identification of root

causes extends only as far as the earliest component failure. If failure is defined with respect to specification, manufacturing and installation defects are included, and in the broadest case, where failure defined with respect to intent, identification of root causes may investigate problems in the specification and design activities.

Of the techniques surveyed, fault trees, Cause-Consequence Analysis has the most elaborate model of causality, explicitly representing the combinations of conditions that will result in a particular outcome. The concepts of *immediate*, *necessary* and *sufficient* causes are used to structure the investigation, ensuring that the analysis is complete and logically consistent.

Classification (of faults and failures)

Several analysis techniques implicitly classify failures, most commonly to provide a set of prompts, or "mini checklists" to suggest possibilities for investigation. Examples include the three failure categories used in Functional Failure Analysis, the HAZOP guide words, and the primary – secondary – command rule in fault trees. The numerous variants of sneak circuit analysis also incorporate implicit classifications in the patterns which are used to identify potential design problems in circuits or plant, and also in the "sneak clues" – another instance of implicit classifications used as prompts. It is interesting to consider the relationships between sneak clues and other sets of prompts – particularly the HAZOP guide words. The correspondence is not exact, but it can be seen that each sneak clue could be suggested by appropriate interpretation of the HAZOP guide words. For example:

Sneak Clue	**Related HAZOP guide words**
• Unintended current path from source to target	• More
• Reverse current flow along path	• Reverse
• Excess current flow along path	• More
• Intended current path created at the wrong time	• Early or Late, or More
• Intended current path can be cut off at the wrong time	• Early or Late, or Less
• Intended current flow can be diverted	• Less (or More for erroneous destination)
• Two paths allow unintended interaction	• As well as
• Person or object can move into flow path	• Other than

The use of prompts based on classifications of failures has largely superseded checklist-based analyses, and it is clear that the development of an appropriate classification / prompt structure is essential to the success of an analysis technique.

Common causes

Single causes that contribute to the occurrence of more than one failure. Common cause failures are indicative of systematic problems in the system development, installation or maintenance processes. Zonal Hazard Analysis (section 2.14) is a method explicitly intended for common cause identification, but standard fault trees and Cause Consequence Analysis can also fulfil this role to some extent through the identification of cut-sets. The potentially serious impact of undetected common causes on system safety makes this an essential aspect of any deductive analysis technique.

Common mode failures

Instances where several components fail in the same way – for example, stress-related failure of structural components. As with common causes, common mode failures are indicative of systematic errors in system development, installation or maintenance.

Contributing factors

Any additional conditions (events or states) which modify the effects of a deviation, fault, failure or hazard. There is an almost unlimited range of potential contributing factors ranging from internal state (e.g. system mode), through environmental and location-dependent factors, to the presence of humans as potential victims of an accident. Identification of contributing factors is an important step in evaluating the risk presented by a hazard but, in general, there are so many that the analysis must focus on identifying and evaluating the most important conditions. Depending on the purpose of the analysis, these may include:

- factors over which the system designers or operators have control
- the conditions most likely to be encountered
- the conditions which will lead to the most serious potential outcome.

Detection

The detection of a deviation or failure is vitally important to the design of protection mechanisms. Detection and mitigation of failures is considered explicitly in almost all inductive safety analysis techniques, either explicitly, as in HAZOP, or implicitly through the investigation of decision points in sequences, as in event trees and Cause-Consequence Analysis. In deductive analyses, failure of detection and protection mechanisms must be included as a necessary event if failures are to propagate.

Certain failures are *inherently* undetectable, at least within the system boundary. This most often occurs when the failed component is a part of a simplex system. Other important cases are those where a component has failed but continues to produce believable information. For example, if the button to call a lift to a particular floor of a building fails, this is inherently undetectable to the lift controller – it cannot predict the arrival of someone wishing to use the lift, so the lack of signal is a normal condition. More complex cases involve sensor "drift", and occur in control systems based on an internal model that gradually becomes inconsistent with external reality. Another important problem in detectability is *dormant* (*unrevealed*) failures, i.e. failure of a component that is not in use at the time, so that the problem only becomes evident when demands are made upon it.

Deviation

Any unintended or unexpected behaviour. The concept is more general than faults or failures, as it includes the possibility of behaviour that is both safe and correct with respect to intent, but which has not previously been recognised as a possibility. The suggestion and investigation of deviations is the basis of HAZOP.

Effect

The conditions resulting from a deviation or failure.

Equivalence

Among the techniques not discussed in detail in Chapter 2 are two (Failure Mode Effect Summaries [71] and Gathered Fault Combination [80]) whose purpose is explicitly to identify and collect together groups of failures whose effects are equivalent. This is important both to help manage the safety process, by reducing the number of individual conditions which must be considered, and in the design of detection and protection mechanisms. Failures are equivalent if they have the same effects *in combination with any other failure or failures*.

Intrinsic Safety

The intrinsic safety (sometimes known as *conceptual safety*) of a system is the degree of safety achieved if all active control is ceased and all protection mechanisms removed. For example, an aerodynamically stable aircraft has a limited degree of intrinsic safety in that, if the pilot makes no control inputs, the aircraft will tend to continue in straight, level flight.

Mitigation

The mitigation of a hazard or failure is the action that is taken to limit its effects once it has occurred. A huge range of mitigation strategies is possible, including using redundancy to mask the fault, building in containment or applying active control or protection mechanisms to limit its effects. Mitigation is considered explicitly by most safety analysis techniques. In inductive analyses, the effectiveness (or otherwise) of mitigation strategies form decision conditions; in deductive analyses, the failure of mitigation mechanisms will appear as contributors to the undesired outcomes being investigated.

Normal Behaviour

Many analysis techniques require the investigation of whether the normal behaviour of the system is safe. Unfortunately, like failure, there are many alternative ways of defining normal behaviour. For a system that has been in use for some time, normal behaviour may simply be that which is usually observed. For safety analysis of a design, normal behaviour is usually interpreted as the way in which the *designer*, having interpreted the specification, intends the system to work. This means that the investigation is considering the correctness of both the specification and design with respect to the original intent (i.e. what is really required for safety).

Paths

The explicit identification of unintentional paths, or routes between a source and a target, is unique to Sneak Path Analysis. This is a potentially very powerful concept, but difficult to systematise.

Propagation

The sequence of effects resulting from an initial deviation or failure. This is explicitly modelled by the successive states in inductive analyses; in deductive analyses, each intermediate event represents a step in the propagation of a basic failure up to its eventual system level effects.

Time

Few safety analysis techniques incorporate an explicit model of time. A simple model of time is implicit in the majority of techniques (causes always precede effects); if more explicit measurement is required, this must be incorporated through the description of events (e.g. "valve remains open for more than 5 minutes").

Transformation

The way in which the effects of a failure are modified by propagation around a system, and the intervention of protection and mitigation mechanisms. In traditional hydro-mechanical systems, the transformation of a failure is normally limited, and there is no explicit modelling of this concept in any of the techniques surveyed.

3.4 Classifying Hazard and Safety Analysis Techniques

So far as can be ascertained from published literature, there has been relatively little work on developing a structured classification of safety analysis techniques. The broad groupings of techniques normally identified have traditionally been related mostly by their role in the systems lifecycle, rather than by the underlying method or concepts. These traditional groupings are hazard identification, risk assessment, preliminary (requirements-setting) safety analyses, detailed (confirmatory) safety analyses, and particular risk assessments, including zonal hazard analyses. Of course, many techniques have multiple roles, and can be included in more than one of these classifications. The only conceptual grouping traditionally identified has been the distinction between deductive (searching for possible causes of specific events) and inductive (identifying the effects of known causes) analyses.

In 1992, Fenelon [28] suggested that the traditional classification of safety analysis techniques into inductive and deductive could be generalised. He suggested two new categories – exploratory and documentary – to complete the matrix shown in Figure 18. Techniques such as HAZOP, which provide a structure for analysis of completely new systems, were classified as exploratory. The documentary class was reserved for notations that are used primarily for recording known properties of completed systems, and which do not necessarily have a significant analytical role. Fenelon suggested that an important step in improving safety analysis should be the development of a single analysis method which could function in all four of the roles identified in his classification structure, and this was the basis for the development of his Failure Propagation and Transformation Notation (FPTN), which is discussed in section 4.6.

		Cause	
		Unknown	Known
Effect	Unknown	Exploratory	Inductive
	Known	Deductive	Documentary

Figure 18 - Fenelon's analysis technique classification matrix

For the purposes of reviewing, comparing and identifying key properties of existing safety analysis techniques, Fenelon's classification structure is an interesting starting point. However, it soon becomes clear that there are two key problems with Fenelon's matrix. The first is simply that there are insufficient classifications to adequately distinguish between techniques with quite different and important roles. For example, Functional Failure Analysis (FFA, section 2.7) is a commonly used preliminary / requirements-setting inductive analysis. Failure Modes and Effects Analysis (FMEA, section 2.9) is an inductive technique principally used (in Europe) for detailed (confirmatory) safety analysis. Under Fenelon's classification, both of these techniques, although obviously and significantly different in nature, can only be described as "inductive".

The second problem is that the structure (implicitly) assumes that safety analysis will only ever work towards more concrete information, which is not strictly the case. It would be preferable to express the role of an analysis technique in a more dynamic way, by defining the starting and finishing conditions, e.g. one role of Fault Tree Analysis is to move from a state where effects are known and causes unknown, to a state where both effects and causes are known, as shown in Figure 19.

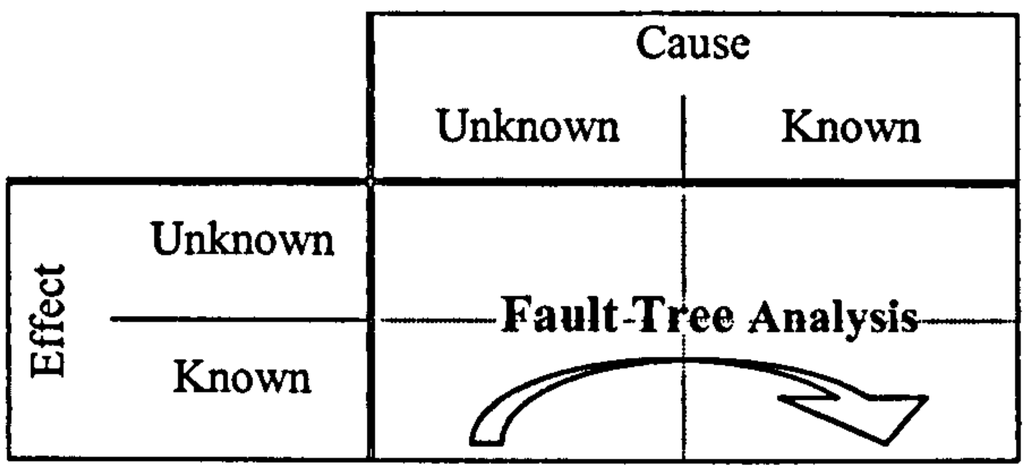


Figure 19 - Representing fault tree analysis as a change of state

The expressive power of this scheme can be greatly improved by including the distinction noted in section 3.1 between *concrete* (known, certain) conditions, and those which are *projected* (suggested or expected). Using the expanded matrix shown in Figure 20, the distinct roles of FFA and FMEA, which were problematic in Fenelon's matrix, can now be represented.

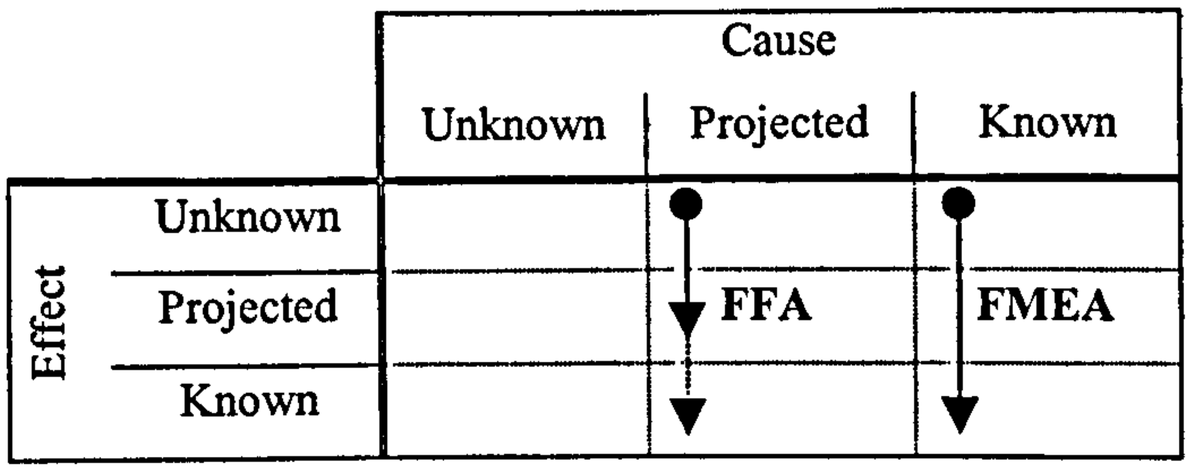


Figure 20 - Expanded classification matrix

Note that in this example FFA is shown as identifying projected effects, with a dotted arrow representing the possibility of continuing to the identification of certain effects. This reflects the fact that safety analyses working from projected preconditions can never be certain about the outcomes. This is also true to a certain extent even when working with the most concrete data available, as in the case of FMEA. The "known" effects identified by FMEA are also only predictions; truly concrete data can only ever be produced by testing, or through accident analysis. For the purposes of this simple classification, however, a prediction based on known starting points will be considered to produce known outcomes.

In the same way that it has been observed that identification of hazards is, in part, a decision about management, projected failure information can also be seen as a management tool. Since these projected properties are abstractions, they can be used to control complexity, and to divorce the specification of important system properties from the low-level detail of the system.

The main purpose of mapping the roles of different techniques is to help understand what combinations of techniques will produce a satisfactory system safety process. In crude terms, the minimum requirement for a "joined up" safety process that can deliver evidence that the hazards in the system have been identified and their causes investigated could be represented by drawing a "time line" diagonally through the matrix, starting at project inception with a completely unknown state, and ending fully documented behaviour (Figure 21). Any combination of techniques that will effectively traverse this leading diagonal should, theoretically, suffice.

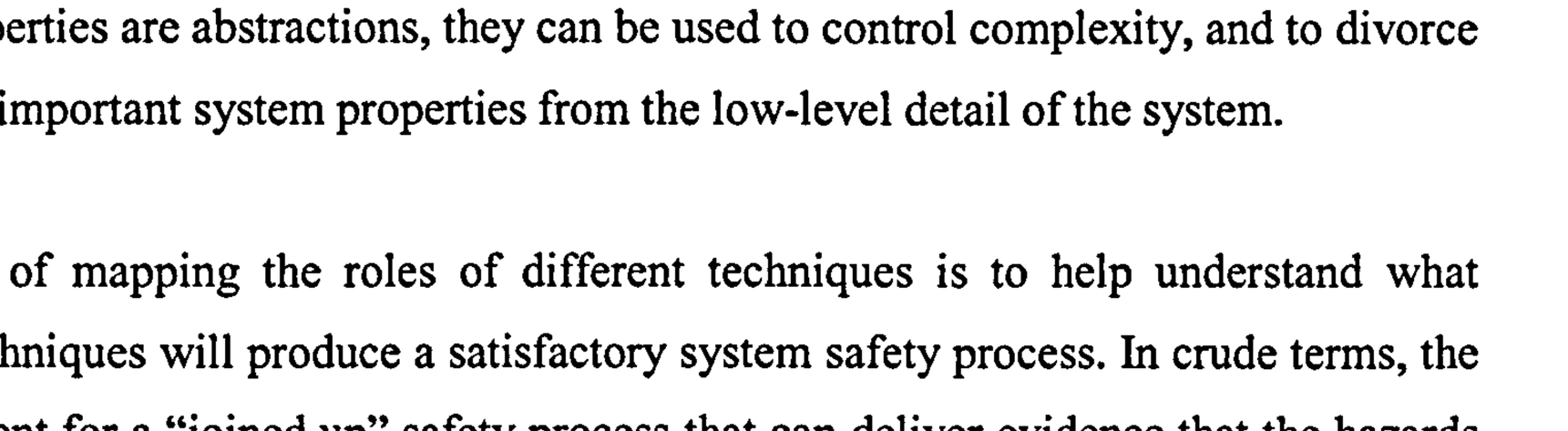


Figure 21 - Minimal set of techniques represented as a time line

In reality, there are many reasons why, even as a minimum, this is unrealistic for a real project. Very few (if any) projects actually proceed strictly top-down as this implies. One of the few documented examples of a major project which really started with a "clean sheet of paper" and defined every component from scratch was the development of the main engines for the space shuttle – a project specifically criticised by Feynman in his appendix to the Rogers Commission report on the Challenger accident [30] for taking an approach which placed very severe demands on untried components.

The vast majority of projects use existing components and subsystems, which means that significant amounts of low-level, concrete information is known very early in the project. A more realistic model of the analysis processes in a real project would therefore be "outside in" (Figure 22). Hazards are identified and projective analyses applied to define requirements for safety related behaviour. Subsystems and components are then designed and selected to meet these requirements. Finally, confirmatory analyses are used to demonstrate that these components and subsystems meet (are "no worse than") their projected behaviour (requirements).

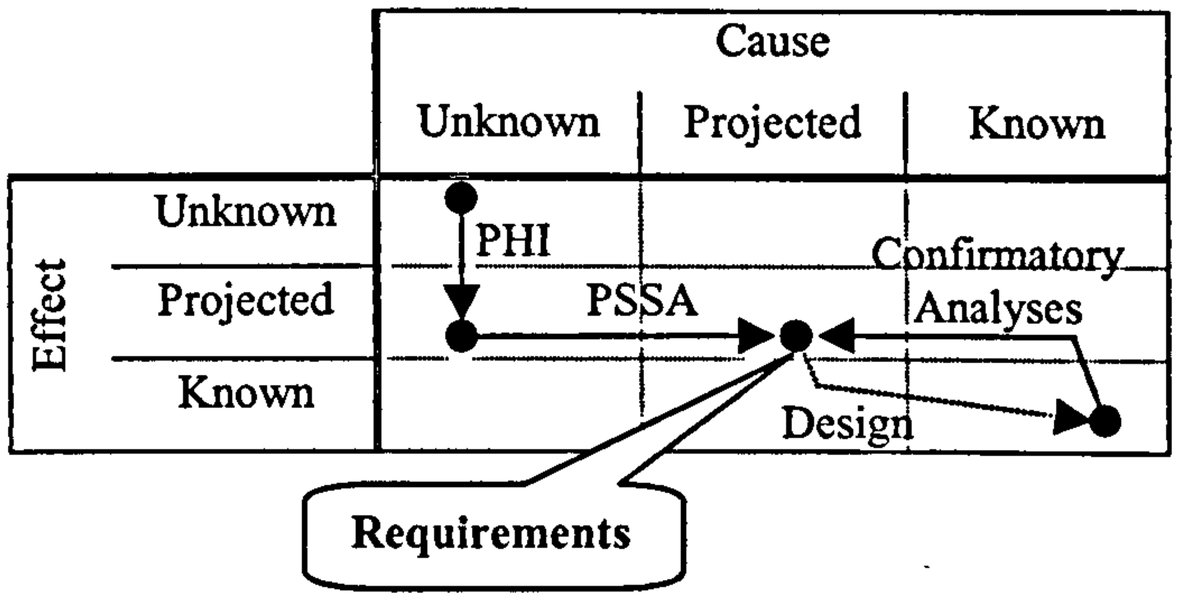


Figure 22 - "Outside in" model of analysis in a project

This model demonstrates two important features. The first is that activities other than analysis (in this case, the design step) can move the state of knowledge about the safety properties of a system. The second is the "closure" obtained by using the confirmatory analyses to demonstrate that the product of the design step meets the projected (required) properties. It is also important to observe that any of the activities represented in this process model may be flawed, resulting in an imperfect understanding of the system properties.

One of the biggest problems of safety critical systems developments is that they are, by nature, open loop. Development times are typically long, and data from testing of systems may only be available some considerable time after the design and implementation work has been completed. In some cases, of course, there is no acceptable way of testing some of the most critical functions in realistic conditions; the nuclear power industry is a good example of this. Even after systems deployment, so long as everything works as intended and accidents are avoided, engineers get relatively little feedback. It can be impossible to decide which features of a design are most effective in maintaining safety. It is only when accidents occur that hard data about failures becomes available – and accidents in current safety critical systems are so rare that the data obtained can only be regarded as point observations, from which it is impossible to draw

statistically significant conclusions. All of this means that closure within the design and assessment process – validating results and designs by multiple means – is vitally important.

This leads to the concept of complementary analysis techniques. Ideally, where safety is critical, each analysis carried out should be tested by comparing its results with those obtained using an alternative technique; preferably one with a substantively different approach. The classic pairing of complementary analyses is Fault Trees and FMEA. Fault trees work down from top events (effects, which may be either known or projected) to find the (initially unknown) causes; event trees work the opposite way, to determine the (initially unknown) effects of known causes. When these two techniques are applied to the same system, a comparison of their results should show identical chains of causality. Similar pairings can be identified for every analysis process.

This model provides a framework for understanding the roles of different analysis techniques, and how combinations of techniques can be selected to ensure closure within a project.

3.5 Conclusions

This chapter has discussed some of the concepts upon which traditional system safety analysis techniques are based. The notions of hazard, fault and failure, which are essential to safety analysis, have been explored at some length, and a range of other concepts has been briefly introduced. Section 3.4 has described a new classification of safety analysis techniques, which allows explicit distinctions to be made between projective analyses used early in the safety lifecycle to set requirements, and the more concrete analyses used in a later, confirmatory role.

These concepts and classification schemes are used in the development of safety analysis principles and new analysis techniques in later chapters.

Chapter 4

A survey of computer system safety analysis techniques

Considering the high profile of safety critical computing both as an industry and as a research area, there is surprisingly little published literature specifically discussing safety analysis techniques for safety critical computer systems. The most plausible reason for the apparent lack of activity in this area is that, until relatively recently, the prevailing view was that safety was essentially a systems engineering discipline. Computer system safety was equated with correctness; safety requirements were derived by analyses at the system level and, since correct implementation of these requirements would give the necessary system level behaviour, there was no need to extend the safety analysis into the computer system itself. Safety critical computing research was concentrated into activities which could help to guarantee the satisfaction of requirements.

This view is no longer widely held, as increasing levels of system complexity and integration mean it is no longer credible to stop the safety process at the boundary of the computer system. It is clear, however, that this change in perspective has not been matched by any significant advances in the range or capability of the safety analysis methods available.

This chapter reviews proposals that have been made for analysis techniques specifically for computer systems, the most significant of which are modifications of HAZOP and Fault Trees. Petri Net analysis is also included here because, although not fundamentally computer specific, it has been applied almost exclusively to the analysis of computer systems. The chapter also reviews one completely new analysis technique, Failure Propagation and Transformation Notation. This technique includes an explicit model of failure classes, and relevant classification research is also reviewed.

4.1 Inductive Methods

Functional Failure Analysis is widely applied to software systems (especially in the automotive and aerospace sectors), and is mandated in many standards. The technique requires no modification to work with software; functions are identified and failure modes suggested using the broad categorisations of failure in exactly the same manner as for system level FFA. This technique is undoubtedly the most widely applied projective analysis of software.

Two significant problems have been observed with this technique in practice. The first is that, since there is no investigation of the potential causes of a projected failure, every failure suggested must be assumed to be possible. This is a conservative assumption, requiring investigation of means of mitigating every failure that is suggested, and has the disadvantage of potentially leading to over-engineering and excessive complexity. The second, more significant problem is that there is no good published guidance on the interpretation of "function provided incorrectly" for software systems. The most common approach seems to be to assume that this equates to "function delivers wrong value" – implicitly excluding other potentially serious problems such as incorrect timing of a function.

Despite the widespread use of FMEA across many engineering disciplines, few proposals for the application of FMEA techniques to computer systems or software have been published. Raheja [65] presents a method called Software System FMEA (SSFMEA), but on inspection, this method is actually found to be a software Functional Failure Analysis. Dhillon's bibliography of FMEA concepts and applications [22] cites only one paper which is specifically targeted at software – Reifer's short paper of 1979 [68]. The approach described in this chiefly concerned with the production of quantitative data for the failure of mission or safety critical software functions. The brief method description contained in the paper implies that the method of working is actually *deductive* – the critical functions are identified, then analytic methods applied to determine what software conditions can cause failures of these functions. Finally, (unspecified) software reliability functions are used to supply failure rates for these software conditions. In summary, the method does not appear to be the type of inductive analysis based on known failure modes that FMEA implies.

4.2 HAZOP

ICI, with its extensive use of HAZOP for process plant assessment, has been using a technique known as Computer Hazard and Operability Studies (CHAZOP) for several years to examine the control capabilities of programmable systems. CHAZOP is actually a checklist, derived from an extensive database of incidents and accidents maintained by the company's Computer Aided Production group. The checklist is divided into sections, and proposed systems are examined by answering the questions in all relevant sections. This procedure has been applied to all new Programmable Logic Controllers (PLCs) and computerised controls installed on company sites since 1988. The questions are at a relatively high level, concerning aspects such as power supply,

control of code etc., and are not a suitable basis for analytical study of general computer safety issues.

Outside ICI, the use or adaptation of HAZOP for computer systems was first suggested in a paper by J. V. Earthy in 1992 [25]. This paper recommended a first assessment at the level of the interfaces between the processor, storage devices and peripherals, followed by more detailed analysis of data flow diagrams. The paper did not propose guide words, but noted that the analysis is only valid if the design representation studied accurately reflects the system as it is actually built. This short paper was quickly followed by a more extensive proposal from Burns and Pitblado [12]. This suggested three related HAZOP studies for programmable systems that control or monitor plant or machinery:

1. An initial "conventional" HAZOP studying the plant to be controlled, using the standard guide words and method.
2. A more detailed Programmable Electronic System (PES) HAZOP study of the computer or PLC systems controlling the plant, considering deviations in *signals* and *actions*, using the guide words *No*, *More*, *Less* and *Wrong*. The paper is not clear at what level this is to be applied, i.e. whether the signals and actions to be considered are internal or external to the control system. The suggested list of source documentation for the study includes ladder diagrams (PLC programs), which implies a detailed internal study, but this conflicts with their stated intention (and the normal role of HAZOP) of aiding initial analysis.
3. A human factors HAZOP, using new guide words for deviations in *information* and *actions*. Again, the suggested source documentation, which includes some detailed information that would only be known at a relatively late stage in design, appears to conflict with the assertion that the techniques are for preliminary hazard analysis.

Cambridge Consultants' modification of HAZOP described by Chudleigh [15] is more specific than either Earthy or Burns and Pitblado's work. Data flow diagrams are identified as a basis for the analysis, a table of guide words and the parameters to which they apply is presented, and a brief description is given of the manner in which they are applied, which includes analysis of *processes* as well as *flows*.

Fencott and Hebbron [27] develop a version of HAZOP for use on Ward & Mellor [81] essential models. Their approach is based on the analysis of individual "response threads" (lists of required responses to events) extracted from the model, rather than all communications over a given data

flow. They emphasise the importance of identifying appropriate *property words* to capture the intent of the responses, and develop and classify guide words for use with each property word.

Since the publication of these early papers, the evolution of software and computer system HAZOP has been shaped primarily by the MoD sponsored work which culminated in the publication of Defence Standard 00-58 [79]. This work was contemporaneous with, and influenced by, the development of SHARD, and is described in Chapter 7.

4.3 Fault Trees

There are two distinct approaches to adapting fault trees for computer systems analysis. The first is work such as that by Dugan, Bavuso and Boyd [23, 24], which is exclusively aimed at improving the modelling of failures of complex hardware, providing new gates to deal with sequence dependencies and cold spare redundancy. This is simply an extension of system level fault tree analysis, and does not attempt to investigate either the logic implemented in the software or the effects of hardware failure on the software. This work is interesting because of the way in which it demonstrates that even highly successful analysis techniques need to be updated to meet the demands of innovative technology.

The second approach involves using fault trees in a way that is substantially different to the standard system level method. Software Fault Tree Analysis (SFTA) was first described in detail by Leveson and Harvey in two papers in 1983 [51, 52]. The aim of the technique is to show that the logic contained in the program cannot produce output conditions that have been determined to be hazardous.

The starting point for the method is system level analyses that identify output conditions of the software which cause, or contribute to, hazards. A separate software fault tree must be constructed to investigate each potentially hazardous output condition. The software analysis starts by identifying the code responsible for producing the output; from here the analysis proceeds backwards, using inspection of the code to deduce what steps would have been necessary for the program to reach this point. Impossible conditions (contradictory conditions under an AND gate) are identified and eliminated. If the code cannot produce the undesired output, then this pruning should eventually eliminate the entire tree. If the potentially hazardous output condition can be produced as a result of a combination of input conditions, then the tree remaining after pruning will define the necessary conditions. Of course, if the output condition can be produced regardless of input conditions, the analysis has discovered an error in the code.

Leveson and Harvey describes the application of the technique to an assembly language program of some 1250 lines – a study which resulted in the discovery of previously unconsidered failure modes.

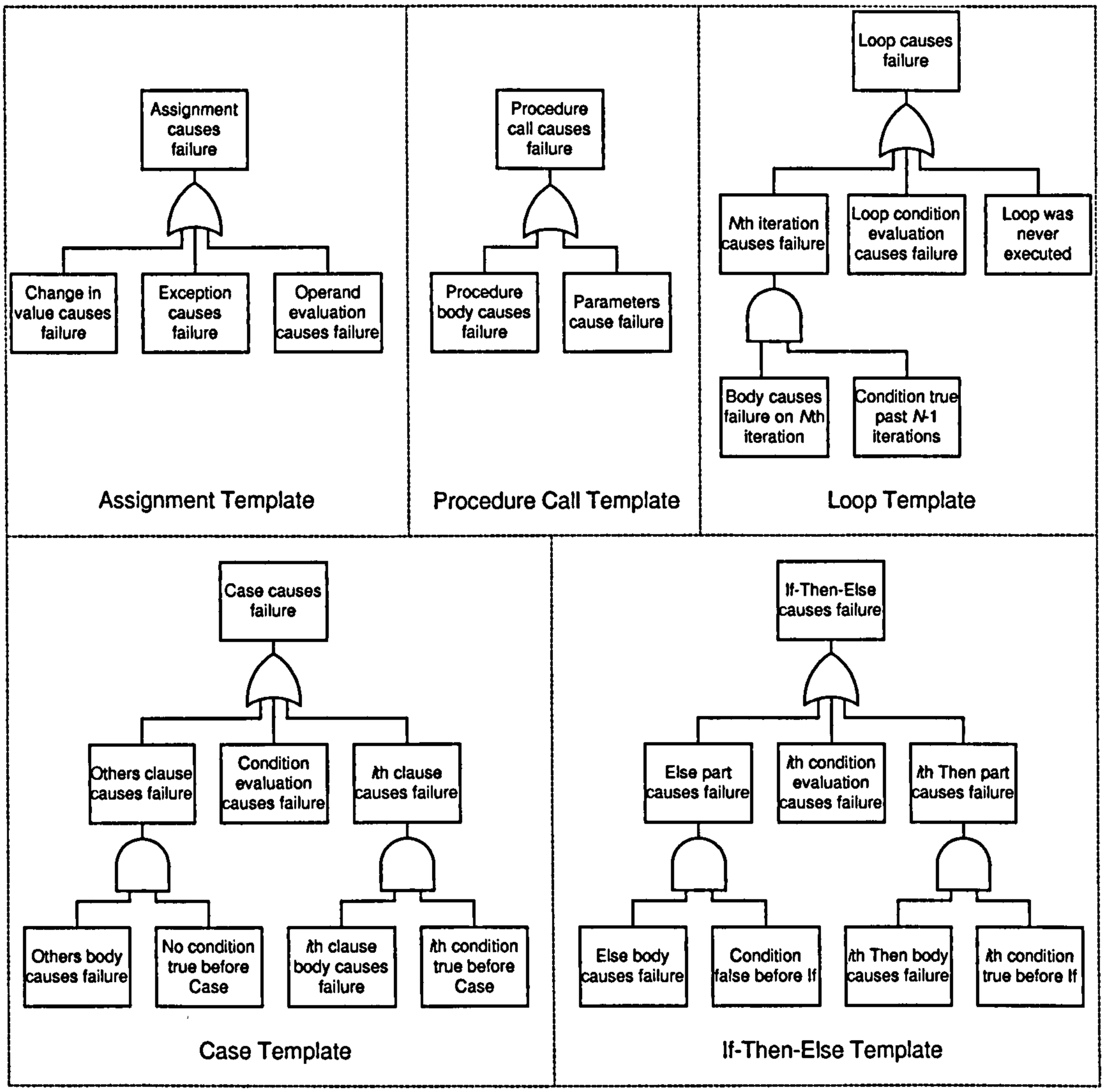


Figure 23 - Leveson's fault tree templates for Ada statements

SFTA was elaborated by Leveson and her collaborators, and a later paper [53] presented a thorough update of the technique. This paper concentrates on the analysis of Ada programs, since this is now the language of choice for many safety-critical applications, and presents a set of templates that encapsulate the failure semantics of various language constructs. The basic templates for simple code statements are shown in Figure 23; the paper also includes a set of templates for Ada tasking.

As previously, analysis starts with the code responsible for the actual output condition; however, once this is identified, the appropriate template is instantiated (i.e. the event boxes are completed with specific details). Each branch of the template tree is then examined to identify the immediately preceding statement and the appropriate templates are added to the tree and completed. Again, this process is repeated until either contradictory conditions can be identified, at which point sections of the tree can be eliminated, or until input conditions are reached. Note also that there are situations in which some of the event boxes can be eliminated immediately. The simplest cases are conditionals with no "else"; parts of the assignment template can also be removed for simple assignments where there is no evaluation of the new value and/or no actions such as type casts which could potentially cause an exception. An example of a segment of a software fault tree, demonstrating the pruning of impossible conditions, is shown in Figure 24.

Although software fault tree analysis using Leveson's method and templates can be made to work, there are many problems with it. The graphical notation of fault tress is an exceedingly cumbersome way to represent what is, in effect, a static data-dependency analysis of code. The tree fragment shown in Figure 24 is an analysis of just 12 lines of code. In addition to its size, the method and notation are fundamentally unsuited to the analysis of code containing loops, especially if there are data dependencies between iterations of the loop. As the *loop* template in Figure 23 shows, an entire sub-tree must be constructed for each iteration of the loop. Whilst this can be avoided by careful rearrangement in simple cases, there is no easy way to simplify loops which manipulate two or more mutually-dependent variables. In addition to these fundamental issues, there are some less serious issues, such as errors in the published templates (notably the sequential composition rule), which make it difficult to start using the technique.

Despite these problems, there have been some sizeable applications of the method, such as the Ontario Hydro nuclear protection system study [9]. Researchers in the Dependable Computing Systems Centre (DCSC) at York have also carried out some unpublished trials, which suggest that, provided some pragmatic simplifications are made, the technique has value in some specific circumstances, such as retrospective analysis of legacy code, where the hazard directed inspection can be very helpful. It should be noted that Leveson herself has noted that the technique is really only likely to be tractable where the proportion of critical code in a system is very small.

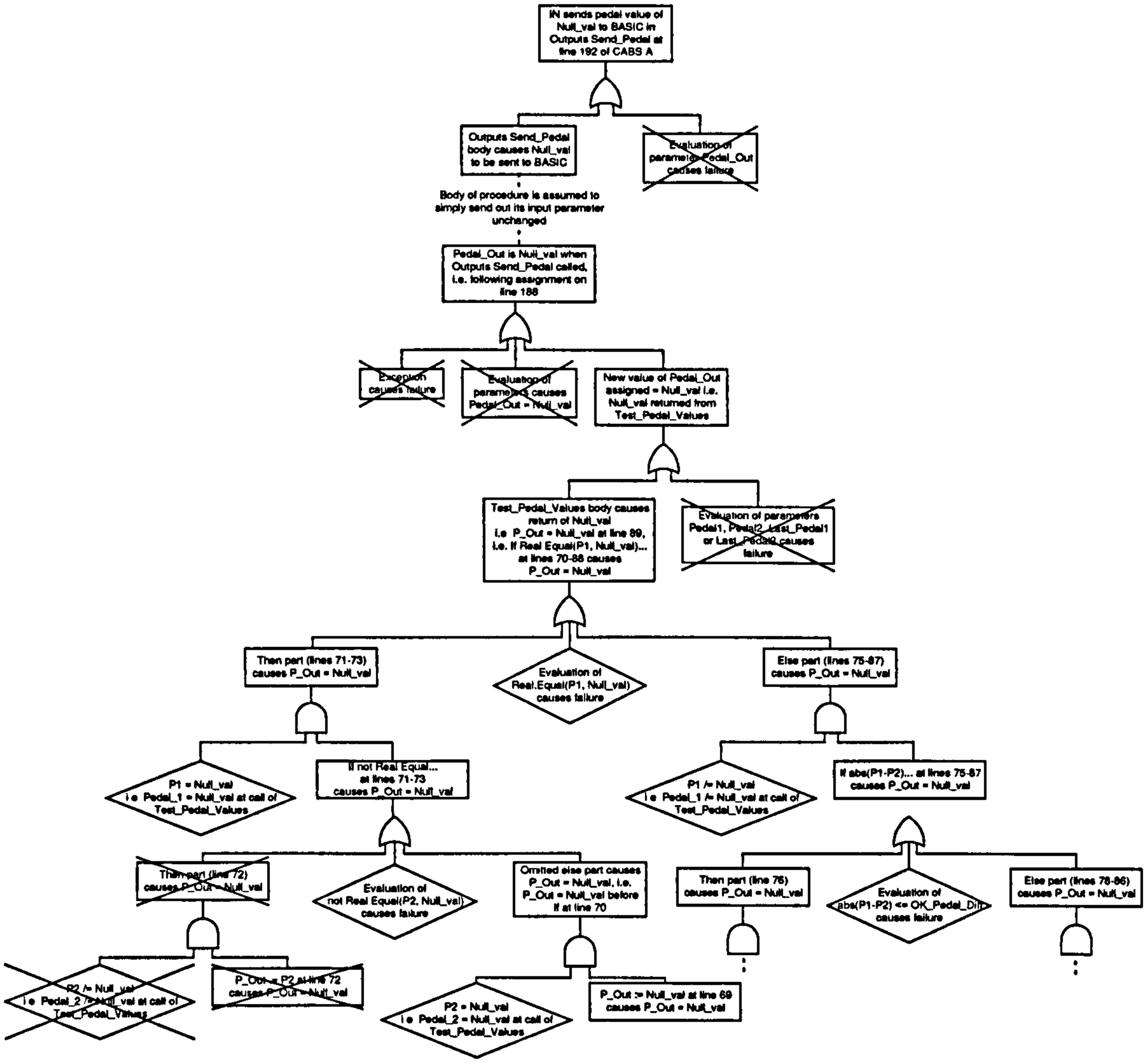


Figure 24 - Fragment of template based SFTA analysis

Leveson [48] also notes that the construction of software fault trees by this method is equivalent to the derivation of weakest preconditions, and this idea is developed by Clarke and McDermid [18]. The equivalence is shown, although it is not as obvious, and requires more work, than Leveson appears to imply.

