The Integration of Hazard Evaluation Procedures and Requirements Engineering for Safety-Critical Embedded Systems

by

Eamon J. Broomfield

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy of Loughborough University

1997

9/9001085

Abstract

Although much work has been done on assessing safety requirements in programmable systems, one very important aspect, the integration of hazard evaluation procedures and requirements engineering, has been somewhat neglected. This thesis describes the derivation and application of a methodology, *HAZAPS* (HAZard Assessment in Programmable Systems). The methodology assists at the requirements stage in the development of safety-critical embedded systems. The objectives are to identify hazards in programmable systems, construct and model the associated safety requirements, and, finally, to assess these requirements. *HAZAPS* integrates safety engineering and software modelling techniques. The analysis of more than 300 computer related incidents provided the criteria used to identify, select and modify safety engineering techniques.

There are four stages in the *HAZAPS* process: a) safety-critical subsystems are identified using domain analysis; b) Fault Tree Analysis (FTA) is used to identify the contributors to the hazards and to specify safety requirements; c) task synthesis and a new modelling technique (the Event Time Diagram or ETD) are then used to understand and analyse operational tasks associated with the safety requirements; and, d) the principles of HAZard and Operability Studies (HAZOP) and Failure Mode and Effects Analysis (FMEA) are used to assess the safety requirements. A *HAZAPS* tool has been developed for the Windows environment using an expert system shell. The *HAZAPS* tool can be used to record all information generated during the *HAZAPS* process.

This thesis makes a novel contribution to the field of re-use of incident data to permit feedback into the design of safety-critical software. It represents a novel synthesis of hazard evaluation procedures with requirements engineering. The methodology conceived and developed in this study has been tested on industrial systems and proved to be effective.

This thesis is dedicated to my wife, Siobain, who gave me continuous support
and encouragement without which this thesis would
never have been written.

Acknowledgements

I am grateful to the DTI/EPSRC who funded this work as part of the Safety Critical Systems Research Programme initiative.

Thanks also to Dr. Paul Chung and other members of the Safe Plant Design Group at Loughborough University. Special thanks to Professor F.P. Lees for his rare but invaluable comments.

I am indebted to the different companies that provided raw incident data and design details, and tested the methodology.

Glossary of Terms

AIChemE	American Institute of Chemical Engineers
DAF	Dissolved Air Flotation
DFDs	Data Flow Diagrams
ESA	European Space Agency
ES-FMEA	Embedded-System-FMEA
ES-HAZOP	Embedded-System-HAZOP
ETD	Event Time Diagram
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes & Effects Criticality Analysis
FMES	Failure Modes and Effects Summary
FPTN	Failure Propagation and Transformation Notation
FTA	Fault Tree Analysis
GAC	Granular Activated Carbon
HAZAPS	HAZard Assessment in Programmable Systems
HAZOP	HAZard and OPerability Study
HFTA	Highlevel Fault Tree Analysis
HID	Human Input Device
HRA	Human Reliability Analysis
HSE	Health & Safety Executive, UK
IEC	International Electrotechnical Commission
INPO	Institute of Nuclear Power Operations
MIL-STD	Military Standard, USA
MoD	Ministry of Defense, UK
MORT	Management Oversight & Risk Tree
O&SHA	Operating and Support Hazard Analysis
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OOP	Object Oriented Programming
PAISLey	Process-oriented Applicative and Interpretable Specification Language
PHA	Preliminary Hazard Analysis
REVs	Requirements Engineering Validation System
RGS	Rapid Gravity Sand filters
RLP	Requirements Language Processor
RSL	Requirements Statement Language
SADT	Structured Analysis Design Technique
SFTA	Software Fault Tree Analysis
SHA	System Hazard Analysis
SRS	Software Requirements Specification
SSHA	Subsystem Hazard Analysis

Table of Contents

CHAPTER 1 : Introduction

Despite the widespread use of 'embedded systems' (real time control systems used for automated production lines, weapon systems, antilock braking systems, medical systems, etc.), the most efficient and effective way of designing, building, testing and maintaining safety-critical systems is by no means obvious. Our understanding of the methods and procedures used in the development of safety-critical systems is rapidly improving through research [Redmill, 1993; Malcolm, 1994] and the establishment of standards and guidelines [MoD 00-55, 1991; MoD 00-56, 1993; MoD 00-58, 1995; IEC Standard, 1995; MIL-STD-882C, 1993; RTCA DO-178B, 1992]. However, many of the methods proposed for requirements capture and specification have their origins in traditional software engineering rather than the safety aspects of systems engineering; and the techniques proposed for hazard evaluation are not proven for software systems. There is also a false perception that, if enough effort and money are used in the building of a safety-critical system, the system will be safe. Yih et al. [1995] discuss three nuclear power plant projects (the P20 project of Chooz B, France; the shutdown system of the Darlington plant, Canada; and the primary protection system of Sizewell B, UK) all of which had high verification costs and yet still faced doubts about their potential safety performance.

1.1 System safety and software

The safety of a system can be defined as the likelihood that the system will not lead to a state in which human life or the environment is endangered [IEE Report, 1992]. The subjective nature of this definition introduces the first problem when building safety-critical systems, that is, how to recognise that the system has safety implications in the first place. For example, in the case of the London Ambulance incident [Arthur, 1992], the safety implications were not recognized (or, if they were, the project managers did not know how to deal with them). On the other hand, recognizing a system as 'safety-critical' is no guarantee that a safe system will be built, as is illustrated by the two cases [Borning, 1987] where well-established, well-understood ballistic missile early warning systems alerted operators to imaginary attacks. IEE [1989] emphasises the subjective nature of safety

Safety is a concept which is not fixed in either time or space. Attitudes to safety, and tolerance of danger, are subjective and variable over time and circumstance. Industries have attempted to address this problem by identification of more objective aspects of safety, with particular definitions of words such as 'hazard' and 'situation'.

When assessing the safety of a system, it is usual to use the word 'hazard' because this allows us to focus on situations in which there is an actual or potential danger. A 'hazard' is defined as

... a physical situation with a potential for human injury, damage to property, damage to the environment or some combination of these [Jones, 1992]

Although a computer does not in itself contain hazards, it controls (or fails to control) equipment that can contribute to a hazardous situation. This stresses the significance of the software system interface. The internal behaviour of the software (i.e. internal states and state transitions) results in changes in the external behaviour of the system via the interface and this may result in a hazardous situation. Safety-critical software is defined in MoD 00-55 standard as *software used to implement a safety-critical function* i.e. as a property of the *use* of the software, *not* as an intrinsic property of the software [Ravn et al., 1990]. Safety is a property of the total system, not the sum of the safety properties of the individual parts.

1.2 Problems and proposed solutions

The Ariane V rocket was exploded after take-off in June 1996 because there was a major software design fault introduced during the conception of the system and the software had not been tested in relevant conditions [Abbott, 1996; Amelan, 1996]. The director of the European Space Agency stated

There have been errors of conception, of specification and of verification in testing [Irwin, 1996].

Problems associated with programmable safety-critical embedded systems can be described in terms of the three areas mentioned in the above quotation (i.e. conception, specification and testing). The following three sections outline these problem areas and describe the approach used in this thesis to solve these problems.

1.2.1 Conception

Problems arise at the conception of a system because the expertise required is often partitioned into different disciplines (e.g. pneumatics, electrical engineering, mechanical engineering, medical instrumentation, software engineering etc.). It is difficult to integrate the different views and there is a tendency to view the total system from one perspective only.

In this thesis a novel methodology is proposed which will unify disparate expertise into models, methods and procedures that can relate the needs of **all** users in an integrated fashion. The methodology is based on the generification of tasks and a new graphical technique. This methodology permits special consideration to be given to interfaces at which the different disciplines interact because this is where problems are likely to occur.

1.2.2 Specification

A major difficulty in developing safety-critical systems is in determining how to specify safety requirements. The safety requirements must express constraints on both the target system and the embedded system in a coherent manner.

It is desirable to interrelate hazards in the environment to programmable states in the embedded system. There are several approaches to identifying hazards in embedded systems, for example (a) starting with a top level hazard and using a deductive technique to identify programmable events that are related to this top level hazard, or (b) starting with a programmable event and using an inductive technique to identify top level hazards associated with this programmable event. Neither (a) nor (b) is feasible because the search would become intractable.

In this thesis a new system model is proposed which will use a combination of inductive and deductive techniques and integrates hazard evaluation procedures and requirement engineering techniques. Existing safety and software engineering techniques will be customised so that results can be logically and systematically passed from one technique to another.

1.2.3 Testing

Testing of safety-critical systems presents an even greater challenge than testing of non-safety-critical systems because it is not sufficient to test only software and hardware but the environment and human interactions with the system must also be considered. Traditional testing techniques used for testing programmable systems are not adequate for safety-critical systems. For a safety-critical system one must identify what should not happen as well as what should happen.

In this thesis patterns will be abstracted from incidents and a generic framework constructed. This framework will subsequently be embedded in a methodology that can be used by developers when building new systems. The framework will be accessed by adopting an operational approach which will consider the software, hardware and human interaction associated with the system.