4.4 Petri Net Analysis

Basic Petri nets consist of a set of *places* and *transitions*, and input and output functions defining the mappings from places to transitions and vice versa. The current state of the system is denoted by marking places with *tokens*. A transition is enabled only when each of its input places contains at least as many tokens as there are arcs from the place to the transition. Once enabled, a

transition may fire, at which point the enabling tokens are removed from its input places, and a token deposited at each of its outputs. They have been used to analyse for many kinds of system properties, including deadlock and reachability. The use of Petri Nets for safety analysis was first proposed by Leveson and Stolzy [54] in 1987. The example they used to illustrate their paper, a simplistic level crossing, is shown in Figure 25.

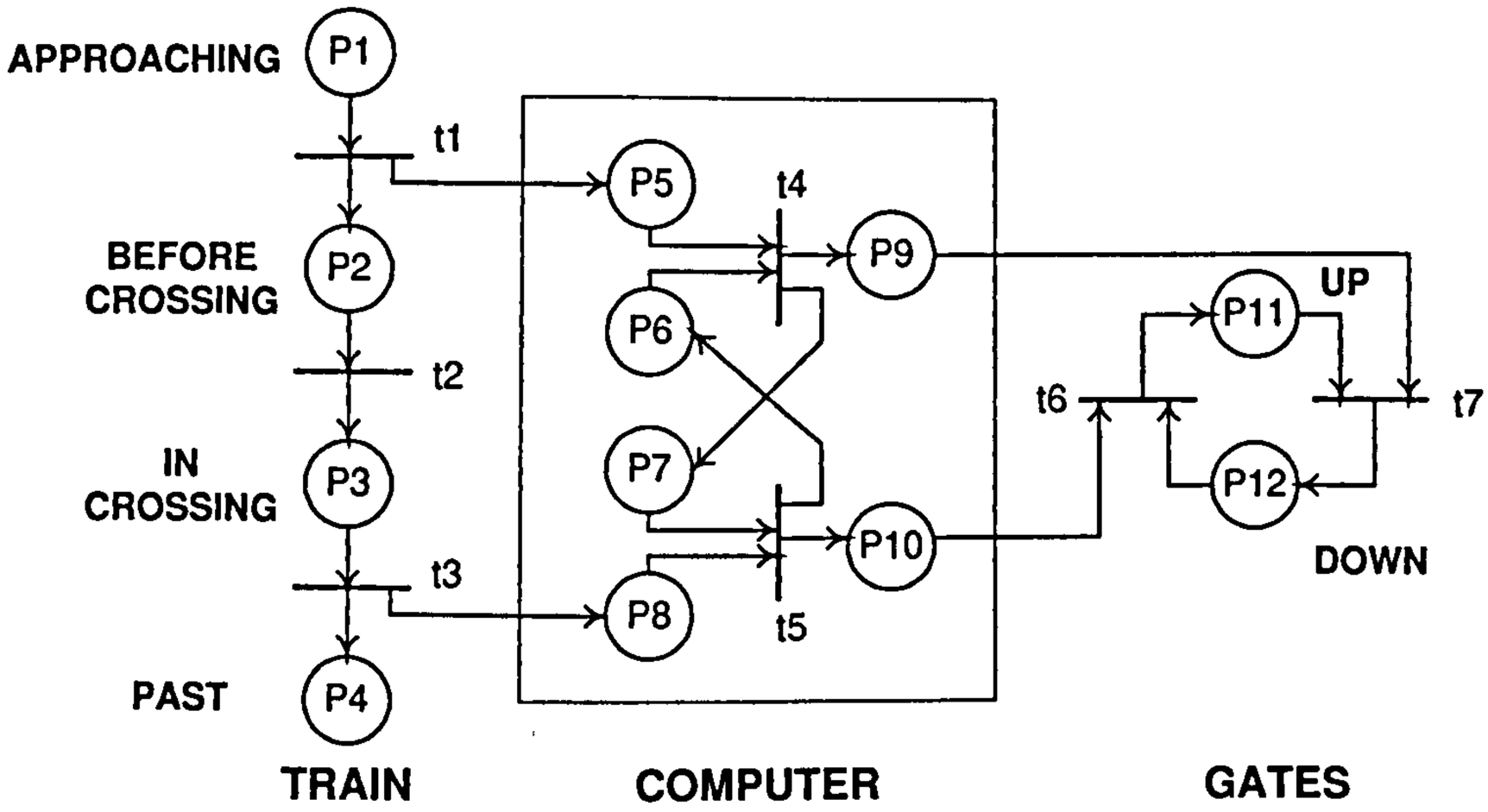


Figure 25 - Simple Petri Net representation of a level crossing

The first step in safety analysis of a Petri net is to determine the hazardous states. In the example system, any state in which places P3 and P11 (representing train within crossing and gates up respectively) are occupied simultaneously is hazardous. Forward or backward reachability analysis can then be used to determine whether the dangerous states can be reached from the initial system state. However, these techniques are extremely expensive, and only practical for very small systems. The reachability graph for the example system contains only 13 states, of which two are hazardous, and is shown in Figure 26.

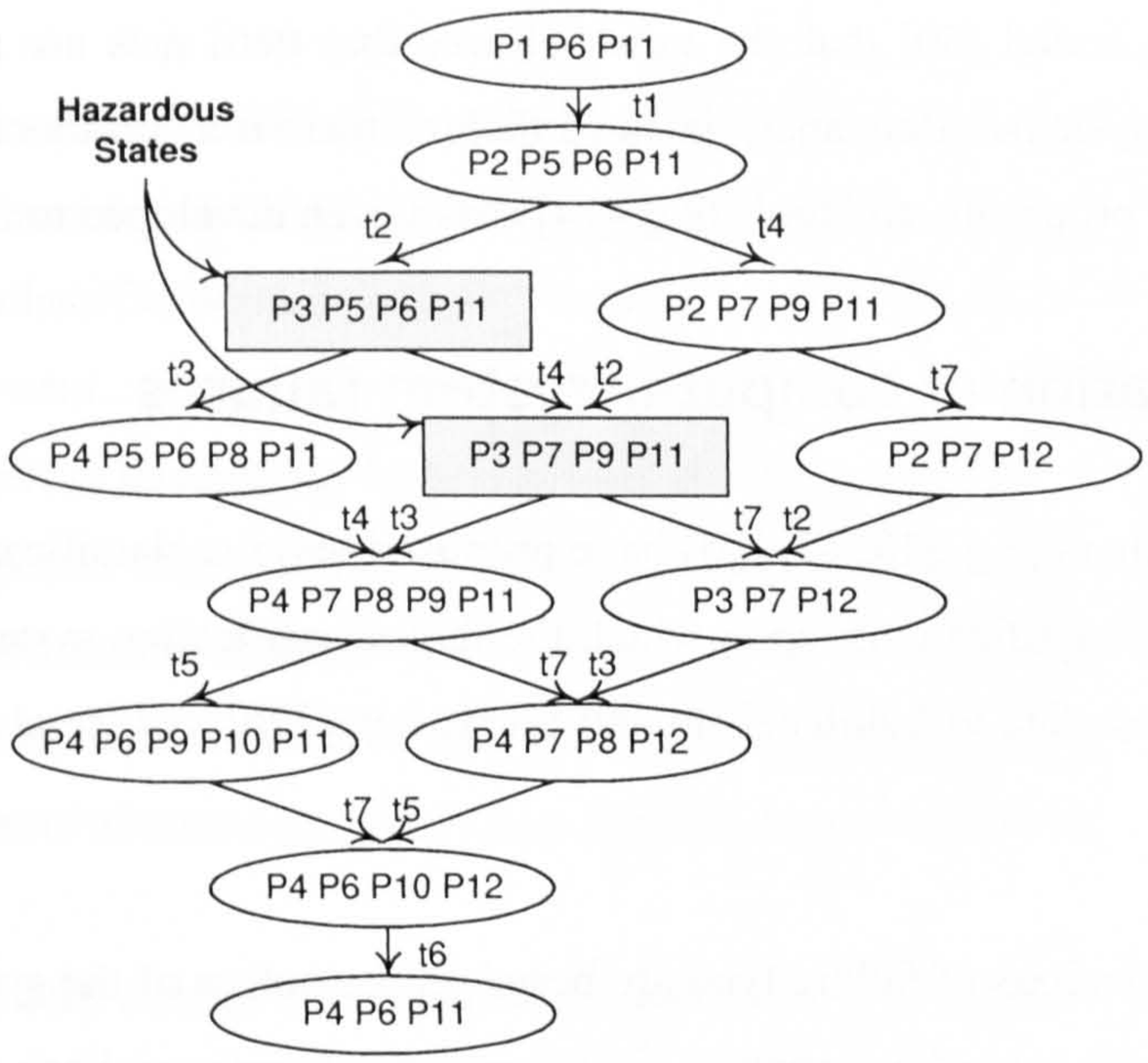


Figure 26 - Reachability graph for the level crossing example

Leveson and Stolzy propose instead the identification of *critical states*. A critical state is defined as being a state from which there are paths that lead to both hazardous and safe states. Critical states are identified by working backwards through the predecessors of identified hazardous states until a state is found from which there is a path to a safe state. Various design measures, such as imposing timing constraints or precedence of certain transitions, can then be taken to ensure that the transition to the safe state is always taken. This is a conservative approach, in that the analysis does not check whether the critical state could actually ever have been reached. An example critical state from the level crossing example is shown in Figure 27.

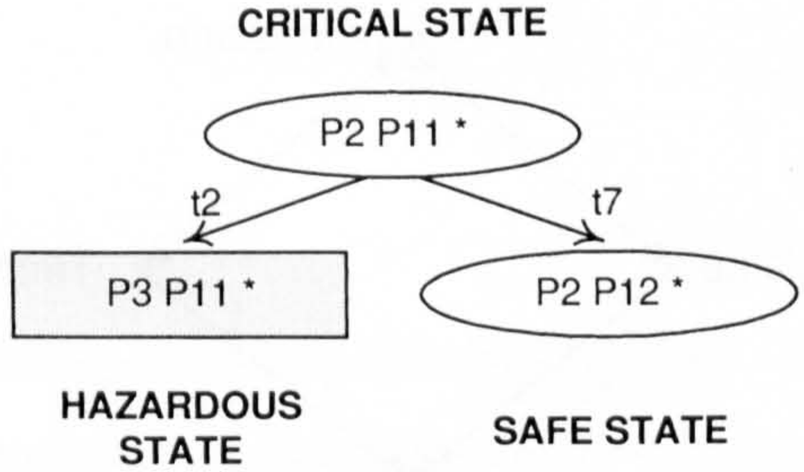


Figure 27 - Example of a critical state

Leveson herself has stated [50] that she now believes that Petri nets are not particularly well suited for computer systems safety analysis; nevertheless, there is a considerable body of research work based on these proposals, and tools (e.g. [14]) have been developed to implement them.

4.5 Classification of computer system failures

A number of researchers (e.g. [19, 47, 75]) have proposed various classifications of failure type. The most developed classifications, upon which the fault classification system of FPTN (section 4.6) is based, are those due to Ezhilchelvan and Shrivastava [26] and Bondavalli and Simoncini [7].

Both of these classifications of failure type are based on the notion of the system as the provider of a *service* (sometimes termed a *response*), or a *sequence* of services. Each service consists of a particular value, delivered at a particular time or, more generally, within a defined interval.

Ezhilchelvan and Shrivastava propose a hierarchy of failure, as shown in Figure 28. The definitions of the failure types identified in the hierarchy are:

- **Omission** — no service is delivered
- **Value** — a service is delivered within the correct interval, but with an incorrect value
- **Timing** — the correct value is delivered, but outside the correct interval
- **Emission** — a response which is expected, but has either (or both) incorrect time or value
- **Byzantine** — a very general class of failure, which includes every failure mode of a component.

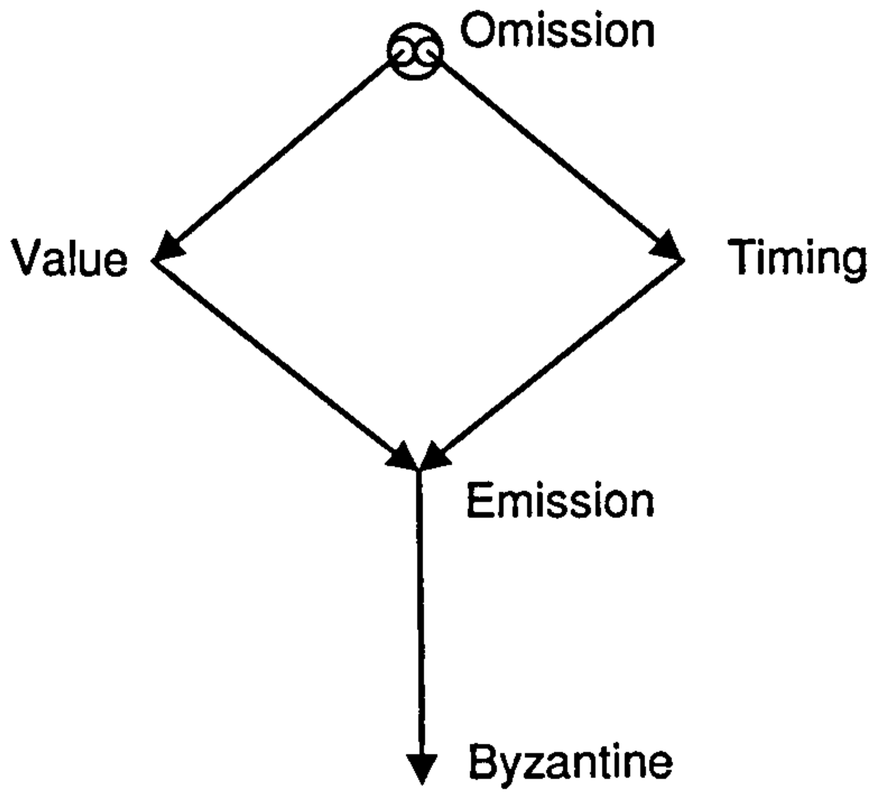


Figure 28 - Ezhilchelvan and Shrivastava's fault / failure ordering hierarchy

There are some points about Ezhilchelvan and Shrivastava's hierarchy which are worth noting:

- The hierarchy does not include normal behaviour as one of the classes. This is reasonable, as it is explicitly a classification of failures, but safety analyses must also consider the normal behaviour of a system.
- The hierarchy itself is interesting; Ezhilchelvan and Shrivastava argue that each failure in the hierarchy is a special case, or proper subset, of the level below. Thus, omission can be regarded as either a timing failure, where the delivery time is infinitely late, or a value failure where the value delivered is null. Similar relationships apply to timing and value, which are by definition subsets of emission, which is in turn a subset of byzantine.

By placing the failure classes in this hierarchy, Ezhilchelvan and Shrivastava have done more than simply identify failure types; they have also produced an ordering implicitly based on another failure property, which they identify as *tolerability*.

Bondavalli and Simoncini's classification of failure type considers the time and value domains separately:

Value is subdivided into

Correctly valued

Subtle Incorrect (undetectable by the service recipient)

Coarse Incorrect (can be detected by the service recipient)

Omission

Time is subdivided into

Correctly timed

Early

Late

Infinitely Late

Again, there are some interesting points about this classification. The inclusion of both infinitely late and omission seems redundant; it is clearly an attempt to provide symmetry by allowing an omission to be considered to be either a time-domain or value-domain failure. The paper emphasises the importance of detectability as a property of failure but, in this scheme, it is only considered with respect to value domain failures.

4.6 FPTN

Fenelon's Failure Propagation and Transformation Notation (FPTN) [28, 29] is a method of studying and expressing the failure behaviour of complex systems. It is rather different to the other methods surveyed here, in that its main purpose is not the expression of the causes or external consequences of faults, but rather the study of how faults are propagated around a system, and how that system can transform the fault, for example through the effects of protection mechanisms.

FPTN uses a diagrammatic notation that was designed to resemble data-flow design and specification methods such as CORE [74] and Mascot [39]. It is modular and hierarchical; each module is represented by a box, with arrows indicating incoming failure modes on the left, and outgoing failure modes on the right. The notation also explicitly represents the *type* of each failure mode by appending a type letter to the fault name (e.g. *v* for value, *t* for timing, etc.). The notation is shown in Figure 29 and Figure 30. Fenelon states that an FPTN module can be considered as containing a "forest" of fault trees linking input failures to output failures; a more appropriate model might be to consider each module as containing a cause-consequence chart, since multiple effects of a single event can be represented.

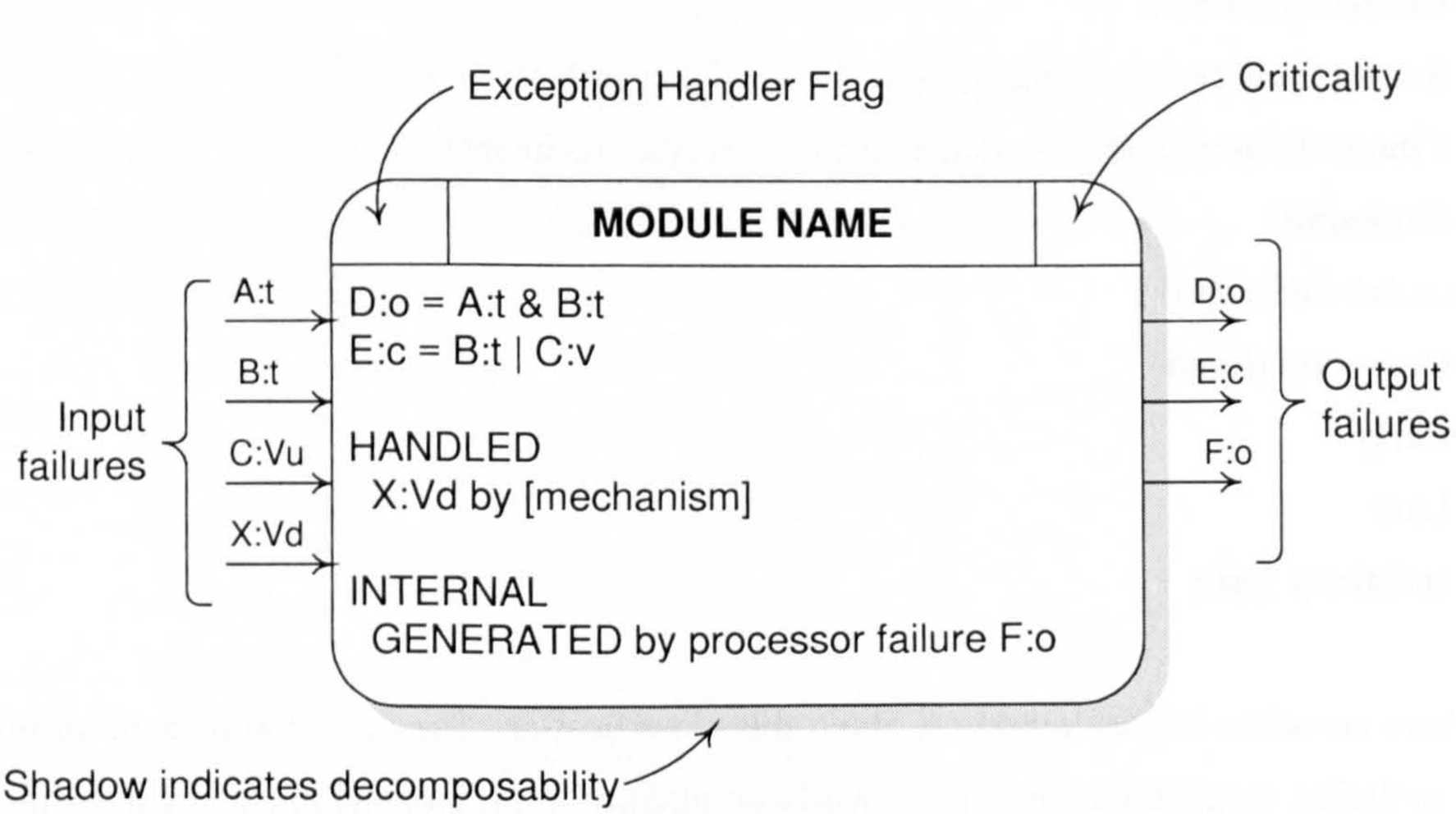


Figure 29 - A single FPTN module

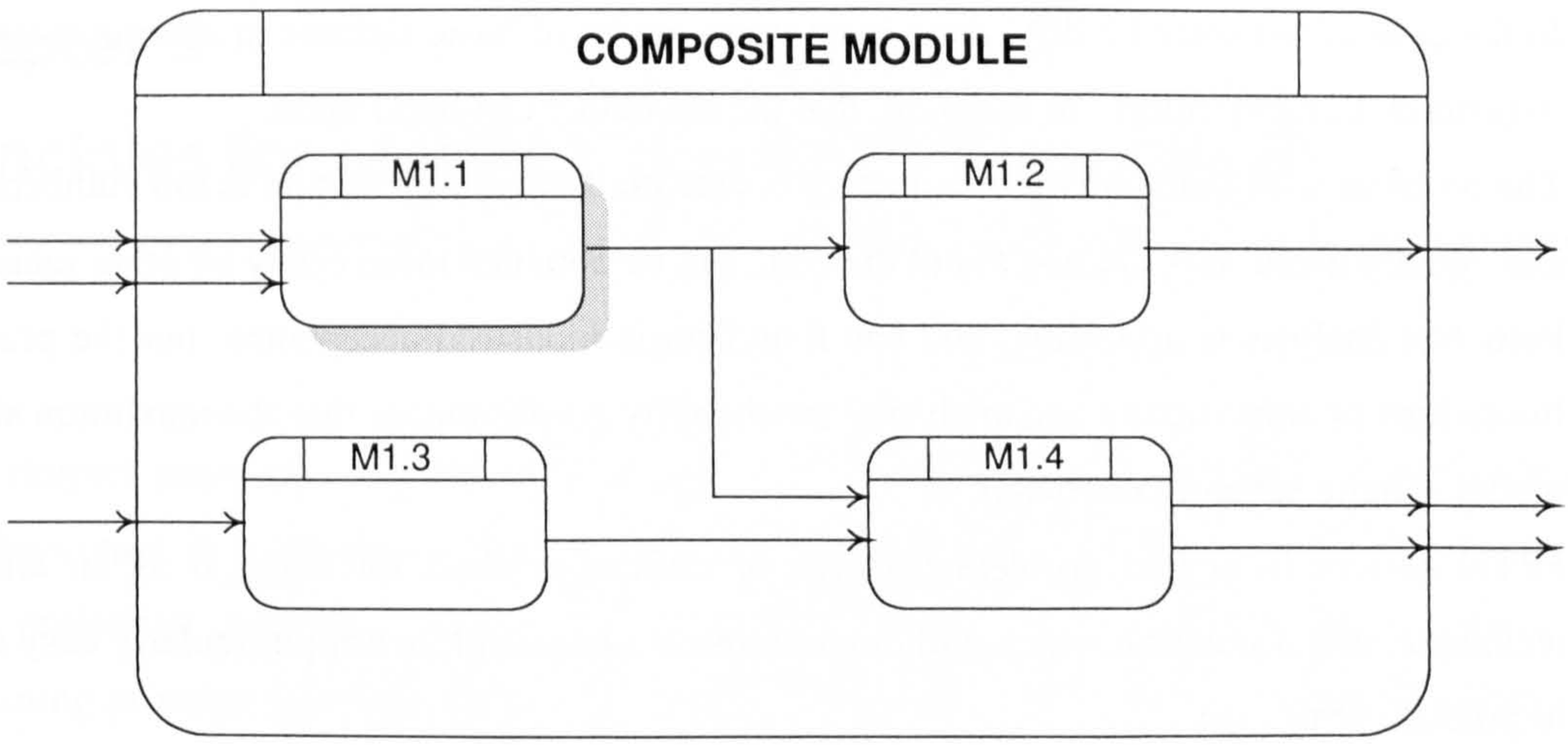


Figure 30 - FPTN module showing hierarchical decomposition

Fenelon proposes the use of FPTN notation as an architectural modelling tool in the early stages of a safety analysis process, to structure the analysis of the code. The system is decomposed into a set of FPTN modules, representing the operational structure of the system. The analysis begins by tentatively connecting the modules, using hypothetical failure modes that are believed to exist in the system. Then, starting at the bottom of the module hierarchy, a modified version of software fault tree analysis is used, taking the output failure modes the module is expected to exhibit as top events. This analysis will determine the faults that can be generated within each module, or result from a failure mode input from a module lower down the hierarchy. The input and output failures shown in the model are then modified, if necessary, to reflect the results of this analysis.

Unfortunately, no detailed method has ever been published for the application of the technique in this way, and the majority of the case studies which have used the notation have simply used it as a means of summarising, and representing in a very compact form, failure information which has been derived using more traditional methods.

4.7 Conclusions

This chapter has surveyed the few safety analysis techniques that have been proposed for computer system safety analysis. Apart from the widely applied Functional Failure Analysis, and more recently some HAZOP variants, none of these techniques has achieved any significant industrial acceptance, and most have not progressed beyond the status of academic toys. A number of reasons can be suggested for this:

- In the case of software FMEA, there is no good model of basic failures in computer systems in general, but particularly in software, that the technique can build upon.
- The problem with both software fault trees is essentially that the notation is too cumbersome, and equally useful information about the code can be obtained more easily by other means.
- Petri Net analysis is appealing, and has found some industrial acceptance, but the practical limitations of constructing and analysing reachability graphs means that the maximum size of model is quite severely restricted.
- FPTN suffers from two problems; a lack of defined method for using it as an analysis technique, and a notation which, although extremely powerful, is not particularly easy either to produce or to read.

More fundamental than all of these issues, however, is that none of these techniques seem to address the genuine needs of industry. Computer system safety analyses must either offer information that can be obtained no other way, or else must be the simplest, cheapest way of obtaining information, if they are to find a place in industrial development processes.

Chapter 5

Principles for computer system safety analysis

This chapter proposes a number of principles derived from the concepts in Chapter 3 and from the discussion in Chapter 4. The first five of these principles are general, and apply equally to both projective and confirmatory hazard and safety analyses of any type of system. The remaining principles are specific to computer system analyses, and attempt to define approaches to solving (or at least managing) some of the problems discussed in previous chapters.

5.1 Principle 1: Safety analysis must have value as part of the engineering process

This is a very basic and seemingly obvious principle that is, unfortunately, forgotten in a surprising number of industrial situations. There is no point in carrying out any sort of analysis unless it is clear that it is a necessary, value-adding step in the delivery of a safe system (or its supporting safety case).

This means that:

- **The right analysis must be used.** In defining a safety process, it is essential to understand what information is available, and what is required, at each point in the project, and to select analyses to match. This is a problem for computer systems, where the range of proven techniques is very limited and there may be no existing methods suitable for use with new specification and design notations, or novel system architectures.
- **The analysis should be timely.** This is most important for projective and requirement setting analyses. As well as starting the analysis sufficiently early, and integrating it closely with the design process, this also implies a requirement to select analysis techniques that can deliver results sufficiently quickly to satisfy project timescales.
- **The *results* of the analysis must be used.** Not only is an unused safety analysis a waste of time and money, it is also extremely demotivating for the analysts whose work has been ignored. This is a particular problem for the later, confirmatory, analyses that contribute to the safety case rather than to shaping the system; the safety case must be seen as an important deliverable in its own right, and must be seen to be used and maintained after delivery.

- **The analysts should understand what they are doing, and why.** Analysts must be aware of their role in the overall process; they must understand who the "customers" are for their work, what sort of skills and expectations their customers have, and must tailor their products accordingly. Lack of such understanding is a common problem, observed on many industrial projects.

  A typical example was the analyst who delivered to the design team a complex 20 page preliminary system fault tree, unsupported by any sort of commentary, but with the conclusion "no derived requirements identified". In fact, the tree contained many assumptions to do with redundancy, fault detection and management strategies and the ability to match historically observed component failure rates, all of which were effectively safety requirements. However, to the analyst, they were obvious from the fault tree and did not require further clarification or comment; to the design team, "no derived requirements" meant carte blanche to develop or modify the design as they chose.

In recent years there has been a significant improvement in the understanding of the need for, and role of, safety engineering in general. However, experience with real projects across a range of industries has shown that a worrying proportion of analyses are still carried out too late, or because they are required by standards, with no expectation that they will contribute anything significant to the project. This is really an issue in the specification and management of safety processes, but for those charged with selecting or developing safety analyses, it prompts consideration of how to maximise the value of any given technique. This is the motivation for the remainder of these principles.

5.2 Principle 2: Method is more important than notation

Safety analysis is a creative process, and the single most important attribute of a safety analysis technique is its ability to structure and direct the analysts' thinking. Effectively, the method for a technique encodes past experience of an effective approach. This means that a well-defined method, supported where necessary by additional rules or guidance, is vital. Good quality final representation of the analysis results is important for effective communication, but can never compensate for poor content.

A method should:

- Clearly identify the role and capability of the technique
- Provide, if possible, a means for estimating the resources required
- Identify the source information required, and any other prerequisites such as skills or knowledge of analysts
- Clearly describe the analysis procedure
- Note any options or alternatives
- Provide guidance, noting especially any known pitfalls
- Identify what should be recorded, with format and notation if necessary

A good test of how well a method is defined is how easy it is to teach engineers to use it. Experience on case studies and teaching various safety courses consistently shows that HAZOP is the easiest technique to teach; it combines a well-defined method with relatively simple concepts. Fault trees are also notable for a well-defined method, but some of the important concepts, such as the immediate cause rule, are quite difficult. The problem is that it is not immediately obvious *why* such rules are necessary, nor that they will necessarily lead to complete and correct trees.

Comparing these established techniques, where detailed, structured method descriptions are available, with newer techniques, especially for computer system safety analyses, it is clear that many of the proposals consist of powerful, well-defined notations, often embodying important concepts, but sadly unsupported by a well-defined and workable method. Fenelon's FPTN [28, 29], discussed in section 4.6, is a notable example of this problem. The published papers describe an extremely powerful and flexible notation, and suggest a number of alternate roles within the safety lifecycle for which it might be appropriate, but do not give methods for any of these roles. In effect, in its current form, FPTN is only suitable for representing information that has been collected using other methods, such as fault trees and event trees.

Leveson's software fault trees [52, 53] also suffer from this problem to some degree. The published papers and the "Verification of Safety" chapter of Leveson's book [50] present the basic principles, the template structures, and examples of their application, but the method descriptions are not sufficient for practical application of the technique. The deficiency becomes particularly obvious when attempting to teach the use of SFTA. Some of the templates (notably that for assignment) are not self explanatory, and rules for combination of templates (for example

the combination of assignment and procedure call templates required to model a function call) are lacking.

Summary

The main value of a safety analysis technique is the way in which it structures and directs the creative process of analysis. Notations that are so complex that they hinder this process, through either the need to complete excessive detail, or simply the time taken to draw complex diagrams, should be avoided.

5.3 Principle 3: Techniques should be as simple as possible

One of the notable features of several recent proposals for new safety analysis techniques (especially for computer systems) has been an attempt to provide means for capturing and recording every conceivable aspect of both the normal and faulty behaviours of a system in a single method and notation. This trend is typified by Fenelon's FPTN [28, 29], discussed in section 4.6, although there are a number of newer (as yet unpublished) approaches which allow, or require, the analyst to represent even more of the possible behaviour of a complex system in a single table or diagram.

The rationale for this trend is that, as systems become more complex, understanding their safety properties requires methods and notations which can model complexity. Fenelon's intention with FPTN was that it should be a single notation which was capable of supporting every phase of the system safety lifecycle from concept to documented product; further, the structure of the analysis should reflect the structure of the system itself, hence the "modular" approach.

There are some sound arguments in favour of this sort of integrated approach. The advantages of using the same notation throughout the lifecycle should include:

- only one notation for project staff to deal with, potentially reducing confusion, and minimising training requirements;
- only one tool is required to support safety analysis throughout a project;
- comparisons, whether between alternative design proposals or between requirements and achievement, are easier to make.

Unfortunately, practical experience and anecdotal evidence show that these highly capable notations simply do not fit the way that engineers on real projects need to work. The preference for simple techniques is well illustrated by cause consequence analysis (section 2.13). This

technique was first proposed in 1970, only a few years after fault trees were developed. It is undoubtedly significantly more powerful then either fault trees or event trees, whose characteristics it combines and yet, despite a number of relatively high-profile case studies in the nuclear industry in both Europe and Canada, it has not achieved any significant level of usage in any industry.

The practical problems hampering the adoption of techniques such as cause-consequence analysis and FPTN include:

- Speed and cost (especially of reworking)

  This is undoubtedly the biggest single issue. Particularly in the early stages of a project, ideas and designs typically change relatively rapidly. There is simply no need for a complete, detailed and fully accurate assessment; what is required is a good indication of the most significant issues as quickly, and at as low a cost as possible. Effort expended on the production of extremely detailed or attractively represented analyses is generally wasted, since it will soon be altered or superseded.

- Low perceived cost-benefit

  Related to the straightforward issue of speed and cost noted above is the problem of the engineers' perception of the value of safety analysis. An effect that was observed in all of the SHARD and LISA case studies reported in Chapter 7 and Chapter 8 was that engineers were almost always aware of the most significant problems and issues with a system or design before any formal safety analysis had been carried out. Although most of the project staff who participated in the case studies were convinced of the value of a rigorous safety process, there was significant resentment about being required to spend a lot of time on techniques which merely confirmed something which was already known. Clearly, the more difficult or time-consuming a method, the more unwelcome it will be.

- Over complexity

  There is a limit to how much information any one engineer can effectively manage. In the main LISA case study, described in section 8.4, participants jokingly referred to "one headful" as a measure of system complexity; in fact this is a serious issue. When anything becomes too large for a single person to fully understand, the potential for inconsistency, miscommunication and misunderstanding increases enormously. When studying systems that are already extremely complex, simple analysis methods have the significant advantage of not diverting concentration away from the subject system.

There is, perhaps, a case to be made for using one of these powerful notations as the basis of a (computer supported) integrated safety analysis toolkit. Such a toolkit would maintain the complete notation as its internal representation; the user would be presented with simple, familiar analyses such as FMEA or fault trees, with the computer performing a mapping between the (partially populated) underlying model and the user's view(s). As the project developed, options were confirmed and information was collected, the model would become more complete; the final phase would be to present the user with the integrated view for final checking and completion.

Attractive as this idea is, there are practical problems. It is difficult to map simple techniques such as FMEA or fault trees onto a complex model unless their use has been tightly constrained; this also limits the scope for allowing users to customise techniques to their specific needs. It is interesting to note that the ASAM project at York [31], which developed just such an integrated toolkit, started out with a number of relatively complex, tightly defined models, which were gradually abandoned or relaxed as the project progressed. The underlying model used for the majority of the safety analyses in the eventual production release of the toolkit represented everything as a set of conditions (which could be characterised as hazards, failures, events etc) and simple relationships.

Successful safety analysis techniques manage the problem of complexity by (implicit) abstraction, focusing analysts' attention to specific aspects of system behaviour. For example, the majority of safety analysis techniques either do not model time at all (e.g. FMEA) or, like fault trees, abstract away many details of timing behaviour. Apart from certain gates such as the priority-AND, or the sequence dependency gates suggested by Dugan [23, 24], in which *ordering* of events is significant, there is no explicit representation of time in a fault tree.

Summary

There are significant advantages to selecting (or developing) techniques which are as simple as possible for a given analysis requirement. They should focus on the most significant issues, should be quick and easy to use, and should not require disproportionate amounts of time to be spent on writing up or drawing out complex notations.

5.4 Principle 4: Techniques should guide without unnecessarily constraining

Principle 2 emphasised that the key role of any analysis technique is to guide a creative process. A successful technique will help the analysts to derive the necessary results as quickly and efficiently as possible, highlighting key issues, and providing a structure that will help to ensure completeness of the analysis. However, it is important that the technique is not excessively rigid, and does not prevent analysts from exploring novel ideas where this is appropriate. This is particularly important for the analysis of computer software, where designers can build anything that the design notation or programming language will permit; effectively, there are few of the physical laws that constrain the possible behaviours of traditional technologies such as hydraulics, electrical and mechanical systems.

There are various ways in which a technique can become too restrictive. The simplest is in the projective techniques such as functional failure analysis and HAZOP that use guide words to structure the analysis. These are intended to prompt the analysts to think about particular types or classes of behaviour. This clearly carries with it a risk of complacency ("we've done all the guide words so we must have thought of everything"), to which the obvious solution is to attempt to define guide words covering all possible situations that can be conceived in advance. As section 7.3 explains, this was a mistake that was made with the early work on the SHARD analysis method. Initially, a relatively large set of tightly defined guide words was proposed in an attempt to provide concrete guidance on the interpretation of system properties and resultant potential erroneous behaviour. Analysts working on the early case studies found that this was too restrictive; they were able to suggest important deviations which did not fit into the pre-defined structure, and it was eventually found necessary to revert to a smaller set of much more general guide words.

Balancing structure and guidance with flexibility is equally important in the more detailed, lower-level confirmatory analyses; fault tree analysis provides a good example. This technique has many rules (section 2.10), but these are really guidance to ensure that the analysis is correct and complete (e.g. immediate cause, primary – secondary – command, completion of gates), that correct tree syntax is observed (e.g. named intermediate events), and that sufficient information is recorded to ensure future comprehensibility. The method does not place arbitrary restrictions (e.g. there is no limit on the number of events that can be connected to a single gate), or force the analyst to work within predefined structures (e.g., there is no restriction on the structure of the

tree created during initial analysis; reduction to sum-of-products form for numerical analysis is a separate process). It provides a place marker (the undeveloped event) for use where there is insufficient information to complete part of the tree, which is also very useful for ensuring that analysis proceeds with a flow, following the most significant causal links, and leaving awkward or insignificant conditions for later completion.

Excessive constraint becomes particularly noticeable when computer based tools are used to support or record a safety analysis. There are few tools that permit the user to significantly customise standard forms, for example, and in a number of industrial studies, engineers have admitted that details are invented or defaults accepted because the tool will not move on until something has been entered in every field.

A related issue is the development of fully automated hazard identification and safety analysis tools. These generally combine a system modelling tool with an expert system based on a database of hazard and failure information. The creative process of analysis is reduced entirely to the programming of the expert system, and there is no opportunity for original thinking with each new system analysed. One of the biggest limitations of current tools is that they are unable to make judgements about the relative importance of issues identified, and the user is potentially left with the task of sifting through large volumes of undifferentiated output. There is undoubtedly value in such tools, particularly in the potential speed of analysis and guaranteed completeness, but their limitations must be realised. For the immediate future, such automated analyses, even of well-understood technologies such as chemical process plant will continue to need to be supported by creative manual analyses.