1.3 Structure of the thesis

The structure of the thesis is as follows:

Chapter 2: *Hazard evaluation procedures and requirements engineering.* This chapter is related to sections 1.2.1. and 1.2.2 above. The purpose of this chapter is to review how people identify and specify requirements for programmable systems. Different strategies for developing safety-critical systems are described. Various requirements and software engineering techniques are investigated to analyse their suitability for safety-critical systems. Ways of integrating hazard evaluation procedures and software engineering techniques are compared and contrasted with the objective of identifying the limitations of current approaches.

Chapter 3: *Using incident analysis to construct a methodology.* This chapter is related to section 1.2.3 above. The purpose of this chapter is to demonstrate how the knowledge gained from analysing incidents can be reused when developing and testing safety-critical programmable systems. The benefits of incident analysis are described. The derivation of a model for analysing the sequence of events associated with an incident is demonstrated and discussed. It is shown how analysis of incidents can be used to construct a generic framework which can later be incorporated into a methodology for assessing the safety of programmable systems.

Chapter 4: *The HAZAPS methodology.* This chapter is related to sections 1.2.1, 1.2.2 and 1.2.3. The purpose of this chapter is to define the underlying principles of the HAZAPS methodology. It describes the development and application of the models and methods used in the methodology, including the derivation of a system model used to interrelate safety engineering and software engineering techniques. It outlines the procedures used for each stage of the HAZAPS methodology.

Chapter 5: *The HAZAPS tool.* The tool developed to support the HAZAPS methodology is discussed. The requirements, design, implementation and operation of the HAZAPS tool are described.

Chapter 6: *Applications of HAZAPS.* The purpose of this chapter is to evaluate the HAZAPS methodology using an illustrative example and two case studies; one carried out on a rotary screen line printing press and the other on a water treatment plant. The objectives of applying HAZAPS were to a) show typical results, b) demonstrate the benefits, c) identify any major difficulties, and d) show how HAZAPS can be applied to different application domains. The results of the assessments are discussed.

Chapter 7: *Conclusions and further work.* This chapter describes what has been achieved. It discusses the novelty and limitations of the methodology and also shows how the methodology can be refined and extended.

CHAPTER 2 : Hazard Evaluation Procedures and Requirements Engineering

The addition of software to a system increases the complexity of the system and makes it more difficult to evaluate the safety of the system. Table 2.1 details the advantages and disadvantages of using programmable systems. To benefit from the advantages of using software in safety systems, strategies must be developed to overcome the problems outlined in Table 2.1 and thus ensure a safe system.

2.1 Strategies for building safety-critical programmable systems

The literature reveals a number of different approaches to building safety-critical systems; authors tend to select an approach which best relates to their particular specialist knowledge/expertise. These can be classified under the headings *fault avoidance, fault forecasting, fault removal* and *fault tolerance*.

2.1.1 Fault Avoidance

Fault Avoidance has been defined as:

> *The use of design techniques and implementation methods which aim to prevent, by CONSTRUCTION, fault occurrence or introduction.[Schoitsch et al. 1990]*

Techniques that can be used for fault avoidance include structured methods, prototyping, formal methods, quality control measures and use of computer tools. De Panfilis [1991]

Table 2.1: Advantages and disadvantages of using programmable systems

Advantages [Parnas et al, 1990; Barlow & Smith, 1990; Prosser, 1993]	**Disadvantages** [Jones, 1991; Parnas et al, 1990; Smith & Wood, 1989; Yount, 1985]
Flexibility	Requirements often poorly thought out and planned
Fewer devices/functions, thus saving on space, weight and power	No satisfactory procedures yet established for validation of software
Improved performance	Increases uncertainty and makes it difficult to predict failure modes
Easier to modify and reconfigure in the field	Trivial errors can have major consequences
Lower costs	Makes it difficult to impose standard approach to design
Allows self-test and early warning diagnostics	Difficult to control software changes
Provides more information to operators	Exhaustive testing almost impossible, since number of possible paths through the software is usually very high
	Susceptible to common mode failures whereby a single failure defeats redundancy
	Data and programs can be corrupted by interference

proposes prototyping as a fault avoidance technique. The advantages of prototyping are that it supports a better understanding of user requirements and allows refinement of high level requirements in an iterative and incremental manner, thus minimising design errors. Bowen & Stavridou [1993] suggest there is an enormous gap between what can be achieved by fault tolerance and fault removal and what is required to build ultra-high dependability systems. They suggest that fault avoidance using formal methods in conjunction with other techniques may help close this gap. Although there are many procedures and methods that can be used for fault avoidance, in practice, no technique can guarantee avoidance of all errors. Nevertheless, facilities can be built into these techniques to make it easier to forecast, tolerate and remove faults at a later stage.

2.1.2 Fault Forecasting

Fault forecasting has been defined as:

> *The use of techniques to estimate, by EVALUATION, the presence, creation and consequences of faults.* [Schoitsch et al., 1990].

Fault avoidance, fault removal and fault tolerance are often discussed, whereas fault forecasting tends to be overlooked. This is partly because fault forecasting is usually classified with fault avoidance. Fault forecasting techniques include FTA, HAZOP and FMEA. The advantage of fault forecasting is that safety can be considered explicitly and consequences that must be avoided can be investigated without the need to focus on all requirements [Rushby, 1994]. In other words, fault forecasting emphasises safety analysis and hazard control rather than focusing on correctness [Leveson, 1991]. Integrating fault avoidance and fault forecasting techniques provides a very powerful combination to aid in the development of safety-critical systems.

2.1.3 Fault Tolerance

Fault Tolerance can be described as:

> The built-in capability of a system to provide continued correct services by REDUNDANCY despite of a limited number of faults (hardware or software) [Schoitsch et al., 1990; Bowen & Stavridou, 1993].

Yount [1985] describes several techniques that have been used to implement fault tolerance: basic (similar) redundancy, software fault tolerance techniques, dissimilar processor hardware, and dissimilar backup systems. He believes that it is better to use a fault-tolerance, rather than a fault avoidance approach to safety-critical systems development. He also maintains that fault-tolerant architectures are more cost effective than fault avoidance architectures because of the increasing cost of verification and validation in the latter. Others [Laprie, 1993; Prorok et al., 1992] claim that fault tolerance techniques can increase the complexity of a system and thus decrease dependability. Bowen & Stavridou [1993] maintain that the (supposed) benefits of using a fault-tolerance approach are contentious, and that the overall gain in using this approach may not be significant. In practice, both fault avoidance and fault tolerance approaches are used when building safety-critical systems.

2.1.4 Fault Removal

Fault Removal has been defined as:

> *The use of techniques to minimize the presence of faults by VERIFICATION throughout the development phases* [Schoitsch et al., 1990].

Prorok et al. [1992] maintain that we should not depend on fault avoidance or fault tolerance techniques to ensure a safe system, but that safety-critical software should be rigorously tested before release. They emphasize the importance of a testing framework which allows progressive testing throughout the development of the product. Schoitsch *et al.* [1990] describe fault removal techniques including testing on several system levels as part of the overall design strategy. The testing procedures are white box testing (module testing), blackbox testing (functional testing) and safety tests (functional tests with respect to safety). Test cases are derived from the user requirements specification and by simulation.

2.1.5 Selecting a strategy

At present, there is no way of ensuring a fault-free programmable system, nevertheless, by careful application of relevant techniques, the overall risk can be reduced. The best strategy seems to lie in selecting a combination of techniques and trying to ensure that the disadvantages of one technique are overcome by the use of another technique. Before selecting a combination of techniques, therefore, it is essential to identify the strengths and weaknesses of each technique and to establish at which stage in the lifecycle the technique can be most effectively applied. Rushby [1994] claims that, since faults in the requirements specification are the primary source of catastrophic failures in critical systems, in order to make a major advance in the development of safety-critical systems, we should focus on techniques that can be used early in the lifecycle. This is not surprising as the percentage of software errors reported in the requirements phase varies from 44% to 66%; it is also worth noting that estimated costs for correcting such errors at a later stage vary from 200 to 1000 times the original cost (Bowen & Stavridou, 1993; Reifer, 1979; HSE, 1995; Jaffe et al., 1991). It is important to adopt a system-wide multidisciplinary approach to cope with the continual increase in complexity of embedded systems. If, for example, the original specification is incomplete or incorrect, then, even the use of formal proofs and/or fault tolerant techniques based on that specification will not ensure the delivery of a safe system. Improved requirements elicitation methods and specification techniques (fault avoidance approach) should be integrated with traditional safety engineering techniques (fault forecasting approach) to minimise the risk of errors being introduced later in the product lifecycle. The next section examines requirements engineering techniques and their applicability to safety, and the following section discusses hazard evaluation procedures and their integration into software development.

2.2 Requirements engineering

There are a number of requirements engineering techniques that one can use. The choice of technique depends on the particular phase of the requirements process and on the specific application. Wallace & Ippolito [1993] state:-

> *The major objectives of the software requirements process are to fulfil the system and software objectives, develop software requirements based on and traceable back to the system requirements and to provide complete, consistent, testable and understandable information from which the software may be designed.*

The first step in fulfilling these objectives is to 'identify' the system requirements. Jirotka & Goguen [1994] discuss terms used to describe this process, namely to *capture*, *specify*, *elicit* or *construct* requirements. The term *elicit* implies that the requirements are to be found amongst the managers, users etc. *Specifying* and *constructing* are terms taken from other engineering disciplines, the first implying that someone with the necessary technical skills can produce the requirements more or less as a matter of routine, and the second implying that the requirements do not exist but must be synthesized from the engineering process. Finally, *capturing* implies that requirements are elusive but are waiting to be *caught*. From the safety perspective *specifying* and *constructing* are important because they relate to interaction with other disciplines; *capture* refers to those difficult to define requirements related to the interface between the software and the system. All these terms are used for requirements engineering because all of these activities are part of the requirements engineering process and moreover these activities are closely coupled together in a process which is concurrent and iterative. Some requirements engineering techniques are better suited than others to particular activities.

Different frameworks have been used to evaluate requirements engineering techniques. Yadav et al. [1988] developed a framework to compare Structured Analysis Design Technique (SADT) and Data Flow Diagrams (DFDs) based on four dimensions: *syntactic, semantic, communicating ability* and *usability*. Evaluation criteria are associated with both the *syntactic* and *semantic* dimensions. These evaluation criteria are related to levels of abstraction, viewpoint, complexity, completeness and correctness. Lindland et al. [1994] use similar dimensions (i.e syntactic, semantic) for their framework but subsume *communicating ability* and *usability* into a single dimension *pragmatics* which relates the *model* to audience participation by considering, not only the *syntax* and *semantics* dimensions, but how the *audience* will interpret them. A model is a global term to describe what is used to analyse the problem, and *audience* refers to anyone involved in modelling. The framework of Lindland *et al.* [1994] was proposed as a means of determining the quality of the requirements engineering process. Each dimension has a *goal* and *means*, where *means* has associated *model properties* and *modelling activities*. For example, *syntactic* quality has *goal* (correctness) and associated *means* has *model property* (formal syntax) and *model activity* (syntax checking). The author considers that the following dimensions are important in the requirements process for safety-critical systems:

- ✦ Multi-disciplinary dimension - stressing the importance of cooperation between participants of different disciplines,
- ✦ Modelling dimension - signifying the importance of capturing and understanding the safety requirements,
- ✦ Specification of external system behaviour - highlighting the importance of the software system interface.

2.2.1 Multi-disciplinary perspective

Ross [1977] stated that :-

> *Our efforts have for too long been misplaced towards the system end of the scale. The real solution lies toward the human end of the scale where the real needs must be recognised and channelled into strengthened machinery for system building.*

Today our efforts are still focused on trying to achieve this objective, the emphasis has shifted from a technology centred view to a user centred view; in effect, for complex systems, the emphasis is on eliciting rather than specifying requirements. Kedzierski [1988] analysed the time spent on different activities during the development of complex software and found that over half the time is spent gathering all the necessary information and solving problems that arise from unsatisfactory information and communication. This is supported by results [Herbsleb et al., 1995] indicating that the major sources of designers' uncertainties are in obtaining sufficiently detailed knowledge of the application domain and how the software will function. For safety-critical systems, since the knowledge and expertise is distributed among people of different disciplines and since requirements elicited are important for subsequent safety analysis, effective communication during the requirements process is paramount. Ill-conceived requirements, often arising from uncertainties, are a recipe for disaster.

For simple systems, the software analyst can elicit requirements from the user and specify them in a model (using graphical or mathematical notation), without regard to whether other participants in the project can understand or interpret the specification method. This approach is not acceptable for safety-critical systems, as a software analyst may be 'naive' about safety principles and, in practice, cannot be expected to have specialist knowledge of a complex system of which the software is only a subsystem. It is vital that those responsible for identifying safety concerns can both understand and interpret the external behaviour described in the requirements specification. The problem lies in identifying models suitable for specifying safety-critical systems that are also amenable to interpretation by other participants. For ill-defined problems, Martin [1980] suggests that .... *the optimal strategy would be to choose a set of projections that are 'orthogonal' to or 'different' from one another in many separate senses, so that they each represent a very different 'slice' through both the problem situation and the user's body of experience.* From the safety perspective one *'slice'* should allow all participants to analyse the external interfaces and their associated safety implications.

2.2.2 Modelling

Models are invariably used in the design and building of systems. Since the only perfect model of a system is the system itself, models are necessarily idealized and do not perfectly match the real world situation. No amount of mathematical analysis will reveal discrepancies between the model being used and the real situation [Parnas et al, 1990]. Also, models may be perfectly adequate for one purpose but woefully inadequate for another. For example, for a given application, a model may prove very useful for data analysis, but completely inadequate for safety analysis. The general approach is to abstract pertinent features from the model so that we can focus on relevant issues; this may involve successive model building using analytic or synthetic methods. Different approaches to modelling include participative, structured, object oriented and formal methods.

2.2.2.1 Participative Methods: The participative approach [Benyon & Skidmore, 1987] ranges from highly socially-oriented methods to the more technically-oriented traditional prototyping methods. At the social end of the scale, we have ethnography which is a method of capturing the social activity (e.g. procedures, practices) of the users of a proposed or existing system and involves the ethnographer spending several months observing the users. Sommerville et al. [1993] discuss the requirements for a user interface to a flight database which is used to provide real-time information to air traffic controllers and describes how sociologists can contribute to requirements engineering using ethnography. They state *Ethnography is distinct from traditional systems analysis in that it focuses on the participants and their interactions in a system rather than the data, its structure and its processing.* Ethnography could be useful for the development of safety-critical systems as it provides a means of capturing interdisciplinary knowledge. The problem is converting this knowledge into structured information. The Viewpoint technique provides a possible means of solving this problem. It takes Viewpoints of different participants (agents) involved in a project where Viewpoints are *partial or incomplete descriptions which arise because of the different responsibilities or roles assigned to the agents* and the analysis of *Viewpoints embraces the relations between views, between views and agents, and between agents* [Finkelstein & Sommerville, 1996]. Although the Viewpoint technique is very flexible, this flexibility may give rise to problems when integrating the different Viewpoints at the end of the process. The proposed methodology of Jarke et al. [1993] is similar to Viewpoints in that it is based on organising requirements knowledge according to four related 'worlds' (*system, subject, usage* and *development*). The methodology focuses on the embedding of systems in their environment rather than on systems functionality and structure. The *system* world is related to the *subject, usage* and *development* world via representation, interface and process. The *development* world is related to the *usage* world via participation (e.g. prototyping).

The classical prototyping approach involves the following iterative cycle [Pressman, 1994] (a) quick design, (b) building prototype, (c) customer evaluation of prototype, (d) refining prototype. This approach can be very useful for safety-critical systems where one is often dealing with uncertainty and complexity, however, Jaffe et al. [1991] point out that prototyping has the same limitations as testing in that behaviour can only be guaranteed for certain inputs, not for all inputs.

2.2.2.2 Structured Methods: In the early 1970's Structured Methods were developed as a means of defining the behaviour of a system independently of the means of implementation.

It was introduced as a technique for partitioning, structuring and expressing ideas and relies heavily on graphics to indicate structure and relationships. Ross [1977], the originator of Structured Analysis Design Technique (SADT), makes a number of salient points relevant to structured analysis in general:-

- ✦ SADT is based on structured decomposition, and enables structured synthesis to achieve a given end;
- ✦ SADT incorporates any other language; its scope is universal and unrestricted;
- ✦ The structured decomposition may be carried out to any required degree of depth, breadth, and scope;
- ✦ The universality of SADT makes it particularly effective for requirements definition for arbitrary systems problems.

Data Flow Diagrams (DFDs) are the basic entity in many structured methods (excluding SADT). A DFD models the processes that transform data in a system and the interfaces between those processes by emphasising data flowing, being stored, and being transformed. The components of a DFD are *data flows*, *transformation processes* and *data stores* (Fig. 2.1). *Data flows* represent an imaginary route along which data passes. The details of each *data flow* is defined in a *data dictionary*. Each data transformation process in the DFD has inputs and outputs. The *data flows* that enter are processed and a new or modified *data flow* leaves the process. A *data store* is a store for various items of data in the system. *Data stores* hold data transferred or received via *data flows*, and represent accumulations of data within the system. DFDs may be partitioned into levels that represent increasing functional detail. The top level DFD is usually referred to as the *context diagram*. The bottom levels represent processes that are sufficiently simple to be implemented from the specification alone.

Extensions to DFD notation have been proposed to model real-time systems [Pressman, 1994]. The extensions proposed by Ward [1986] (known as the 'Ward & Mellor' approach) are also shown in Fig. 2.1. These extensions are: a control transformation process, continuous data flows, event flows and buffers. Finite State Machines [Mealy, 1955] can be used to define a control transformation. The continuous data flow is introduced to deal with real time data (e.g. the data from a sensor which is continuously monitoring the state of a plant). Event flows are used to represent flow of control data. The buffer stores control information. The Ward & Mellor approach appears to be an appropriate technique for modelling safety-critical systems as it contains all the basic modelling primitives that are needed (i.e. events, time, control and data flows).