Summary

Techniques should not force analysts to work in structures that limit creative thinking. It is important to distinguish *rules* (essential steps of the method) from *guidance* (options, open-ended prompts and suggested approaches).

5.5 Principle 5: The role of the technique should be clear

Study of current practice in a number of industries shows that there is confusion about the role and capability of safety analysis techniques. In some cases, such as with the introduction of ARP 4754 [70] / 4761 [71] in the civil aerospace sector, this is due to changes in working practices required by standards; in others, there seem to be genuine misconceptions about what information a particular technique is capable of providing.

Perhaps the most notable instance of this problem is in the misuse of projective techniques, particularly functional failure analysis (section 2.7) and related methods. These techniques very explicitly investigate *projected* failures to assess the acceptability of design proposals and to set requirements for further design development. Whilst they may provide a basis for a safety argument, they do not provide any sort of *evidence* of achievement; this must come from other sources.

Unfortunately, in the software industry in particular, it has become common for functional failure analysis to be used as a form of evidence; because the design deals acceptably with all of the projected failures in the FFA, it is considered to have acceptable safety behaviour. This is, obviously, a flawed argument; evidence is required both that the real failure modes of the system are as predicted, and that the implementation behaves as required by the design in the presence of these failures. The primary cause of this problem seems to be the lack of suitable methods for obtaining data about real system failure modes, a problem examined in detail in the discussion of principle 7 (section 5.7). The difficulty of obtaining concrete data makes it very tempting to treat any available information as fact.

As well as producing weak safety arguments, using techniques in inappropriate roles leads to practical problems in conducting the actual analyses. One of the most interesting of these concerns the "stopping problem"; analysts ask what criteria should be applied to decide when "enough" analysis has been done. If the role of the analysis is clearly defined, this question should be unnecessary; it will be clear when the required product has been delivered. The role of projective analysis is essentially to answer questions such as "which is the better (best) alternative?", "can this design produce a safe solution?" or "what must be done to ensure this system meets safety targets?". In some cases, answering these questions may actually require very little analysis – in general negative answers (e.g. rejection of designs with serious flaws) require less work than positive ones. Unfortunately, this is not allowed for in many safety processes, which require analysis of every function in a design, without recognising that this may add no useful information.

For confirmatory analyses, the stopping criterion will be the delivery of sufficient evidence to complete every part of the safety argument. In many cases, there is no practical alternative to analysis of every component; for example, a subsystem supplier may not know which failure

modes of his system are important in the complete platform safety argument, and must therefore supply data about every known failure mode.

Summary

Appropriate and efficient use of safety analysis is an issue for the specification of safety processes, requiring understanding of the capabilities of each technique. Documentation of analysis methods should include a clear statement of the role(s) for which it is suitable, required source material and expected products.

5.6 Principle 6: Safety analysis starts at the system level

Safety is a property of a complete system (platform) in a given context. This means that analysis of deviations and failures in subsystems and components must always be considered in terms of their contribution to system level effects. Information about component or subsystem failure modes that does not relate to system level hazards is a specific form of component data, *not* safety analysis.

This principle presents serious problems for critical computer systems, and two distinct cases can be identified. The first of these cases applies to completely bespoke development, where every aspect of the computer system hardware and software is custom designed for a specific application. Here, the problem is that, very often, the computer is the most complex part of a system, and needs a longer development time than mechanical, electrical and other components. There is therefore pressure to start the development of the computer system as early as possible, with the attendant risk that high-level analyses will be skimped, or carried out in parallel with the design activities which they should be guiding. The computer system design then has to be modified when the analyses are eventually delivered and identify requirements that it does not satisfy. Since by this stage considerable effort has been expended, there is reluctance to make significant changes, so "fixes" are implemented, often leading to a system which is more complex and less robust than could have been achieved if the complete requirements had been identified in time. Worse still, there may even be pressure to carry out "safety analyses" on the design before the system level information is available to provide the necessary context.

This is essentially a management problem. Analysis and design must be seen as an integrated process, and managers must be prepared to resist pressure to start design work until preliminary analyses are complete. There are, however, technical challenges, particularly the need for efficient projective analyses to establish requirements as quickly as possible.

The second case is where pre-existing components such as hardware, operating systems or other software are incorporated into a design. Here, the problem is very much like that for component suppliers in other engineering disciplines; sufficient data must be available about failure modes that the system integrator can select appropriate components and complete a system level safety argument. However, although similar in principle, the complexity of computer system makes complete and accurate description of failure behaviour practically impossible. It is therefore probable that some degree of re-analysis will be required once the complete system context is known, and this implies requirements for the availability of design information. If this cannot be obtained, reverse engineering is (usually) possible, but costly.

The main technical challenge of this case is to develop improved means for specifying complex safety requirements, and describing the properties of completed components, so that systems designers can select appropriate components, and work around their known limitations if necessary. To some extent this is being addressed by work such as that of Jones [40] with his concepts of "relies" and "guarantees", properties which software components can be guaranteed upon to provide, so long as they can rely on specified characteristics of other components.

An alternative approach where detailed information about the failure behaviour of a particular component or subsystem is not available is to assume that it will behave in the worst conceivable way in all circumstances, and design to contain this. This conservative abstraction makes complete analysis at system level possible, and has the added benefit that modifications to the component or subsystem do not affect the system level safety argument. Unfortunately, this approach is normally only viable for relatively small and extremely independent system components. Containing the worst conceivable behaviour of large or tightly coupled components is either infeasible, or requires prohibitively complex or costly mechanisms.

Summary

Safety is a system property, and all analysis must ultimately demonstrate how component level functions and failures contribute to the system level behaviour. As the "outside in" model of safety analysis discussed in section 3.4 shows, it is not necessary for analysis always to proceed strictly top down, but some initial high-level analysis is always required in order to identify high-level safety requirements.

5.7 Principle 7: Projective analyses are key to software safety

This thesis has made a strong distinction between the projective analyses used early in the development lifecycle to set requirements, select between alternatives or determine the overall acceptability of a proposal, and the confirmatory analyses used later in the process to assess achievement and contribute evidence to the safety case.

Software is an expression of logic and mathematics which does not degrade after it has been implemented. It may be affected by failures of the hardware on which it is executed or which supplies its inputs, but the software itself can only contain flaws as a result of incorrect specification or implementation.

The available evidence shows that by far the most significant cause of software failing to function as intended is errors in the elicitation or specification of requirements. The most common safety related problem in current software development processes is that of complete omission of safety requirements from specifications which focus primarily on the intended functionality. It is therefore vital that the development process includes activities to review both the completeness and appropriateness of safety related specifications. This is precisely the area that projective analysis addresses, hence its importance for software. Techniques such as functional failure analysis and HAZOP provide the structure for a systematic, hazard-directed review of a design with respect to *intent*, providing feedback on the specification as well as design activities, at a stage in the process when correcting errors is relatively inexpensive. HAZOP has the additional advantage that, being structured around the flows within a system, it requires analysts to think about a system in quite a different way than the normal functional view.

As the process in Figure 31 shows, projective analyses establish the conditions which are tested by later confirmatory analyses, which for software are invariably deductive. Inductive techniques such as FMEA are theoretically capable of following the propagation of low-level failures through to system level effects that can be compared against intent. Unfortunately, as the discussion in Chapter 4 shows, there are currently no workable techniques that fulfil this role for software – hence the reliance on deductive methods.

Used in this confirmatory role, deductive analysis can address two questions:

1. are the identified hazards managed acceptably, i.e. are the potential hazard causes sufficiently improbable and / or is there appropriate detection and mitigation? This is the primary safety assessment, corresponding to the demonstration that achieved behaviour is "no worse than" requirements shown in the simple "V" safety lifecycle model in section 2.2.
2. does the behaviour revealed by the analysis agree with the way the designers claim the hazards are managed? This is effectively a design and implementation review step. If the analysis demonstrates that the system behaviour is not what the designers claim, there is a fault in the process; even if the system behaviour appears acceptable, this should be a cause for concern.

The important point to note is that both of these questions start from the identified hazards and requirements resulting from the projective analyses; deductive analysis cannot compare achievement against the original intent.

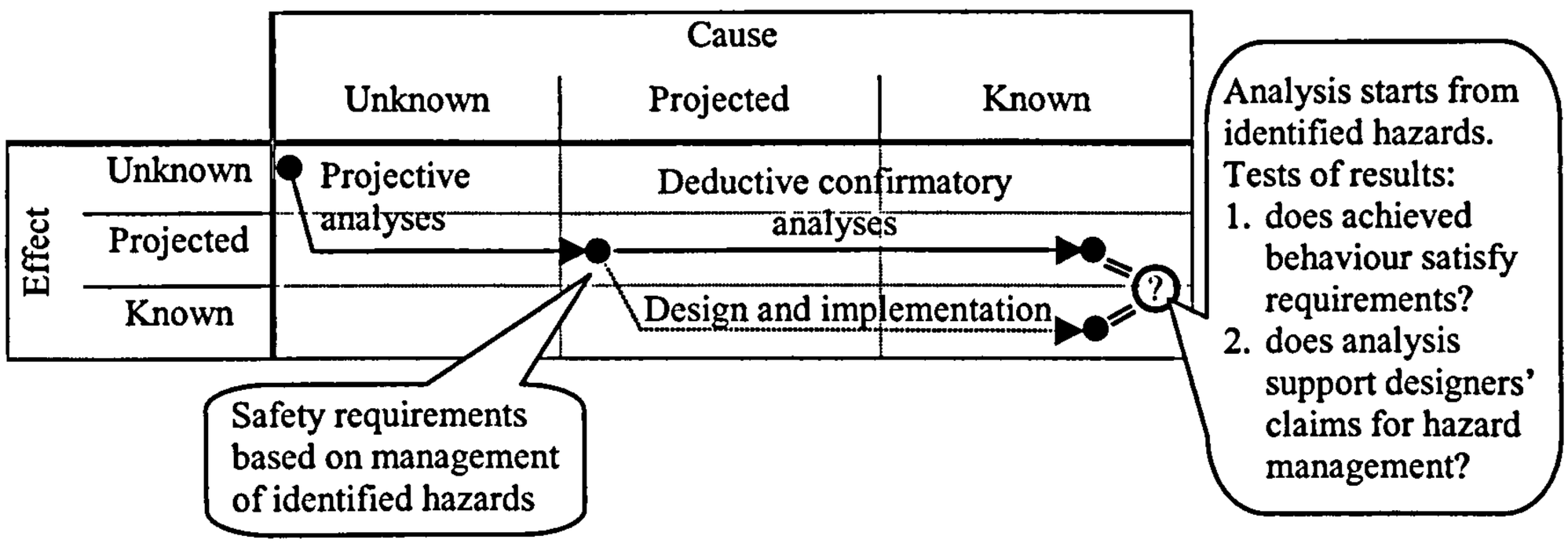


Figure 31 - Process with deductive confirmatory analysis

It is also important to note that, for software, confirmatory analysis using techniques such as software fault trees is effectively another form of static analysis. Because the software does not degrade, there is nothing that software fault trees can reveal which cannot potentially be identified using methods such as weakest preconditions, path coverage or data dependency analyses. The one advantage software fault trees has over such techniques is that it is explicitly hazard-directed. This focuses attention on safety issues and, if the proportion of safety critical functionality in a program is very small, it may be a cost-effective way of limiting the amount of analysis required. However, there is no fundamental reason why other methods should not be adapted for use in a hazard-directed manner, and a number of research proposals (such as Clarke and McDermid's weakest preconditions for failure [18]) have been made in this area. These

approaches may ultimately prove far more tractable then fault trees, which are limited by their inability to represent program constructs such as loops (c.f. the discussion in section 4.3).

Summary

Two factors make projective analyses especially important for the achievement of software safety. The first is that they provide a systematic process for reviewing safety related requirements, and the available evidence shows that the specification process is the source of the majority of flaws in software. The second is that there are currently no analysis techniques which are capable of comparing the safety related behaviour of finished software with the original intent; the deductive techniques which are available for examining completed code are simply hazard-directed static analyses.

5.8 Principle 8: Safety analyses must consider hardware and software

As the discussion of principle 7 in section 5.7 above has observed, for software on its own there is little that a confirmatory safety analysis can deliver which cannot be obtained from other types of static analysis. However, as safety is a system property, it is not acceptable to assume perfect execution of the software; potential unintended behaviour of the underlying hardware must also be taken into account. This is something that conventional static analyses of software do not allow for.

In current industrial practice, the usual method of including computer hardware failures in safety analyses is simply to assume that any failure within the computer system will be sufficient to cause any failure mode of concern at the computer's outputs. Thus, in system level fault trees, it is common to see a basic (or undeveloped) event called simply "computer hardware failure" (see, for example, the fault trees for inadvertent aircraft braking developed in figure 5.1-2 of ARP 4761 Appendix L [71]). If the fault trees are quantified, a conservative rate for this event can be derived by summing the individual failure probabilities of the computer system components. Whilst this is clearly sufficient to ensure safe treatment at the system level, it is of no use to software designers, who need to know what the symptoms of various hardware failures will be in order to develop detection and protection strategies.

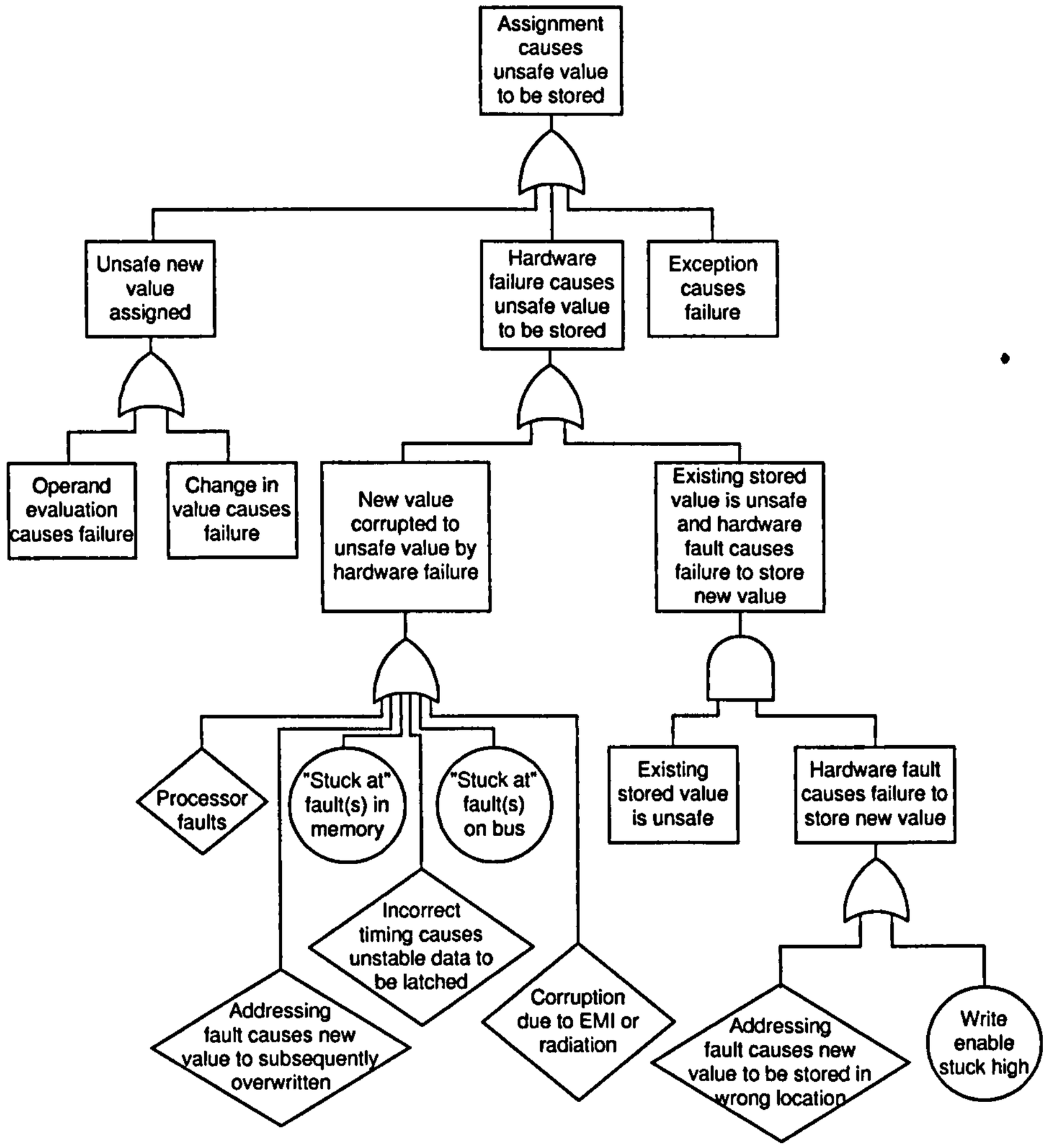


Figure 32 - Fault tree for assignment showing some hardware failure contributions

The major obstacle to a detailed combined safety analysis of hardware and software is yet another instance of intractable complexity. To illustrate this, consider Figure 32, which shows Leveson's software fault tree template for the assignment statement [52] expanded to include just a few of the hardware failures which could conceivably cause an unsafe value to be assigned. This is clearly a completely unmanageable approach. This fault tree exhibits side effects ("addressing fault causes new value to be stored in incorrect location" – what will be corrupted at the location where the new value is actually stored?), there are additional software events to study ("existing stored value is unsafe"), and to study every statement in even a small section of code at this level of detail would be impossible.

It is obvious that all of the hardware events represented in Figure 32 are common cause failures that could potentially affect any step of software execution. Some, such as faults in particular memory devices, will have a limited scope (they can only affect data stored in that specific

device); others, such as bus or processor faults, can potentially affect any operation. This means that detailed inductive analyses are as intractable as deductive approaches; it is easy to identify individual hardware failures to study, but impossible to consider every condition of the system in which they might arise.

In fact, since the state space of even simple programs is very large, there can be no tractable approach (at least for manual analysis) based on evaluating the effect of individual hardware failures in every possible system state. This rules out methods based on conventional event or state models. The challenge for safety analysis is to find meaningful ways of identifying the critical parts of the system and modelling the potential impact of hardware failures at a more abstract level.

Unfortunately, classical common cause / common mode analysis techniques are little help. Zonal analyses are more plausible, since it is possible to apply them at a higher level of abstraction, but it is not clear what would constitute a zone within a computer system – or, indeed, if they can be considered to exist at all in single processor systems.

Summary

At the system level, computer hardware failures are generally managed by a conservative assumption that any internal hardware failure will be sufficient to cause any failure mode at the computer's outputs. This is not sufficient for the needs of software designers, who need more precise modelling of the effects of hardware failure.

The development of tractable, effective methods to integrate detailed hardware and software analyses is still an open research issue, but existing analysis techniques can provide useful information to computer systems designers provided that care is taken in their application. At the very least, hardware failure modes should be included when considering possible causes of deviations in HAZOP type projective analyses of computer systems; at this level of abstraction, they do not make analysis intractable.

5.9 Principle 9: Techniques should use familiar concepts and models

Computer system safety analysis needs to be an integral part of both the overall system safety process and the computer hardware and software development process. This means that the safety concepts on which it is based should, so far as possible, be the same as traditional system level safety analyses; similarly, the models of computer systems the analyses are based on should be standard models that software and hardware engineers will be comfortable working with.

Section 3.3 has identified a range of concepts which underlie a number of the most important system level safety analyses. The majority of these can readily be applied to computer systems, and many of the more concrete concepts, such as fault (error) propagation, detection and mitigation are already familiar to software engineers. The discussion of principle 7 in section 5.7 has emphasised the importance of projective analysis to computer system safety, and the fundamental principles of projective analyses such as the use of failure classifications as prompts are also readily applicable to computer systems.

One of the hardest concepts to explain, as was experienced on some of the SHARD case studies, is the idea of failure with respect to *intent*; the problem is making it clear that intent means *what is really needed for safety*, and not what the designer believes is right.

The need to base safety analyses around familiar models of computer systems is primarily a practical issue; analysis proceeds faster and more accurately when the participants can easily interpret the system representation they are working from. In practical terms, this means that, unless analysis techniques are very general and can readily be interpreted for a range of specification and design notations, it will be necessary to produce guidelines which "tailor" the method for new representations. Leveson's software fault trees [52, 53] are an extreme example of this, requiring new templates for each new language.

The principle of working in terms of familiar concepts also applies to the final presentation of computer system safety analyses – particularly the results of confirmatory analyses that will be incorporated into the system safety case. This does not necessarily mean sticking to traditional notations; indeed, Fenelon [29] reports that systems safety engineers were generally averse to software fault trees, since the template based approach gave them an unfamiliar structure. However, the results of any analysis that is not based around the investigation of hazards using

combinations of events and conditions to model causes and effects will require careful explanation.

Summary

One of the principal aims of safety analysis is to help engineers to take a fresh look at the systems they are working on. However, this does not imply a need to force people to work with new and unfamiliar concepts. To ensure that safety analysis can be effectively integrated with both the systems safety and computer system development processes, techniques should be based around standard safety engineering concepts and, wherever possible, should work directly from system models which are produced by the specification and design activities.

5.10 Conclusions

This chapter has proposed, discussed and attempted to justify some basic principles for selecting, and more especially designing, computer system safety analysis techniques. Many of these principles are unashamedly based on frustration with attempts to define safety processes or techniques that do not recognise the pragmatic needs of industry. Perhaps surprisingly, it is not only academic proposals that display this lack of understanding. A number of industry-led standards have also been criticised for impractical requirements. A current example is IEC 61508 [38], a supposedly cross-industry standard, which has been heavily criticised because its criticality assignment rules are based on a model of control and protection functions familiar to the process and nuclear industries, but which is inappropriate for other application domains such as aerospace systems.

The principles defined here do not claim to define all the requirements for a successful computer system safety analysis technique; however, they provide some general "ground rules", and identify a number of key issues which new proposals must address.

Chapter 6

Putting principles into practice: background to the development of new analysis techniques

From the discussion of the principles in Chapter 5, and the review of computer system safety analysis literature in Chapter 4, three significant deficiencies in the range of available techniques are readily apparent. These are:

1. Although a small number of projective computer system safety analyses have been proposed, the only technique that is widely applied is a variant of Functional Failure Analysis. There is a need for alternative techniques that can fulfil this role.
2. There are no inductive analyses capable of comparing the safety related behaviour of finished code with the original intent.
3. There are no practical analysis techniques that allow the detailed investigation of the interactions between software and the underlying hardware.

Two of these deficiencies, those in projective analysis and hardware / software interactions are addressed by new analysis techniques which are developed in Chapter 7 and Chapter 8. This chapter discusses the background to the development of these new techniques, and explains why these particular subjects were chosen.

6.1 Projective analysis for software safety

The research work described in this thesis began with an informal survey of the current practices and perceived future needs of companies developing safety critical software, primarily avionics and railway signalling systems. The conclusion of this survey was that the most immediate technical problem that the companies were facing was in the identification of software safety requirements sufficiently early in the development process. In many cases, projects carried out thorough hazard identification, risk assessment and preliminary system safety activities, but were unable to clearly identify software safety requirements from the system level information. One of the most common symptoms of this problem was that, as top level software designs were proposed, the software teams struggled to decide what integrity level should be assigned to the functions they had defined.

The original motivation for investigating projective analyses for software was therefore to provide software developers with a means to assess their design proposals against system level requirements and intent. However, a number of (then recently published) papers on software safety had argued persuasively for the development of techniques which could satisfy the safety analysis requirements of every stage of a software project with a single method (or, at least, a single notation).

The starting point for the work was therefore the proposal of a set of objectives for an ideal computer system safety analysis technique:

1. The method should be capable of being applied at all stages of the system design and development lifecycle from initial design through to validated implementation.
2. The method should not involve an excessive increase in the work required at early stages of the design. Ideally, it should allow the system designers to identify quickly which areas of the design are most critical, and concentrate further analysis work on those areas.
3. The analysis should help drive design development through the comparison of alternatives and the refinement of specifications.
4. It should be possible to have a high degree of confidence that thorough application of the method will lead to consideration of all credible failure modes.
5. The analysis should be in a form which allows the design to be checked and approved incrementally, permitting closer integration of design / implementation and verification / validation activities.
6. The results of the analysis should be in a form that is suitable for inclusion in a safety case.

The development of a completely new analysis technique that could satisfy these principles, with the attendant problems of technology transfer – particularly the difficulty of convincing potential users of its value – was considered undesirable. Instead, it was concluded that the best approach was to identify an existing technique, possibly from completely outside the field of computer systems safety, which satisfied as many of the requirements as possible, and to modify it as required for the new role. There was no existing technique that could be used throughout the system development lifecycle, as required by the first objective, so the criterion was changed to finding a technique which could operate in all four of the roles (exploratory, inductive, deductive and documentary) identified by Fenelon [28], but with an emphasis on the needs of the early phases of development.

Objective 6 was also problematic, and it soon became obvious that no technique could be expected both to operate effectively in an exploratory role and, at the end of the lifecycle, to deliver concrete data for inclusion in a safety case. A more reasonable requirement was that the technique should provide a structure that could form the basis of a detailed safety argument.

Even with these concessions, no existing safety analysis technique was able to satisfy all of the objectives, but the role in which HAZOP was used in the chemical process and nuclear industries was closest to the type of role envisaged for the new computer system analysis. The then recent publications by Burns and Pitblado [12], Earthy [25] and Chudleigh [15] showed that other researchers had also recognised the potential of this technique. As section 4.2 explains, although the use of variants of HAZOP for the analysis of software specifications and designs represented an exciting new suggestion, the problem with all of these early publications was that they concentrated on describing and explaining traditional HAZOP to an audience of computer scientists. None of these papers presented detailed technical proposals for the type of software analysis that seemed to be required, and the most thoroughly developed work (that of Burns and Pitblado) was flawed.

One of the important features of HAZOP is that it is structured around flows. This means that, in the context of a chemical plant, the first focus of analysis is the properties and behaviour of the flows in the pipelines connecting the major components (storage tanks, reactor vessels, pumps etc.) of the plant. These active components are brought into the analysis as causes or targets of unintended behaviour in the flows connecting them. There appeared to be a number of potential benefits to using a similar approach for computer systems, the most significant of which was to force engineers to take an alternative view to the normal process model used in most design methods. Other suggested benefits of considering information flows between components included:

- Earlier analysis

  In many design methods, the interfaces between parts of the system are defined before the component implementation is finalised. A suitable analysis of these interfaces could provide a useful input to the later stages of design elaboration.
- Simpler specifications

  The intended behaviour of an interface is likely to be simpler to both specify and understand than that of an active component.
- More restricted failure behaviour

In general, the failure modes conceivable for interfaces are more restricted than those of active components. This may help to contain complexity and limit the size of the results.

The papers by Burns and Pitblado, Earthy and Chudleigh had largely concentrated on the process of HAZOP, detailing aspects such as the role of the technique, membership of the HAZOP team, and the conduct of HAZOP meetings. It was therefore decided to start by investigating three specific technical issues:

- the selection of an appropriate set of guide words for use with computer systems
- the identification of specification and design notations with which HAZOP would integrate well, and
- whether an analysis structured around the flows between components could perform as well as one structured around the active components of the system.

Chapter 7 describes the resultant development of SHARD (Software Hazard Analysis and Resolution in Design), and explains how these original intentions were modified as case studies and other practical work shaped the principles defined in Chapter 5.

6.2 Analysis of hardware / software interactions

Unlike the rather uncertain beginnings of SHARD, the second new safety analysis technique was developed in response to a very specific requirement. British Aerospace Military Aircraft and Aerostructures (BAe MA&A) had begun work on a cockpit display system, which was one of the first military applications to employ software segregation of functions of different integrity levels on the same processor. The production of safety evidence for this system would require not only an evaluation of the interactions between the application code and the segregation system software, but also a detailed investigation of the possible effects of hardware failures.

This project was therefore directly relevant to one of the specific deficiencies identified by the safety analysis principles, and also extended the research to address some of the safety issues of using operating systems in safety critical applications.

6.2.1 Use of operating systems in safety critical applications

Traditionally, safety critical computer systems have tended to be bespoke, with custom-written software running on a single, often also custom-designed, hardware platform. In mass-market products such as domestic appliances and automotive applications, the most significant design

driver is normally unit cost, which is reflected as a requirement for minimal hardware and compact, efficient code. These have typically been easiest to achieve with a fully bespoke system. In the aerospace and military sectors, factors such as size, weight, power consumption and heat dissipation have been dominant. There has also been a requirement to use "hardened" components, capable of operating outside the temperature ranges and E.M.I. levels acceptable to normal parts. Again, these factors have tended to predicate bespoke development.

Although it has not been the primary driver, safety engineering has benefited from the bespoke approach. Every part of the system is specified, designed and implemented for the application, and this allows a very high degree of visibility of, and control over, both process and product. From the point of view of safety engineering, specific advantages of this approach include:

- the ability to examine and provide input to every part of the software (and potentially hardware) design
- the ability to include safety-specific features at all levels in the software and hardware
- access to designers and engineers at all levels
- a design that is tightly focused on the particular requirements of the application. This can permit relatively simple designs which are easy to analyse
- no, or very few, design variants to assess.

In some markets (e.g. automotive), hand-crafted assembly code has been common, allowing very direct assessment of the code that will actually run, without the need for complex assessment and certification of compilers etc.

It is becoming generally accepted that completely bespoke development is no longer viable except for the most critical of applications. There are many reasons for this change, of which cost is probably the most significant. As hardware costs have fallen, development (particularly software development) is now the dominant cost, even in mass markets. Other significant factors in the move away from bespoke systems have been:

- Reduced need for bespoke hardware

  Computer system components have become generally smaller, faster and more highly integrated. Chips with extremely low power consumption and heat dissipation have been developed to meet the needs of the portable computing market. This means than there is no longer so much need for bespoke hardware to achieve the required functionality within the physical constraints imposed. This has even begun to alter the perceived need for hardened components, with suggestions that it would be cheaper and more effective simply to rely on replication where adverse environments are expected to impact hardware reliability.

- Requirements for maintenance and upgrade

  There are many factors that make the maintenance and upgrading of safety-critical systems problematic. These include:

  - Hardware obsolescence

    As the pace of general computing hardware development has accelerated, components used in bespoke safety critical systems (which typically have a long in-service life compared to, say, desk-top computers) become obsolete and difficult or impossible to replace during the service life of the system. For example, the Eurofighter aircraft, which is not even in service yet, uses components that are no longer generally available.

  - Performance limits

    Many bespoke systems are designed such that the hardware is just sufficient to run the application software. New requirements for additional functionality can be impossible to meet without costly migration to new hardware.

  - Inadequate design and safety information

    Many system upgrade projects have been complicated by inadequate knowledge of the original design rationale, assumptions and essential safety related information. This has occurred where systems were developed to outdated working practises with less documentation than is currently considered necessary, or where a small development team has not adequately recorded knowledge which everyone in the team shares, but which may not be obvious to an outsider. To ensure that changes do not cause unexpected problems can require a serious reverse engineering task (c.f. for example SHARD case study 1 in Chapter 7).

Related to the problems of hardware obsolescence and performance limits is a desire to find ways to allow safety critical systems to follow the general computing "power curve". Because of the need to use components with a good track record of reliable operation, safety critical systems projects are usually restricted to using hardware that is no longer state of the art even when design begins. By the end of a lengthy development, the product may be a couple of generations behind leading technology, giving rise to a perception of poor performance when compared to contemporary non safety critical systems.

There are many possible approaches to reducing development costs and addressing some of these issues. These include:

- moving to program generators
- increased use of standard components and hardware
- increased reuse of software components
- use of commercial-off-the-shelf (COTS) software.

All of these offer significant benefits in one or more of the problem areas identified, but all also pose new challenges for achieving and assessing safety.

Essential to many of these changes will be the use of system architectures more similar to those found in general computing systems. The most important requirement on the safety process will be the ability to produce satisfactory safety arguments for operating systems which can provide flexible scheduling, guarantee independence of separate processes on the same processor, and provide cross-platform commonality to allow applications to be migrated to new or upgraded hardware. Whilst significant progress has been made in some areas (notably in the guaranteeing of timing properties of different scheduling schemes on a range of architectures – see, for example [2, 42, 46]), the design and analysis of operating systems presents many new challenges for safety engineers.

6.2.2 Monolithic Software vs. Operating Systems

A notable feature of current-generation safety critical systems (i.e. those in active service) is that the great majority are monolithic software systems, with the application software interfacing directly with the hardware and providing its own scheduling, device drivers and support functions, with no discernible operating system layer (Figure 33). These systems almost invariably have static address maps (i.e. data is stored at fixed locations in the memory map, with no paging, address translation or other dynamic memory management). They also typically have relatively primitive scheduling, either running simply as one large program, or using a simple static cyclic schedule.

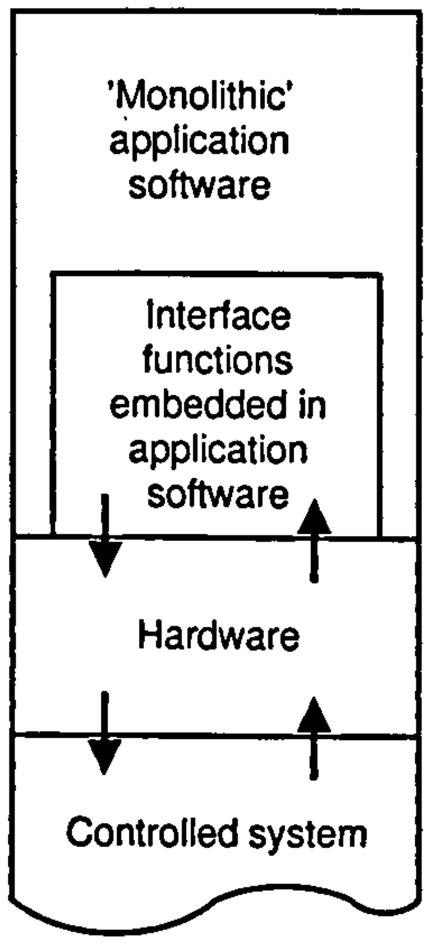


Figure 33 - Monolithic system structure

A typical example of this type of system is the engine management systems found on Ford cars built in the early 1990s (although it should be noted that these management systems were not considered to be safety critical at the time they were developed). These systems were based around custom-designed processors, with I/O features that are more typical of microcontrollers than standard microprocessors. The software was custom-written in assembly language, both to maximise the performance of the system and to minimise code size (necessary because of the limited addressing capacity of the processor). The code was organised largely as a single program calling subroutines, with minimal interrupts for critical timing events. I/O and device management was all performed directly within the application code. RAM was extremely limited, and all variables were statically assigned to fixed locations. Changes to the memory map or execution order of the functions could necessitate rewriting several code modules because of the optimisations that relied on such structural features.

This type of architecture has both advantages and disadvantages for safety engineering. The principal advantages are:

- Obvious relationship of hardware and software functions

  This is perhaps the biggest single advantage of monolithic systems. Since every function is part of the application code, it is easy to trace which hardware components and lines of the software are involved in any given function. Also, the static memory management and simple scheduling mechanisms mean that system behaviour is limited and predictable. This greatly facilitates and simplifies the analysis of critical functions.

- Team knowledge of both hardware and application software

  In a monolithic development, it is common for the same team of developers and analysts to work on all parts of the software, and to have detailed knowledge of the underlying hardware. This can assist in providing a "seamless" approach to safety.

The disadvantages include:

- Lack of partitioning

  Many system-level safety arguments rely on demonstration of independence between diverse or redundant components. With a monolithic system, this is very hard to demonstrate (i.e., the application cannot easily be "partitioned", and claimed to consist of independent software processes resident on the same hardware). The typical response is to physically partition the application to separate processors where independence is required. This is also the most common approach for managing functions of different integrity levels; if it is not acceptable to develop all of the software to the same integrity level so that it can be co-located, the different integrity levels are allocated to separate hardware. Whilst this is simple and usually satisfactory from the safety viewpoint, it can be inelegant and extremely inefficient, requiring high levels of inter-processor communication, or resulting in very poor load balancing.

- Small changes can necessitate huge amounts of re-analysis

  The tight coupling of the hardware and software, and lack of ability to demonstrate independence between functions, can mean that relatively small design changes can result in a requirement for a disproportionately large re-analysis of the safety properties of the system. For example, if a safety argument relied on a proof that a critical variable could never be overwritten, the entire proof may have to be repeated if the variable is moved to a different location in memory. Perhaps the most extreme cases of this type of re-analysis are those related to timing behaviour, where reorganising a fixed cyclic schedule can require complete reworking of potentially extensive calculations and proof of timing properties such as ability to meet deadlines.

- Problems of ensuring safety after maintenance or upgrade

  Modification of monolithic software systems is also problematic, again largely as a result of the tight coupling of hardware, software, timing characteristics etc. Side effects of apparently minor changes can be extensive and potentially dangerous.

The primary functions of an operating system (Figure 34) are to manage the interface between software tasks and the underlying hardware, and also, in some cases, to manage interactions between a set of tasks. It provides facilities for allocating resources such as memory, access to hardware devices and I/O to tasks and ensuring that they do not interfere with each others' use of these resources. It also provides a scheduling regime which determines which tasks will be executed and in what order, and may allow different priorities of task, so that critical tasks can pre-empt lower priority activities.

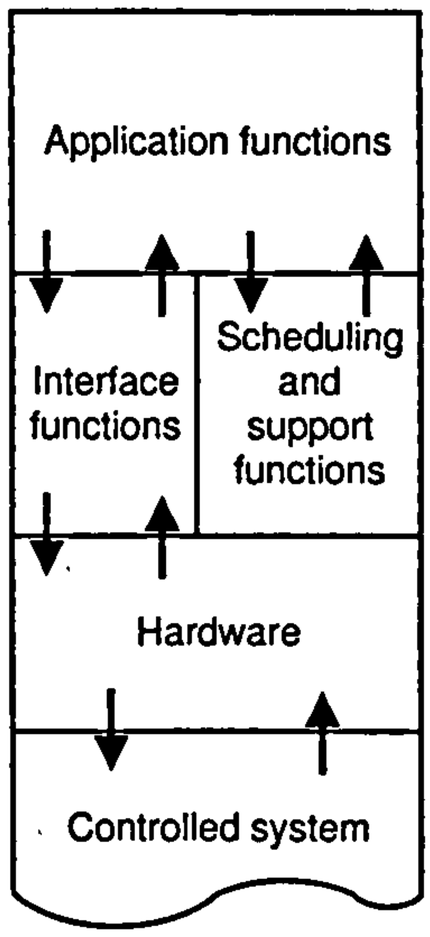


Figure 34 - System structure with operating system

In this respect, operating systems, almost by definition, address to some degree all of the disadvantages of bespoke systems noted above. Partitioning is supported by the allocation of resources to tasks and, if strong enough, this should enable the re-analysis required after modification to be limited to effects within the single task or group of tasks affected by the change. Portability may be greatly enhanced, provided that the same operating system application program interface is available on different hardware platforms, as the uniform interface it provides removes (or at least substantially reduces) the dependency of the application software on the hardware.