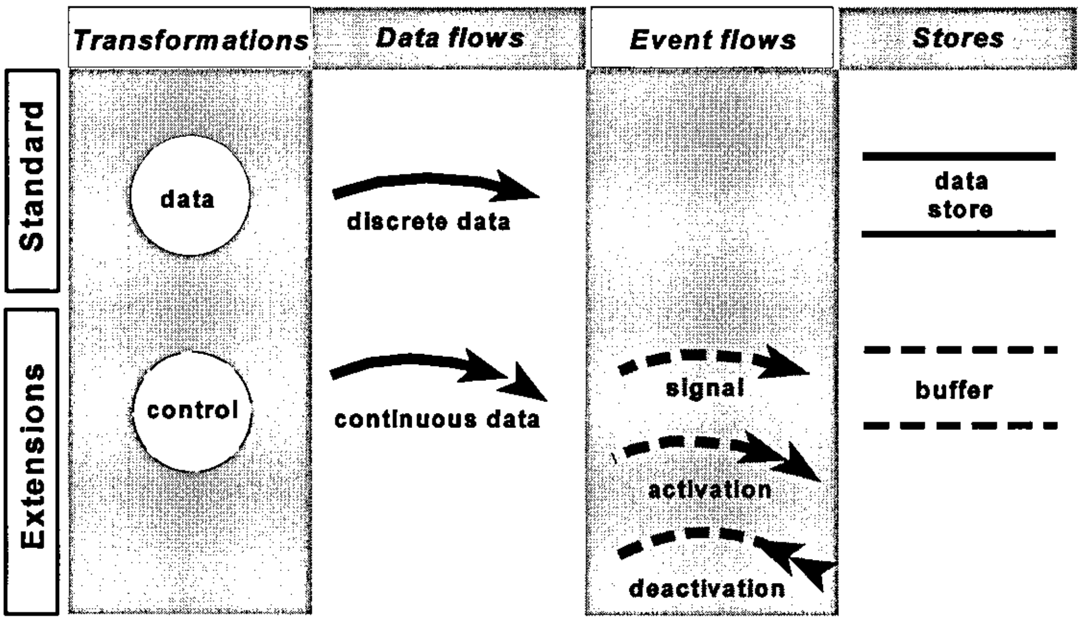


Fig. 2.1: Standard and extended DFD notation [after Ward, 1986]

2.2.2.3 Object Oriented Methods: There are many object oriented methods such as OMT [Rumbaugh et al, 1991], BOOCH [Booch, 1991], and COAD/YOURDON [Coad & Yourdon, 1991]. The fundamentals of the object oriented approach are that:-

(i) each object has associated with it data and operations;

(ii) groups of similar objects can be defined in terms of a common class where these classes may be arranged in a hierarchy;

(iii) objects communicate by sending messages to each other whereby the receiving object uses one of its operations to respond to a message.

Advantages of taking an object oriented approach rather than using the structured methods discussed above, include: reusability, easier maintenance, the models are closer to reality, and there is smoother transformation from the analysis model to the implementation model. Rumbaugh et al. [1991] points out that structured methods and object oriented approaches are similar because they both support three different models of the system (object, dynamic and functional). The difference between the two approaches is that the object oriented approach emphasises the object model whereas structured methods emphasise the functional model. This has important consequences when identifying safety concerns because structured methods favour a top down decomposition and the object oriented approach favours a bottom up method (this feature makes reuse possible). For safety-critical systems, the dynamic model is paramount. It is difficult to determine whether an object oriented approach or a structured methods approach produces more suitable dynamic models for safety-critical systems because there are so many different dynamic models used in the various object oriented approaches and, in some cases, the same dynamic models are used in both structured methods and the object oriented approach.

The object oriented lifecycle consists of object oriented analysis (OOA), object oriented design (OOD) and object oriented programming (OOP). Cuthill [1993] states:-

> *The OOD properties of encapsulation, abstraction, inheritance and refinement reinforce the safety-critical design features of modularity, functional diversity and traceability.*

However, one of the major difficulties of all requirements techniques is in establishing how to capture the requirements in the first place. Although many object oriented approaches provide excellent detail on OOD and OOP, they provide very little help on OOA. As Embley et al. [1995] pointed out in their survey of various approaches (including OMT, BOOCH and COAD/YOURDON), all approaches adopt a design perspective rather than providing real analytical support. Jacobson et al. [1993] propose both a requirements and an analysis model, the requirements model being constructed using *use cases* which model the interaction between the user and the system. *Use cases* have been adopted in the proposed amalgamation of OMT and BOOCH methods [Rumbaugh, 1996]. Another approach to aid requirements capture is *concept maps* [Umphress & March, 1991]. A *concept map* is a graphical representation of ideas that we have about objects and links between these ideas (e.g. nouns are concepts, and verbs are links). If we are to use an object oriented approach for safety-critical systems, we must have an understandable and precise method of relating safety concerns and the requirements model.

2.2.2.4 Formal Methods: Bowen & Stavridou [1993] carried out a survey on the use of formal methods in different industries (aviation, railway, nuclear power plants, medical and ammunition control). They discuss the use of formal methods in a number of safety-critical standards. Although the standards RTCA DO-178B, IEC and MoD 00-55 have a formal methods content, formal methods are only mandatory in MoD 00-55. It is interesting to note that MIL-STD-882C does not even mention formal methods. It is often claimed that the use of formal methods leads to concise, unambiguous and exact specifications without explaining why this is true. Rushby [1995] describes why formal methods are useful and, in particular, discusses their relevance to safety-critical systems. He points out that the major difficulties with software are complexity and discontinuity of behaviour. All possible behaviours must be considered under all circumstances and the discontinuous nature of the input/output relationship (i.e. a small change in input can result in a large change in output, or, local actions can have nonlocal consequences) must be modelled. Rushby [1995] states that formal methods provide us with a means of ... *identifying and grouping "essentially similar" pieces of behaviour together so that all members of a group can be dealt with at a single shot. .... By composing small pieces of behaviour together to yield larger and larger parts of the complete behavior, we eventually cover all possible end-to-end behaviors without having to enumerate them explicitly.*

Some disadvantages of formal methods are:-

- ✦ A formal specification does not ensure completeness. It may be proved to be correct, but this does not guarantee that all system requirements have been addressed.
- ✦ Formal methods are not understandable to the non-computer specialist. In describing formal specifications, McHugh [1993] writes, *While precise and supporting analyses based on theorem proving, the formal specification languages of the computer scientist often fail to satisfy the communications role that is the primary concern of IEC880 and similar standards.*
- ✦ A formal mathematical proof is time consuming and only suitable for very small programs, a few thousand-lines at most [Jesty et al, 1991].
- ✦ Formal methods do not model 'real world' entities, instead they focus chiefly on syntax and semantics of the language in question [Jackson & Zave, 1993].

Formal methods have been recommended and criticised for the wrong reasons, most of this controversy arises because of incorrect suggestions as to why they should be used and when they should be used. Formal methods are very useful for requirements analysis but not for requirements capture. At present, a reasonable approach to requirements engineering for safety-critical systems might involve the application of an informal technique, followed by a semi-formal (possibly scenario-driven) method, and finally, a formal method with the ability to model timing behaviour.

2.2.3 Specifying external system behaviour

We can associate the difficulties of specifying the behaviour of embedded systems with those of real time systems, namely:-

- How to handle complexity;
- How to model static and dynamic relationships;
- How to specify behaviour in order to facilitate testing .

This might suggest that it is worth investigating real-time specification methodologies, however, the question is not simply one of semantics and syntax, but of how real world behaviour can be specified. The ideal situation would be to model the environment explicitly. In discussing embedded systems, Zave & Yeh [1986] state ... *the best way to derive the requirements for a system is to model its environment (probably bottom-up,* ***synthesizing*** *many views and diverse pieces of information), and then work* ***'outside-in'*** *to the specification of requirements for an appropriate system. This is followed by top-down design and implementation of a system to meet the requirements.* ***Synthesizing***, in this context could be interpreted as elicitation (using, for example, the Viewpoint approach discussed above). ***Outside-in*** emphasises the importance of specifying an operational or external behaviour.

Davis [1988] compared a number of techniques for the specification of external system behaviour: Natural Language; Finite State Machines; Decision Tables and Decision Trees; Program Design Language; Structured Analysis/Real-Time (e.g Ward & Mellor);

Statecharts; Requirements Engineering Validation System (REVs); Requirements Language Processor (RLP); Specification and Description Language; Process-oriented Applicative and Interpretable Specification Language (PAISLey). Davis's criteria included:-

- ✦ The technique should encourage users to think and write in terms of external product behaviour in preference to internal product components.
- ✦ The resultant software requirements specification (SRS) should be helpful and understandable to non-software specialists.
- ✦ The resultant SRS should serve effectively as the basis for design and testing.

Davis [1988] scores each of these criteria on a scale from 0 (poor) to 10 (excellent). Adding up the respective values, shows the highest scoring techniques (using all criteria, not just the three quoted above) were RLP (50), Statecharts (48), REVs (45) and PAISLey (42).

2.2.3.1 Requirements Language Processor (RLP): RLP [Davis, 1988] uses as the organisational unit of the SRS, a stimulus response sequence which is a trace of a two-way dialogue between the system under specification and its environment. Typical dialogues are based on scenario generation, and correspond to user-oriented, user-known, external system features. The language used is dependent on the particular application (e.g. Ballistic Missile Defence, Patient Monitoring). Once the language has been defined, RLP can check the requirements for correct syntax and semantics. Davis [1988] pointed out that the choice between RLP and REVs (see below) should be based on the questions a customer might ask. For example, if the questions are about features rather than particular stimuli, RLP should be chosen.

2.2.3.2 Statecharts: Statecharts [Harel, 1987] are extensions to Finite State Machines. These extensions permit hierarchical decomposition of states, and the specification of transitions dependent on global conditions. The addition of these extensions make Finite State Machines more suitable for the specification of external behaviour of real-time systems. Statecharts have been used in a modified form to specify the safety properties for a traffic alert and collision avoidance system [Craigen et al., 1994]. The statecharts were changed to emphasize transition logic by incorporating a tabular representation of predicate calculus. The resultant SRS was found to be more reviewable and tractable than the pseudo-code version which had been created previously.

2.2.3.3 Requirements Engineering Validation System (REVs): The original motivation for REVs was based on a US Department of Defense directive which emphasized *the need for early software visibility, risk reduction, through software requirements analysis prior to the second Defense System Acquisition Review Council review of a weapon system (DSARC II) and greater 'front end' development* [Alford, 1977]. REVs has three main components (a) a translator for the Requirements Statement Language (RSL); (b) a centralized data base; (c) a set of automated tools for processing the information in the data base. Requirement Statement Language (RSL) has an associated graphical notation, R-nets [Bell et al., 1977]. R-nets are an extension to Finite State Machines and are used as the basic unit to represent the system's external behaviour; the fundamental entity being a stimulus. The R-net can be viewed as a column of a state transition matrix [Davis, 1988].

2.2.3.4 Process-oriented Applicative and Interpretable Specification Language (PAISLey): PAISLey, an executable specification language for embedded systems, includes specification methods and analysis techniques, and takes an operational view of the proposed system [Zave, 1991].The specification language is founded on the principles of asynchronous processes and functional programming; it predicts by simulation the performance of complex behaviour which would be difficult to determine analytically. PAISLey representation emphasizes the cyclic nature of the behaviour of components and its notation integrates data, processing and control in a unified whole. This is in contrast to RSL (see above) which emphasizes sequences and does not integrate data, processing and control [Zave, 1986]. Another functional programming language, Haskell has been used on a safety-critical project and it has been claimed that functional programming can be used as the key technology in the development of complex systems of significant size [Chudleigh et al., 1996].

The following conclusions can be drawn from Davis's data. RLP got the highest overall score (50) because it provided the best automatic test generation facility whereas PAISLey scored the lowest (42) because it is difficult to understand. On ability to represent external behaviour, RLP once more scored the highest (8), just ahead of REVs (7). These methods got a higher score than Statecharts (5) because Statecharts require specification of explicit external signals for an entity before defining the entity's internal structure. PAISLey scored 3. RLP, Statecharts, REVs and PAISLey facilitate an operational approach to requirements specification.

2.3 Integration of hazard evaluation procedures and software development

Although many hazard evaluation procedures are available to the safety engineer, it is difficult to determine which of these are appropriate for systems containing embedded software. Even in traditional safety engineering, the selection of a technique is by no means obvious as no single hazard evaluation procedure is suitable for all purposes. Safety-critical software standards recommend that hazard analysis be applied throughout the lifecycle, therefore, it is necessary to select technique(s) suitable for varying levels of detail depending on the associated objectives. The results from different phases of the lifecycle must be integrated so that software faults that contribute to hazards can be identified. In his discussion on hazard identification for chemical plants, Ozog [1985] states:-

> *Hazard assessment should be a continuous process throughout the life of a facility. There are optimum times to conduct studies - during conceptual design, design freeze, and pre-startup periods, as well as while the plant is being operated.*

The purpose of any hazard evaluation procedure is to identify hazards and either eliminate them or reduce the associated risk to a tolerable level; for programmable systems this includes identifying in the design process all errors that may lead to hazardous conditions in the system in which the software is embedded. Also, if hazard evaluation procedures are initiated early in the software lifecycle, it is easier to modify the system, thus saving time and reducing costs. As discussed above, the ideal way to achieve a safe system would be through fault avoidance, however, as this is not feasible, we look at what fault forecasting techniques might be suitable to complement fault avoidance techniques. A problem with many hazard evaluation techniques is in determining how to incorporate them into the software development lifecycle; this problem is further exacerbated by the imprecise semantics of traditional hazard evaluation procedures. We must identify the hazardous characteristics of the proposed system, and consider these in relation to the requirements engineering process. In the next section we look at hazard evaluation procedures and describe how they have been incorporated in software development.

2.3.1 Hazard evaluation procedures

2.3.1.1 Checklists: The Checklist approach is popular, easy to use and can be applied at all stages of the project. It can simply consist of a list of potential hazards with the onus being on the safety analyst to associate the list with the proposed system and environment. Checklists can be *generic*, *application specific* or *sector specific*. Sources of Checklists are reported in Leveson [1995], Wells & Wardman [1994] and Hammer [1980]. Checklists can be compiled for the complete development of programmable systems or for particular stages in the development. The Health and Safety Executive [1987] provides guidelines - including Checklists for safety requirements specification, hardware, software, installation, testing, operations and maintenance modification. It should be noted that only a multidisciplinary expert team familiar with all these aspects can respond to all items on the Checklist. Lutz [1993] compiled a safety Checklist for use in the analysis of software requirements. It was derived from common safety-related errors discovered during system testing. The focus was on inadequate interface requirements and discrepancies between the documented requirements and the requirements actually needed for correct functioning of the system. Kolb & Ross [1980] consider Checklists to be *'hard to compile, easy to misuse'*. No matter how much effort is expended on creating a Checklist, there is no guarantee that it is complete.

2.3.1.2 Fault Tree Analysis (FTA): FTA analysis is a top-down approach that focuses on a top event (e.g. accident) and then analyses the system to determine what single event or combinations of events could have led to the top event. The underlying logic of FTA is not difficult to understand, however, the construction of a meaningful Fault Tree requires in depth understanding of the system and expertise in the associated domain. Taylor [1994A] suggests a series of steps for constructing a Fault Tree:-

- ✦ choose top event [this could be selected, for example, from a serious consequence identified in a What-If study or a Failure Modes and Effects Analysis (FMEA)];
- ✦ identify possible causes in general terms (using case histories or morphological searches);
- ✦ localise hazards to specific places in the plant;
- ✦ identify various chains of disturbances, searching for individual component failure modes;
- ✦ identify individual causes of component failures.

Once the Fault Tree has been constructed, the major contributors to hazards can be identified and the Fault Tree used to eliminate hazards or reduce the risk to a tolerable level. It is important that the level of detail is not so complex as to make the evaluation of hazards intractable. Fussell et al. [1974] point out that FTA is of major value in:-

- ✦ directing the analyst to ferret out failures deductively;
- ✦ pointing out the aspects of the system which are important with respect to the failure of interest;
- ✦ providing a graphical aid to those in system management who are remote from the system design changes;
- ✦ providing options for qualitative or quantitative system reliability analysis
- ✦ allowing the analysis to concentrate on one particular system failure at a time;
- ✦ providing the analyst with genuine insight into system behaviour.

2.3.1.3 HAZard and OPerability Study (HAZOP): HAZOP is the most popular technique used for chemical process plant, and is also used in many other systems (e.g. mechanical, electrical and transport) [Robinson, 1995]. The aim of HAZOP is to generate credible causes of deviations from design intention and to identify consequences [Wells & Wardman, 1994]. The general HAZOP approach involves using *triggers* or *guidewords* and applying them to a model of the system. This approach can be used from the conceptual design stage of a project, right through to the commissioning stage. It is systematic, employs a team approach, and allows an exploratory approach to the identification of hazards. A HAZOP study results in a number of recommendations for design, equipment or operating philosophy improvements [AIChemE, 1985]. There are different types of HAZOP. Standard HAZOP uses the guidewords NO, MORE, LESS, AS WELL AS, PART OF, REVERSE, OTHER THAN that can be applied to different process parameters (e.g flow [CIA, 1977]). The AIChemE [1993] guidelines describe two variations of HAZOP, Knowledge-Based HAZOP and Creative Checklist HAZOP. In Knowledge-Based HAZOP the guidewords are replaced by the team leader's knowledge and specific checklists. Creative Checklist HAZOP is very similar to Preliminary Hazard Analysis. At an early stage in the project only materials and block layouts of plant are known and consequently only 'block' hazards can be identified; a hazards checklist (e.g. fire, toxicity) is used. Another variation, Human HAZOP, has been proposed to analyse human interaction with the process where standard guidewords are interpreted in terms of human error (e.g. NO is interpreted as NOT DONE, and REVERSE is interpreted as LATER THAN or MISORDERED [Whalley, 1988]).

It should be noted that HAZOP is a time consuming process, it identifies many more OPerability problems than HAZards, and combinatorial explosion results if process deviations are analyzed in conjunction with input/output states of computer hardware. Taylor [1989] points out weaknesses of HAZOP when applied to process systems. For the *more curious and rare modes*, the analysis depends on expertise. HAZOP's process does not get down to the deep causes of some types of accident. Taylor [1989] suggests bolstering the HAZOP process with past histories in the form of generalised Fault Trees.