Unfortunately, there are disadvantages to the use of operating system in critical systems. The overall size of a system consisting of application code and an operating system will usually be substantially larger than an equivalent monolithic solution. Since operating systems add a level of indirection between application code and hardware, the analysis of software / hardware

interactions becomes more complex as well as larger. Operating system facilities such as memory management, scheduling and partitioning become as critical as the most critical task they are managing, and so must be designed, developed and assessed to equivalent levels of integrity. Few currently available operating systems fully meet the integrity requirements for systems with serious safety implications. Also, there will still be a substantial analysis task when an operating system is migrated to a new hardware platform.

6.2.3 The safety challenges of operating systems

Computer hardware and operating systems are inherently single systems. It is simply not feasible to have a "standby operating system" that is fired up in the event of failure of the primary system, in the way that a standby generator might be used in an electrical supply. This means that the basic model of software safety management must always be rather different to physical components. Even if independence of failure behaviour of functions in the application software can be shown, there will always be a need to implement replication at the level of complete hardware units to provide for hardware or operating system failure. Even this does not avoid systematic failures in the design or implementation of the hardware and operating system unless diverse units are used.

This observation implies that there might be little point in demonstrating independence of software components within a single system since replication at the "box" level is required anyway. However, there are significant benefits to be obtained in the areas of performance and flexibility, and potentially some safety benefits.

The inability to partition critical and non-critical functions on the same processor can lead to very poor utilisation of resources. For example, if only 10% of the code in a system is critical, but it must be partitioned out to a separate processor (or replicated processors), this may result in one or more processors sitting idle whilst others are heavily loaded. In the worst case, a system which it would have been possible to run on a single processor if independence could have been shown between functions may require two or more processors when partitioned in this way.

Assuming that issues of replication (and perhaps diversity) to cope with hardware and operating system failure can be dealt with at a platform or system level, the use of operating systems introduces new challenges for safety analysis within a single processing unit.

Perhaps the most significant change is that the use of operating systems makes it more realistic to regard a large software system as a set of components, with definable boundaries. Whilst this is clearly desirable, it challenges existing principles of development for safe software.

The first practical problem, as noted in the discussion of principle 6 in section 5.6, is that there is currently no accepted way of specifying or describing the complete safety-related behaviour of an individual software component, rather than the overall behaviour of a complete system. In practise, many of the common defensive programming techniques implicitly take the view that each software function is a (relatively) independent component, which is responsible, so far as possible, for defending itself against potential failures of other system components. Thus it is common to implement input and output checks, comparison of results from calculations on real data with predictions from models, feedback checks and other similar features even where data sources and system behaviour should be trustworthy. The problem is that very often these techniques, which can be extremely expensive in terms of computation time, are implemented simply because they are possible, rather than with any analytical understanding of the potential failure modes to which the component is really likely to be subjected.

The second practical problem is that, if software is to be regarded as a set of components, it is vitally important to be able to construct robust arguments for the independence of the behaviour and failure modes of these components. (As noted above, demonstrating independence from the underlying operating system and hardware is not possible.)

This presents further problems. Safety engineering always starts by identifying hazards at the platform level, and propagating safety requirements down to system and component level. Once components and systems have been developed, their actual behaviour and failure modes are assessed to demonstrate that the achieved platform level properties are acceptable. How, then, is a project attempting to develop generic computing hardware, or a generic operating system for safety critical systems, to determine its requirements and describe its achievements, when its eventual application context is not known?

Although superficially similar to designing any other engineered artefact (such as, say, a pump or valve), the main problem is one of complexity. The designer of a pump may not know what systems his component will eventually be used in, but its capabilities and failure modes can be expressed relatively simply in familiar engineering terms, and the platform designer can select a pump that meets his requirements.

It is perhaps better to compare an operating system with, say, the electrical power supply to a complete plant, rather than with a single component. The selection of electrical power supply characteristics will determine some of the requirements for components – suitable operating voltage, for example. A complete electrical system failure could disable all of the controlled components in a plant, or perhaps cause common damage to all of them through excessive voltage or current. In a similar way, selection of an operating system will set requirements for the development of other software components, in terms of interfaces, scheduling model etc., and operating system failures have the potential to disable or interfere with the behaviour of every other software function in a system.

This analogy is over-simplistic, but helps to identify further characteristics of computer hardware and operating systems that make them particularly problematic:

- Complexity

  Interfaces to, and behaviour of, even the simplest operating systems are complex and difficult to specify completely (although it must be acknowledged that this problem is no worse than that of fully describing the behaviour and interfaces of hardware).

- Unknown behaviour in case of hardware failure

  The sheer complexity of computer hardware makes it impossible to predict all its potential failure behaviours. This problem affects both monolithic systems and those with operating systems – even if they can be shown to be perfectly correct with regard to their specification, they cannot guarantee continued correct, or even predictable, behaviour if the underlying hardware fails. However, operating systems compound the problem due to the level of indirection between application code and hardware.

- Lack of independent action in components (all tasks rely on the operating system)

  The designer of the plant in the example above could attempt to ensure that individual components were designed to behave in the safest way possible in the event of anticipated problems with the power supply – for example, fitting springs to make valves self-closing when un-powered. This sort of independent protection within components is not possible with software, as it is not possible to guarantee any behaviour of software in the event of hardware or operating system failure.

- Un-needed functionality

  The more general a computer, the fewer of its capabilities will be required by any particular application. This means that complete analysis of an operating system in any given context must include demonstration that the functions that are not wanted cannot in any way interfere with the required behaviour.

It is clearly not possible to construct a complete safety argument for any operating system supplied as a "stand-alone" component (i.e. without knowledge of its eventual application context). However, it should be possible to develop "sub-arguments" – effectively, a description of a set of properties that the operating system is claimed to guarantee, together with the evidence to support the claims. Even this will require a combination of several different techniques, covering different aspects of correct behaviour and appropriate management of a range of failure behaviour, perhaps using the type of evidence structure discussed in section 2.2.6.

6.2.4 First steps

The cockpit display system project proposed by BAe MA&A did not require full analysis of a complete operating system. The primary objective was to develop a tractable technique for investigating the effects of hardware failures on the integrity level segregation system. The technique that was developed – LISA (Low-level Interaction Safety Analysis) – and the results of the case study are described in Chapter 8.

6.3 Conclusions

This chapter has described the background to the development of two new computer system safety analysis techniques. The first, SHARD, was motivated by general observations about deficiencies in existing safety processes. Its development, described in Chapter 7 was evolutionary, and resulted in the principles for safety analysis proposed in Chapter 5.

The motivation for the second technique was a project requirement that combined a hardware / software interaction analysis problem with investigation of some of the issues in assessing the safety of operating systems. This chapter has discussed some background to this problem, and the development of LISA is described in Chapter 8.

Chapter 7

Software Hazard Analysis and Resolution in Design (SHARD)

This chapter describes Software Hazard Analysis and Resolution in Design (SHARD), a projective computer system safety analysis technique based on HAZOP.

At the time that work on the SHARD technique began, it seemed that the major problems in using HAZOP for computer systems analysis would be with technical details, such as the selection of an appropriate set of guide words, and the identification of specification and design notations which were well suited to this type of analysis. The initial proposals for SHARD, published in a paper at the COMPASS conference in 1994 [58], concentrated almost exclusively on these technical aspects.

The initial attempt to define a variant of HAZOP for computer systems was targeted exclusively at software designs (hence "Software" in SHARD). The scope of study was further restricted by considering only the application code; it was assumed that correct behaviour of both the computer hardware and operating system could be demonstrated by other means, and could be relied upon. Initial studies were targeted at dataflow and process network type design notations, where all dataflows are explicit in the design diagrams. Since the analysis was intended specifically for use in developing and refining a design, the method steps were written to include the identification of strategies for managing any safety related problems identified as an integral part of the technique.

Note that the acronym SHARD was not actually applied to the work described here until the publication of the second paper in 1995; however, for clarity, the name SHARD will be used consistently to identify all of the work described in this thesis.

The development of SHARD was shaped by four major case studies, which are described in more detail in sections 7.2 to 7.6. As these case studies progressed, it became clear that the original technical concerns, although valid, were relatively minor compared to some more fundamental issues concerning the role and meaning of analysis early in the design and development lifecycle. There were also some significant problems of working practices that had to be addressed if the

technique was to be successfully adopted in industry. A second paper [57], published at the COMPASS conference in 1995 reported the work on the early case studies, and described some of these newly identified issues. This chapter expands on the content of both of these papers.

At about the same time that the first SHARD case studies were being conducted with various British Aerospace operating companies, the MoD published a draft of Def Stan 00-56 [78] which, for the first time, identified HAZOP as a technique that could be mandated on defence development programmes. The lack of suitable guidance and documentation for the use of HAZOP as suggested in that standard was identified as a serious problem. The MoD responded by forming the Hazard Identification and Analysis Interest Group (HAIG), a panel drawn from a range of military and associated industries, which was charged with investigating, and later with producing, guidelines for safety critical systems HAZOP suitable for use in the context of Def Stan 00-56. The work of this group eventually led, via a contract with Cambridge Consultants Ltd., to the publication of Def Stan 00-58 [79].

Most of the work on SHARD was conducted between 1994 and 1996, concurrent with the major period of activity of HAIG. Many of the ideas and issues which the work on SHARD raised were discussed at the group's meetings and, subsequently, addressed directly to the authors of Def Stan 00-58 by way of commentaries on early drafts of the standard. Since the publication of the standard significantly predates the submission of this thesis, a number of the issues, ideas and developments that are described here as part of the SHARD development process are reflected in the standard. The primary authors of the standard have also now published their own book [67], which amplifies on many of the issues addressed briefly in the standard. It also tackles a number of the criticisms of the standard that were not addressed by modifications in the final issued draft. The current status of the standard, Redmill et al.'s book, and their relationship to the SHARD work are described further in section 9.4.

7.1 Initial Technical Approach

The guide words used in process industry HAZOP studies have evolved over approximately thirty years, and there is now a high degree of confidence that, when used in a systematic way by an experienced team, they will ensure an acceptably complete identification and investigation of the potential hazards in new chemical plants. One of the primary areas of concern about early proposals for software HAZOP was the uncertainty over how (or, indeed, whether) the same set of guide words could be interpreted in the new context. In view of this, it was decided to use

existing research into the classification of computer system failures to structure a new set of guide words.

The set of failure classes proposed by Bondavalli and Simoncini [7] (discussed in section 4.5) was selected as the most appropriate, specifically because of their consideration of the detectability of failures, a critical point when considering strategies for handling failures. However, the set was augmented by the term *commission*, to describe those situations where a faulty system produces an output when a correctly functioning system would have produced no output at all.

Following the model used by both Bondavalli and Simoncini and Ezhilchelvan and Shrivastava [26], every information flow in a software design was considered in terms of *services*; each transmission of information was regarded as a separate service, and for each service, failures were defined in the three groups shown in Table 6.

Group	**Failure classes**
Service provision	Omission, Commission
Service timing	Early, Late
Service value	Coarse (detectably) incorrect Subtle (undetectably) incorrect

Table 6 - Failure classifications used to structure SHARD guide words

As each failure corresponds to a single information transmission, multiple failures in a sequence of information transmissions must be considered as a sequence of individual failure events.

The initial attempt to define the SHARD guide words, applied in the first and second case studies, assumed that these failure classifications would need to be refined, in much the same way that physical properties (flow rate, temperature, pressure etc.) are used to refine the interpretation of the guide words in a process plant context. It was suggested that a customised table of guide words should be constructed specifically for each project, using the combinations of communications protocols and data types available in the design notation to produce guide words with relatively precise meanings.

For example, two of the most important communications protocols supported by the MASCOT family of design notations [39] are the *pool* and *channel*. A pool is a single writer, multiple

reader protocol. The write operation is destructive, overwriting any data already in the pool. Readers may access the pool at any time, and will read the most recent data written. Thus, if the writing process is late or stops running, the reader processes will not be held up; they will simply keep reading old data. A channel protocol is a single writer, single reader protocol with synchronisation; the reader cannot proceed until the writer has delivered the data. Physical device input and output have similar properties to the pool protocol; a process may read from or write to a device at any time without being held up. MASCOT supports a standard range of data types, including Boolean, enumerated types, various integer and real number representations, and complex data types such as records and arrays.

Table 7 shows an early attempt to interpret the failure classes from Table 6 for some combinations of communications protocol and data type for MASCOT 3. The intention was that for each flow analysed, the protocol and data type would be used to select the appropriate row of the table; the entries in that row (e.g. "No write, Unwanted write" etc.) would then be used as the guide words for that flow.

Flow		**Failure Categorisation**					
		Service Provision		**Timing**		**Value**	
Protocol	**Data Type**	**Omission**	**Commission**	**Early**	**Late**	**Subtle**	**Coarse**
Device input	**Boolean**	No read	Unwanted read	Early	Late	Stuck at 0 Stuck at 1	N/A
	Value	No read	Unwanted read	Early	Late	Incorrect in range	Out of range
Device output	**Value**	No write	Unwanted write	Early	Late	Incorrect	N/A
Pool	**Enumerated**	No update	Unwanted update	N/A	Old data	Incorrect	N/A
	Value	No update	Unwanted update	N/A	Old data	Incorrect in range	Out of range
	Complex	No update	Unwanted update	N/A	Old data	Incorrect	Inconsistent
Channel	**Boolean**	No data	Extra data	Early	Late	Stuck at 0 Stuck at 1	N/A
	Complex	No data	Extra data	Early	Late	Incorrect	Inconsistent

Table 7 - Example guide words for MASCOT 3

The main source documentation for safety analysis was a single design drawing, but it was acknowledged that it would generally be necessary to refer to additional material, such as data dictionaries or timing diagrams, to establish all the information necessary for a complete analysis. The main steps of the analysis process suggested were:

1. Identify and consistently label the flows in the diagram. For each flow, identify its *source* (where the flow originates) and *sink* (destination). Assemble any additional material from data dictionaries, timing diagrams etc. required to completely describe the operation of the flow.
2. Review the design to ensure that the *intended* operation is clear.
3. Construct a table of guide words, interpreting the six major failure classifications for the combinations of communications protocol and data type used in the design.
4. Work through the flows in a systematic order, using the combination of protocol and data type to select a set of guide words from the table constructed in step 3, and considering the deviations from intended behaviour suggested by each guide word.
5. Determine and record the potential causes of the suggested deviations. Note that identifying the causes of a deviation will require study of the active component (process) or data store that is the source of the flow.
6. Determine and record the expected effects of the suggested deviations. Except at the top (context) level of a design, the extent of identification of effects should be limited to the active component (process) or data store which is the flow sink. Effects that propagate beyond the immediate destination of the flow will be considered as part of the analysis of further data paths that originate from this component.
7. Reduce the set of suggested deviations to a set of *meaningful failure modes* by discarding those for which the potential causes are acceptably improbable, and those for which no hazardous effects have been identified. Note that whenever a deviation is discarded, a justification must be recorded
8. For each meaningful failure mode identified, suggest alternative management strategies. These may take the form of design modifications to remove its causes or limit its effects, or a set of requirements that must be satisfied by lower-level design elaboration to achieve acceptable system-level safety properties.

9. Select of one of these strategies to pursue, and record a justification for the selection. In the worst case, if no acceptable management strategy can be suggested, the only acceptable course of action may be a redesign.

7.2 Case Study 1

The first case study attempted was a project with British Aerospace Systems and Equipment (BASE) in Plymouth. This was a current development project to integrate an existing gun control system into the command and control system on a new class of frigate. Unfortunately, the case study was never completed, since project pressures meant that BASE staff were unable to release sufficient time for significant trials. However, participation in planning, design and review meetings, and discussions about effective means of working, provided valuable insights which contributed significantly to the development of SHARD.

It was obvious from this case study that the process issues, which had initially been assumed to be less of a problem than technical concerns, were in fact more significant. It was clear that, for a new technique to be adopted successfully, it was vital that as well as providing the benefits implicit in satisfying principles 2, 3 and 4 in section 6.1, it should be flexible enough to fit into existing working practices. In the environment of this case study, this meant that the analysis could be produced sufficiently quickly to be included in the regular design review meetings. Also, as the membership of the design review meetings frequently included staff who understood the design requirements and operating environment, but were not necessarily fluent in the design notations used on the project, the analysis must be presented in a highly accessible form.

The project involved modification of a successful existing system, which had been in service for over ten years. This meant that the proportion of completely new design was relatively small. However, the safety processes in place within the company when the system was originally developed meant that there had been relatively little safety-related design rationale captured, and a major reverse-engineering effort was required to understand the principles of its operation and the implications of the modifications. This balance of activity meant that the case study also highlighted the importance of principle 5. Not only must incremental development be supported, there must also be provision for development of subsystems, for reverse engineering where changes are required to existing systems, and for some form of compatibility checking, where a partial analysis could be compared with safety requirements and information about a larger system. All of these requirements suggested a much more flexible working approach than was

implied by any of the classic texts on HAZOP, all of which assumed a "clean sheet" top-down system development, albeit with the incorporation of standard components.

Relatively little actual analysis was produced on this case study. Since the primary focus of this case study was the technical issues of guide words etc., no attempt was made to convene a full HAZOP-style team meeting, but a small number of information flows were analysed by the author working with two engineering staff from BASE. The initial technical approach outlined in section 7.1 was used exactly as proposed. Progress was good, and some useful results were produced relatively quickly. Since there was little new design, there were few recommendations for design changes; most of the actions recorded in the analysis took the form of questions to be resolved by the team working on the reverse engineering of the legacy system. This quick and effective work contrasts with the difficulties encountered in team working in the second case study, and foreshadows the eventual conclusions about working practices in section 7.4. The eventual documentary output was about fifteen pages of tables containing some seventy individual entries in total.

The system design had been created using the Yourdon [82] notation in the Teamwork tool. The analysis conducted in the case study was on selected flows from the top (context) level diagram, which was then supported by further analysis of the same part of the system on lower (more detailed) level design drawings. The technical approach was found to be a good match to this design notation; the identification of flows and their intended properties was straightforward and, at the level of the drawings used in the study, there was little detail contained in the process "bubbles", so all the staff involved found it easy and natural to concentrate on the flow properties. This meant that the expected properties of the active processes were captured as relatively high-level descriptions of their contributions to the causes, propagation or mitigation of particular failure modes. It would have been relatively easy to take these general assertions / assumptions and convert them into requirements, and one of the BASE project staff spent some time considering how requirements derived in this way could be incorporated into the requirements management and tracking tool used on the project. Unfortunately, project time pressures meant that effort had to be diverted away from the case study before this could be tried out in practice.

The flows selected for analysis were chosen by the BASE project staff purely on the basis of their knowledge of and / or interest in specific aspects (mostly the new parts) of the system design. However, since these were almost exclusively interface functions to the new command and

control system, the analysis inevitably required detailed understanding of many original parts of the system. There was no attempt to work through the design in any logical order, or to ensure in any systematic way that the propagation and chaining of events was checked for completeness and consistency. Although this is inconsistent with the systematic working envisaged in the SHARD method, it was surprisingly successful; this was probably as much to do with the interest of the staff involved as with any technical merit of the approach or the analysis produced.

In retrospect, it is clear that the relatively informal, small group approach used from necessity in this case study was actually very successful, and anticipated later conclusions about working practices. Also, since this was a live project, the relatively limited information that was available was much more representative of the type of role envisaged for HAZOP, i.e. as a means of reviewing and refining design proposals. This is the only reverse-engineering case study on which the HAZOP / SHARD principles have been tried. Although the ideas were immature at the time of this study, the results were promising. The engineers involved did not appear to have any problem in interpreting the guide words to prompt questions about the (undocumented) operation of an existing system. Later case studies, which were carried out on recently completed systems, with the intention that the results obtained from SHARD could be compared with what was already known about the system, were actually hampered by participants' preconceptions and over-detailed knowledge.

7.3 Case Study 2

The second case study was the largest team study carried out during the development and evaluation of HAZOP / SHARD. At the time of the study, the first draft of Def Stan 00-58 [79] had just been released for comments by Cambridge Consultants Ltd. As part of the process of developing and refining the guidelines, early drafts were released to selected groups who were able to offer trials on real systems. BAe MA&A, working with the DCSC offered a system for study. However, in order to integrate with the work on SHARD, the case study was conducted using the procedural aspects (i.e. team working and management) of the Def Stan 00-58 guidelines, but applying the guide words and other technical details outlined in section 7.1. This case study was of considerable importance in the development of SHARD, not least because of the opportunities it presented for discussion with experts and practitioners involved on a daily basis in the development and approval of safety critical systems.

The system studied was a recently implemented avionics subsystem (in test at the time of the case study), which provided a range of utility functions such as test and configuration management

and maintenance data collection. It was not directly involved in the control or navigation of the aircraft, but could contribute indirectly to hazards through selection of test modes at inappropriate times, or incorrect maintenance or configuration data reporting. Detailed specification and design information were available, expressed in COntrolled Requirements Expression (CORE) notation [74]. Since the design of the system was effectively complete, the results from the study were not able to influence the development of the design. However, a complete hazard analysis for the system had already been performed using other techniques, so there was an opportunity to compare the results achieved, and the effort expended, with the previous analysis.

The team recruited for the study consisted of:

- The study leader (the author).
- A *recorder*, drawn from the team which had specified the system, who both participated in discussions and recorded conclusions using a simple tool built from the Microsoft Excel spreadsheet package.
- A member of the design / implementation team.
- A representative from the Independent Verification and Validation team.
- A second member of the group which had specified the system, whose role was intended to be to act as the user / customer, and ask questions from that perspective.

This team composition is exactly as recommended in the Def Stan 00-58 guidelines. The team met for five sessions, each consisting of a short (3 hour) morning or afternoon, with a tea break in the middle. Again, this is close to the 00-58 recommendations.

The first day began with a short presentation explaining the purpose of the study and basic HAZOP concepts. The team then spent approximately half a day studying an example system defined in the CORE notation. The hope was that this would enable people to become familiar with the concepts, and iron out a number of problems with procedure and interpretation before the study proper commenced. Study of the avionics system itself began mid-afternoon on the first day, and continued through both sessions on the second day. At this point it was decided that no further useful progress could be made by continuing analysis, and the team spent the morning session of the third day discussing their conclusions from the study.

The actual progress made by the case study in terms of number of flows examined was relatively small. After two days' study, the team had recorded and agreed upon the analysis of just seven flows, three of which were so closely related that they could reasonably have been grouped together without any significant loss of information. However, amongst these, the existence of a

suspected hazard not described in the previous hazard analysis was confirmed. Several factors influencing the rate of progress can be identified:

- Problems with the design representation.

  Only the design and IV&V team members were completely familiar with the CORE drawings produced by the tool used on the project. This meant that significant periods of time were spent explaining the notation to the rest of the team. Whilst it is normal for a HAZOP procedure to include time spent discussing the intended operation of the system, there were occasions where explanations had to be repeated or clarified as the designer had incorrectly assumed that some aspects were clear from the notation. In addition, the particular tool used on the project had a number of limitations, and this had led in some instances to representations that were not a clear recording of the actual intent (for example, drawing single processes as several boxes to circumvent limitations on the number of flows that could be attached to a single process). Predictably, these were the parts of the design that caused most confusion, and where progress was slowest.

  Another problem was that the CORE diagrams were backed up by textual descriptions of the processes, data types etc. These had to be referenced frequently, and this was a slow process involving manual searching of extremely large printouts. It would have been preferable to have had access to the design database on-line, perhaps with projection facilities so that the whole team could view the information.

- Time spent discussing issues that should have been resolved off-line.

  Despite the expertise gathered in the study team, it was found that there were many questions which no-one present was able to answer. The correct procedure would have been to mark these as actions for resolution outside the meeting, and reconvene later. Unfortunately, there were so many unresolved issues that, in order to make progress, the team resorted to making assumptions, which later work occasionally proved to be inappropriate.

- The complexity of the system studied.

  The study system was, perhaps, a poor choice in that, as a test and data collection facility, it had a large number of complex interfaces with other aircraft systems. This inevitably led to the problem of incomplete knowledge described above. In many cases, data flows were discovered to be so interrelated that it was almost impossible to undertake an analysis of any one in isolation. Understanding was also hampered by lack of information about the criticality, or even usage, of a lot of the data that the system handled. For novice analysts, a more independent, less data-oriented system would have been a better choice.

- Discussion of procedural issues

The study was not conducted strictly as a "proper" HAZOP. As an experimental trial, it was considered important to discuss issues related to the method as they arose. A further complicating factor was that some members of the team found it difficult to take a fresh look at a system with which they were extremely familiar. They were reluctant to spend time on areas of the design they felt were already fully documented, or broke into discussions with "pet" ideas. The members of the team who were attempting to work strictly according to the HAZOP procedures found this extremely frustrating.

- "Context free" analysis

  The team was attempting to study the system at a relatively low level of detail without having results of higher-level analyses on which to draw. This last point meant that the scope of consideration for every deviation inevitably grew wider and wider as people attempted to understand the implications of failures – in some cases to the extent that discussions encompassed ground crew training and maintenance procedures.

From the point of view of development and refinement of the analysis method, the most important outputs of the study were the conclusions reached by the team in discussions both during and after the study meetings, and these are considered in the following sections.

7.3.1 Working as a team

The study highlighted the benefit that a team conclusion may carry more weight than individuals' conclusions, but also raised concerns about the effectiveness of carrying out the analysis as a team, including:

- When the number of questions that must be resolved off-line becomes excessive, the study may have to be reconvened many times as the answers to queries suggest new deviations or interactions that must be explored. In the worst scenario, the team approach may be less effective, and introduce significantly greater delays, than a study carried out by one or two people who can contact others with the necessary expertise as soon as the need arises.
- When studying design at a low (very detailed) level, a lot of the analysis is either repetitive, or relatively straightforward and obvious, merely confirming or elaborating upon earlier analyses at a higher level. In such cases, the expensive team analysis approach is simply not necessary; worse, the lack of interesting progress means team members become bored and demotivated. At the level of design detail at which the analysis was conducted, this case study presented no compelling evidence that the team study actually provided a more thorough analysis than would have been possible with other techniques, or with one or two analysts working individually.

Conclusions about the size and composition of the team included:

- The size of the team is critical; too small, and it may be prone to bias, too large, and discussions of relatively simple points may become unnecessarily protracted.
- The composition of the team in terms of designers / users / experts was good, but the complexity of the system was such that it would never have been possible to assemble a team with the expertise necessary to analyse all parts of it without referring a significant number of questions to external experts.
- The leader's role is crucial to the success of a HAZOP team. Implicitly, it is necessary that the team leader has a strong personality and is able to be assertive in controlling discussions, especially when issues become contentious, or the study becomes "adversarial", with the designer(s) pushed into a defensive role. This was actually a significant problem in the case study, partly because the leader was the youngest member of the team, and an "outsider", but also because of the personalities of some of the other team members, with a relatively reserved designer confronted by a rather bullish user representative.

  To help the recorder, who has a very difficult task, it is important that the leader summarises the discussion frequently, and helps to ensure that the records made reflect the important points.

  To make good progress, the leader must also be assertive in enforcing time keeping, and deciding quickly when an issue is not going to be resolved in the meeting.

The overall conclusion of the discussions with the team, and also when this case study was later compared with the other trials, was that the team approach is really only appropriate for analysis at a relatively high level. This is, of course, the role that traditional HAZOP takes in the process industries, where the HAZOP team reviews the entire plant proposal at the level of the piping and instrumentation diagrams.

As the design progresses and more detail is introduced, the analysis will inevitably become more routine, and should only be expected to confirm or elaborate earlier analyses. It should only be necessary to reconvene the team to consider areas where low-level analysis contradicts or casts doubt on the original work. At low levels of design detail, relatively small changes can result in the need for substantial re-analysis. This also contra-indicates a team approach, as the costs of re-convening the team to agree the impact of such changes will be infeasibly high.

7.3.2 Recording the results

The recording of results was one of the areas of this case study that was least successful, highlighting a number of practical problems:

- It was difficult for the recorder to participate in the discussion and also record what has been said. There were two aspects to this. The first was the difficulty the recorder had in producing an accurate and unbiased record of a debate in which he was participating. The second was simply the time taken to type (possibly large) amounts of text. A more effective summing up of each item by the study leader might have helped, especially if the computer used for recording had been linked to a projector so that the entire team could have viewed and agreed what was written. The problem with this is that it would inevitably have slowed proceedings still further, and there is a danger that, where the item was contentious, it would have proved difficult to find a form of words acceptable to all participants.
- In many instances, team members gave examples of specific types of problem; the recorder, recognising that there was a more general case, attempted to determine and record this "on the fly", occasionally falling behind the current discussion as he struggled to complete his explanation of previous items. Again, if this had to be done during the meeting, it would have been preferable for it to be discussed and incorporated into the leader's summing up. Otherwise, the more practical approach would have been to record the example as given, and note the need to improve the written record by providing a more complete description or explanation at a later time.
- Even for the relatively small portion of the system analysed in the meeting, team members began to have problems tracking and navigating what had been written, and resorted to sketching additional diagrams on a whiteboard to represent the failure modes that had been identified, and how they related and propagated around the system.

  It was clear that, for large systems, the output of the process might be a huge volume of paperwork. It was suggested that the tabular records should, wherever possible, be supported by a diagrammatic representation such as fault trees, or cause-consequence diagrams.

7.3.3 Comparison of SHARD results with previous analyses

One of the stated intentions of this study was to compare the results from SHARD, and the effort it required, with those from previously completed Functional Failure Analyses. Unfortunately, the study was so dominated by discussion of procedural issues, and the problems with the design representation described above, that no meaningful comparison could be made about effort. The

consensus opinion of the team members was that, even if the process issues had been resolved satisfactorily, the SHARD approach would still have been substantially more expensive than FFA. However, this was tempered by noting that the team approach had significant potential benefits, particularly if applied early in the development process. It was also agreed that the SHARD approach provided more structure to the analysis; several team members observed that it would therefore be an easier technique for novice analysts to apply.

Comparing the results of the SHARD and FFA studies produced some interesting observations, although the small number of items actually analysed using SHARD meant that the significance of these was rather limited. Also, the two approaches were not strictly independent; some of the staff involved in the SHARD study had also been involved in preparing the FFA, and therefore inevitably brought their knowledge of previous conclusions into the meetings.

It was observed that, for the small number of flows examined, the SHARD study had identified (as causes or effects of deviations) all of the safety related failure modes contained in the related FFAs. However, the SHARD analysis was more explicit about the relationship between these failure modes, making it much more obvious how they were propagated around the system.

The SHARD analysis appeared to be much more sensitive to the areas where the design had been constrained by tool limitations, most notably producing analysis that was extremely uninformative for the functions which had been split due to constraints on the number of flows into a component. By contrast, the authors of the FFA had managed to present reasonable descriptions of these "virtual sub-functions". It was not clear whether this should be regarded as a weakness of the method. It was pointed out that, as a design aid, the identification of parts of the design that could not be analysed meaningfully could be an indication that there were problems to be resolved. However, this is not an effect that has been encountered on any subsequent case study.

7.3.4 Technical conclusions

Although the study concluded that CORE was a generally suitable design representation on which to base a HAZOP-type analysis, the example presented for this study was found to have a number of drawbacks. It became clear that the effectiveness of the study was being impaired because the majority of the drawings represented only a single process on each page, necessitating constant searching through the specification to follow data flows. The specification could not be pulled apart into individual sheets and laid out on the table, as the flows which

crossed page boundaries were not always grouped according to destination page, and frequently appeared in a different order on different pages. This contributed to team members' difficulties in navigating and understanding the propagation of failure modes around the system. A number of instances were found where structures had been introduced into the design to work around limitations of the notation (or the tool), and the analysis results for these parts of the design were, at best, uninformative.

Some of the team members complained that the study was too data-oriented, to the exclusion of consideration of the processes and data stores in the design. Of course, all the data in the flows studied originated or was consumed in processes, so a better investigation of causes of deviations would have rectified this problem. The rather limited consideration given to causes was possibly a result of either the cumbersome representation of detailed design information in the CORE print-outs, or the large number of unresolved questions that were raised about the design intent.

This case study also identified a number of significant flaws in the technical approach outlined in section 7.1. Perhaps the most important of these was that the table of "refined" guide words was much more of a hindrance than a help. Initially, there were significant problems in constructing a table of guide words for use with CORE; most of the suggestions were contentious, and it took a long time to agree on words that everyone found acceptable. As the analysis progressed, the agreed table was repeatedly called into question, and much time was spent revising it as situations arose where the pre-defined words were not appropriate. This then necessitated re-visiting previously completed pages of the analysis to see whether the revised words prompted further deviations for consideration.

One of the areas where this was most significant was in the definition of detectable and undetectable failure. Although academically attractive, the team members found the distinction difficult to grasp. For example, it was not always clear whether the team was discussing whether a failure was *intrinsically* detectable (i.e. given unlimited resource, a detection mechanism could be devised), or whether it was *actually* detected in the proposed design. This had not been considered in the introductory briefing, and the practical solution adopted was to consider *intrinsic* detectability, and then note in the justification or actions whether the detection mechanisms actually in place were considered sufficient.

In discussion after the study, the team members agreed that they felt constrained by the relatively tight definitions of the guide words, and would much have preferred to work just with the six

basic failure classes. The expectation was that each of these guide words would have prompted several deviations, but these would have been specific to the flow under study, and would therefore have been more meaningful.

Asked to consider whether it would have been better to use the traditional HAZOP guide words, the team members concluded that they were not as relevant as the software failure classes and that, given that their main desire was to work with a smaller set of more general prompts, the larger number of traditional guide words would probably have been more onerous.

The primary technical conclusion of this study was that, whatever guide words are used, their most important function is as discussion starters, and it is important not to unnecessarily restrict the freedom of the team to interpret them in novel ways. However, suggesting how the words should be interpreted in particular contexts is seen as important for ensuring uniformity across multiple studies conducted independently on component parts of a large system.

Another area of significant difficulty was in the recording of actions. The primary intention of SHARD was that the actions should be recommendations or requirements for design development; given that the study was conducted on a completed design, this was meaningless in this case. Whilst they accepted the argument for closer integration of analysis and design, the team were concerned that there should be seen to be some degree of independence between the two activities. The suggested approach was to use the SHARD analysis strictly as an analysis, and to record recommendations and requirements separately, citing the analysis as the source of new requirements.

7.4 Postscript to Case Study 2 - Revised working practices and technical approach

Following Case Study 2, the suggested technical approach of SHARD was revised to incorporate all of the significant recommendations. The complex table of pre-assigned guide words was dropped, retaining just the six main failure classes (omission, commission, early, late, detectable value and undetectable value) as guide words. The analysis steps were also modified to remove design revision as an integral activity; instead, a much freer recording of justifications, comments and recommendations was substituted as the final step, meaning that the method description was applicable unaltered even where it was used purely for retrospective analysis.

Process issues were also added to the description of the technique. Being aware from Case Study 1 of the need for flexibility of approach, three distinct ways of working were outlined. The first was the traditional team approach, in line with process industry practice. The second was a simple design – review – redesign structure, following the practice of many organisations, where a design is circulated for peer review and comment. In this case, the safety analyst(s) would simply be added as additional peer reviewers each time a design was circulated.

The third, and most complex, suggested process model attempted to retain the concept of analysis as an integral part of the design activity. This process model is outlined in Figure 1. The intention of this process outline is that the designers should actually produce their own analysis, which can be used in support of their proposal. The analysis would form part of the package for peer review, and be discussed as part of the normal review meetings. If necessary, a HAZOP team could be convened to tackle difficult or contentious issues.

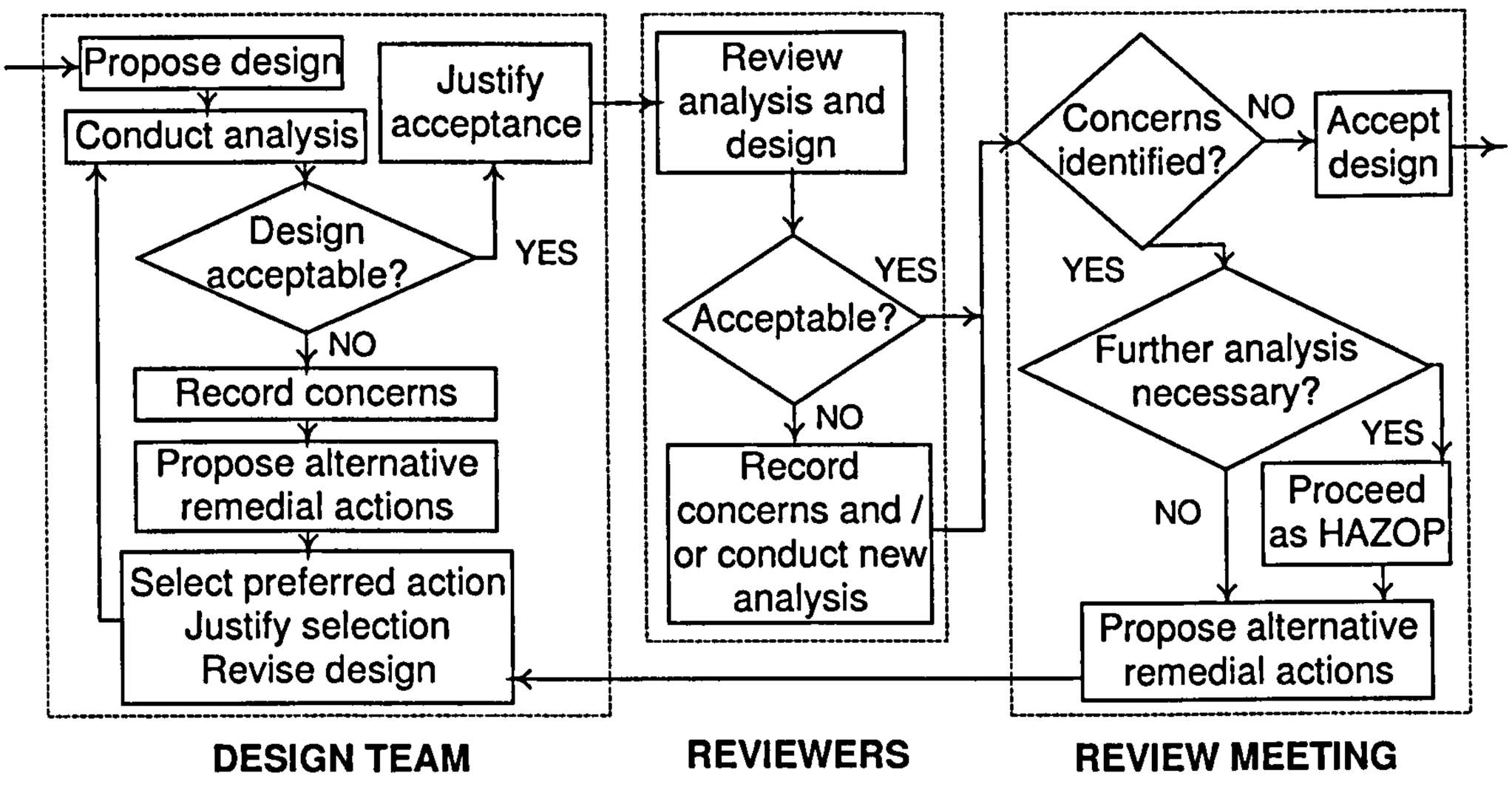


Figure 35 - Outline of SHARD analysis as an integral design activity

7.5 Case Study 3

The third case study undertaken was the most extensive in terms of effort and size of analysis produced, but was carried out without significant industrial participation.