2.3.1.4 Failure Modes and Effects Analysis (FMEA): FMEA (originally designed for reliability studies) is a methodical study of component failures. The process involves recording component failures on a data tabulation sheet and analyzing them individually so that the consequences of the failures can be identified. A criticality ranking can be assigned for each failure mode (Failure Modes & Effects Criticality Analysis; FMECA) which helps to focus on those areas of the design considered to be the most dangerous. FMEA helps to determine areas of a design where redundancy should be implemented, and to identify compensating features for those single point failures where elimination is impractical

[Reifer, 1979]. It is a bottom up approach that can be very thorough if all failure modes can be identified, however, it is very time consuming. AIChemE [1993] guidelines emphasise the importance of selecting the level of resolution, as this determines the detail to be included in the FMECA tables; the guidelines mention a plant level (e.g feed system) and a system level (e.g feed pump). Also, since all failures are not safety-related, FMEA is not an effective method of hazard identification, unless one can target safety-related components. Taylor [1994A] states:-

The success of Failure Modes and Effects Analysis depends on the analyst having a good understanding of what failure modes can occur within a process plant, and what modes it is necessary to distinguish to ensure a reasonably complete analysis.

2.3.1.5 What-If: What-If [Nolan, 1994; AIChemE, 1993] is similar to HAZOP (described above) in that it is a safety review based on team effort and it uses an exploratory approach. The difference is that, in What-If, instead of selecting a process parameter (e.g. temperature) and applying a guideword (e.g. NO), a more application specific What-If question is posed (e.g. What-If MOTOR M51 stops during startup?). It is not systematic. A What-If review is usually combined with a Checklist and is useful for identifying possible accident scenarios. Nolan [1994] lists the following advantages of What-If :-

- ✦ It can be accomplished with a relatively low skill level.
- ✦ It is fast to implement, compared to other qualitative techniques.
- ✦ It can analyze a combination of failures.
- ✦ It is flexible.

2.3.2 Selecting a hazard evaluation procedure

The underlying principles of the techniques described above are not complicated, the difficulty lies in determining which technique to use, when to apply that technique, and how to integrate that technique into the software lifecycle. Montague [1990] states that, for the chemical industry, the choice of technique is dependent on a number of factors, namely:-

the objectives of the study, the complexity of the chemical process, the age of the plant or process, the data requirements of the study, the resources available for the study, the level of expertise required in the use of the technique, and the potential consequences of accidents.

Important considerations based on these factors include:-

- objectives may demand quantitative and/or qualitative techniques;
- level of detail may indicate a coarse or fine screening technique;
- the complexity of the system determines how sophisticated the technique should be;
- if a new system is being built, the technique must be particularly suited to Preliminary Hazard Analysis (PHA);
- regulations and standards may dictate the choice of technique;
- available expertise may limit the choice of technique;
- the consequences of potential accidents determine how much in-depth analysis is required.

Hazard evaluation techniques are often combined. For example, Rushton [1994] carried out a survey on the use of HAZOP in the offshore industry, and found that the majority of correspondents selectively use FTA and FMEA in conjunction with HAZOP. This is not surprising as the completeness of individual methods has been shown to be very poor. Suokas & Rouhiainen [1989] carried out HAZOP studies on storing and loading/unloading of sulphur dioxide and ammonia. HAZOP identified a total of 77 accident contributors, two further techniques (Action Error Analysis and Work Safety Analysis) discovered 23 further contributors.

The choice of technique is dependent on the part of the lifecycle at which it is to be applied. There are various suggestions as to which techniques are appropriate for different phases in the chemical industry. Nolan [1994] proposes the Checklist approach for the Concept/Exploratory phase, followed by What-If for the pre-design stage, and finally HAZOP for the detailed design stage. FMEA has been suggested for both the pre-design and design stage and FTA for the detailed design stage [AIChemE, 1985] (Fig.2.2). In the standards (MIL-STD-882C and MoD 00-55), different types of hazard analysis and their associated phases have been described (Fig. 2.2). These include [MIL-STD-882C]:

Preliminary Hazard Analysis (PHA): To identify safety-critical areas, to provide an initial assessment of hazards, and to identify a strategy for controlling these hazards.

Subsystem Hazard Analysis (SSHA): To verify subsystem compliance with safety requirements contained in subsystem specifications and other applicable documents. To identify hazards and to recommend a strategy for dealing with hazards and risks.

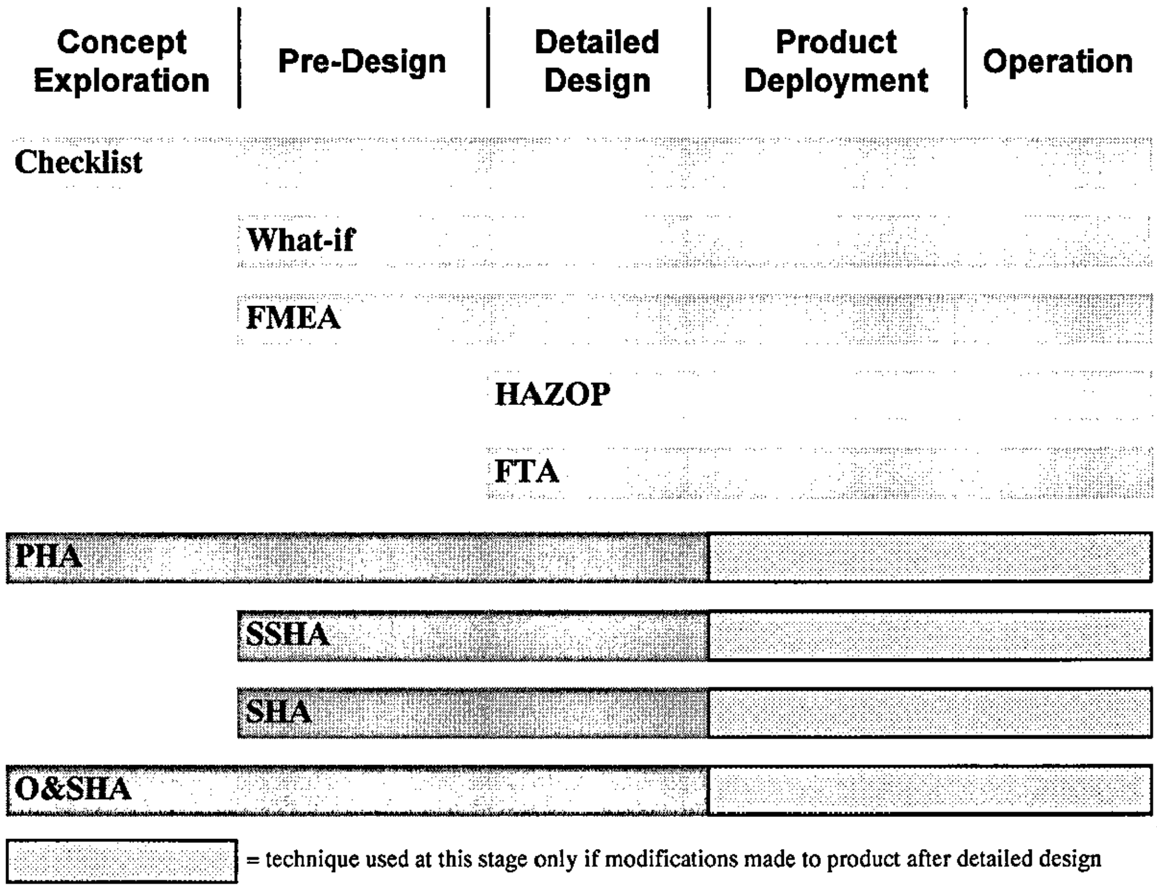


Fig. 2.2: Hazard evaluation procedures and associated types of hazard analysis (based on information from Nolan, 1994; AIChemE, 1995; MIL-STD-882C, 1993)

System Hazard Analysis (SHA): To verify system compliance with safety requirements contained in system specifications and other applicable documents, to identify hazards and risks associated with subsystem interfaces and system functional faults, to recommend actions necessary to eliminate hazards or reduce the risk to an acceptable level.

Operating and Support Hazard Analysis (O&SHA): To evaluate activities for hazards or risks introduced into the system by operational and support procedures and to evaluate the strategy for dealing with these hazards or risks.

The MoD standard 00-56 subsumes SSHA and O&SHA, into SHA. The difficulty is in correlating the different hazard evaluation procedures with the different types of analysis. MIL-STD-882C does not specify any hazard evaluation techniques to be used. The MoD standard [00-56] suggests using HAZOP at the PHA stage. Note that Wells & Wardman [1994] distinguish between PHA and HAZOP. They suggest that PHA starts at the point of a dangerous disturbance rather than a process deviation. The IEC standard states that FTA, HAZOP and FMEA are highly recommended for systems with high integrity levels, however, it does not state how they should be applied. Using information from the traditional application of hazard evaluation techniques, one might propose the matching shown in Fig.2.2. However, looking at previous workers' methods of assessing hazards in programmable systems (Table 2.2), shows there is no definitive matching of techniques and phases in the software lifecycle. Checklist and What If procedures are not included in Table 2.2 as they are not commonly used when integrating hazard evaluation procedures and software modelling techniques.

2.3.3 Approaches to integrating hazard evaluation procedures in the development of safety-critical programmable systems

An analysis of how hazard evaluation procedures have been integrated into software development (Table 2.2) shows that:-

- ✦ all combinations and a number of permutations of FTA, HAZOP and FMEA have been used ;
- ✦ the same hazard evaluation procedures have been used for evaluation of different stages of the system lifecycle;
- ✦ different software engineering modelling techniques have been used with the same hazard evaluation procedure.