The case study system was a computer-assisted braking (CAB) system. The system manufacturer had provided outline requirements and a number of alternative design proposals, one of which was analysed in the case study. It was later decided to create a new design proposal to the same requirements, so that both the new design and accompanying reworked analysis could be published. The system description, along with partial analysis results, was included as an exercise in the "Safety and Hazard Analysis" module of the MSc in Safety Critical Systems Engineering at York in 1996 and subsequent years. The same system has also been used as an example in doctoral theses by a number of other researchers at York, e.g. [64].

The analysis and consequent redesign were conducted by the author working with one other researcher. The work was carried out over an elapsed period of approximately two months, and consumed some 300 man-hours effort. The six basic failure classes were used directly as guide words, and the results of the analysis were recorded in a worksheet in the Microsoft Excel spreadsheet package. In all, 73 deviations on 9 main data flows in the context level diagram were analysed; an example page from the spreadsheet is shown in Table 8. Although this may not appear to be a substantial volume of output, a large proportion of the analysis was repeated at least twice, as modifications introduced into the design in response to analysis findings invalidated other parts of the analysis which had previously been completed.

7.5.1 Analysis and Design Revision

This case study was the first significant attempt to use SHARD in the way it was conceived, i.e. as an integral part of the process of developing a design for a safety critical system. Initially, an attempt was made to analyse the complete original design exactly as proposed. However, it soon became obvious that there were substantial flaws in the design. There followed a protracted debate about whether it would be more productive to complete the analysis in order to identify as many significant problems as possible at once, or whether it was preferable to revise the design immediately, scrap the partially completed analysis and start again.

The course actually followed was an attempt to continue the analysis assuming that the changes suggested had actually been incorporated into the design. This was a mistake, as the result was an analysis of uncertain status that did not correspond completely to any version of the design. This led to confusion, and several drafts of the analysis which were internally inconsistent (or even contradictory), as well as inconsistent with the most recent design drawings.

It became clear that it is essential to the success of the method that the analysis produced is linked with one identifiable, frozen version of the design. If it becomes clear that the design is sufficiently flawed as to necessitate a substantial reworking which will invalidate parts of the analysis and make it not worthwhile continuing, this should be recorded as an analysis conclusion. A new version of the analysis should be produced with the design update, ensuring that any material that is copied over from the abandoned version is thoroughly checked to ensure its correctness and consistency with regard to the new design.

7.5.2 Order of analysis

The repeated analysis of similar variants of a design gave the opportunity to test and compare a number of variants of the analysis method. The most significant product of this experimentation was the identification of a preferred order of working through a design in order to minimise the analysis effort.

The first part of a new system to be analysed should always be the inputs and outputs, which can generally be expected to be relatively static features of a system. Even if the internal design is completely reworked, it is likely that the majority of the system's interfaces will be unaffected (although, of course, the causes of output deviations and the immediate effects of input deviations will change). The analysis of inputs and outputs effectively determines two critical pieces of safety information; which output deviations are hazardous and must be avoided, and the quality of the input data. If possible, it would be preferable to carry out a quick input/output analysis before the first proposal for internal design of a system is produced.

For the analysis of the complete design, it became clear that by far the most effective way of working is *backwards* through the system, starting with the outputs, then considering the flows that immediately precede the outputs and so on back to the inputs. This makes the analysis a rather more deductive process, similar to fault tree analysis; there is still an inductive phase as each flow is considered, checking that the expected effects of deviations have been included in the analysis of downstream flows, but this becomes essentially a confirmatory activity.

There are two main reasons why this approach is effective. Firstly, the deductive approach actually limits the size of the analysis. Since the computer system can only cause or contribute to a hazard via its outputs, if analysis reveals that a particular output has no hazardous failure modes, then any internal flows and processes which contribute only to that output cannot have hazardous effects. For the purposes of SHARD, this means that relatively little analysis of these components is needed – in many cases, no more than recording the absence of hazard contributions as a justification for considering these parts of the design acceptable. In contrast, when analysis is started from the inputs of a system and worked forwards to its outputs (as recommended in Def Stan 00-58), it can be extremely difficult to determine which identified input deviations will eventually contribute to potentially hazardous output failures, and it is necessary to record and trace the propagation of every identified deviation right through the system. Considerable effort may be expended in investigating and recording deviations that will eventually be found to have no safety effect.

Secondly, analysts always seem to find it harder to identify and express the expected effects of a deviation than to suggest its possible causes. The exception to this is in studying the outputs, where deviations affect the "outside world". These effects are typically relatively obvious, and can be determined with some certainty (e.g. incorrect data values correspond directly to incorrect actuator movements; omitted data means parts of a display will be missing). Once these effects have been established, the effects of deviations in internal flows can be described in terms of their contribution to external consequences that have already been recorded.

Working back from outputs in this way also tends to counteract a phenomenon observed in every case study, which is that analysts seem to be much more comfortable with the "immediate cause" rule than with SHARD's equivalent "immediate effects" rule. The intention in SHARD is that the effects of a deviation should only be described in terms of the process or data store that is the sink of the information flow; effects that propagate further should be included in the analysis of downstream data flows. Case study experience showed that analysts are unsatisfied leaving effects "hanging" for future investigation; once a deviation has been suggested, the natural inclination is to immediately trace it right through to its eventual output effects. In several instances in case studies, the effects recorded for internal deviations were developed in this way, but without any recording of how the conclusion had been reached. This does not occur when working back from outputs, as the completion of the propagation sequence is already known.

7.5.3 Revisions to the guide words

This case study also resulted in a further simplification to the guide words. As the example output in Table 8 shows, it became obvious that the need to consider detectability of a deviation is not limited to value domain failures; it is possible to have detectable or undetectable omission, commission and timing failures. Detectability is a completely separate characteristic, and in this case study this was managed by adding a column for detection and protection to the table used to record the analysis.

Since the study was still using the six original guide words, this resulted in detectability effectively being considered twice for value failures and, partly as a result of the confusion this caused, detection and protection were not particularly systematically investigated or recorded. Reviews after the completion of the case study again concluded that the best approach would be to first record the *intrinsic* detectability of the deviation, and then to consider what detection and protection measures, if any, had actually been incorporated into the design.

7.5.4 Integrating application, operating system and hardware analyses

Another significant outcome of the CAB case study was the appreciation that the original assumption that application software could be analysed in isolation, leaving the safety of the underlying hardware and operating system to be demonstrated independently, was not acceptable.

The CAB system was a multi-channel design, with triple redundant processors communicating with each other and with bespoke output hardware via a replicated communications bus. On each processor, the application was structured as a sequence of tasks running in a scheduling environment that permitted pre-emption. A number of the safety features of the design, such as the selection of which channel was master in any given cycle, were controlled by altering the timing behaviour of the system. Some critical information was determined via a voting system, involving a number of rounds of inter-processor communication. The hardware and operating system characteristics were so significant as potential causes of critical deviations in these areas that it was not acceptable to omit them entirely from the analysis.

The practical resolution reached in the case study was to record hardware and scheduling contributions to hazards, but not to attempt to develop them further. In effect, they became a set of implicit requirements which, in a complete safety process, would need to be identified and extracted from the SHARD records, and satisfied by other analyses. Although they were not

developed, efforts were made to describe the failure modes of concern as fully as possible, so that the engineers working on these aspects of the system had the maximum possible information.

This solution is represented in fault tree terms in Figure 36. The SHARD analysis completes the investigation of causes for deviations in the application software and its inputs, but hardware faults and deviations in operating system software are left as undeveloped events.

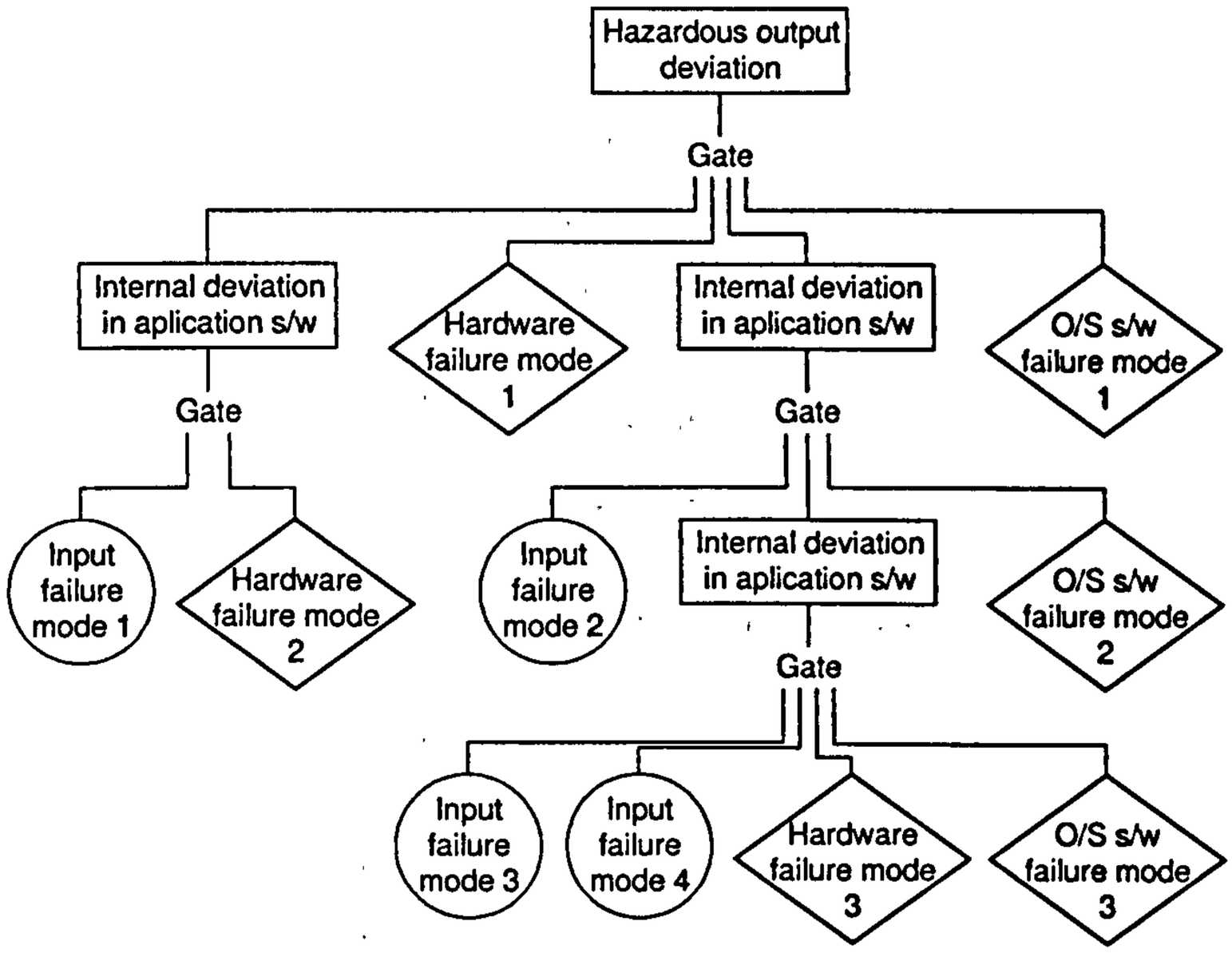


Figure 36 - Fault tree representation of inclusion of hardware and operating system failure modes in SHARD

Flow ID 1 **Source** OUT **Data Type** Record (4 x pressure values)
Flow Name Actuator_control **Destination** Hardware **Protocol** Signal
Additional Information Written by OUT processes on all 3 processors. First to complete is used by hardware, rest are ignored.

ID	Guide Word	Deviation	Possible Causes	Co-effectors	Detection / Protection	Effects	Haz Cat	Justification/Design Recommendations
1.1	Omission	No output records sent. Results in no change in braking pressure (stuck at).	Failure of OUT process on all 3 processors, or failure of both CAN buses	N/A	None possible	a,b,d,e	Cat	Rec: Ensure reliability of processor hardware and CAN bus implementation adequate. Must be at least 1 order of magnitude better than hydraulic components of system.
1.2	Commission	Additional output records sent. Result is unwanted change in braking pressure.	Spurious CAN messages.	N/A	Only has effects when spurious message is valid, but with out-of-sequence ID.	b,c,d,f	Cat	Rec: Check integrity of message ID generation in OUT process. CAN protocol with CRC checks etc. considered sufficient to prevent incorrect interpretation of other CAN messages.
1.3	Early	N/A	N/A	N/A	N/A	N/A	N/A	Just: Meaningless. Output can never precede initiating change in demand.
1.4	Late	Output records sent late.	Failure of scheduling software or clock hardware on all 3 processors. CAN bus overloading.	N/A	N/A	b,e	Cat	Rec: Clock failure only considered likely as result of failure of synchronisation procedure. Check integrity of synchronisation. Check possible causes of CAN bus saturation.
1.5	Late	Unacceptable response time to change in demands. (end-to-end latency)	Scheduling failure of tasks / messages	N/A	N/A	b,e	Cat	Rec: Investigate watchdogs to limit process over-run. Ensure there is no way in which BASIC process can be prevented from completing every cycle.
1.6	Value, detectable	N/A	N/A	N/A	N/A	N/A	N/A	Just: Hardware has no error detection / correction capability. All value errors must be considered undetectable.
1.7	Value, undetectable	One or more components of output record not proportional to demand	Undetected algorithmic failure in any process. Corruption of CAN message.	N/A	CAN protocol. TMR on BASIC processes.	a,b,c,d,f,g	Cat	Rec: Apply every possible defensive programming strategy to detect and reject incorrect values.

Table 8 - Sample of SHARD output for the Computer Assisted Braking system

7.5.5 Process issues

This case study confirmed the appropriateness of a small team approach to this type of safety analysis. For contrast with the majority of the work, which was undertaken with two analysts working together, some parts of the analysis were attempted by one researcher working alone. This proved to be extremely frustrating, and significantly less progress was made per hour of effort than in the joint work. Subjectively, the main problems were either running out of ideas, or becoming bogged down in detail and being unable to find an acceptable generalisation. Also, there was a tendency for the individuals to doubt the correctness and completeness of their own work, and almost as much time was spent discussing and reviewing each contribution as would have been spent carrying out the work jointly.

Working together, the researchers found that the flow of ideas improved; suggestions made by one person would be developed by the other, or would prompt related ideas. Also, the ability to discuss and agree on the important issues and the wording of the record was extremely valuable. Most of the work was conducted on a large white board, so that both participants could read and amend the record easily; each session was then transcribed into the worksheets.

Another helpful factor was the complementary personalities and expertise of the researchers involved. In terms of Belbin's classification of team members [4], one of the participants was clearly a "shaper", and the other a "completer". The resultant combination of creativity and adherence to the procedures of the analysis method was extremely effective. Both researchers had a good background in computer systems, but one had expertise in the automotive application domain and was able to explain the background to some of the requirements, whilst the other had more expertise in architectural design and operating systems for multi-processor systems.

7.6 Case Study 4

The fourth SHARD case study was the first to be carried out largely by engineers in an industrial context, with relatively little input from the author or other researchers.

The system studied was part of the weapons control system on a frigate. The study concentrated on the functions which were intended to provide a secure interlock to prevent simultaneous firing of two missiles from launchers at opposite ends of the ship – potentially hazardous if the missiles' trajectories intersect above the vessel. This functionality was to be implemented in a

combination of hardware and software components. The controllers of the two missile launchers communicated via a direct primary data link; if this failed, there was an indirect communication path via other systems that could be used as a fallback. The two controllers used a sequence of messages to interlock their operations.

The main work on the study was carried out by Matt Tucknott of BAe Dynamics, with assistance from a number of engineers working on aspects of the system. The case study was initiated with a day's meeting at which the principles of SHARD were first presented briefly, and then the rest of the day was used to actually study a small part of the system in a HAZOP team format. The intended operation of the system was explained, and then two flows were examined in some detail. This was a particularly interesting exercise, as the author had no prior knowledge of the system, but was able to guide the meeting using the SHARD process.

The meeting rapidly identified certain features of the design as being of particular concern. Chief among these were the implementation of the secondary data link, and the interaction between hardware failures and the software protocols in the two launchers which detected failures of the primary link and switched to using the secondary link for synchronisation. The BAe staff were encouraged by this, as their own analyses had already highlighted these as areas of potential concern; the fact that they had been identified so rapidly suggested a promising technique.

After the initial meeting, the BAe staff took over the running of the case study, which was largely carried out by Matt Tucknott working together with one of the design engineers. A second meeting was called when it became clear that technical difficulties were being encountered with the method; this meeting was extremely helpful and productive for both the author and the BAe engineers, and a number of important guidance principles emerged. Most of these principles encoded ideas that the author had already implicitly applied (particularly in the CAB case study) but which were not explicit in the method documentation. In particular, the SHARD concept of chaining of deviations was made much more explicit, and the fault tree-like *primary – secondary – command* rule for identifying causes of deviations was added to the method guidance.

The discussions also emphasised the need for further clarification of the role of SHARD. In particular, there was confusion about the status of the deviations suggested by the guide words. The Venn diagram shown in Figure 37 was drawn to clarify the relationship between suggested, expected, possible and actual deviations, defined as:

- *possible* deviations – the ways in which the system can actually deviate from its intended behaviour. This set could only be determined by a notional "perfect observer", with complete knowledge of the system.
- *actual* deviations – the subset of the possible deviations which will actually be experienced when the system is tested or in service.
- *suggested* deviations – all of the deviations prompted by the guide words and considered in the SHARD analysis. These will include some deviations that are not actually possible, and will almost certainly exclude some which are possible.
- *expected* deviations – The subset of the suggested deviations for which credible causes are identified in the analysis. Again, since the analysis is projective and working from incomplete or uncertain information, the set of expected deviations will not perfectly match the set of possible deviations.

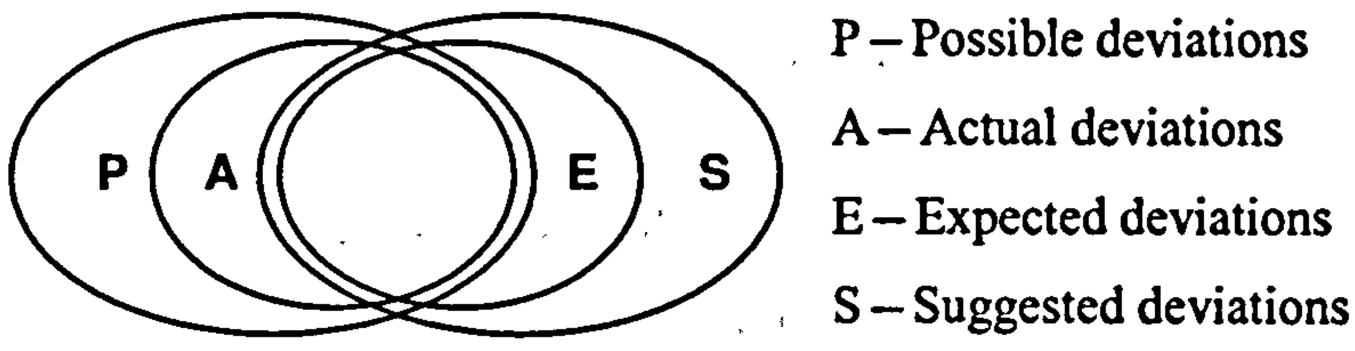


Figure 37 - Relationship between possible, actual, suggested and expectd deviations

This emphasised two rules of the analysis method. The first is the importance of interpreting the guide words imaginatively to create the set of suggested deviations for analysis. The second, which is where the case study experienced difficulties, is in understanding that the fact that a deviation can be suggested does not mean it necessarily exists. Only the expected deviations need to be considered as potential hazards, and there is no need to consider the detection and mitigation of deviations for which no plausible cause can be found.

Because of the complex exchange of messages used to synchronise the activities of the two controllers in case study 4, it was essential that the analysis considered many alternative sequences of deviations. There was some debate as to whether it was better to consider a corrupted exchange to be a result of omission and commission deviations, (relative) timing deviations, or a combination of the two. Although any of these options was valid, any corrupt sequence can be represented using only omission and commission deviations, whereas timing deviations alone are not always sufficient. For simplicity and consistency, an approach based on omission and commission deviations was therefore considered preferable.

After the second meeting, the case study progressed well, concluding with the issuing of internal company memo describing the findings of the study, and recommending that SHARD should be adopted as part of BAe Dynamics' safety process.

7.7 Other case studies

In addition to the four major case studies described in sections 7.2 to 7.6, a number of smaller case studies were conducted during the development of SHARD. The author, working with other researchers, has applied the technique to a range of small teaching examples. These exercises, which have included examples from rail, manufacturing and aerospace industries, have been conducted in a very similar manner to the CAB case study described in section 7.5, and the results have been consistent with the findings of that study.

There have also been a number of industrial trials, of which the most significant is one which was conducted by Chris Harper, formerly of British Aerospace Airbus, the conclusions of which were reported to an internal BAe / DCSC workshop held in York in April 1996. This trial, which was a retrospective analysis of a partially completed design, was supportive of the SHARD approach. The most significant contribution of this study was that an attempt was made to quantify the costs and potential benefits of the technique. The figures produced are distorted by the fact that the analysis revealed a problem that the live project development team had not discovered until rig testing; it was calculated that the cost of the analysis had been less than $1/25^{th}$ of the cost of the subsequent redevelopment. More realistically, the conclusion of this study was that the cost of the analysis was broadly compatible with other approaches such as Functional Failure Analysis, and that SHARD should be considered as a candidate technique for inclusion in future safety programme plans.

7.8 The SHARD method

This section presents a step-by-step description of the SHARD method, complete with guidance notes. This method description, augmented by practical examples of each of the steps, forms the basis of a handbook that will be used for the next round of industrial case studies and trial applications of the technique.

7.8.1 Introductory notes

Software Hazard Analysis and Resolution in Design (SHARD) is a technique for investigating the expected safety-related behaviour of safety critical or safety-related computer systems. Despite its name, SHARD should be viewed as a safety investigation of the complete computer system, and not just the software application.

SHARD is intended to help assess the suitability of proposed computer system designs, and to derive safety-related requirements for detailed development of a design. SHARD analysis should primarily be viewed as a part of the design process, rather than as a safety assessment or audit technique. The analysis should be "owned" and managed by the design team, even if other groups are involved in its production.

SHARD analysis of a design is structured around the *information flows* between the components of the system. At the top level, the technique considers inputs to the computer system from sensors and other data sources, and outputs to actuators, displays and other systems. Within the software, the analysis focuses on data flows between software functions. This provides an alternative to the largely function-based view of systems typically taken during design, and can help designers and safety analysts gain new insights into the requirements for, and potential problems with, a system design.

SHARD uses a small set of *guide words* to prompt consideration of possible *deviations* from the intended behaviour of each information flow. For each deviation considered, the analyst(s) must determine whether it has *plausible causes*, and whether it can cause or contributes to a *hazardous effect*. If so, the analyst must assess the protection and mitigation (if any) that is already built in to the design.

For any plausible and potentially hazardous deviation that does not already have adequate protection or mitigation, the analyst(s) should recommend appropriate actions to improve the design. If no reasonable remedial actions can be identified, this is an indication that the overall design concept may be unsatisfactory.

The analyst(s) may also make recommendations about further safety activities that should be carried out to provide the necessary evidence that the expected behaviour of the system is realised in practise.

Note also that SHARD is *predictive* and *requirement setting*. It does not provide *evidence* about actual safety characteristics achieved; rather, it should provide confidence that, provided that the system is built as described and any recommendations are met, it can achieve the required level of safety. Since SHARD is carried out on the software design, the quality of the analysis, and the strength of the conclusions that can be made, are very dependent on the strength of the mapping from design to implementation. If the eventual implementation varies significantly from the design (or from the analysts' understanding of the design), this will significantly reduce the value of the analysis.

The principles of SHARD analysis are extremely simple. It is important to understand, however, that the analysis process requires both creative thinking in the interpretation of the guide words, and careful, methodical investigation of the potential causes and effects of deviations if it is to provide constructive input to the design process.

7.8.2 Recording the Analysis

SHARD analysis should be recorded in a tabular format. A new page should be started for each information flow considered, and the details of the flow should be recorded at the top of the page. The columns used in the table should include *at least*:

- Guide word
- Deviation
- Possible causes
- Effects
- Detection and protection
- Justification or design recommendations.

It is helpful to add an index column so that every deviation can be given a unique ID for reference. Other columns, such as identification of additional factors contributing to effects, or an assessment of the severity of the effects, may be added if required.

If the SHARD analysis is conducted by a team, it is important to ensure that the team members are able to review and approve the record that has been made. This can either be done by using a whiteboard or OHP (or a computer with projection facilities if available) to record the analysis, or by circulating copies of the record *quickly* at the end of the meeting.

7.8.3 SHARD Steps

Figure 38 shows a simple flow chart for the steps of a SHARD analysis. Each of these steps is now described in more detail. In these descriptions, the term "analyst(s)" is used to encompass everyone involved in the study, regardless of the actual working practice adopted (see section 7.4).

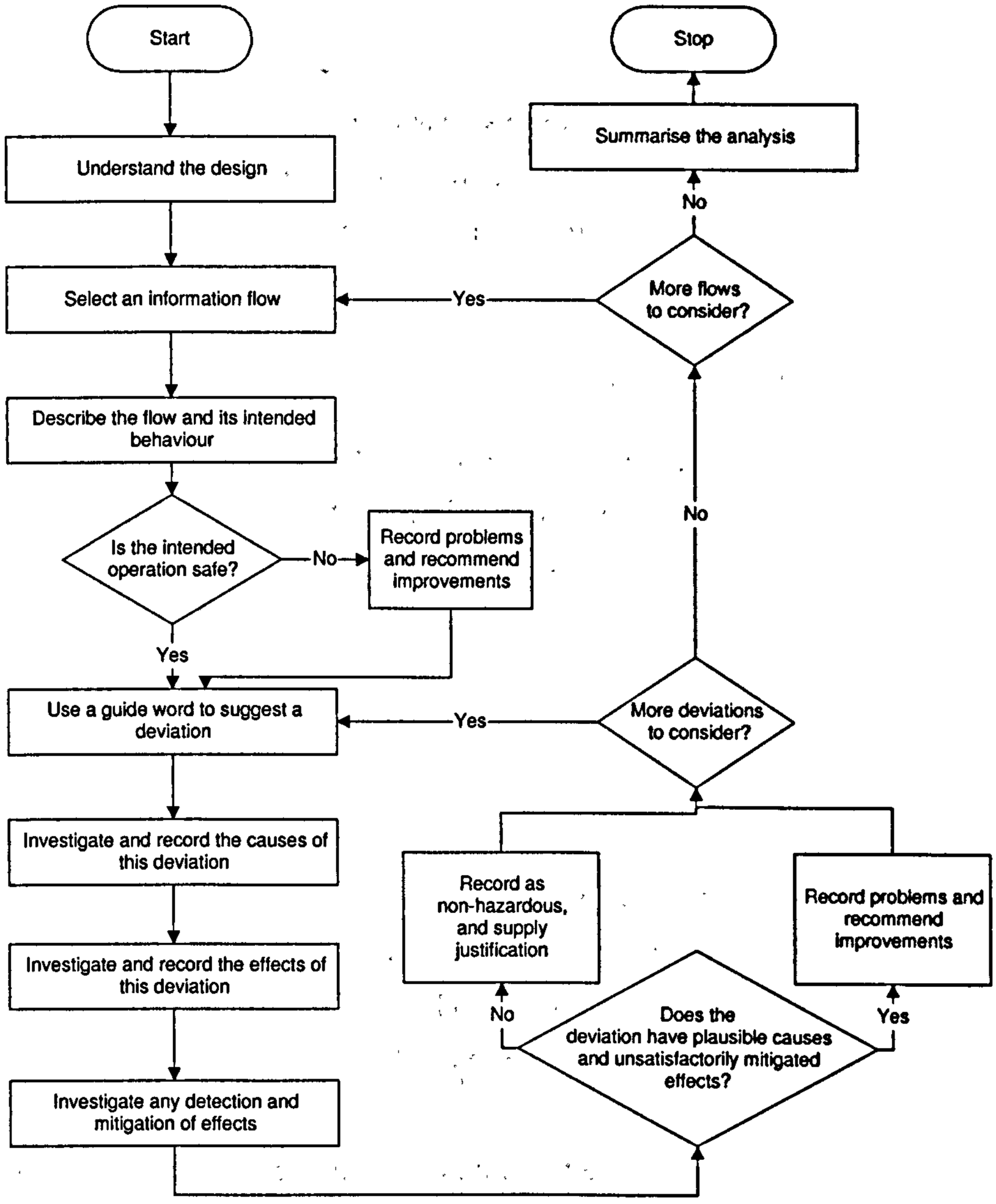


Figure 38 - SHARD flowchart

The descriptions of each step in this section are extensive, and may appear somewhat daunting. It is important to note that the intention of this section is to provide as much guidance as possible. In practice, most of these steps are simple, if occasionally time-consuming. Experience shows

that the first parts of any design analysed always take a disproportionate amount of time and effort, as analysts work to understand the system requirements and design proposals. As this understanding is gained, the analysis rapidly speeds up, and people develop the ability to rapidly focus on points of genuine concern.

Step 1: Understand the design.

When a design drawing is presented for analysis, the first step is for the analyst(s) to make sure they understand clearly what the design intent is. In a team SHARD study, the best way to achieve this is for the designer to quickly "talk through" the design, outlining the function of each component and information flow. Analysts working individually or in small teams should similarly talk through the drawing with the designer(s) before starting work.

Clearly, this step also requires that analysts are able to read the notation used to represent the design, and are able to understand the meaning of special symbols, shapes and annotations. Whilst it is possible to conduct a study with a team where some members are not familiar with the design notation, case study experience has shown that this

This step should also include a brief review of relevant safety requirements, or the results of earlier hazard and safety analyses applicable to the part of the systems under study.

It is important to ensure that the analyst(s) understand the way the system should work not only in normal operation, but also in unusual circumstances, such as

- system start-up / power-up
- system shut down / power down
- any special modes for maintenance or testing
- error recovery modes (e.g. restarting after unintentional power interruption).

It is quite likely that, in the early stages of design, the appropriate behaviour for all of these circumstances will not be defined; the SHARD analysis itself should contribute to understanding how to manage exceptional circumstances safely. However, where system level hazard and safety analyses have identified the required behaviour, the analysts should be aware of this.

Step 2: Select an information flow.

SHARD analysis, like all hazard analyses, should start at the system level. This means that it should start from the context level design, and work down to more detailed drawings.

SHARD analysis always proceeds backwards, starting with the outputs of a system or function, and working back through its internal information flows and components to its inputs. This is important, as working this way allows the effects of deviations in the behaviour of internal and input flows to be expressed in terms of their contribution to output deviations that have already been studied.

Compared to starting with inputs and working forwards, working back from the outputs cuts down considerably on the work involved in recording the effects of a deviation, and makes it easier to understand which deviations can cause or contribute to a hazard.

All of the outputs should be analysed before internal flows are investigated, working back from the flows closest to the outputs, and the inputs should be studied last. In the system shown in Figure 39, for example, the flows would be studied in the order Out 1, Out 2, Out 3, I1, I5, I6, I4, I2, I3, In 1, In 2, In 3.

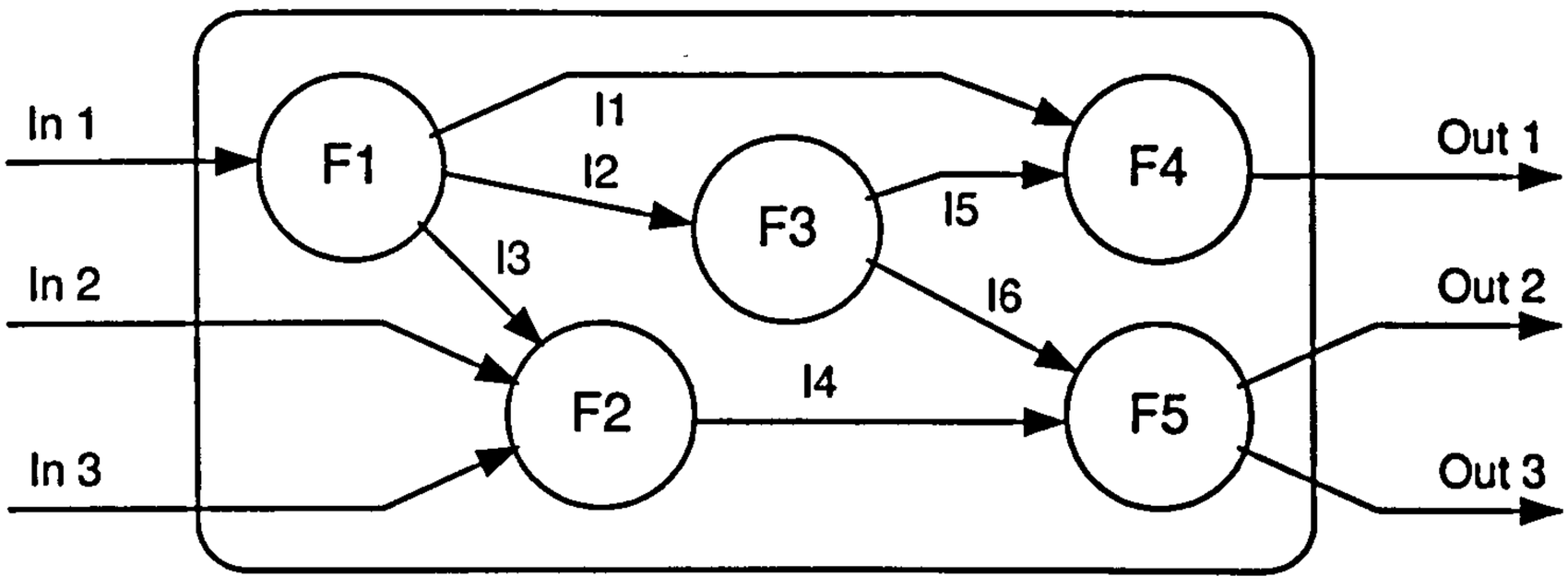


Figure 39 - Example to illustrate order of analysing flows in SHARD

Note that some design notations allow complex or compound flow types, i.e. one line on the diagram may represent the flow of many pieces of information, transmitted either separately or in a data structure such as a record or array. In studying a design that uses these flow types, the analyst(s) must decide whether it is necessary to separate the data items and analyse each one individually. This is an important but potentially difficult decision.

The following rules should be applied:

- In the case of a compound data flow (i.e. one line represents multiple data items transmitted separately), it is almost always necessary to analyse each individually. At the very least, there are likely to be timing characteristic, and possibly issues of sequencing, to be considered.
- In the case of a complex data type such as an array or record, the primary factor in the decision is how the data is used. If it is always used as a unit, then it may be possible to analyse it as a single flow. If the data items are separated at the destination and used for different purposes, then separate analysis will be necessary.

  For example, in the lift controller design, it would have been possible to draw a single output flow to the motor power management unit. This single flow would actually consist of a record containing three elements – required_speed, direction and start/stop. Although they have a common source and destination, the effects of deviations in these three elements are clearly quite different, so separate investigation would be required.

  If in doubt, always analyse each data item separately.
- Communications systems are a special case. In these systems it is common to find that the unit of information is a packet consisting of a block of data which is being moved, wrapped in the control data required for routing, error detection / correction etc. In such systems it is often impossible to determine the effects of deviations in the message data. The most tractable approach is to separate the control fields for individual analysis, but to treat the message content as a single data item. It is usually necessary to assume that every packet contains the highest criticality data the system will handle, and analyse deviations in the control structures of such a packet.

  For example, in a digital telephone exchange, it is impossible to study the precise effects of corruption of the digitised audio data being moved. However, in considering protocol-level deviations that might result in errors such as the loss or mis-routing of packets, it should be assumed that the conversation being damaged is a call to the emergency services.

Step 3: Describe the flow and define its intended behaviour.

Ensure that the intended behaviour of the selected flow is clearly understood. This should include defining what the data represents, how it is transmitted, and also how it is generated and used. It should also include details of how this flow should behave when the system is in states other than normal operation, as discussed in step 1.

For analyses at the top (context) level of a system, it is possible that these details will not have been decided; again, conclusions from the SHARD analysis may help to determine the most appropriate implementation.

Where the communication mechanism used for an information flow is defined, it has a significant effect on what deviations from its intended behaviour are possible, and whether or not they can be detected. In describing a flow, the analyst should consider:

- Data type and coding

  How is the data represented? For example, what format is used for real numbers and integers? Is there internal redundancy or error checking within the data, such as parity bits? How are enumerated types mapped onto binary representations?

- Timing

  Is this a periodic (regular) or sporadic (possibly interrupt-driven) communication? How frequently does it occur? What is the acceptable delay and jitter?

- Protocol

  Some design notations permit a range of different communication protocols. These will not only alter the possible behaviour of the system, they may also affect the way that the analysis is conducted.

  For example, the MASCOT family of notations provides (amongst others) *signal* and *pool* type protocols. The essential properties of these are summarised in Table 9. It can be seen that the pool type protocol is equivalent to a construct such as that shown in Figure 40.

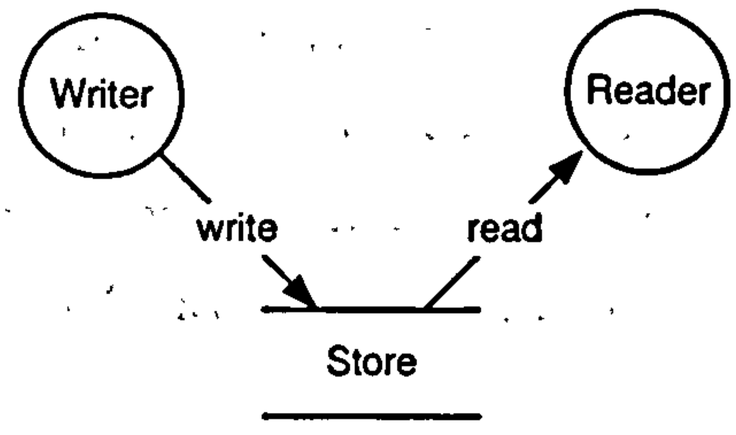


Figure 40 - Write to / read from store as equivalent of Mascot pool protocol

There are two possible approaches to analysing such a communication. *Either* it can be considered as two separate flows, the writer part and the reader part, which are analysed separately, *or* it can be analysed as a single flow, but with multiple interpretations of each of the guide words to consider the different effects of deviations in reader and writer behaviour.

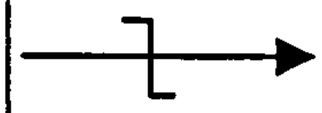
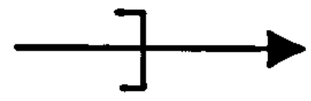
Protocol		Signal	Pool
Symbol			
Data properties	Read	Destructive	Non-destructive
	Write	Destructive	Destructive
Dynamic Effects	Reader	Can be held up	Cannot be held up
	Writer	Cannot be held up	Cannot be held up

Table 9 - Properties of MASCOT Pool and Signal communication protocols

It is sensible to ensure consistency of analysis approach across an entire project. This requires the project safety team to study the chosen design notation to identify the preferred approach in situations such as that described above. Step 1 of the SHARD analysis will then include a review of the project-specific analysis approach.

A brief description of the intended behaviour of the flow, together with its data type, communication protocol and timing information should be recorded.

Step 4: Ensure intended operation is safe

Compare the description of the generation, transmission and use of the data from step 3 with the overall functional description and safety requirements reviewed in step 1. Note any inconsistencies or inadequacies, and make recommendations as to how they should be rectified (see step 8).

It is important to remember that system safety must always be considered with respect to the system *intent*, and not the *specification*, which may itself be faulty.

If it is not clear how a part of a design is intended to work, or if information is missing, it is acceptable for the recommendation recorded to be a requirement for further investigation to clarify the design or supply the missing information. This will, of course, mean that the analysis will have to be revisited once the additional information is made available to ensure that it is acceptable.

Step 5: Use a guide word to suggest a deviation.

Having investigated the information flow in normal operation, the next step is to study what unintentional behaviour is possible, and what its potential effect on safety would be. This is the main body of the analysis and SHARD uses a set of guide words to prompt consideration of possible deviant behaviour. SHARD uses only a small set of guide words, but each may potentially be interpreted in several different ways to suggest a range of deviations.

The guide words are based on the concept of a *service*. A service is the communication of a piece of information, with a specific value, at a particular time. The guide words suggest ways that this might go wrong. They are:

Omission:

The service is never delivered, i.e. there is no communication.

Commission:

A service is delivered when not required, i.e. there is an unexpected communication.

Early:

The service (communication) occurs earlier than intended. This may be *absolute* (i.e. early compared to a real-time deadline) or *relative* (early with respect to other events or communications in the system).

Late:

The service (communication) occurs later than intended. As with early, this may be absolute or relative.

Value:

The information (data) delivered has the wrong value.

A single guide word may have many interpretations, each suggesting different deviations.

Each deviation that is considered should be recorded with as clear a description as possible. It is not satisfactory to simply note the guide word, as other people reading or updating the analysis in future may not interpret the guide words in exactly the same way.

There are several important points to note.

1. There are situations in which there are no meaningful interpretations for a guide word at all. A simple example is in the case of an interrupt; this is simply a stimulus, indicating that an event has occurred, and carries no data at all. There is therefore no meaningful interpretation of the guide word "value". This must be recorded (i.e. enter N/A as the deviation) – do not simply omit the guide word.

2. The fact that a deviation can be suggested does not necessarily imply that it is likely to occur, or even that it is possible. The role of step 6 (investigation of causes) is to determine which of the suggested deviations have plausible causes.

3. Some plausible deviations could be suggested by more than one of the guide words. For example, consider the case of passing of a data record containing several elements. The guide word "omission" could suggest that, although the communication has taken place, one or more of the elements in the record is missing (*partial* omission). Alternatively, the same deviation could have been suggested by the guide word "value" – since the record has been passed, it could be considered that there has been no omission, but the missing data constitutes an incorrect record value. It does not matter for the purposes of SHARD which view is taken in such situations; however, it is important that similar instances are treated consistently throughout the analysis.

4. Timing and value deviations can occur together. For example, an output could be both late and have the wrong value.

Case studies have identified the following "variants" of guide words as generally helpful prompts. Note, however, that these are intended for guidance, and are not necessarily exhaustive, or applicable in all circumstances.

Omission:

- *Total* – there is no communication at all.
- *Partial* – part of a communication is missing; may suggest either an incomplete data structure, or one of a series of related communications is missing (e.g. a packet missing from a sequence that carries a long text).

The following cases of omission are applicable to iterated communications:

- Single – a single iteration is omitted
- Repeated – several (or possibly all) consecutive iterations are omitted
- Periodic – every n^{th} iteration is omitted.

For example, consider an output that is supposed to occur at regular intervals over a period of time from t_1 to t_2. The guide word "omission" could suggest the following different deviations:

- just one of the intended output events is missed (single omission);
- the output never occurs in this period (repeated omission);
- the output occurs as intended for the first few repetitions, then ceases (a different case of repeated omission);
- the output occurs throughout the period, but every 3^{rd} iteration is omitted (periodic omission).

These cases are shown graphically in Figure 41. Each of these cases might have different effects and need to be considered separately.

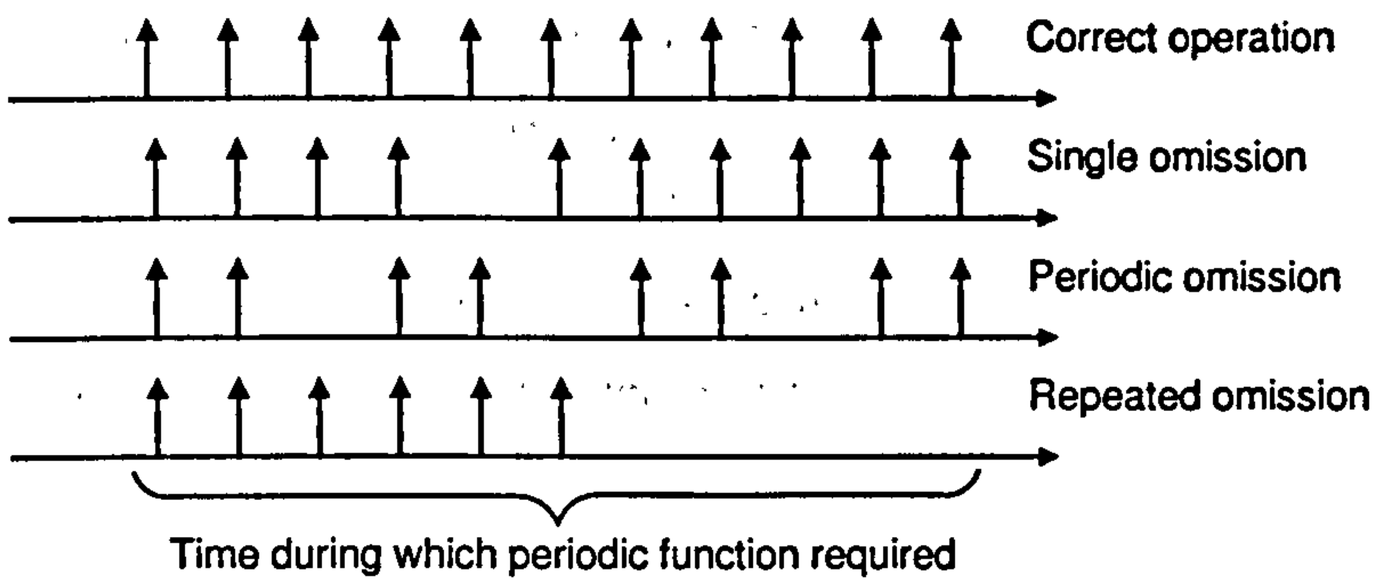


Figure 41 - Illustration of different cases of *omission* failure

Commission:

- Repetition – an expected communication is repeated when it should not have been.
- Spurious – there is a completely unintended communication.

Early and Late:

As noted above, the timing guide words may be interpreted with regard to absolute (real) time, or relative to some other system event. There are two important cases of relative timing:

1. Events (communications) in sequences.

   All possible incorrect orderings of a set of events can be represented as some combination of omission and commission. For example, if the intended sequence of events ABCD was to actually occur as ABD, that it clearly an omission. The incorrect

sequence ABDC could be considered to be an omission (of C) followed by a commission (of D). However, it is much more natural to think of this second example as a timing problem – either C is late with respect to D, or D early with respect to C. Particularly in multi-processor systems, there are many practical interpretations of this form of lateness.

2. Events (communications) in a system with a cyclic structure.

   The great majority of embedded computer systems have some form of cyclic schedule. Even those that use a dynamic schedule often work by breaking time down into a set of short periods or "frames". In such systems, it is often convenient for analysis purposes to study the timing of events and actions with respect to the beginning or end of a cycle or frame.

   In practice, the beginning and end of a cycle also almost always define the boundaries between timing failures and omission or commission. If an event occurs at the wrong time, but within the correct cycle, it is simply a timing failure. If, however, an event is so early or late that it falls in the wrong cycle, it must be considered to have been omitted in the cycle where it was required, but will turn up as a commission failure in the cycle into which it has fallen.

Value:

Value deviations are so application specific that no additional generic prompts have been identified. In practice, the most important properties of value deviations are related to the detectability of the deviation, which is discussed in step 8.

Step 6: Investigate causes

Having suggested a deviation, the next steps are to consider whether it has plausible causes (step 6) and/or potentially hazardous effects (step 7).

Propagation and Transformation

For the investigation of both causes and effects of a deviation, the concepts of *propagation* and *transformation* of deviations are extremely important. These are the way in which unintended behaviour of one component can cause other components to also behave incorrectly.

Consider the sequence of functions, data flows and data stores in Figure 42. If function A fails to produce the raw_data output, function B will be unable to produce the filtered_data output which should have been placed in the store. The failure of A has *propagated* to B.

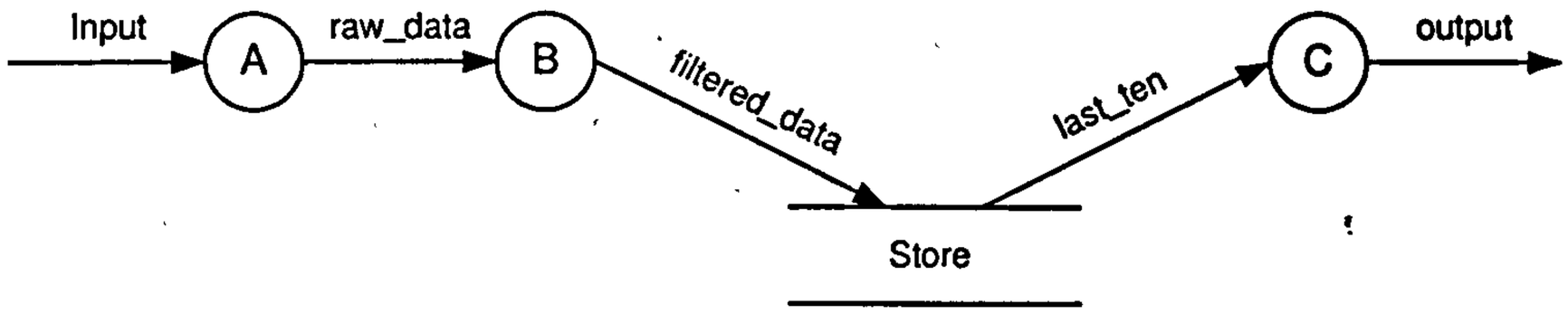


Figure 42 - Illustration of failure propagation and transformation

As the data in the store has not been updated, function C will also be forced into unintended behaviour. Exactly what it does will depend upon implementation details. If C can detect the lack of new data, it may take action such as extrapolating from preceding values, or perhaps simply flagging the fault. If C cannot detect the lack of new data, it will simply collect the out-of-date data in the store and produce an output which, although it may be mathematically correct given the input, is faulty in that it no longer represents the state of the real world. In any of these cases, the deviation has been *transformed* from the original omission into some sort of value error.

In carrying out a SHARD analysis, the individual information flows in the system are considered individually. Clearly, however, these sequences, or chains, of propagated deviations will be important, as the effect of a deviation in one flow will usually be to cause a deviation in one or more of the other parts of the system.

The primary – secondary – command rule

In searching for potential causes of a deviation in an information flow, the *primary – secondary – command* rule should be used. This provides prompts for groups of potential causes to consider. It is similar to the primary – secondary – command rule used in fault tree construction. The simple arrangement of processes and communications shown in Figure 43 illustrates some important cases to consider.

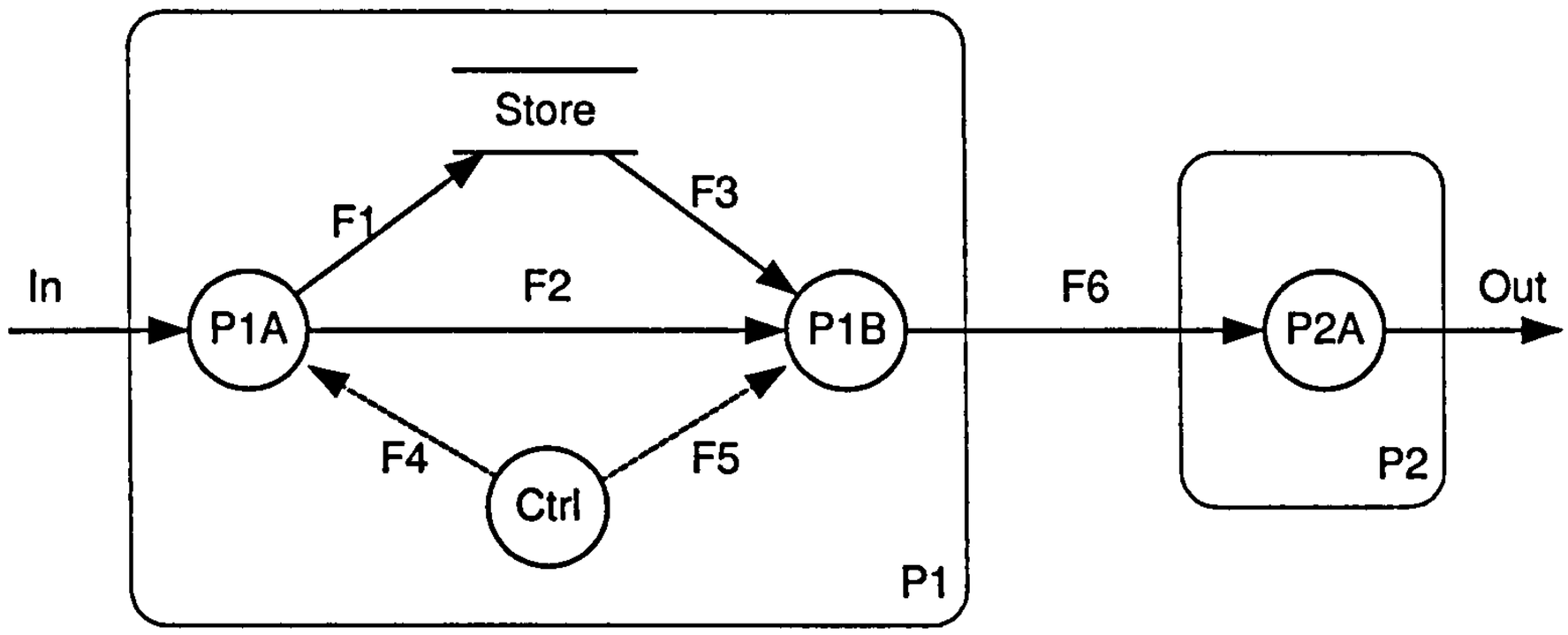


Figure 43 - Simple system illustrating generic causes in SHARD

Primary causes of a deviation are those which are failures of the information flow itself. Consideration should include:

1. Failure of transport medium

   This prompts investigation of hardware causes directly affecting the flow of the data. This is of particular interest for flows between hardware units (e.g. F6 in Figure 43), although hardware failures within a single unit should be considered.

2. Failure of flow source

   This represents cases where there is a fault within the process in which the flow originates. Note that cases where the source of the flow fails due to external problems are considered separately; therefore, for software, causes in this group are always design or coding faults. Typical examples considered are infinite looping, excessive execution times meaning processes are swapped out before producing output, exceptions etc.

3. Failure of flow initiator

   The flow initiator is the active process that initiates the data transfer, and is not necessarily the same as the flow source. For example, in Figure 43, the data source for F3 is the passive store; the initiator is process P1B. Note again that this group of causes does not include failures due to external causes.

Secondary causes of a deviation are those cases where the incorrect behaviour is caused by deviations in components that are not part of the flow itself, but have influence over it. There are two major categories of secondary cause:

1. Hardware failure affecting flow source or initiator
2. Incorrect inputs to flow source

This group of causes includes deviations in all of the *data* inputs to the source of a flow. In Figure 43, the causes of deviations in flow F6 would therefore include deviations on flows F2 and F3.

Command deviations occur when a flow is capable of operating correctly (i.e. there are no primary causes of deviation) but it is "told to do the wrong thing". Specific cases include:

1. Incorrect control of flow source or initiator

   This group of causes includes deviations in the *control* inputs to the source of a flow. In Figure 43, this group of causes for F6 would include deviations in flow F5.

2. Incorrect scheduling of flow source or initiator

   Includes cases where there are problems with the underlying operating system which prevent processes that could otherwise run correctly from being scheduled on time (or ever).

As with the deviations suggested by the guide words, the fact that a potential cause is suggested by the primary – secondary – command rule does not necessarily mean that it is possible. Equally, the rule is only guidance, and may not suggest some of the causes that are actually possible. It should therefore be applied carefully, and each suggested cause should be reviewed to decide whether it is plausible. Record the plausible causes of the deviations identified with as much detail as possible. Where the causes arise from deviations in other flows, it is sufficient to note this; when these flows are analysed, the appropriate cause / effect pairs should be cross-referenced.

If no plausible causes can be found for a deviation, this should be recorded; this will be sufficient justification for accepting that this deviation cannot represent a contribution to a hazard. However, it is good practice to examine (briefly) the potential effects of the deviation anyway, in case later analysis suggests a cause that has been overlooked.

Step 7: Investigate effects

Evaluate the potential effects of each deviation identified. At the system outputs, the effects of deviations should be investigated with reference to system hazard identification and risk assessments, to decide which deviations must be treated as potential hazard causes. For each deviation, record the system level hazards that it can cause or contribute to. If an output deviation cannot cause or contribute to a system level hazard, this should also be recorded.

Within a system, the majority of the effects will normally be causes (or contributions to causes) of deviations in other flows which have already been studied. In this case, all that is necessary is to record an appropriate cross-reference. Any deviation that can cause or contribute to a deviation that has already been identified as a potential hazard cause must also be considered as a potential hazard cause *unless* there are mechanisms included in the design that are sufficient to detect and mitigate the deviation.

When examining the effects of any deviation, but especially deviations on the system outputs, note any *contributing factors* that may alter the effects of the deviation. For example, if the system has alternative *modes* of operation, consider whether the effects of a deviation will be different in each mode.

Step 8: Examine detection, protection and mitigation

For each deviation that has been identified as a potential hazard cause, determine whether the system already incorporates mechanisms to detect and protect against, or mitigate, the effects of the deviation.

It is helpful to start this investigation by considering the *intrinsic detectability* of the deviation, i.e. whether would ever be possible to detect. If a deviation is intrinsically undetectable, this should be noted. If not, consider whether the mechanisms in place are suitable, likely to be effective, and whether they mitigate the deviation sufficiently that it no longer needs to be considered as a potential hazard cause.

Note that it is very rare for a deviation to "disappear" completely, i.e. for a process to be able to completely mitigate the effects of faults on its inputs. Detecting the fault and flagging it may be the *specified* behaviour for a function, but it is still a deviation from the *intended* fault-free behaviour of the system. In general, the only components that can ever completely stop the propagation of a deviation are voters, which may be able to completely ignore, or mask the effects of, a fault on one of their input channels.

Step 9: Decide on acceptability and make recommendations

The last part of the analysis of each deviation is to decide whether the management *of that specific deviation* within the design is acceptable, to justify that decision, and to make

recommendations where necessary. If no plausible causes have been found for a suggested deviation, or if it has been shown that it is not a potential hazard cause, this will normally be sufficient justification for acceptance (the only exceptions being cases where the analysis has revealed serious non-safety-related flaws in the design, which may necessitate redesign and repeat analysis).

There are three main types of recommendations that can be made in SHARD:

a. Accept the design as proposed. This is implicit if no other recommendations are made, and does not need to be recorded.

b. Further investigation required. This conclusion should be recorded (with details) if the analysis could not be completed because there were questions that could not be answered from the information supplied.

c. Design modifications required. In deciding on recommended changes to the design, it is important to make a realistic assessment of how much rework is justified by the problems that have been identified. In making recommendations, think about the *hierarchy of risk reduction actions*. In decreasing order of preference, these are:

   1. *Eliminate the hazard.* Ideal, but often impossible, and prone to involving extensive redesign.
   2. *Reduce the probability of the hazard arising.* A good option if it can be achieved with simple changes. If additional detection and protection mechanisms are required to achieve the reduction, the implications of the additional complexity should be carefully considered.
   3. *Mitigate the effects of the hazard.* As with 2, beware of excessive complexity.
   4. *Provide alerts and warnings.* A "last resort", which should only be considered if there are no other viable options.

   It is not necessary to specify complete detail of design changes, but it is helpful to indicate the preferred strategy.

In the worst case, where very serious safety problems have been identified, complete redesign may be the only acceptable option.

Repeat steps 5 to 9 for each guide word, then repeat from step 2 until all of the flows in the system have been examined. Note that if the analysis concludes that the design is badly flawed and completely unacceptable, it is preferable to stop the analysis at that point, and develop a new design.

Whilst conducting the analysis, it is important always to consider the design exactly as it is presented. DO NOT assume any changes to the design (even corrections to "obvious" mistakes, or the implementation of analysis recommendations). This results in an analysis that is not consistent with any issued design level. It is also very easy to overlook the assumed changes. Anything that needs to be altered, no matter how trivial, should be recorded in the analysis recommendations, where it can be tracked.

Step 10: Summarise the analysis

The final step in the SHARD process is to produce a report summarising the analysis findings and highlighting key recommendations.

7.9 Conclusions

This chapter has described the development of the SHARD analysis method, highlighting the case study results which led to refinements of the method, and also to the principles for effective safety analysis proposed in Chapter 5. It has also presented a complete SHARD method description, containing all of the guidance notes and additional rules derived from the practical work undertaken.

Chapter 8
Low-level Interaction Safety Analysis (LISA)

This chapter discusses the development of the second of the two new safety analysis techniques based on the principles discussed in Chapter 5. Low-level Interaction Safety Analysis (LISA) was developed in response to a request from BAe MA&A. The requirement was to provide evidence about the safety of a new cockpit display system, which was one of the first military avionics systems to employ a segregated operating system (i.e. one in which software functions of different integrity levels are run on the same processor). The LISA technique was developed specifically to help provide an argument of correct and robust partitioning of this segregation system.

Since the case study was integral to the development and evaluation of the technique, the chapter starts by presenting the case study requirements in more detail, and discussing the principles that were used to structure and define the scope of the work. The LISA concepts and method are presented, and the chapter concludes by examining results from its practical application to the case study system.

8.1 Background

8.1.1 Case Study Analysis Requirements

The cockpit display system case study that motivated the development of this analysis approach, and which was used to test and refine the method, was a large, multi-processor system. Hazard analysis had shown that there was only one critical hazard associated with the system, which was the continued display of incorrect primary flight information to the pilot. The system also contributed to a number of less critical hazards. The design work was largely completed at the time the case study started, but development was in progress and, as well as providing evidence for inclusion in a safety case, there were opportunities to influence some design decisions.

In order to achieve the redundancy and diversity required for a critical hazard, the aircraft was fitted with a pair of identical systems (live and hot stand-by). Within each system, displays were generated as a series of symbols, which were output to screen driver hardware. Further hardware read back the final output to the screens, and converted it back into a list of symbols. For the

critical flight information, a monitor function within the display system performed a computation to determine the input data that should have resulted in the actual output. This was compared with the original input data, and any discrepancy was taken as an indication of internal failure, in which case the display was blanked, control switched to the hot stand-by, and the failed system re-initialised as the new stand-by. The arrangement of one system is shown schematically in Figure 44.

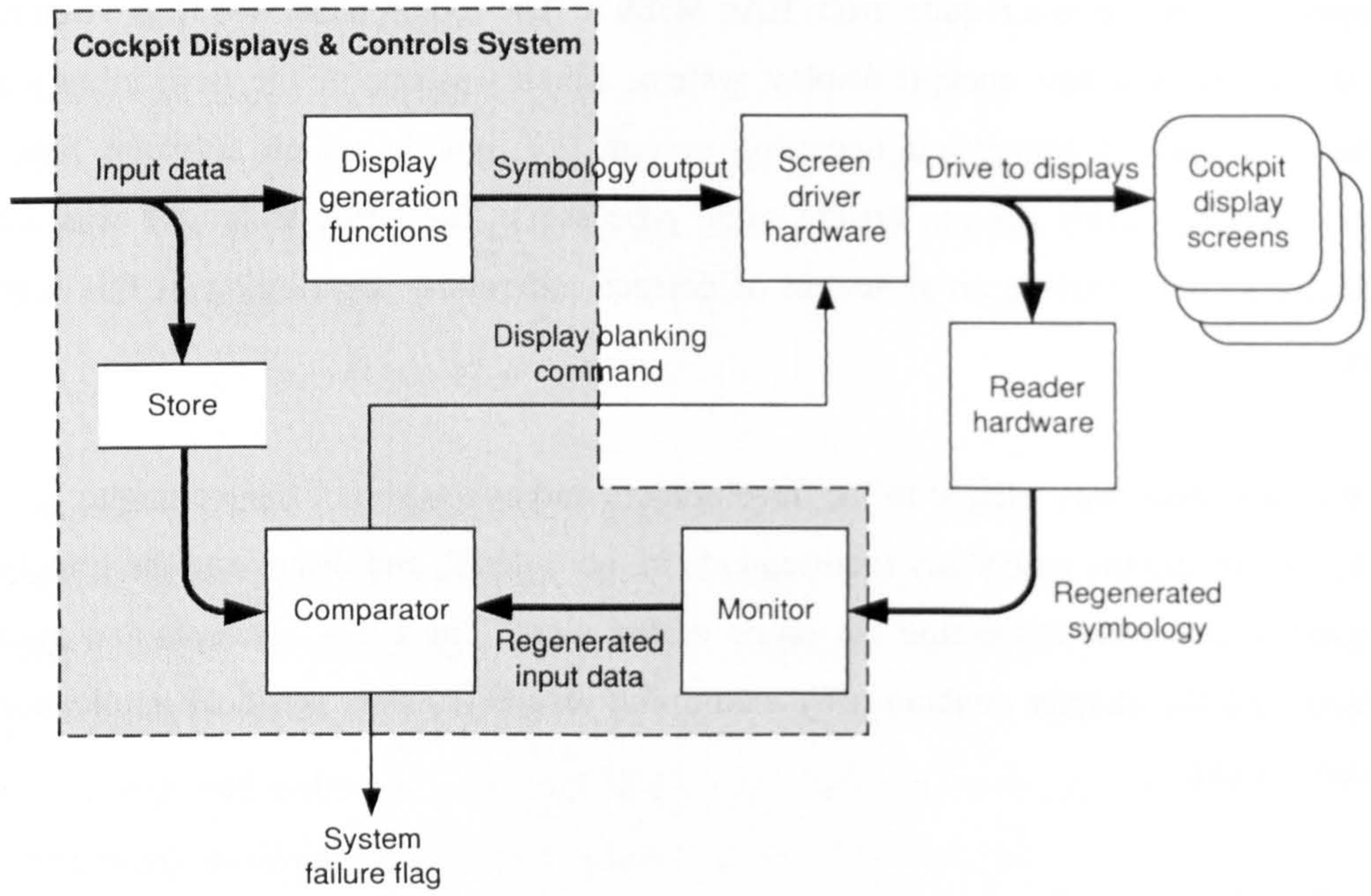


Figure 44 - Cockpit display system schematic

The system was developed in accordance with project specific standards, which predate, but are very similar in approach to UK Defence Standard 00-56 [78], and parts of 00-55 [77]. The hazard analysis of the system proposal determined that the monitor and comparator functions were the most critical parts of the system and, in accordance with the standards, these were developed to the highest integrity level (Risk Class 1). The remaining functions were all developed to Class 2 standard.

The relative proportions of Class 1 and Class 2 software, together with the relatively high levels of data sharing between the two classes of software, meant that the conventional approach of partitioning out the Class 1 software to separate processors would have resulted in very inefficient use of resources in a heavily loaded system. The design team therefore took the decision to implement a segregated software system, with a core of Risk Class 1 software

operating on all processors to implement initialisation, communication and scheduling / synchronisation tasks, in addition to the critical monitor function.

Although this was a completely bespoke development, the Class 1 software effectively provided some of the functions of a rudimentary operating system. These included:

- memory and I/O protection
  (the static memory map meant that dynamic memory management was not required)
- synchronisation of activity across all processors
- scheduling

The Class 1 software did not provide the uniform interface to hardware that would be expected of a full conventional operating system (i.e. application functions were able to access hardware directly).

Evidence was required that the segregation system genuinely provided an acceptable level of independence between the Class 1 functions and the rest of the system. No existing analysis technique could be identified which could provide this evidence so, with the agreement of the certifying authority, it was decided to develop a new approach. It was decided that the Class 1 software which implemented the segregated software environment should be analysed independently of the rest of the system – in effect, it was to be treated as an operating system to be analysed in isolation, without detailed application knowledge. The primary reason for this decision was that it was hoped to make the Class 1 analysis completely independent of the Class 2 software, so that changes could be made to non-critical parts of the system without the need to repeat the complete safety analysis of the Class 1 parts – realising one of the benefits of operating systems identified above.

8.1.2 Case Study Safety Principles

The first challenge for a "context free" analysis of an operating system (i.e. not based on detailed knowledge of the application safety implications) is to determine what properties are required of the operating system in order to ensure safe operation of the application.

In discussion with BAe MA&A, the primary requirements identified for acceptability of the segregation scheme were:

1. **Data flow corruption must be prevented**

   -Modification of Risk Class 1 data by Risk Class 2 software shall not occur

2. **Control flow corruption must be prevented**

   - No action of the Class 2 software shall prevent the Class 1 software from executing when it should
   - Modification of Risk Class 1 code by Risk Class 2 software shall be prevented

3. **Corruption of the execution environment shall not occur**

   i.e. corruption of processor registers, device registers and memory access privileges shall not occur.

From these a further, secondary, requirement was identified:

4. If any of the primary requirements 1 to 3 is violated, this shall be detected, and the system caused to blank the screens and shut down promptly, setting a failure flag so that the stand-by unit can assume control of the displays.

Note that requirement 2 must be interpreted carefully, in that requirement 4 means that Class 2 software errors may cause a shutdown, thus preventing further Class 1 execution. This is acceptable, as it is the defined safe state of the system. However, the self-shutdown procedures, which are executed if any hardware failures or internal inconsistencies are detected, are a part of the Class 1 software, and this gives a further derived requirement:

5. The ability of the Risk Class 1 self-shutdown procedures to run to completion (i.e. safe shutdown) must be guaranteed regardless of the actions of the Class 2 software.

It was further agreed that requirement 4 applied only to cases of single failure, i.e. the analysis did not need to consider cases of multiple simultaneous or near-simultaneous failures. This limitation was justified by two arguments:

1. Self-shutdown procedures were very quick and simple. It was considered that, provided that the first failure did not prevent the shutdown procedures from being initiated, the probability of a second, independent failure occurring that could prevent their completion was negligible.

2. In the case of multiple, coincident failures of sufficient severity to prevent orderly detection and shutdown, the probability that the system could continue to provide a coherent and believable display at all was considered negligible (i.e. severe coincident failures were expected to result in a degree of display breakdown that would be obvious to the pilot even without blanking of the screens).

There were also a number of specific safety-related timing requirements, such as the maximum allowable time form detection of certain classes of fault to system shutdown. Most of these timing requirements were expressed in terms of the maximum number of display refresh cycles for which a fault could be permitted to persist, based on assessments of how long it would take the pilot to perceive and respond to changes in the display.

In order to meet the requirement that the analysis produced should be as independent of the application software as possible, it was decided that analysis should proceed by assuming that all application software (except the critical monitor function itself) would always behave in the worst conceivable way. This was interpreted as meaning that any failure of protection, synchronisation or scheduling would always result in the lower integrity software causing the maximum possible interference to the operation of the critical function. If satisfactory protection of the critical function could be demonstrated under this assumption, no future change to the lower integrity software could invalidate the safety arguments.

8.1.3 Case Study Analysis Approach

It was clear that a concrete analysis was required for the case study, the main purpose being to provide evidence of achievement of safe partitioning, not to influence the development of a design. However, it was initially difficult to identify a tractable approach to concrete analysis of such a large and complex system. Traditional inductive analysis was impossible, as there was no data on known (or even expected) failure modes of any software or hardware system components. Equally, a functional deductive analysis was impossible, since there was an explicit requirement not to base the analysis around application functions (i.e. the Class 2 software) which were expected to be subject to change.

The analysis method developed was based around the identification of *arguments of acceptability* for the usage of every *resource* in the system. This method was called LISA (Low-level Interaction Safety Analysis) and is described in sections 8.2 and 8.3 below. Section 8.4 describes the results of applying the technique to the case study system.

8.2 LISA Principles

Instead of analysing the system functionality, the LISA method focuses on the interactions between the software and the hardware on which it runs. A set of *physical resources* and *timing events* is identified, and a set of projected failure modes of these resources is considered. Unlike the design-driving role of HAZOP or SHARD, however, the aim of this analysis is to use a combination of inductive and deductive steps to produce arguments demonstrating that either no plausible cause can be found for a projected failure, or that its consequences would always lead to an acceptable system state.

This section describes the identification and classification of resources, the failures to consider, and the type of argument structures that make up LISA analysis. Section 8.3 then presents a step-by-step description of the analysis method.

8.2.1 Identifying and Classifying Resources

Two classes of resources are identified and used for LISA analysis; physical resources, and time.

Physical resources consist of the processor registers, memory locations, I/O and other special registers. This is, effectively, the programmer's model of the hardware; hardware features such as buses, arbitration logic etc. are considered in terms of the registers that control them. For any specific combination of software and hardware, physical resources can be partitioned into classes based upon the *criticality* of the resource usage. There are five classes of criticality:

- *intrinsically critical*

  those resources which contain safety critical data at any point in the execution of the software, or the program code for safety critical functions; examples include I/O and RAM used by safety critical functions, processor registers etc. For a general purpose operating system, where it is not known in advance which functions are critical, or where dynamic memory management is employed, it may be necessary to regard all memory locations, I/O etc. that is available to application processes as intrinsically critical. However, in the case study system, with its static memory allocation, it was possible to limit this classification only to those resources that were directly used by the Class 1 software (i.e. the monitor function or the segregation system itself).

- *primary control*

these are resources which directly control the use or function of an intrinsically critical resource; examples include memory management unit (MMU) registers, I/O control registers etc.

- *secondary control*

  resources which either provide a backup to primary controls (e.g. a secondary MMU giving redundancy in memory protection), or control access to primary resources (for example, key registers which must be set to particular values before MMU registers can be altered).

- *non-critical*

  resources which are never used by critical software, and do not affect the operation of any part of the hardware which is used by critical functions.

- *unused*

  locations in the memory map which do not correspond to a physical device. The importance of these locations is that there should be no attempts by any part of the software to access them; such an attempt indicates a failure, and must be trapped and handled safely.

The model of time used in the analysis is based on the identification of discrete timing *events* that have associated hardware actions. Examples of these include interrupts, the use of system timers and counters, and synchronisation actions. Again, this is a model familiar to programmers.

Timing events can be identified as either critical or non-critical, depending upon whether they affect the execution of critical code. Note that there will be a set of primary (and possibly some secondary) control resources associated with each timing event. For example, primary resources associated with a timer-generated interrupt will include the control registers for the timer, and CPU registers that determine the response to the arrival of the interrupt.

As a basis for analysis, this model has several advantages:

1. it is possible to ensure that the set of resources analysed is complete (i.e. includes the entire memory map, all non-mapped devices such as processor registers, and all interrupts and synchronisation events);
2. the model is familiar to the system's designers and programmers, so it is possible to discuss safety analysis in well-understood terms;
3. although potentially large, the set of resources in any system is of a fixed and predetermined size, so the effort required for analysis can be predicted reasonably accurately in advance.

8.2.2 Resource Dependencies

From the descriptions of the resource classes, it is clear that there are dependencies between resources; that is, the state of one resource affects the behaviour of another. Indeed, this is explicit in the definition of primary and secondary control resources. However, there are other, less direct dependencies. The most significant of these is that, in most systems, there must be an initialisation phase, in which the software configures the hardware to the state required for the execution of the main body of the application. However, this initialisation code is, itself, run on the very hardware it is configuring, so a circular dependency is created – an example is shown in Figure 45.

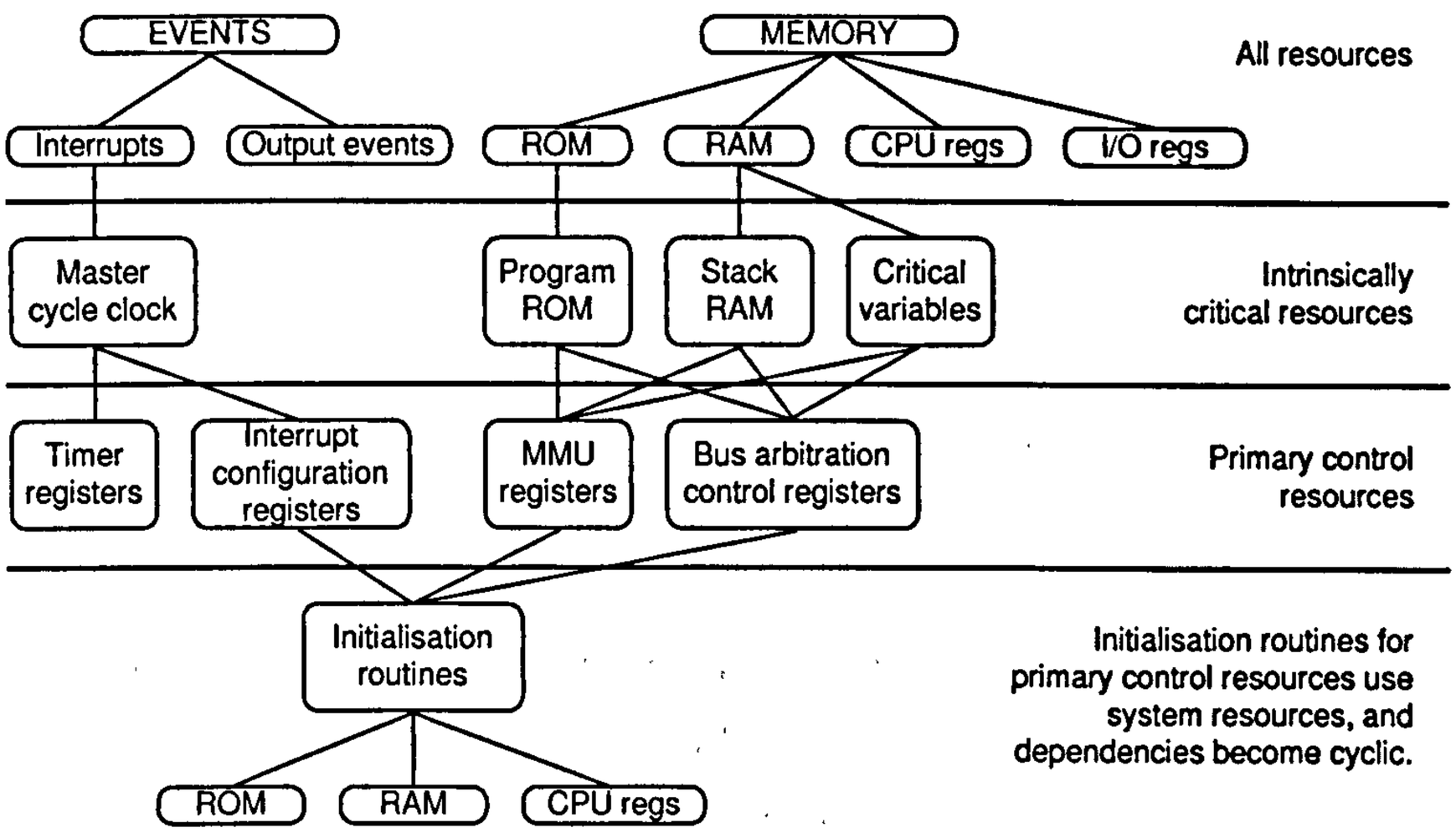


Figure 45 - Illustration of circular resource dependencies

A complete safety argument for the system must therefore demonstrate that the system powers up in a safe state, and respects minimum safety requirements throughout every stage of initialisation. To guarantee the correct execution of the application, it must also be shown either that successful completion of the initialisation guarantees that the hardware is correctly configured, or that it is impossible (or at least extremely improbable) that the main body of the software could fail to detect, and safely respond to, any incorrectness in its execution environment.