2.3.3.1 FTA: FTA has been used at the system level, sub-system level and code level. FTA has also been used in conjunction with the 'Viewpoint' technique [Seward et al.,1995].

Table 2.2: Different approaches to integrating hazard evaluation procedures and software modelling techniques

Author(s)	Hazard Evaluation Procedure			Software Modelling Technique
	FTA	HAZOP	FMEA	
Seward et al. (1995)	▨			Viewpoints
Mojdehbakhsh et al (1994).	▨			DFD
Shebalin et al. (1988)	▨			
Leveson & Stolzy (1983)	▨			
Reunanen & Heikilä (1991)		▨		Ward/Mellor
Chudleigh & Clare (1993)		▨		DFD
MoD 00-58 (1995)		▨		DFD or Object-Oriented
Burns & Pitblado (1993)		▨		
Reifer (1979)			▨	
Klein & Lali (1990)			▨	
Canning (1990)			▨	
Saeed et al. (1995)	▨	▨		Object-Oriented
Maier (1995)	▨		▨	Ward/Mellor
ESA (1991)	▨		▨	DFD or Object-Oriented
Fenelon et al. (1995)	▨		▨	Goal Approach
Hobley & Jesty (1995)	▨		▨	Passport Cross
Fink et al. (1993)		▨	▨	
Taylor (1994B)	▨	▨	▨	

▨ = procedure used

Hazards are associated with Viewpoints and, subsequently, possible contributors to these hazards are identified using FTA. Mojdehbakhsh et al. [1994] identify both system and software safety faults using FTA, and use DFDs to help identify top events for FTA and determine the software safety faults that might contribute to hazards. They point out two limitations of FTA, namely, its informality and the lack of a method of specifying temporal relationships. Shebalin et al. [1988] suggest an FTA approach for safety analysis for distributed systems. Their approach involves identifying undesired system events and then constructing a system fault tree for each of the events. The resultant fault tree nodes which represent hazardous action taken by a component are analysed by associating with them different generic failure modes (*inappropriate command transmission*; *command failure*; and, *incorrect data sent*). These failure modes form the top events for subsequent fault tree analysis. Leveson & Stolzy [1983] use fault trees to analyse the logic of software (SFTA) and to determine safety violations. To achieve this, they present fault tree templates for some basic ADA constructs (e.g. if-Then-Else Statement, While Statement). McDermid [1996] has made some salient points regarding this approach namely:-

- SFTA should be applied top down;
- using the OR gate to represent the sequential composition of statements as specified in Leveson & Stolzy's templates is incorrect;
- SFTA based on the template-approach can be an effective analysis technique.

The major challenge of using FTA from system level down to software code level is to find an effective method of transferring the results at each level of abstraction.

2.3.3.2 HAZOP: Most attempts at integrating HAZOP into the software development process have included a software modelling technique. This is not surprising as, unlike FTA, traditional HAZOP has no associated logical representation, but is focused on piping and instrumentation or engineering line diagrams. The differences between the approaches to using HAZOP focus on how the deviation from design intention is represented. For instance, Reunanen & Heikilä [1991], using the Ward & Mellor approach (see above) analyse the deviations of flows on a state transition diagram, and Chudleigh & Clare [1993] using DFDs (see above) focus on deviations of the modelling entities (e.g. the entity process has associated deviations: FAILURE, ERROR, WRONG PROCESS, INTERRUPTED; and the entity data flow has associated deviations CORRUPTED, NONE, WRONG SOURCE/SINK). Problems arise because of the difficulty of applying traditional HAZOP to programmable electronic systems. The MoD 00-58 standard proposes new guidewords and suggests the approach is valid for functional or object-oriented representations. Burns & Pitblado [1993] also recognise the need for new guidewords, emphasise the significance of human error in computer controlled plants, and mention a number of incidents in which human error played a significant part. They propose new parameters and guidewords for 'Human' HAZOP, namely, INFORMATION (associated guidewords NO, MORE, LESS) and ACTION (associated guidewords NO and WRONG).

It is important to remember that HAZOP has its origins in Method Study [Elliott & Owen, 1968]. The first three stages of Method Study [Currie, 1972] are:-

- ***select*** *- the work to be studied;*
- ***record*** - all *the relevant facts of the present (or proposed) method;*
- ***examine*** *- those facts critically and in sequence.*

It is significant that the purpose of examination in Method Study was to determine alternatives **not** deviations, the assumption being that the method under examination could be clearly defined. The problems that arise with present approaches to software HAZOP appear to be due to the focus on this third stage ***examine*** (e.g. addition and refinement of guidewords) without the necessary attention to the ***select*** and ***record*** stages. Transferring these stages to software, ***select*** could be interpreted as 'determine the right level of abstraction' and ***record*** could be interpreted as 'model effectively'.

2.3.3.3 FMEA: FMEA approaches are generally based on functional failure rather than component failure. Reifer [1979] proposed that the software requirements specification be analysed to identify mission-essential requirements and failure-critical factors, where failure factors are defined as *software errors that are serious enough to cause the program, when executed, to either abort or degrade before the mission objective is realized*. Both mission-essential requirements and failure-critical factors are determined using checklists derived by analyzing previous projects. The requirements are refined until detailed FMEAs are produced and then each failure mode is evaluated and the corresponding effect at the software level determined. Klein & Lali [1990] propose that FMEA be performed for the functional modes of each system, subsystem or component. They claim FMEA, PHA and O&SHA are very similar but point out that, with FMEA, it is difficult to identify safety related failure modes that are not caused by equipment failures. They refine the FMEA method by incorporating:-

- ✦ lack of proper safeguards in the design;
- ✦ lack of operator training to follow procedures;
- ✦ lack of human engineering causing operator error.

Canning [1990] states that, for complex computer systems, FMEA may not be tractable for individual components. She suggests that FMEA be carried out at a functional level and states that this has an added advantage as it ensures that the assessment is independent of the means of implementation

For FMEA to be effective, it is essential to create generic failure modes, regardless of the 'view' (subsystem, functional, component) selected. The biggest drawback of FMEA appears to be its lack of ability to interrelate failures of interacting entities.

2.3.3.4 FTA & HAZOP: Saeed et al. [1995] suggest HAZOP as a means of identifying safety related failure modes in the requirements phase. The guidewords are based on an error classification scheme and vary depending on the applicable domain. Two domains are described, a value domain and a time domain: guidewords for the value domain are ARBITRARY and DETECTABLE and guidewords for the time domain are LATE, EARLY and INFINITELY LATE. Having identified the failure modes, FTA is then used to determine those circumstances that can lead to the postulated failure modes. The graphical notation of Rumbaugh et al.[1991] is used and the principal modelling abstraction is the *interactor* which corresponds to a class in object oriented terminology. The *interactor* has slots: a *name*, a collection of *components*, declarations of *constants* and *variables*, and a *behaviour specification*.

2.3.3.5 FTA & FMEA: The most popular approach appears to be combining FTA and FMEA with a modelling technique. Maier [1995] suggests using FTA or Highlevel Fault Tree Analysis (HFTA) to derive safety requirements, followed by application of FMECA to the user requirements specification. Finally, FMEA and FTA are used to review the software requirements document. FMEA is applied to *nodes* i.e. data stores and processes which have been constructed using the Ward & Mellor approach (see above). Generic failure modes associated with nodes are:-

- the failure to send a message, consequently, the message is missing at the following node;
- the untimely (too early or too late) release of message;
- the sending of a wrong (undesired release) or faulty message.

The resultant failure modes are used to construct the fault tree for the final analysis where the top events are base events from the earlier HFTA. The ESA [1991] approach is similar to Maier [1995] but also makes provision for an object oriented approach. This approach is based on a thread analysis, where FMEA and FTA are applied depending on the level of detail. FMEA is supported with a Checklist of generic failure modes for both functional analysis and an object-oriented analysis. For functional analysis, generic failure modes are given for the following element types: transducer, process, store, channel, controller, documentation and training. For object oriented analysis, generic failure modes are given for the following element types: objects, operations, relationships (e.g. USE relationships), specific objects (e.g. an object which represents the provided interface of another object used by the system to be designed, but which is not part of the design process). Fenelon et al. [1995] use a goal structured approach to guide the construction of a safety case, where a 'goal' represents an objective to be achieved and each of the goals can be represented in a hierarchical tree structure. Each goal has an associated context, strategy and solution. They propose a new technique, Failure Propagation and Transformation Notation (FPTN) to test whether the goals are satisfied. The objective of FPTN is to allow both a top-down mode (FTA) and a bottom-up mode (FMECA) analysis. The basis of the approach is the FPTN module which is a functional analog of Failure Modes and Effects Summary (FMES). Each FPTN module has an associated set of equations and these equations are representations of fault tree cutsets which define how the output failure modes are related to the input failure modes of the module. The failure modes are based on an error classification scheme. The types used are: *omission*, *commission*, *subtle* (e.g. incorrect value), *coarse* (e.g. inconsistent value), *early* and *late*. Hobley & Jesty [1995] base their approach on a modelling technique, PASSPORT Cross, derived from the Business Systems Planning system developed by [IBM]. The PASSPORT Cross consists of four matrices sharing common axes. The axes are labelled: information sets, communication facilities, functional elements and architectural elements. They propose using FMEA on all *sensitive* elements in the matrices and FTA for each hazard identified. The PASSPORT Cross model assists the FTA process by providing information on the inter-connections between various elements.