8.2.3 Safety Arguments for Resources

Having identified the timing events and resources in the system, and assigned appropriate criticality classes, the acceptability of their implementation and use must now be demonstrated. The arguments made must consider both normal (intended) operation and the effects of failures. There are many fault tolerance strategies; the aim of this section is not to discuss or compare these, but rather to consider some general properties which are relevant whatever the system architecture.

Because the criticality of data and calculations is application dependent, it is not possible to make general arguments for the safety of *intrinsically critical* resource usage based on knowledge of the hardware and operating system alone. It is entirely the responsibility of the application designer to demonstrate that the intended (normal) operation of the system is safe. However, study of the underlying hardware is very important in understanding the behaviour of the system in the presence of failures. There are two categories of failure to consider; failúre of the hardware implementing the resource itself, and failures in the configuration and protection of the resource arising from faults in primary control resources.

In both monolithic systems and those with operating systems, the arguments which can be made for the tolerability of hardware failure in intrinsically critical resources depend upon two factors; the improbability of that failure, and the provision of appropriate protection mechanisms for critical data and operations within the application. For example, in a simple, single channel system, the integrity of critical data held in RAM may be checked by storing the same value in two locations and comparing them before use. For temporary storage (e.g. intermediate results in critical calculations) this may not be viable, so the calculation may be repeated, or the results of two alternative algorithms compared. The effectiveness of these strategies depends upon the improbability of two hardware failures resulting in identical but incorrect results.

An alternative strategy may be to argue that intermediate values are stored for so little time that the effective exposure to random hardware failures is negligible. This argument may prove fallacious if a single calculation contains many steps that use previous intermediate results, or if the calculation is repeated frequently, in which case the proportion of the time for which values are stored in the same temporary location may prove too high.

Many of the possible resource protection strategies in application software depend upon the ability to control, or make use of, features of the hardware. For example, storing two copies of

data gives greater protection if the two locations are in separate devices (RAM chips), potentially avoiding sources of common mode failure such as faulty address decoding or 'stuck at' faults on individual devices. This is relatively easy to achieve in monolithic software, where the programmer has direct control over the hardware. For an operating system to offer similar protection, it may be necessary to implement special features (possibly with complementary compiler directives) to provide the application programmer with the necessary control. However, the benefit of this will be that it is also possible to provide generic 'argument fragments' or *patterns* (a concept developed in Kelly's DPhil thesis [43]) which can be applied each time the feature is used.

Failures of intrinsically critical resources arising from faults in, or incorrect management of, primary control resources are of particular concern because they have the potential to cause common mode failures, possibly invalidating any of the above arguments. However, this is an area where generic arguments may be made about the interaction of the operating system and the hardware. Since there will normally be relatively few primary control resources in a system, it is feasible to devote significant time and effort to the analysis of each one.

So, for intrinsically critical resources, there are three essential strands to a safety argument:

1. Safety of normal usage – argument is the responsibility of application developer.
2. Toleration of hardware failure – argument is the responsibility of application developer, but with support from hardware and operating system analysis.
3. Correct management via primary controls – argument is primarily the responsibility of hardware and operating system analysis.

For *primary control* resources, the safety arguments that can be made depend on so many factors that it is impossible to give general guidelines for the safety argument. If there is a *secondary control* resource which duplicates the behaviour of the primary for redundant protection (e.g. an external device which duplicates the memory protection functions of the MMU), it may be sufficient to argue safety simply from improbability of coincident failure, provided that initialisation is not a potential source of common failure. More probably, it will be necessary to identify means of detecting and managing the effects of a failure. For example, if the MMU is incorrectly configured so that a process is denied access to memory locations it requires, can it be shown that the resultant errors are always trapped and lead to the system taking appropriate action?

Similarly, for *secondary control* resources, the argument will depend entirely on the role of the resource, and no general guidance can be given.

For *non-critical* resources, there is no need for an argument of safety, merely a justification of why the resource is considered to be non-critical. For the final resource class, *unused* locations, only one argument is normally required. As any attempt to access such a location is necessarily an error, the argument must show how the system will trap and respond to such an attempt.

8.2.4 Failure mode identification in LISA

Most of the arguments outlined in section 8.2.3 require an assessment of the effects of failures, and it is now necessary to consider an appropriate model of failure. It is infeasible to consider every type and cause of failure of each device in a computer system individually, so it is necessary to make an abstraction.

Experience with HAZOP and SHARD (Chapter 7) has shown that the classification of computer system failures proposed by Bondavalli and Simoncini [7], and modified for SHARD, is applicable to many aspects of computer systems, and the same structure is used as a basis for LISA. Interpretations of the failure categories (those used in the cockpit display case study) are introduced below, together with some suggested modifications and extensions.

As with SHARD, the basic failure categories are:

- *Omission* – a particular service is not provided
- *Commission* – a service is provided when it is not required (i.e. a perfectly functioning system would have done nothing)
- *Early* – the service is provided before the time (either real time, or relative to some other action) at which it is required
- *Late* – the service is provided after the time at which it is required
- *Value* – the timing is correct, but the value delivered is incorrect.

Suggested interpretations of these guide words for both events and physical resources are given in the LISA method description in section 8.3 below.

8.2.5 Failure mode interpretation and arguments of acceptability

Using the failure mode categories to structure the analysis of timing events is relatively straightforward, and is described in the procedures in section 8.3. The procedure for physical resources is more involved, and requires more explanation.

There are two major factors to consider in developing arguments of acceptability for the physical resources in a system – the hardware structure, and the software usage of the resource. These generally divide physical resources into blocks in various ways, and the LISA analyst can develop arguments that apply to complete blocks. For example, any failure causes arising from physical hardware malfunctions will normally be the same for an entire device (e.g. a complete RAM chip, or all devices accessed via the same bus), and can be addressed by the same argument. Similarly, a single argument can often be developed for all resources that are used in the same (or related) way(s) by the software – for example, the entire area of RAM used for the processor stack, or storage of a particular class of variable. An important structuring and effort management principle is therefore the early identification of these blocks of resources, and the development of the *generic arguments* that apply to these blocks.

It is important to note that the physical resource failures being considered in LISA are very low level; they are failures *within a single access* of a physical device. Thus, in general, only the hardware will determine the potential causes and effects of timing failures in accessing resources, because they will be errors in a single processor cycle or single bus transaction, which is normally shorter than (or equal to) the execution time of one single processor instruction. The exceptions to this are cases where the hardware is designed such that devices other than the processor can become bus master, e.g. for direct memory access (DMA) transfers.

The most important components of an argument of acceptability for a physical resource timing failure will therefore be a description of the possible causes (if any), together with the (hardware) mechanisms for detection of bus errors and the (software) response when errors are detected.

The argument of acceptability for omission and commission (access permission) failures will need to consider failures in both the memory protection hardware, and in the software which initialises the hardware, e.g. incorrect configuration of MMU address maps. Mitigation will be difficult to show unless the hardware incorporates independently initialised redundant memory protection devices (which is unusual), and the argument may simply have to appeal to the reliability of the device(s) used.

Value failures similarly need to consider both hardware and software causes and mitigation. For an operating system analysis, it may not be necessary to consider value failures of data (variable storage) or program RAM (i.e. most of the intrinsically critical resources) in detail *provided* that data and program corruption are included in the application safety analysis. Primary and secondary control resources will, however, always need to be addressed.

8.2.6 Selection of analysts

As with all hazard and safety analyses, there are many factors that will influence the selection of appropriate people to conduct a LISA analysis. This is a very low-level technique, requiring extremely detailed knowledge of both the software and hardware design; case study experience has shown that this is difficult and time-consuming for an outsider to obtain. However, LISA is intended to be a demonstration of achieved safety, and it is reasonable to expect that LISA analysts will be required to be as independent as possible.

Case study experience showed that easy access to members of the design team was vital, and it seems likely that the most effective way of applying LISA would be to have a member of the design team working in an explanatory role, together with an independent analyst whose responsibility would be to produce the final report and conclusions.

8.2.7 Review of LISA principles

The LISA principle is simple; the physical resources and timing events in a system are identified, and arguments of acceptability developed for projected failure modes.

Clearly, to produce a complete argument of overall safety using this approach requires that:

a) the set of resources identified for analysis completely and correctly represents the system, and
b) the projected failure modes investigated cover all the possible failure modes of the real system.

Requirement a) is relatively easy to satisfy, as it corresponds to checking for complete coverage of the system address map, and all the interrupts and synchronisation events designed into the system. It is impossible to guarantee that requirement b) has been satisfied; however, this is no worse than determining completeness of any other analysis, and can be resolved for practical purposes by a process of review and agreement that the set of possibilities investigated is acceptably complete.

8.3 LISA method

This section outlines the conduct of a LISA analysis, following the method developed for the cockpit system case study.

Step 1: Agree principles for acceptability

Since the purpose of LISA is to produce arguments of acceptability for the safety related behaviour of system resources, the first step must be to determine the basis for acceptability. This will vary from project to project. If the system to be analysed is being developed specifically for one application, and hazard and safety analyses are available, the LISA acceptance arguments will be able to relate low-level behaviour to system level effects. For systems that are being developed as components without detailed knowledge of the eventual application, the acceptance criteria will depend upon the target integrity level of the system, and might typically include requirements to demonstrate that:

- the intended usage of a resource does not allow low-integrity processes to access / alter high integrity data;
- no plausible cause can be found for a suggested failure;
- a suggested failure has been shown to be mitigated by a completely independent mechanism (i.e. different hardware and independently coded software);
- a suggested failure has been shown to produce no effect on the correct operation of the critical functions of the system;
- a suggested failure may have an adverse effect on the correct operation of critical functions, but the failure can be reliably flagged to application code, which can implement acceptable handling or mitigation.

Step 2: Assemble source material

The minimum required source materials for LISA are:

- an overall description (specification or design) of the intended operation of the system, including strategies for managing expected failure modes;
- a complete system memory map;
- definitions of the purpose and usage of all special device registers, I/O etc. This may take the form of programmers' manuals for each hardware device in the system;
- a specification or design document which describes all the timing events in the system;

- specification or design documents which define the system start-up, initialisation, exception handling and normal and emergency shutdown processes.

Additional sources that may be necessary or useful for some analyses include subsystem specification or design documentation, hardware failure data, and program source code.

Step 3: Analyse timing events

From the system documentation, identify all the timing events which involve a hardware / software interaction. These will include all uses of interrupts, system timers, counters or clocks, inter-processor synchronisations and time-dependent interactions with external devices or other systems. Care must be taken to ensure that events used in every mode of system operation, including initialisation, shut down etc., are included.

For each event identified, describe its intended operation, including the preconditions necessary for the event to be generated (e.g. programming of timers) and the correct response(s) to the event. Ensure that the intended behaviour defined does not in itself create potential safety problems (e.g. check that context switches initiated by interrupts cannot leave critical data in an inconsistent state, that interrupt mask levels in the new context are appropriate etc.)

Step 3.1: Suggest deviations from intended behaviour of events

Use the guide words *omission*, *commission*, *early* and *late* to prompt consideration of possible deviations from the expected operation of each event. Each guide word may suggest more than one deviation.

Omission – the failure of an event to occur. Possible cases of omission to consider include:

- Source (writer) omission

  All events will have a single source, which in this case fails to produce the expected event. Depending on the type of source it may be necessary to consider whether it is plausible for the source to proceed to its next expected action, or whether it experiences a fail-stop condition. A special case of this is *precondition* omission, in which the event is generated when a set of preconditions is satisfied, and the preconditions are never true.
- Transmission omission

  The medium through which the event is transmitted "loses" the event.

- Destination (reader) omission

  The destination process(es) or object(s) that was (were) expected to recognise and respond to the event do(es) not do so. In a multiprocessor system, where events affecting more than one processor are possible (e.g. broadcast interrupts or synchronisation events), it is necessary to consider symmetric (where no recipient responds to the event) and asymmetric (one or some recipients respond) omission.

Commission – the spurious occurrence of an event. Source, transmission and destination may apply as for omission, and it may also be necessary to consider:

- Number of commission errors

  A single unintended event may have a different effect to multiple repetitions

- Repetition *vs.* insertion

  An expected event that is repeated may have a different effect from a completely unexpected event. This particularly applies when events are expected in a predetermined sequence, which is violated.

Again, there may be symmetric and asymmetric cases to consider in a multiprocessor system.

Early and *late* may be interpreted either with respect to real time, or as prompts to consider incorrect ordering (relative timing). They may also prompt consideration of jitter (i.e. where supposedly regular periodic events actually occur at irregular intervals). It should also be noted that it is important to define the boundary between an event that is late, and one that is considered to have failed entirely (omission), and similarly for early / commission.

Step 3.2: Investigate possible causes of event deviations

For each of the deviations suggested in step 3.1, identify possible causes, making certain that both direct hardware failures and indirect causes in software (e.g. incorrect programming of control registers) are considered. If no plausible cause can be found for a deviation, this should be noted.

Step 3.3: Investigate effects of event deviations

Investigate the effects of each deviation for which plausible causes were found in step 3.2. Consider how the deviation will be detected and handled by the system, and ensure that this will

result in a safe system state. If there are no detection and handling mechanisms for the deviation, consider what its ultimate effect on system safety will be.

Step 3.4: Produce arguments of acceptability

Decide which of the principles of acceptability agreed in step 1 is appropriate for each suggested deviation, and produce arguments of acceptability showing how the system design meets the principles. These arguments do not need to be extensive, but should be sufficient for a reviewer reading the analysis to identify all of the components and mechanisms involved. If a suitable argument of acceptability cannot be found for a suggested deviation, this is an indication of a possible flaw in the system design.

Record the investigation of the deviation, and repeat for each suggested deviation of every identified event.

Step 4: Analyse physical resources

Identify physical resources in the system. The primary information source for this is the system memory map, although it may be necessary to consult processor, device and subsystem documentation to identify the function of all of the registers within the blocks allocated to each device. As for events, describe the intended usage of each resource, ensuring that start-up, initialisation, shutdown and other modes are considered, and ensure that the intended use does not in itself create safety problems.

Step 4.1: Group resources by common factors

There is no need (and, indeed, it would be impractical) to analyse every aspect of every resource separately. Resources should therefore be grouped by common characteristics so far as possible. For example, access permissions are typically assigned to blocks of memory locations; the effects of granting inappropriate access (or denying access where it is requires) can be examined for the whole block of locations. Similarly, access timing properties will usually apply to all of the locations within a single hardware device; again, these can be examined once for all locations within the device. If the initial suggested grouping is not correct, this will become apparent as the analysis proceeds; either identical results will be obtained for many resources, suggesting they

should be grouped, or the analysis of a block will show that the effects of deviations vary for different locations within the block, showing that it must be split.

Step 4.2: Classify resources

Resources are classified as *intrinsically critical*, *primary control*, *secondary control*, *not critical* or *unused* (the definition and implications of each of these classes is discussed in section 8.2.1 above). The purpose of this classification is to help determine the argument requirements for each resource (discussed in section 8.2.3 above).

Step 4.3: Suggest deviations from the intended use and operation of each resource.

Use the guide words *omission*, *commission*, *early*, *late* and *value* to prompt consideration of possible deviations from the expected use and function of each resource. Each guide word may suggest more than one deviation.

- *Omission* and *commission* are interpreted as access permission violations. An omission failure occurs if a process that should be able to access a resource is denied permission. Commission failure occurs where a process is granted access to a resource that it should not have.
- *Early* has two interpretations in the case of a physical device such as memory, both leading to (unpredictably) corrupt data:
  - the processor reads from a location in the device, and attempts to latch the data from the bus before it is stable, or
  - the processor writes to a location in the device, and de-asserts data before the device has latched it correctly.

  These may seem unlikely failures, and only possible with poor hardware design. However, there are systems in which parameters such as the number of wait states inserted on accessing a particular device are programmable; the system can dynamically alter its own timing characteristics. In such systems, this type of timing failure is plausible and extremely important.
- *Late* refers to delay in accessing the resource, arising either from effects such as contention for a shared bus, or from the same type of configuration fault that could lead to *early* failures. Unlike *early* failures, lateness cannot cause data corruption. Excess wait states in a device access will merely extend the cycle and result in reduced performance; the data will be held stable until the end of the cycle. Late latching of data being read by the processor (i.e. after it

has been de-asserted by the device being accessed) is highly unlikely, since the processor itself generates the timing control signals that the control the device. In general, therefore, *lateness* is only of interest in our analysis if the delay is great enough to be treated as an *omission*, e.g. by triggering a bus timeout.

- The *value* of a resource is its data content. For control resources, the correct value can often be determined in advance, and the effects of changes predicted. In the case of memory (RAM or ROM) the effect of unwanted changes can only be determined with knowledge of the application software.

Step 4.4: Investigate possible causes deviations in resource use or function

For each of the deviations suggested in step 4.1, identify possible causes, making certain that both direct hardware failures and indirect causes in software (e.g. incorrect programming of control registers) are considered. If no plausible cause can be found for a deviation, this should be noted.

Step 4.5: Investigate effects of deviations in resource use or function

Investigate the effects of each deviation for which plausible causes were found in step 4.4. Consider how the deviation will be detected and handled by the system, and ensure that this will result in a safe system state. If there are no detection and handling mechanisms for the deviation, consider what its ultimate effect on system safety will be.

Step 4.6: Produce arguments of acceptability

As with the event analysis, decide which of the principles of acceptability agreed in step 1 is appropriate for each suggested deviation, and produce arguments of acceptability showing how the system design meets the principles.

Record the investigation of the deviation, and repeat for each suggested deviation of every resource. It is helpful to produce two tables to record the analysis for each block of resources, one containing the investigation of properties such as access permissions and timing which apply to all resources in the block, and one which records deviations, effects and arguments which are specific to a single resource. As generic arguments become apparent (i.e. those that are used repeatedly), a separate table of these should be compiled to reduce the volume of the analysis.

8.4 The Cockpit System Case Study

The case study system design philosophy and analysis requirements have been outlined in section 8.1. This section presents further detail of the system, the analysis work carried out, and some fragments of the analysis output.

8.4.1 System definition

The basic structure of the case study system hardware is shown in Figure 46. The processors were arranged in pairs, each processor having its own private bus, giving access to RAM, ROM and timers, and to the arbitration logic for access to the shared local and system buses. In addition, a secondary MMU on each private bus provided redundant protection of critical memory areas. In order to ensure system consistency and guarantee the critical function access to the system bus and I/O when required, the system employed synchronised cyclic schedules, executing a high integrity code segment on all of the processors at the same time in each cycle.

The segregation of Class 1 code from the rest of the system was achieved through the use of the processor supervisor and user modes – Class 1 software in supervisor mode, and the rest of the tasks in user mode.

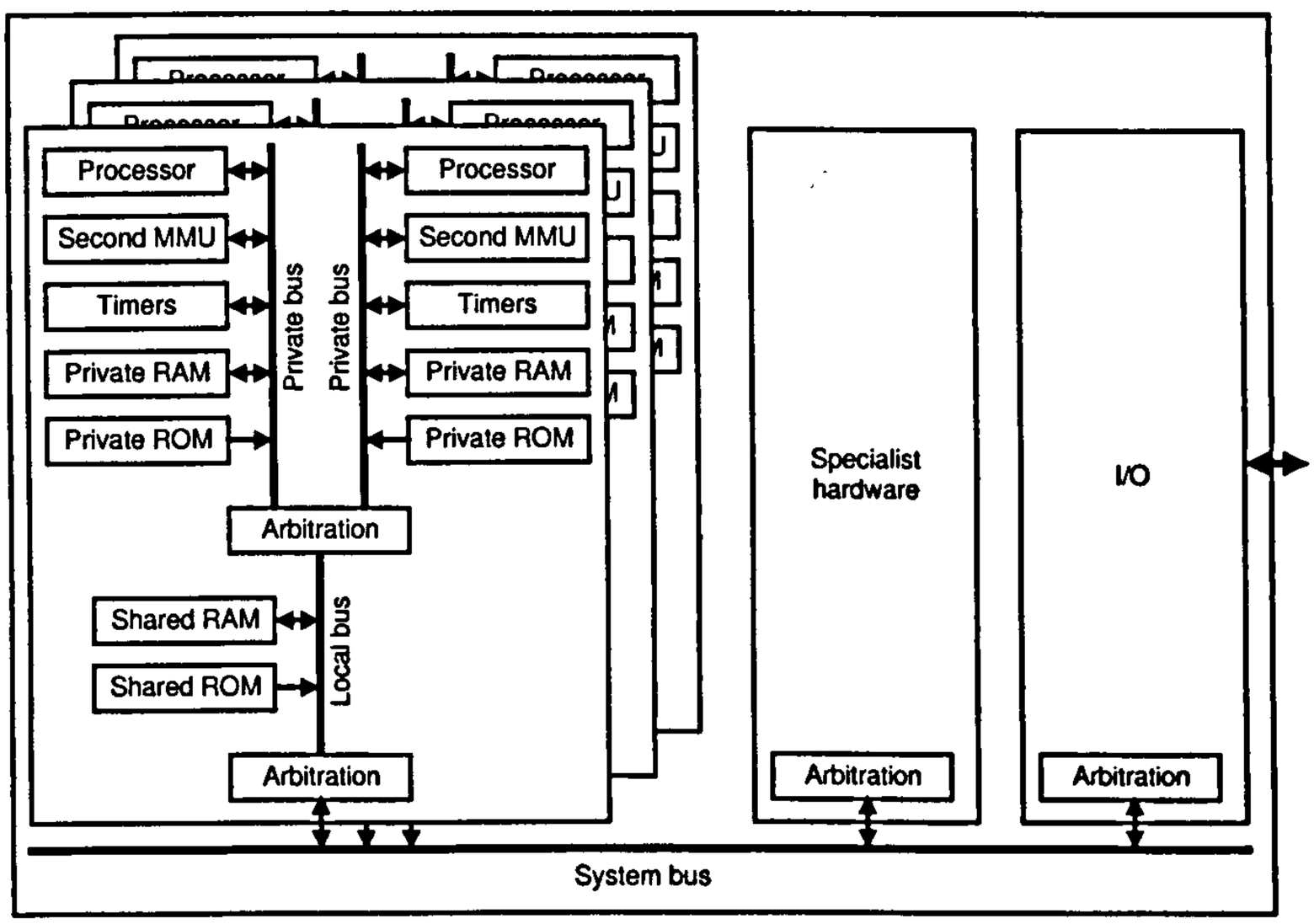


Figure 46 - Case study system hardware

The part of the system selected for analysis (the Class 1 software segregation system) was specified in a document approximately 200 pages long. Device manuals and subsystem specifications took the total source documentation for the analysis to approximately 1250 pages.

8.4.2 Analysis

The LISA analysis of the cockpit system was carried out by one analyst (the author) with considerable assistance and support from the project team at BAe MA&A, and also from Dr. Neil Audsley, who carried out a timing analysis of the system which was included in the final report. The case study extended over an elapsed period of approximately 30 months, during which time it received about 10 man months of direct effort. Of this, approximately 4 months was dedicated to developing an initial detailed understanding of the design and intended operation of the system. The initial analysis took approximately 2 months of effort, and there were then a number of updates of the design, which necessitated repeating or updating parts of the analysis. The final report took a further two weeks to write.

The problem of understanding the design was compounded by the extended duration of the case study. It was found that, without daily involvement in the project, the extremely detailed knowledge required to perform the analysis was rapidly lost, and a significant period of re-familiarisation was required before any updates or modifications were made to the analysis. The design team at BAe MA&A were extremely accessible and helpful, and this helped to compensate for limitations in the analysts' understanding of the system.

Arguments of compliance with the safety principles were considered complete when either:

- the failure had been shown to be mitigated by two or more independent mechanisms
- the failure had been shown to produce no effect on the correct operation of the segregation system or Class 1 application software

or

- the failure had been shown to produce no effect on the correct operation of the segregation system, but there were potential effects on the Class 1 application software. In this case, the need for subsequent application software analysis was noted.

When completed, the analysis consisted of 7 tables of event analysis which, together with supporting notes, occupied 22 pages, and 18 tables of physical resource analysis, which occupied a further 20 pages. The complete final report contained 98 pages of safety analysis, which

included method descriptions, the LISA tables, and 22 pages of Functional Failure Analysis carried out on two custom ASICs. The FFA results were compared with the LISA analysis of the device registers contained in the two ASICs. The comparison showed that the LISA analysis contained every failure mode considered in the FFA, with a few additions and rather more detail, but was rather less readable. This last point may be an issue of style, or may reflect the fact that the FFA was added to the report in response to a specific request for the analysis of these particular devices to be presented in a format which was more familiar to the reviewers, and particular care was therefore taken in the structure and wording of this analysis.

In conducting this analysis, it was found that the method was extremely good at highlighting areas where the design intent was not clearly expressed in the available documentation, and several recommendations were made for improvement in the specification and supporting documentation. No safety related problems were found in the actual design at any stage, although completing the arguments of acceptability for some items required quite extensive investigation, including requests for additional information from subsystem suppliers. In a number of cases, the analysis also suggested additional tests that could be incorporated into the built in test (BIT) routines to improve the detection of certain classes of deviation. Most of these suggestions made use of self-test capabilities already present in the hardware, and all of the suggested additional tests were incorporated into the system. The LISA analysis was also used to investigate the required iteration rate for a number of the continuous built-in test (CBIT) routines in order to meet the worst-case detection and shutdown response times agreed in the safety principles.

In a number of cases where particularly complex mechanisms were involved in the detection and handling of faults, the analysis recommendations also included bench tests involving fault injection to prove that the system response would be as predicted. It was also found that, when design changes were required for other reasons (e.g. to improve system performance), the results of completed parts of the analysis were valuable in developing modifications and in selecting between alternatives.

The final conclusion of the report was that the system was suitable and sufficiently robust for its intended use; however, this was strongly qualified by a number of limitations and exclusions. These included:

- The analysis was conducted on the design *as presented*; there was no attempt to verify that the implementation matched the design, and that this was a task that was essential to ensuring the accuracy and applicability of the report's conclusions.

- Throughout the analysis, it was assumed that previous hazard and risk assessments were correct, i.e. that the hazards were as described, and that the identified safe states were acceptable.
- The safety principles outlined in section 8.1.2 were assumed to be sufficient to guarantee safe operation of the system.
- No attempt was made to ascertain, either quantitatively or in broad qualitative terms, the availability or reliability of the system. It was noted that it is explicit in aircraft level safety analyses that blanking the pilot's displays is acceptable provided that it occurs infrequently, and even then normally only for a few seconds whilst control of the displays is transferred to the hot standby system. Extremely low levels of reliability or availability, resulting in excessively frequent switch-overs, or simultaneous failure of both master and standby systems, forcing the pilot to switch to reversionary instrumentation, could invalidate the conclusions about the acceptability of screen blanking as a safe state.

The report was presented to BAe MA&A and to customer representatives at the Critical Design Review. A number of comments were received; these were mostly typographic errors, requests for clarification, or comments on the system design. No written comments (either adverse or positive) were received about the analysis approach, although two of the customer reviewers commented verbally that they found the analysis thorough and credible. The customer representatives have formally requested BAe MA&A to ensure that the LISA analysis is maintained as the system progresses through the remaining stages of development and commissioning to in-service status.

There was relatively little development of the method over the period of the case study; the initial method outlined at the start of the study was found to be practical both in terms of workload and quality of output. The biggest problem was in obtaining a sufficiently detailed understanding of the study system – as noted above, this took nearly half the total time expended. If the analyst had been drawn from the project team, or at least had more of a background in the application domain, this should not be such a significant burden.

The case study identified few aspects of the analysis approach that had not been anticipated. The only significant change made to the method during the case study was in the identification of blocks of related physical resources that could be analysed together. The initial approach tried was to only group resources that were identical in every respect. After a progress review, a decision was taken to allow grouping where most attributes (usually timing and protection) were

the same, and to document differences individually. This reduced the size of the physical resource analysis tables by approximately 50 percent.

The case study highlighted some areas where features of the study system significantly facilitated the construction of arguments of acceptability. These included:

- An easy to achieve safe state in which the system could stop. This meant that proving that detected failures were handled correctly was relatively easy. The task would have been far harder had there been a requirement for continued operation.
- Redundant MMUs, independently initialised, which permitted a simple and sound generic argument for memory protection safety.
- Only two classes of software to be segregated, allowing the use of processor modes to implement segregation, with supervisor mode used for the Class 1 application and segregation system code, and user mode for all other application code.
- Memory allocation was static, and the physical and logical memory maps of the system were the same (i.e. the MMUs performed no address translation). It was therefore a simple task to identify all the areas of memory that could ever be involved in a critical process.
- The tight synchronisation between processors meant that many failures could be detected and mitigated by related events on other processors.

8.4.3 Sample analysis output

This section contains samples of the three types of tables produced by the case study. Due to the sensitive nature of the case study, details in the tables have been altered, but the style and technical nature of the entries is representative.

Table 10 contains a sample of the timing event analysis, in this case for the master interrupt which prompted Processor 1 to send the broadcast synchronisation interrupt to all processors at the start of each cycle. Note that the "Dis-chgd" column was used as a check column; "YES" was entered into this column when all argument requirements had been discharged, i.e. a satisfactorily complete argument of compliance with the safety principles had been found.

Table 11 contains samples of the generic arguments developed for blocks of physical resources, which were then used in the analysis tables as shown in Table 12. Note that, as so much information was contained in the generic arguments, or the whole device arguments in the top part of the table, the entries in the address-specific sections of the table were mostly extremely brief, as shown here. The only exception to this was in the analysis of special device registers,

where complex arguments were frequently required. Again, the "Dis-chgd" column is a check column for completion of the argument.

8.5 Conclusions

This chapter has presented an overview of some of the safety analysis problems associated with the transition from bespoke, monolithic safety critical systems to systems designed around more conventional operating systems. It has argued that operating systems present considerable challenges for software safety engineering, particularly in the areas demonstrating partitioning within software so that separate and independent application components can be identified, and in the specification and description of the safety properties of these components. The LISA analysis method has been described, along with the major case study that motivated its development.

The work presented in this chapter is of a rather different nature to that on SHARD in Chapter 7. The safety analysis method presented has been extensively exercised on a live system development project, where its effectiveness has been demonstrated. However, this has been the only trial, and further studies are required to determine how generally applicable the technique is.

Master (start of cycle) Interrupt – periodic, 20ms, processor 1 only						
Guide Word	**Deviation**	**Cause**	**Effect**	**Detection / Protection**	**Dis-chgd**	**Justification / Comments**
Omission	No master interrupt (more than 5ms late, as this is the limit imposed by the protection mechanism)	• Hardware failure • Programmable timer not programmed correctly • Interrupt control not programmed correctly • Interrupts remain masked during user mode	Processor 1 will remain in user mode indefinitely	There is a back-up interrupt programmed for 25ms on Processor 2, which is reset on receipt of each broadcast interrupt. If this interrupt ever occurs, Processor 2 will run the shutdown procedure, log Processor 1 failure, and set the system failure flag.	YES	• Failure will be detected and shutdown initiated on first occurrence. • Backup mechanism uses completely diverse hardware (i.e. different processor, no reliance on VME bus)
Commission (single)	Excess master interrupts (interrupt occurs less than 20 ms after preceding interrupt)	• Hardware failure • Programmable timer not programmed correctly	If Processor 1 is executing in supervisor mode, no effect until context switch to user mode, since interrupts masked. In user mode, will cause immediate re-entry into supervisor mode, resulting in shortened (no) time for user mode software. Result will be single-cycle user mode over-run.	Single spurious / early interrupts may not be detected. If this failure occurs in the same cycle as, or in the cycle subsequent to, a user-mode software over-run, then effects etc. will be as multiple commission.	YES	• All segregation mechanisms will function as intended despite mis-timed cycle.
Commission (multiple)	Excess master interrupts	As above.	As above, except multiple-cycle user mode over-run is certain.	Multiple spurious / early interrupts will be detected by the user mode software completion detection mechanism. Repeated short user mode time slots will cause either overrun into a third cycle, or repeated overruns into a second cycle, either of which will result in shutdown.	YES	• Failure will be detected and result in shutdown within a maximum of 4 cycles. • Backup mechanism relies on Processsor 1 hardware, but not on timer hardware, which is most plausible cause of failure.
Early	Master interrupt early by less than the slack time in user schedule	• Hardware failure • Programmable timer not programmed correctly	Small reduction in master cycle time. Possible slip before effects become as commission depends on slack in user mode schedule.	Will not be detected.	YES	• All segregation mechanisms will function as intended despite mis-timed cycle.
Early	Master interrupt early by more than slack time in user schedule	As commission	As commission	As commission	YES	As commission
Late	Master interrupt occurs less than 5ms late (i.e. between 20 and 25 ms after start of cycle)	• Hardware failure • Programmable timer not programmed correctly (NB - a single bit error in timer control registers may increase time by 4ms)	Cycle time extended by up to 5 ms	None.	YES	• All segregation mechanisms function as intended despite longer cycle.

Table 10 - Sample of LISA event analysis output

	Applies to	Justification
G1	Unused location (no hardware present)	• Attempted accesses to non-existent hardware indicate program corruption, and will result in system shutdown via one of the following mechanisms: • These locations are mapped out by MMU, and accesses will result in bus error on read or write; this will result in an exception in either user or supervisor mode, which will be handled to shutdown. • CPU bus timer in custom ASIC provides a redundant mechanism. If an erroneous access is not refused by the memory protection hardware, an access to a non-existent device will result in the bus timer generating a transfer error (TEA) signal; again, the result will be an unexpected exception handled to shutdown.
G2	Processor MMU functionality duplicated by second MMU	• This applies to access permissions to all locations • The two MMUs are always run in parallel, and both must give permission before a location can be accessed. This is 'fail safe', in that both devices have to grant permission for access; if either one denies it, a bus error handled to shutdown will result. Thus a failure in which Risk Class 2 software is granted a permission which would enable it to corrupt a Risk Class 1 resource requires simultaneous failure of both MMUs. • The MMU mapping tables are set up by separate sections of initialisation code.
G3	Read-only device register enforced by hardware	• Writes to these locations are ignored by the device (have no effect). • An attempt to write to such a location is indicative of software error, but it cannot further damage the system state.
G4	Program code and static configuration data areas	• Given the degree of synchronisation within the supervisor mode cyclic schedule, and the mechanisms which exist to detect erroneous user mode behaviour, it is extremely unlikely that corrupt program code could execute for any significant period of time before causing a detectable failure. This also applies to mapping tables etc., as corruption of these will cause similar effects. The only plausible undetectable failure would be omission of a section of user mode code, so user mode functions include cross-checks on successful completion (normal defensive programming techniques are sufficient).
G5	User mode RAM area	• No corruption of user mode data can lead to violation of segregation system integrity, as the worst case behaviour which can be produced is the malicious Risk Class 2 behaviour already assumed.
G6	Unused location (device register within ASIC)	• Attempted reads from unused registers return 0, thus no detection at this level • Attempted write accesses to unused registers will cause a bus error, which will result in an unexpected exception handled to shutdown.

Table 11 – Sample generic arguments used in the LISA case study

EEPROM

Arguments applying to whole device:

Guide Word	Deviation	Causes	Dis-chgd	Justification
Omission	Denial of read access to user or supervisor mode software, or supervisor write permission during loading	Failure of one MMU	YES	• G2 applies
Commission	Granting of write permission except during loading	Requires simultaneous failure of both MMUs	YES	• G2 applies
Early	Processor attempts to latch data off bus before it has stabilised	Incorrect number of waitstates inserted on access	YES	• See note 1 in discussion of timing analysis.
Late	Excessive latency between device access and data read	Incorrect number of waitstates inserted on access	YES	• Local device - no arbitration • See note 1 in discussion of timing analysis.

Address - specific comments:

Start Address	End Address	Description	Criticality	Reason	Dis-chgd	Justification
		Supervisor Mode Area			-	
010000	01FFFF	Supervisor Mode translation tables	Intrinsically critical	Tables used to initialise CPU MMU at initialisation. Affect boundary between Class 1 and Class 2 software	YES	• Region is check-summed and validity checked at initialisation • G4 applies
020000	03FFFF	Run-Time System	Intrinsically critical	Run time system code used by Class 1 software including segregation system	YES	• As above
040000	0FFFFF	Supervisor Mode software	Intrinsically critical	Class 1 code including segregation system	YES	• As above
100000	1CFFFF	Subsystem Supervisor Mode software	Intrinsically critical	Class 1 subroutines from subsystem suppliers – used by Class 1 software including segregation system	YES	• As above
		User Mode Area				
1D0000	1DFFFF	User Mode translation tables	Not critical	User mode area	YES	• G4 applies
1E0000	2CFFFF	User Mode software	Not critical	User mode area	YES	• G4 applies
2D0000	3BFFFF	Subsystem User Mode software	Not critical	User mode area	YES	• G4 applies
3C0000	3FFFFF	**NOT USED**	Not used	-	YES	• G1 applies

Table 12 - Sample of LISA resource analysis output

Chapter 9
Evaluation

This chapter reviews the new work presented in this thesis, and considers the contributions that it has made both to safety critical systems theory, and in the development of two new analysis techniques.

9.1 Contribution to safety analysis theory

This thesis contains two sections that develop theoretical material; the discussion of concepts in Chapter 3, and the principles for computer system safety analysis in Chapter 5.

The primary purpose of the discussion in Chapter 3 is not to advance new ideas, but rather to examine the concepts underlying safety analysis, which are not explicitly identified in most texts. In doing so, some helpful new ideas emerge about some problematic areas of safety terminology and techniques. Standard definitions of hazards have always been unsatisfactory, since most capture very few of the defining characteristics discussed in section 3.1. Some of the alternative views of hazards suggested here have already been incorporated into teaching material, and have proved very useful in helping students to identify hazards in practical examples.

The discussion of faults and failures in section 3.2 is, in effect, a response to a debate about the definition of safety engineering. One view, predominant in American literature, is that analysis of failure is reliability engineering, and that safety engineering is much broader, requiring demonstration that the correct behaviour of a system is also safe, and that appropriate measures have been taken to mitigate hazards which cannot be entirely managed within the system design (e.g. exogenous hazards). The predominantly European alternative view, that safety engineering is fundamentally about identifying and managing failures, is conceptually very similar once it is understood that European literature, in general, defines failure much more broadly. In this second view, failure is defined with respect to intent, so includes any failures within the engineering process that lead to an unsafe system. The discussion in section 3.2 explains the terminological differences in a way that may help to resolve this debate.

Section 3.3 essentially builds up a "library" of ideas that are incorporated into existing successful safety analysis techniques. The significance of these for computer system safety analysis is that,

as the later chapters on SHARD and LISA show, many of these concepts are directly applicable to computer systems. The discussion of the roles and classification of techniques in section 3.4, expanding on Fenelon's proposed matrix, is significant in that it is the first classification to clearly distinguish between concrete and projective analyses.

The major technical proposal of the thesis is the set of principles for defining effective computer system safety analysis techniques in Chapter 5. As the conclusion to Chapter 5 explains, one of the major motivations for developing these principles was the observation that many recent proposals for new analysis techniques do not appear to have understood the practical requirements of industry, nor to have learnt from existing successful approaches in other disciplines. The major test of the principles, therefore, must be whether they can be applied to produce useful, industrially acceptable analysis techniques.

Since the development and evaluation of a new analysis technique requires significant effort and is extremely time consuming, it would be impossible to state with certainty that the principles are correct. However, the success of both the SHARD and LISA techniques (discussed in more detail in sections 9.3 to 9.5) provides some evidence of their value. Perhaps more significant, however, is the difference between the ways in which SHARD and LISA were developed. Work on SHARD started before any of the principles had been proposed. As Chapter 7 shows, the SHARD development process was relatively slow, with significant revisions following each of the major case studies. Many of the principles in Chapter 5 were derived as generalisations of the resolutions to the problems encountered with SHARD. The work on LISA, on the other hand, was begun after the principles had been suggested. The general approach and details of the method were developed in accordance with the principles and, as Chapter 8 shows, the first major application of the technique to a live industrial project was extremely successful, with no significant alteration required to the initial method definition. Again, this cannot be considered more than a single piece of evidence, but it is very encouraging.

9.2 Evaluation of new safety analysis techniques

It is difficult to provide a rigorous evaluation of new safety analysis techniques. All safety critical systems development projects are, in effect, open loop experiments, since there is no feedback to the safety process after the system has been commissioned, except when there are accidents or incidents. The accident rates achieved by current engineering processes are so low that, unless there was a serious mistake, it would be impossible to obtain statistically significant results for comparative trials assessing the performance of a new technique against existing methods. Given

the scale of typical safety critical systems projects, the cost of such a comparative trial, which would have to duplicate the entire development process from the point of divergence onwards, would in any case be prohibitive.

More feasible experimental schemes could be devised, perhaps by evaluating the effectiveness of new and techniques at finding the problems in a deliberately flawed system design. It would not be sufficient simply to measure the proportion of errors found; the interesting result would be how much more (or less) effective the new technique was than other existing methods. Again, there are significant practical problems with such a comparison. The first is that, in general, there will be no existing analysis method for which a new technique is a direct equivalent. SHARD is, perhaps, unusual in this respect, in that it is very close to the version of HAZOP that has been defined in Def Stan 00-58, but there is certainly no existing technique with capabilities similar to those of LISA. This means that it would be extremely difficult to compare the results of two techniques.

Also, safety analysis (especially projective analysis) is inherently subjective. This means that, unless the same people worked on each of the analyses, it would be impossible to decide with certainty whether differences in results were a product of the different skills and abilities of the participants, or reflected the capabilities of the technique itself. However, to have the same people involved in the application of both techniques is also unacceptable, since they would inevitably bring ideas developed during the first analysis into the second.

In general, it seems that experiments using contrived "toy" examples will not give much indication of the (industrial) practicality of techniques, or of their ability to detect the subtle hazards and hazardous failure modes which typically cause problems for real project.

Since the ultimate goal of this research was to produce safety analysis techniques which are genuinely practical and useful to industry, it was decided to base the evaluation of SHARD and LISA on a range of activities, with an emphasis on industrial scale trials. Table 13 shows the activities carried out for each technique, listed in order of the strength of evidence (closely related to the maturity of the technique) that they provide.

Evaluation	SHARD	LISA
Comparison between techniques using toy examples	✓	✓
Academic case studies based on industrial systems	✓	✓
Peer review through publication	✓	✓
Industrial case study (by originator)	✓	
Industrial evaluation (by third party)	✓	
Application to live industrial projects		✓
Industrial acceptance	(✓)	(✓)

Table 13 - SHARD and LISA evaluation activities

The development of both techniques began with a number of small scale experiments comparing alternative approaches, and progressed to case studies based on real industrial systems but carried out by university research staff. At this point, the development of SHARD and LISA diverged.

Because of the extremely high cost (and potential liability risk) of safety critical systems projects, for a company to accept new procedures or techniques voluntarily (i.e. not because they are contractually required or mandated by standards), it must be convinced of the value of the new activity. SHARD was developed because research had identified an analysis requirement in the safety process that was not well satisfied by currently available techniques, and therefore had to be "sold". This involved a lot of effort in arranging case studies, and persuading companies to perform their own evaluation of the technique. Although SHARD has now been provisionally recommended for inclusion in at least two companies' safety processes, it has not yet been applied "in anger" to a live industrial project – hence the qualified "(✓)" in Table 13.

LISA, on the other hand, was developed specifically to meet the needs of one enthusiastic and committed customer, BAe MA&A. This meant that, as soon as BAe MA&A and their customers were convinced that the technique was viable and could add to the safety evidence available on the project, LISA was applied on a live project. This has provided a rather more direct route to industrial acceptance, but only in one specific instance. LISA has not been included in the safety process for any other project, and needs to be subjected to further industrial trials before genuine industrial acceptance can be claimed – again, rating a qualified "(✓)" in Table 13.

It is worth noting that the evaluation contribution of peer review through publication has been ranked lower in terms of the maturity it indicates than industrial case studies. This is not to say that this is not a valuable activity – both SHARD and LISA have been presented at international

safety conferences, receiving generally positive feedback, and provoking interesting and useful discussions. However, as has been repeatedly emphasised, safety analysis is a very practical, "hands-on" discipline, and opinions formed on the basis of a short conference presentation or academic paper cannot be considered to have as much substance as those formed by participation in a large scale case study.

The general conclusion that is drawn from the evaluation activities performed is that both SHARD and LISA are effective, practical, "industrial strength" analysis techniques. SHARD has reached a stage of maturity at which it is reasonable to regard it as "finished"; LISA requires further industrial trials to validate the approach on a wider range of systems. The following sections provide rather more detailed discussion of some aspects of the evaluation of the two techniques.

9.3 Review of SHARD with respect to the analysis principles

As has been noted above, many of the principles for effective safety analysis presented in Chapter 5 were actually prompted by observations made on the early case studies and trials of SHARD. Not surprisingly, therefore, SHARD satisfied the principles very well. However, it is worth reviewing each principle briefly and commenting on its implications for SHARD.

Principle 1: Safety analysis must have value as part of the engineering process

This principle prompted the original attempt to incorporate design revision as an explicit step within the SHARD process, which was later shown to be unsatisfactory. Value as part of a process does not imply tight coupling; a more flexible approach is often more effective.

Principle 2: Method is more important than notation

Considerable effort has been expended in making the SHARD method clear, and in providing as much guidance as possible. That this has been successful has been demonstrated by two large case studies carried out by industrial safety personnel with almost no input from the author or other researchers.

Principle 3: Techniques should be as simple as possible

This principle was suggested by the simplification of the SHARD guide words necessitated by the problems encountered on case study 2. Although the method description is now quite long, most of the text is guidance notes, and the basic principles are still very simple.

Principle 4: Techniques should guide without unnecessarily constraining

In SHARD, this principle is very closely related to principle 3; the guide words were simplified because of the excessive constraint imposed by the original structure. Flexibility is also incorporated into the range of suggested working practices.

Principle 5: The role of the technique should be clear

SHARD is very explicitly a projective analysis for use as part of the design process. The method description makes it clear that the analysis should be owned by the design team.

Principle 6: Safety analysis starts at the system level

Again, the SHARD method explicitly follows this principle, stating that the analysis should start at the system context level.

Principle 7: Projective analyses are key to software safety

This principle was another that directly resulted from the SHARD case studies. As most of these case studies attempted to use the technique for retrospective analysis of completed systems, the frustration of finding problems too late to have a positive impact was obvious. Chris Harper's study referred to in section 7.7 made the financial importance of this principle extremely clear.

Principle 8: Safety analyses must consider hardware and software

SHARD is sufficiently general that it could be applied to any sort of system. However, the guidance notes on identification of causes of flow deviations (the primary – secondary – command rule) explicitly identify hardware faults as potential causes of software deviations.

Principle 9: Techniques should use familiar concepts and models

SHARD has been built around relatively minor modifications to a familiar safety analysis technique. Its effectiveness has been demonstrated on a range of commonly used dataflow and process network notations, although there is considerable interest in extending the range of design methodologies which the technique can support.

9.4 Relationship of SHARD to Def Stan 00-58 HAZOP

One of the most important aspects of SHARD is its close relationship to HAZOP, the primary differences between the techniques being the set of guide words selected, and the fact that SHARD does not mandate a full team to carry out every stage of the analysis. As was noted in Chapter 7, HAZOP is identified in Def Stan 00-56 [78] as a technique which may be mandated on safety critical systems procurement projects. Def Stan 00-58 [79] gives guidelines for the conduct of HAZOP studies on system containing programmable electronic systems (PESs). Since Def Stan 00-58 is guidance rather than a mandatory standard, the author's understanding is that SHARD may be used on MoD supply projects in place of HAZOP provided that justification is given, and that any project-imposed restrictions (e.g. on the use of an analysis team, or the independence of staff conducting the analysis) are respected.

As Chapter 7 explained, the development of SHARD was largely contemporaneous with the development of Def Stan 00-58, and results from SHARD case studies were used in preparing responses to the MoD's calls for comments on drafts of the standard. There is substantially less distinction between 00-58 HAZOP as now published and SHARD than there was in earlier drafts. A recent book by the authors of the standard [67] cites the work on SHARD as an example of alternative guide words which are acceptable within the terms of the standard. The primary benefits of SHARD over 00-58 HAZOP are seen as the more flexible approach to working practices, which is important in making the technique industrially acceptable, and the much more detailed guidance on technical issues such as interpretation of guide words and identification of the effects of fault propagation given in the SHARD method.

9.5 Review of LISA with respect to the analysis principles

Unlike SHARD, the development of LISA was guided from the start by the analysis principles. It does not satisfy them all fully, but the areas where it does not can be explained and justified. It will be interesting to see whether LISA actually grows to conform more closely to the principles as it is tried (and probably modified) on further projects with rather different needs and different system architectures.

Principle 1: Safety analysis must have value as part of the engineering process

LISA was developed to address a specific need of one particular project, which it satisfied successfully. It provides evidence that cannot be obtained with any other current technique, and can therefore be expected to be of value on future projects with similar needs.

Principle 2: Method is more important than notation

As with SHARD, LISA has no specific notation or recording format. The method is reasonably well defined but, as it has never been exposed to independent trials, it is too early to state whether there is adequate explanation and guidance. Given the degree to which the SHARD method was amplified and clarified once independent trials began, it is reasonable to expect that considerably more work is required on LISA.

Principle 3: Techniques should be as simple as possible

LISA addresses a problem that is fundamentally very complex. The guidance notes on the interpretation of guide words in particular explore some quite difficult ideas about how and why particular failures may occur, and cannot be claimed to be easy to read or understand. However, efforts were made to identify the simplest possible method which could provide the necessary information, and to base the technique on familiar models and concepts to make the results as accessible as possible. One or two of the reviewers involved in the case study experienced difficulty in following the arguments of acceptance presented, but these were all related to the complexity of the design under consideration, rather than to the analysis approach. Given this, LISA can probably be justifiably claimed to be as simple as possible for its intended role.

Principle 4: Techniques should guide without unnecessarily constraining

LISA'a compliance with this principle is impossible to assess without independent trials. The method description includes many quite detailed notes on interpretation, which were ideally suited to the main case study. Some care has been taken to express these as guidance rather than rules, but it seems likely that they will not all be applicable in new situations, and may need relaxing somewhat.

Principle 5: The role of the technique should be clear

LISA is explicitly a confirmatory analysis, intended to deliver evidence about the achieved safety of a completed design. In the case study, a number of safety requirements were derived by the technique, but these were all cases where the analysis had revealed a need for extra information or testing to complete safety arguments, and were not significant design changes.

Principle 6: Safety analysis starts at the system level

LISA quite explicitly does not comply with this principle, in that one of the aims was to investigate the provision of safety information when complete context information was not available. However, the method makes clear that the first step is to agree a set of principles to which the arguments of acceptability will have to conform. These principles could be regarded as providing a "virtual" system level context. Interestingly, the suggestion in the discussion of this principle that situations where complete information is unavailable might be managing by assuming worst case behaviour proved to be directly applicable in the case study, where it was used to avoid the need to analyse all the Class 2 software.

Principle 7: Projective analyses are key to software safety

LISA is a confirmatory analysis technique, and this principle does not apply.

Principle 8: Safety analyses must consider hardware and software

Since LISA was explicitly developed to investigate hardware and software interactions, it necessarily satisfies this principle.

Principle 9: Techniques should use familiar concepts and models

The analysis approach used in LISA combines elements of the familiar HAZOP guide word structured approach with some novel concepts in the study of resources rather than system states, and also in the explicit production of fragments of safety argument. However, it is applied to a system model familiar to programmers and, in practice, none of the case study participants or reviewers expressed any difficulty with the concepts.

Chapter 10
Conclusions

Chapter 9 has evaluated the contribution that the work described in this thesis has made to safety critical systems theory, and the practical value of the two new safety analysis techniques that have been developed. This chapter considers how the work described supports the thesis proposition, draws some final conclusions, and proposes some directions for future work.

10.1 Substantiation of the thesis proposition

In Chapter 1, the proposition of this thesis was stated as:

> It is possible to establish a set of principles for defining effective computer system safety analysis techniques. These principles enable sound techniques to be developed to satisfy novel analysis requirements.

This proposition was to be supported:

4. by demonstrating that the principles are based on concepts and methods underlying existing successful system safety analysis techniques;
5. by explaining observed weaknesses in existing computer system safety analysis processes and techniques; and
6. through the development and successful industrial application of two new computer system safety analysis techniques, addressing identified deficiencies in the range of existing analyses.

The principles that are central to this thesis are presented and discussed in Chapter 5, drawing on concepts established in Chapter 3 and Chapter 4. Chapter 6 extends the discussion of weaknesses in existing processes further as part of the justification for the new analysis techniques, which are described in Chapter 7 and Chapter 8.

The work has provided some evidence that the principles are generally sound, and that they can be applied to guide the effective development of new analysis techniques. In retrospect, the conclusion drawn in section 9.5 that principle 7 ("projective analyses are key to software safety") does not apply to the LISA analysis technique suggests that this is an observation (albeit an important one), rather than a general principle. It is also much more related to the definition of a good safety analysis process, rather than to the design of individual analysis techniques. If this

work is to be extended, this suggests two substantial new directions. The first is the identification or derivation of principles for complete safety processes, equivalent to those proposed here for individual techniques. The second is an investigation of whether the principles could be strengthened by a set of related observations, in a similar manner to the guidance notes attached to the steps of the safety analysis methods.

Pragmatically, the two new analysis techniques, SHARD and LISA, developed in Chapter 7 and Chapter 8 are likely to be of rather more practical benefit in the short to medium term than the theoretical work. Both techniques have now been developed to a state of industrial acceptability, and companies aware of the work have already made a number of requests for further practical investigations to support particular needs; some of these are discussed below.

Apart from the (probable) rejection of principle 7, the principles proposed appear to be both general, in that they have been shown to apply to two quite different techniques, and significant, highlighting issues of genuine importance in the design of effective safety analyses. The overall conclusion, then, is that the work presented here does provide substantiation for the thesis proposition.

10.2 Concluding remarks

In her article "High-Pressure Steam Engines and Computer Software" [49], Leveson uses the problems that beset the pioneers of high pressure steam engines to highlight issues in the current state of safety critical systems engineering. One of her main observations is that

> "there are two stages in the early years of a new technology: (1) exploration of the space of possible approaches and solutions to problems (i.e. invention), and (2) evaluation of what has been learned by this trial and error process to formulate hypotheses that can be scientifically and empirically tested in order to build the scientific foundation of the technology. Most of our emphasis so far has been in the first stage of invention; it is now time to give more time to the second."

It is hoped that this thesis has, at least in a small way, contributed to the foundations of a scientific discipline of safety engineering, as well as unashamedly adding some new approaches to specific problems of assessing computer system safety.

10.3 Future work areas

During the course of the research reported in this thesis, many possible directions for future work have been identified. The first few ideas discussed below are obvious or desirable extensions to the main work developed here. The remainder are more extensive subjects which would develop some of the concepts and principles in new directions.

10.3.1 Extension of SHARD to StateCharts

A number of companies have requested an investigation of the potential for using SHARD with StateCharts [33]. This was considered briefly early in the development of the technique, but at that time the guide word structure was still based around the identification of the communications protocols and data types of flows, which did not map well onto the states and transitions of StateCharts. However, it was noted that in the popular StateMate tool, the state charts are complemented by other notations, including the process network based activity model, with which SHARD is readily compatible.

Now that SHARD is more mature, this would be an interesting area to revisit. It seems unlikely that the concept of flow, which provides the structure of an analysis in SHARD, will be appropriate for state transition diagrams, and will probably need to be replaced. However, there are fairly obvious interpretations of most of the guide words, and much of the value of the SHARD development lies in the extensive guidance notes, many of which may also prove applicable.

10.3.2 Extension of SHARD to Object Oriented Design

Some time was spent at an early stage in the development of SHARD investigating its suitability for use with object oriented design notations. An attempt was made to interpret the guide words to produce a customised table similar to that shown for MASCOT in Table 7 - Example guide words for MASCOT 3, and an analysis of a small example design in HOOD [8] was attempted. The study was not a success; the biggest problem was that the relationships between objects represented in HOOD are not such a good match for the flow / service concepts of SHARD as the communication paths represented in dataflow and process network notations. Another significant problem in the example was that HOOD does not insist that all data flows are represented in the design drawings, which meant that the supporting documentation had to be searched to ensure completeness.

There is still a lot of interest in from industry in the use of HAZOP-like methods to analyse object oriented designs, and they are identified as suitable source material for HAZOP studies in Def Stan 00-58. Despite the negative results of the trial, re-investigating the application of SHARD to object oriented design notations is an obvious extension to this work.

Object-oriented design methods are increasingly being employed on safety critical systems projects, and a number of organisations have requested the investigation of ways to use HAZOP, SHARD and similar techniques with OOD. This was briefly investigated early in the development of SHARD, but this work was discontinued after unpromising early results. This is a subject that should be revisited now that SHARD is more mature. It seems likely that the outcome would be a combination of extensions or modifications to SHARD, and rules for making object oriented designs more amenable to hazard-directed analysis.

10.3.3 Integration of technical and human factors analysis in SHARD

It is apparent that the human / computer interface is a source of many of the problems with current safety critical systems. Despite the poor quality data, Mackenzie's study [56] clearly demonstrated that HCI failings were implicated in a huge majority of the accidents investigated. HCI-HAZOP was one of the three interlinked analyses described by Burns and Pitblado [12], and there are several other HAZOP-related human interface analysis techniques such as SUSI [16].

The integration of SHARD with these types of techniques is an obvious step. At the simplest level, it can be seen that human errors represent one source of incorrect input to the computer system, and can easily be incorporated within the standard SHARD approach. Similarly, faulty system outputs may result in erroneous displays, causing incorrect actions by human operators. However, for maximum effect, the analyses need to be much more integrated than this simplistic linking of inputs and outputs. Human(s) and computers must be considered as parts of a single system, and the analysis must model and study the feedback loops that involve both. This is a more complex problem, involving consideration of factors such as mode confusion, and the potential for users to misunderstand how the computer system operates.

10.3.4 Further trials and extensions of the LISA approach

Although LISA has been successfully demonstrated on a very large case study, more trials are needed to prove the wider applicability of the technique. A range of single and multi-processor systems, with different hardware and software architectures and scheduling models are needed.

Since the case study described in Chapter 8 involved a system with only two classes of software to be partitioned, and with a fixed memory map, trials are particularly needed on systems with more independent processes, and with dynamic memory management.

Interest has also been expressed in extending the purely qualitative approach taken in the current definition of LISA to include guidelines for quantitative arguments, for example in the case of some of the arguments for the safety of intrinsically critical resource usage described in section 8.2.3.

Another obvious development of LISA would be the implementation of tool support for the method. Ideally, this should link to system specification or design tools so that memory maps and other information could be imported or, preferably, linked such that analysis affected by design changes could be automatically identified.

10.3.5 Further validation of the principles

The next step beyond extension of the analysis techniques is to identify further work that could help to validate the principles proposed in Chapter 5. This will almost certainly involve a combination of development of more new techniques and wider peer review. Sneak Analysis is a very appealing starting point for this work, since it is a very powerful technique, considering extensive classes of conditions that are not addressed by HAZOP or Functional Failure Analysis; however, it is also significantly more complex than either, having multiple levels of prompt. The discussion in section 2.11 has noted that sneak analysis for software has already been proposed, and it would be very interesting to try to find ways of extending this into an integrated hardware / software sneak analysis.

10.3.6 Design for analysis and safety argument construction

During the work on LISA, many features of the case study system were identified as being beneficial either for the analysis itself, or for their contribution to the arguments of acceptability. The analysis also highlighted a number of areas where the system design made the safety analysis unnecessarily complex. During the process of design revision, many of the recommendations arising from the analysis work were for ways of making the system simpler to analyse, rather than significant functional changes. No systematic attempt was made to either record the specific features which facilitated or hindered the analysis, or to consider whether they could be extended to identify general principles for analysis-friendly design. This is a potentially extremely

extensive research area, and can be seen as a complement to already relatively well established rules for design for testability.

10.3.7 Formal specification of safety properties

To date, the majority of formal methods work has concentrated on the specification or proof of correct software behaviour. Provided that expected "input" failures, perhaps including a limited range of hardware failures, can be described, there is no reason why proofs cannot be constructed to show that specific outputs of a component behave as required even in the presence of these failures. It is also theoretically possible (though likely to be rather more difficult) to prove a range of general safety-related properties, such as demonstrating that no single input failure can cause any output failure. The problem with the use of formal techniques in this was is not lack of capability of the formal methods, but rather the inability of safety analysts to determine and specify the properties that must be demonstrated, and this will need to be the initial focus of work in this area.

References

[1] *Air Accidents Investigation Branch UK*, http://www.gtnet.gov.uk/aaib/aaibhome.htm, 1999.

[2] Audsley N.C., *et al.*, *Fixed Priority Preemptive Scheduling: An Historical Perspective.* Journal of Real-Time Systems, vol. 8 no. 2, 1995, pp. 173-198.

[3] Ballard R.D., *The Discovery of the Titanic*, 1998. Madison Press Books, Toronto. ISBN 0-75380-529-4.

[4] Belbin R.M., *Team Roles at Work*, 1993. Butterworth-Heinemann. ISBN 0750626755.

[5] Bell R. and Reinert D., *Risk and system integrity concepts for safety-related control systems.* Microprocessors and Microsystems, vol. 17 no. 1, 1993, pp. 3-15.

[6] Boeing Commercial Airplane Group, *Statistical Summary of Commercial Jet Airplane Accidents: Worldwide Operations 1959-1996*, 1997. Airplane Safety Engineering, Boeing Commercial Airplane Group, P.O. Box 3707, Seattle, WA 98124-2207.

[7] Bondavalli A. and Simoncini L., *Failure Classification With Respect to Detection*, in *First Year Report Task B: Specification and Design for Dependability.* 1990, ESPRIT BRA Project 3092: Predictably Dependable Computing Systems.

[8] Booch G., *Object-oriented Development.* IEEE Transactions on Software Engineering, vol. SE-12 no. 2, 1986, pp. 211-221.

[9] Bowman W.C., *et al.*, *An Application of Fault Tree Analysis to Safety Critical Software at Ontario Hydro* in *Probabilistic Safety Assessment and Management: Proceedings of the International Conference on Probabilistic Safety Assessment and Management (PSAM)*, Beverley Hills, CA, 1991. Ed. Apostolakis G., Elsevier Science Publising Co., Inc. ISBN 0444015949.

[10] British Standards Institution, *BS 4778: Quality Vocabulary Part 2: Quality Concepts and Related Definitions*, 1991. British Standards Institution. ISBN 0-580-19694-1.

[11] Burdick G.R. and Fussell J.B., *On the Adaption of Cause-Consequence Analysis to U.S. Nuclear Power Systems Reliability and Risk Assessment*, in *A Collection of Methods for Reliability and Safety Engineering Report V, ANCR, 1273.* 1976, Idaho National Engineering Laboratory.

[12] Burns D.J. and Pitblado R.M., *A Modified HAZOP Methodology for Safety Critical System Assessment* in *Directions in Safety-critical Systems: Proceedings of the Safety-critical Systems Symposium*, Bristol, 1993. Ed. Redmill F. and Anderson T., pp. 232-245. Springer-Verlag.

[13] Carpenter R., Kalla-Bishop P., Munson K. and Wyatt R., *Powered Vehicles*, 1974. Jupiter Books, London. ISBN 0-904041-06-9.

[14] Cha S.S., *AeSOP: An Interactive Failure Mode Analysis Tool* in *COMPASS '94: Proceedings of the Ninth Annual Conference on Computer Assurance*, NIST Gaithersburg MD, 1994. pp. 9-16. IEEE, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 0855-1331. ISBN 0-7803-1855-2.

[15] Chudleigh M., *Hazard analysis using HAZOP: A Case Study* in *SAFECOMP '93: Proceedings of the 12th International Conference on Computer Safety Reliability and Security*, Poznan-Kiekrz, Poland, 1993. Ed. Gorski J., pp. 99-108. Springer-Verlag. ISBN 3540198385.

[16] Chudleigh M. and Clare J.N., *The benefits of SUSI: Safety Analysis of User System Interaction* in *SAFECOMP '93: Proceedings of the 12th International Conference on Computer Safety Reliability and Security*, Poznan-Kiekrz, Poland, 1993. Ed. Gorski J., pp. 123-132. Springer-Verlag. ISBN 3540198385.

[17] CISHEC, *A Guide to Hazard and Operability Studies*, 1977. The Chemical Industry Safety and Health Council of the Chemical Industries Association Ltd.

[18] Clarke S.J. and McDermid J.A., *Software Fault Trees and Weakest Preconditions: A Comparison and Analysis.* Software Engineering Journal, vol. 8 no. 4, 1993, pp. 225-236.

[19] Cristian F., *Questions to ask when designing or attempting to understand a fault tolerant distributed system.* 3rd Brazilian Conference on Fault Tolerant Computing, Rio de Janeiro, 1989, .

[20] Cullen, The Hon.Lord, *The Public Inquiry into the Piper Alpha Disaster*, 1990. HMSO, London. ISBN 0-10-113102.

[21] Dawkins S., McDermid J.A., Murdoch J. and Pumfrey D.J., *Issues in the Conduct of PSSA* in *Proceedings of the 17th International System Safety Conference*, Orlando, Florida, 1999. pp. 77-86. System Safety Society, P.O. Box 70, Unionville, VA 22567-0070.

[22] Dhillon B.S., *Failure Modes and Effects Analysis - Bibliography.* Microelectronics and Reliability, vol. 32 no. 5, 1992, pp. 719-731.

[23] Dugan J.B., Bavuso S.J. and Boyd M.A., *Fault Trees and Sequence Dependencies* in *Proceedings of the Annual Reliability and Maintainability Symposium*, 1990. pp. 286-293.

[24] Dugan J.B., Bavuso S.J. and Boyd M.A., *Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems.* IEEE Transactions on Reliability, vol. 41 no. 3, 1992, pp. 363-376.

[25] Earthy J.V., *Hazard and Operability Studies as an approach to Software Safety Assessment* in *IEE Computing and Control Division Colloquium on Hazard Analysis Digest No 1992/198*, 1992. pp. 5/1 - 5/3. Institution of Electrical Engineers, Savoy Place, London WC2R 0BL.

[26] Ezhilchelvan P.D. and Shrivastava S.K., *A Classification of Faults in Systems*, 1989. University of Newcastle upon Tyne.

[27] Fencott C. and Hebbron B.D., *The Application of HAZOP Studies to Integrated Requirements Models for Control Systems* in *SAFECOMP '94: Proceedings of the 13th International Conference on Computer Safety, Reliability and Security*, Anaheim, CA, 1994. Ed. Maggioli V., pp. 83-92. Instrument Society of America. ISBN 1-55617-536-1.

[28] Fenelon P. and McDermid J.A., *Integrated Techniques for Software Safety Analysis* in *IEE Computing and Control Division Colloquium on Hazard Analysis Digest No 1992/198*, 1992. Institution of Electrical Engineers, Savoy Place, London WC2R 0BL.

[29] Fenelon P. and McDermid J.A., *An Integrated Toolset For Software Safety Analysis.* Journal of Systems and Software, 1993, pp. 2/1-2/16.

[30] Feynman R.P., *Appendix F: Personal Observations on the Reliability of the Shuttle*, in *Report of the Presidential Commission on the Space Shuttle Challenger Accident.* 1986, National Aeronautics and Space Administration.

[31] Forder J., Higgins C., McDermid J.A. and Storrs G., *SAM - A Tool to support the Construction Review and Evolution of Safety Arguments* in *Directions in Safety-critical Systems: Proceedings of the Safety-critical Systems Symposium*, Bristol, 1993. Ed. Redmill F. and Anderson T., pp. 195-216. Springer-Verlag.

[32] Geake E., *Did ambulance chiefs specify safety software?* New Scientist, vol. 136 no. 1846, 1992, p. 5.

[33] Harel D., *StateCharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, vol. 8 no. 3, 1987, pp. 231-276.

[34] Health and Safety Executive, *The Explosion and Fires at the Texaco Refinery, Milford Haven, 24 July 1994*, 1997. HSE Books. ISBN 0-7176-1413-1.

[35] *Archives of the hise_safety_critical mailing list*, ftp://ftp.cs.york.ac.uk/hise_reports/sc.list/archive99.txt, 1999.

[36] Horwell V., *Secrets of the Dead*, Channel 4 Booklets, ed. Snyder P., 1999. Channel 4 Television, London.

[37] International Electrotechnical Commission, *Guide for Hazard and Operability Studies (HAZOP)*, 1997.

[38] International Electrotechnical Commission, *IEC 61508: Fundamental Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems*, 1999.

[39] JIMCOM, *The Official Handbook of Mascot Version 3.1*, 1987. Joint IECCA and MUF Committee on Mascot.

[40] Jones C.B., *Developing Methods for Computer Programs including a Notion of Interference*, (DPhil Thesis), Oxford University, 1981.

[41] Jones-Lee M.W., *Vauling Safety in Transport Project Appraisal* in *Environment and Transport*, Tinbergen Institute, Amsterdam, 1994.

[42] Joseph M., *Real-Time Systems: Specification, Verification and Analysis*, 1996. Prentice-Hall.

[43] Kelly T.P., *Arguing Safety - A Sytematic Approach to Managing Safety Cases*, (DPhil Thesis), University of York, 1999.

[44] Kletz T., *Hazop and Hazan: Identifying and Assessing Process Industry Hazards,* Third ed., 1992. Institution of Chemical Engineers. ISBN 0-85295-285-6.

[45] Koivisto R. and Heino P., *Qualitative and Quantitative Safety Assessment Supporting the Process Plants' Design* in *Probabilistic Safety Assessment and Management: Proceedings of the International Conference on Probabilistic Safety Assessment and Management (PSAM)*, Beverley Hills, CA, 1991. Ed. Apostolakis G., Elsevier Science Publishing Co. Inc. ISBN 0444015949.

[46] Kopetz H., *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 1997. Kluwer Academic, Boston. ISBN 0792398947.

[47] Laprie J.C., *Dependability: Basic Concepts and Terminology.* Dependable Computing and Fault Tolerance, vol. 5 1992, .

[48] Leveson N.G., *Software Safety: Why What and How.* ACM Computing Surveys, vol. 18 no. 2, 1986, pp. 125-163.

[49] Leveson N.G., *High-Pressure Steam Engines and Computer Software.* Computer, vol. 27 no. 10, 1994, pp. 65-73.

[50] Leveson N.G., *Safeware: System Safety and Computers*, 1995. Addison Wesley, Reading, Mass. ISBN 0-201-11972-2.

[51] Leveson N.G. and Harvey P.R., *Analyzing Software Safety.* IEEE Transactions on Software Engineering, vol. SE-9 no. 5, 1983, pp. 569-579.

[52] Leveson N.G. and Harvey P.R., *Software Fault Tree Analysis.* Journal of Systems and Software, vol. 3 1983, pp. 173-181.

[53] Leveson N.G. and Shimeall T.J., *Safety Verification of Ada Programs using Software Fault Trees.* IEEE Software, vol. 8 no. 4, 1991, pp. 48-59.

[54] Leveson N.G. and Stolzy J.L., *Safety Analysis Using Petri Nets.* IEEE Transactions on Software Engineering, vol. SE-13 no. 3, 1987, pp. 386-397.

[55] *Ariane 5 Flight 501 Failure: Report by the Inquiry Board*, http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html, 1996.

[56] MacKenzie D., *Computer-Related Accidental Death: An Empirical Exploration.* Science and Public Policy, vol. 21 no. 4, 1994, pp. 233-248.

[57] McDermid J.A., Nicholson M., Pumfrey D.J. and Fenelon P., *Experience with the Application of HAZOP to Computer-Based Systems.* COMPASS '95: Proceedings of the Tenth Annual Conference on Computer Assurance, 1995, pp. 37-48.

[58] McDermid J.A. and Pumfrey D.J., *A Development of Hazard Analysis to aid Software Design* in *COMPASS '94: Proceedings of the Ninth Annual Conference on Computer Assurance*, NIST Gaithersburg MD, 1994. pp. 17-25. IEEE, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 0855-1331.

[59] McDermid J.A. and Rook P., *Software Development Process Models*, in *Software Engineer's Reference Book*, McDermid J.A., Editor. 1991, Butterworth-Heinemann: Oxford.

[60] Mersey, Rt.Hon.Lord, *Report on the loss of the S.S."Titanic"*, 1912. Alan Sutton Publishing, Stroud, Gloucestershire. ISBN 0-86299-723-2.

[61] Nielsen D.S., Platz O. and Runge B., *A Cause-Consequence Chart of a Redundant Protection System.* IEEE Transactions on Reliability, vol. R-24 no. 1, 1975, pp. 8-13.

[62] Nordland O., *A Discussion of Risk Tolerance Principles.* Safety Systems, vol. 8 no. 3, 1999, pp. 1-4.

[63] O'Connor P.D.T., *Practical Reliability Engineering,* Third ed., 1991. John Wiley & Sons. ISBN 0-471-92902-6.

[64] Prasad D.K., *Dependable Systems Integration using Measurement Theory and Decision Analysis*, (DPhil Thesis), University of York, 1998.

[65] Raheja D., *Software System Failure Mode and Effects Analysis (SSFMEA) - A Tool for Reliability Growth* in *Proceedings of the International Symposium on Reliability and Maintainability*, Tokyo, 1990. pp. IX-1 - IX-7.

[66] Rankin J.P., *Sneak Circuit Analysis.* Nuclear Safety, vol. 14 no. 5, 1973, .

[67] Redmill F., Chudleigh M. and Catmur J., *System Safety: HAZOP and Software HAZOP*, 1999. John Wiley & Sons Ltd., Chichester. ISBN 0-471-98280-6.

[68] Reifer D.J., *Software Failure Modes and Effects Analysis.* IEEE Transactions on Reliability, vol. 28 no. 3, 1979, pp. 247-249.

[69] Roberts N.H., Vesely W.E., Haasl D.F. and Goldberg F.F., *Fault Tree Handbook*, 1981. Systems and Reliability Research Office of U.S. Nuclear Regulatory Commission, Washington, DC, 20555.

[70] SAE, *ARP 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, 1996. Society of Automotive Engineers, Inc, Warrendale, PA.

[71] SAE, *ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 1996. Society of Automotive Engineers, Inc, Warrendale, PA.

[72] Shrivastava P., *Bhopal: Anatomy of a Crisis,* Second ed., Business and the Environment, 1992. Paul Chapman, London. ISBN 1853961922.

[73] Stephans R.A. and Talso W.W., *System Safety Analysis Handbook,* Second ed., 1997. System Safety Society, Albuquerque, New Mexico.

[74] Systems Designers, *CORE - The Method*, 1985. Systems Designers Scientific.

[75] Tambidurai P., *Interactive Consistency with Multiple Failure Modes* in *International Workshop on Hardware Fault Tolerance in Multiprocessors*, Urbana Il, 1989.

[76] Taylor J.R., *Risk Analysis for Process Plant, Pipelines and Transport*, 1994. Chapman and Hall, London. ISBN 0-419-19090-2.

[77] UK Ministry of Defence, *Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment*, 1996.

[78] UK Ministry of Defence, *Defence Standard 00-56 Issue 2: Safety Management Requirements for Defence Systems*, 1996.

[79] UK Ministry of Defence, *Defence Standard 00-58: HAZOP Studies on Systems Containing Programmable Electronics*, 1996.

[80] Villemeur A., *Reliability Availability Maintainability and Safety Assessment*, 1992. John Wiley and Sons Ltd. ISBN 0-471-93048-2.

[81] Ward P.T. and Mellor S.J., *Structured Development for Real Time Systems*, Yourdon Press Computing Series, 1985. Prentice-Hall. ISBN 0138546541.

[82] Yourdon E., *Modern Structured Analysis*, Yourdon Press Computing Series, 1989. Prentice-Hall, Englewood Cliffs. ISBN 013598632X.