2.3.3.6 HAZOP & FMEA: Fink et al. [1993] propose combining FMEA and HAZOP. The failure modes and guidewords used were not generic but created specifically for the application in question. Their main conclusions were that HAZOP is useful for considering complex and interrelated procedures, but FMEA is more systematic and appropriate for considering single components.

2.3.3.7 FTA, HAZOP, FMEA: Taylor [1994B] suggests an integrated approach which considers the interaction between hardware, software and operator. He proposes functional analysis for the system specification; HAZOP, FMEA, simulation and sneak analysis for the system model; and Human Reliability Analysis (HRA) for operating procedures. This is a very comprehensive approach, however, the method of integrating the techniques is not described.

2.4 Conclusions

Different strategies for building safety-critical systems have been described. The emphasis has been on fault avoidance and fault forecasting, rather than fault tolerance and removal. In order to facilitate a fault avoidance approach, properties of requirements engineering techniques which are particularly suited to safety-critical systems have been identified. To investigate the feasibility of a fault forecasting approach, different hazard evaluation procedures and their integration into the software development life cycle have been discussed.

2.4.1 Requirements engineering and safety-critical systems

Three important dimensions of the requirements process have been identified for developing safety-critical systems: a multi-disciplinary dimension, a modelling dimension and the specification of external behaviour. The key to tackling the difficulty of requirements engineering for safety-critical systems is the modelling technique (i.e. the modelling technique must lend itself to multi-disciplinary participation and facilitate the specification of constraints on external behaviour). Despite the problem of identifying candidate object classes in the first place, object oriented modelling has been promoted as the ideal method for multi-disciplinary participation. However, safety objectives are usually expressed as requirements and this implies a functional approach. This apparent difficulty can be overcome if the choice of modelling technique is viewed as being dependent on the level of abstraction. It is possible to take the benefits of both an object oriented and a functional approach if they are used at different levels of abstraction. Whatever modelling technique is used, it must be possible to specify safety constraints. Different techniques for specifying external system behaviour have been briefly discussed. Although some of these specification techniques have been used in the development of safety-critical systems, none of them deals explicitly with safety constraints.

2.4.2 Limitations of present approaches to integrating hazard evaluation procedures and requirements engineering

Major concerns about the approaches described above include ambiguity of hazard evaluation procedures, inconsistency of modelling techniques and incompleteness.

2.4.2.1 Ambiguity of hazard evaluation procedures: For FMEA there is no clear definition of what *failure* is applied to, it can be applied to a requirement, a component, a data store or an object. For HAZOP the *design deviation* has different meanings depending on whether it is based on DFDs, state transition diagrams, or time and value domains. For FTA, a top event is used to identify an accident in the real world or is based on a failure mode in the software. Before a standard methodology for integrating hazard evaluation techniques and software development can be derived, precise meanings for the concepts associated with FTA, HAZOP and FMEA must be defined.

2.4.2.2 Inconsistency of modelling techniques: The same basic components of modelling techniques have been used for analysis but in conjunction with different hazard evaluation procedures (e.g. Chudleigh & Clare [1993] and ESA [1991] used DFDs but the former used HAZOP for flows and the latter used FMEA for the data stores and processes). In some cases, the choice of software modelling technique was dictated by the design rather than the usefulness of the particular technique for hazard evaluation. When techniques were used in combination, nearly all used some software modelling to allow the transfer of results from one level of abstraction to the next. On a superficial level, a functional approach can be associated with FTA & HAZOP and an object oriented approach with FMEA. However, a closer look at the different approaches shows that FMEA is often used to take a functional view, and FTA & HAZOP can be applied to objects that have a specified behaviour.

2.4.2.3 Incompleteness of approaches: At present, there is no ideal way of combining hazard evaluation procedures to ensure completeness and the problem is further exacerbated by the addition of software. There is no optimal sequence for applying the different hazard evaluation procedures to the development of software. FTA, HAZOP and FMEA have been used for PHA, SSHA and SHA, however, it is noticeable that there is an absence of techniques for O&SHA. Only the approach of Fink et al. [1993] specifically addresses this problem, but even this approach is application-specific.

CHAPTER 3 : Using Incident Analysis to Construct a Methodology

Various approaches can be used to try to identify reasons for the failure of safety-critical systems. Many of these are based on the categorisation of faults. Kershaw [1993], for example, categorised faults according to type (random failures of components; systematic errors in design; errors in the interface to the outside world; maintenance errors; common mode errors). Grady [1993] categorised faults according to where they occur in the development of a system (at the requirements stage, design phase, in the coding etc.) and further subdivided these into type (e.g. design phase faults were subdivided into hardware interface, software interface, user interface and functional description).

The approach taken for this study, however, was to analyse 'incidents' or '*... all accidents and all near-miss events that did or could cause injury, or loss of, or damage to property or the environment*' [AIChemE, 1992] and to use this analysis as a basis for a generic safety assessment methodology, HAZAPS (*HAZ*ard *A*ssessment in *P*rogrammable *S*ystems). The objective of this analysis was to determine, not simply the immediate causes of incidents, but the root causes, such as fundamental human error, failure of the technology, and inadequate development processes for the system. Examination of the root causes of incidents can be used to help prevent the occurrence of other similar incidents [AIChemE, 1989].

Data on computer-related incidents are available in many forms from various sources. Raw incident data from the avionics sector can be obtained from the Royal Air Force [RAF, 1989] and the US National Transportation Safety Board [NTSB]. Kletz [1995] described a number of computer-related incidents in the process industry. Neumann [1995] analyses incidents across many different industries where programmable systems are used. Incident data may be used to construct a methodology that can be applied in the development of safety-critical systems. Raw incident data are very useful, however, in order to benefit from raw data, it is necessary to become familiar with the domain concepts of the source industry. Classified data (i.e. incidents that have been mapped into categories) are not as useful because the classification scheme by definition necessitates a specific and, in some cases, biased interpretation of the incidents. Another problem with classification schemes is that they are difficult to reuse. However, classified data can be useful in identifying where the effort needs to be focused. For example, Lutz' [1993] analysis showed that the two most common causes of safety-related software errors are:-

- *inadequate interface requirements;*
- *discrepancies between the documented requirements and the requirements actually needed for correct functioning of the system.*

The knowledge gained from incident analysis is useful for the development of any system. It is particularly important for the development of safety-critical programmable systems because these systems are highly complex and, the more complex the system, the more difficult it is to determine what might go wrong (complexity is related to uncertainty and all risk arises from uncertainty).

3.1 Major benefits of incident data

3.1.1 Incidents are representative of the 'real world'

Incident data are representative of the 'real world'. The crash of the British Midland aircraft in 1989 (where both engines were shut down at the same time during flight) is illustrative of this point. *The chances of simultaneous failure of both engines are commonly estimated at somewhere between one in 10 million and one in a million* [Neumann, 1995]. Another example is the crash of the Eastern Airlines L-1011 near Miami because all three members of the flight crew were preoccupied with a blown 'gear-down' indicator light. This blown indicator light would not inhibit safe landing [Jentsch, 1993]. Incident data are based on the software operating in real situations with real users and real hardware.

3.1.2 Incidents provide insight into when, why, how failures occurred

Incident data give insight into when, why and how failures occurred and, in particular, how they propagate in a system on both a macro and micro scale. Often there are many different causal factors in an accident. By investigating incidents we can analyse the interrelationships between different contributors to accidents. Bradley [1995] analysed a number of catastrophes including Bhopal, Challenger, Chernobyl, Piper Alpha, Three Mile Island and several DC-10 accidents. He used a classification scheme to identify contributors to accidents. This scheme includes errors in five areas: design, failure of equipment with no directly attributable human error, management, operation, and repair. For each accident, a disaster sequence was identified based on these contributing factors. All accident sequences included two or more of the five contributing factors, and one (the DC 10 incident in Chicago, 1979 where an engine fell off on takeoff) included all five. A single failure does not, in general, result in a catastrophe, this is usually due to a combination of different failures which occur in sequence. It is difficult to hypothesize about what faults might lead to failures and result in incidents, however, by working backwards from incidents, causal sequences can be readily identified.

3.1.3 Incidents provide an invaluable source of experiential knowledge

The experiential knowledge gained from a study of incidents can be used to construct a methodology for assessing safety before a system is put into operation. It is useful to identify and categorise contributing factors to incidents but, ultimately this knowledge must be embedded in a framework for use by the developer of safety-critical systems. This framework must be effective, efficient and easy to use. In the next section, the strategy used to derive a framework based on computer-related incidents is described.

3.2 Strategy used to derive a framework based on computer-related incidents

When building new systems, one way of using incidents is to store them in a database or hazard log. However, the disadvantage of doing this is that, for every new system, it would be necessary to (a) search the database for all relevant incidents, (b) match them with the subsystems or components, and, (c) understand and analyse the sequence of each selected incident to see if it could occur on the new system. This would be difficult even if used for rebuilding a similar system within the same industry and would not be feasible for use in a different industry. The strategy used for utilising incident data in this work is shown in Fig. 3.1. In summary, patterns were abstracted from incidents and a generic framework was

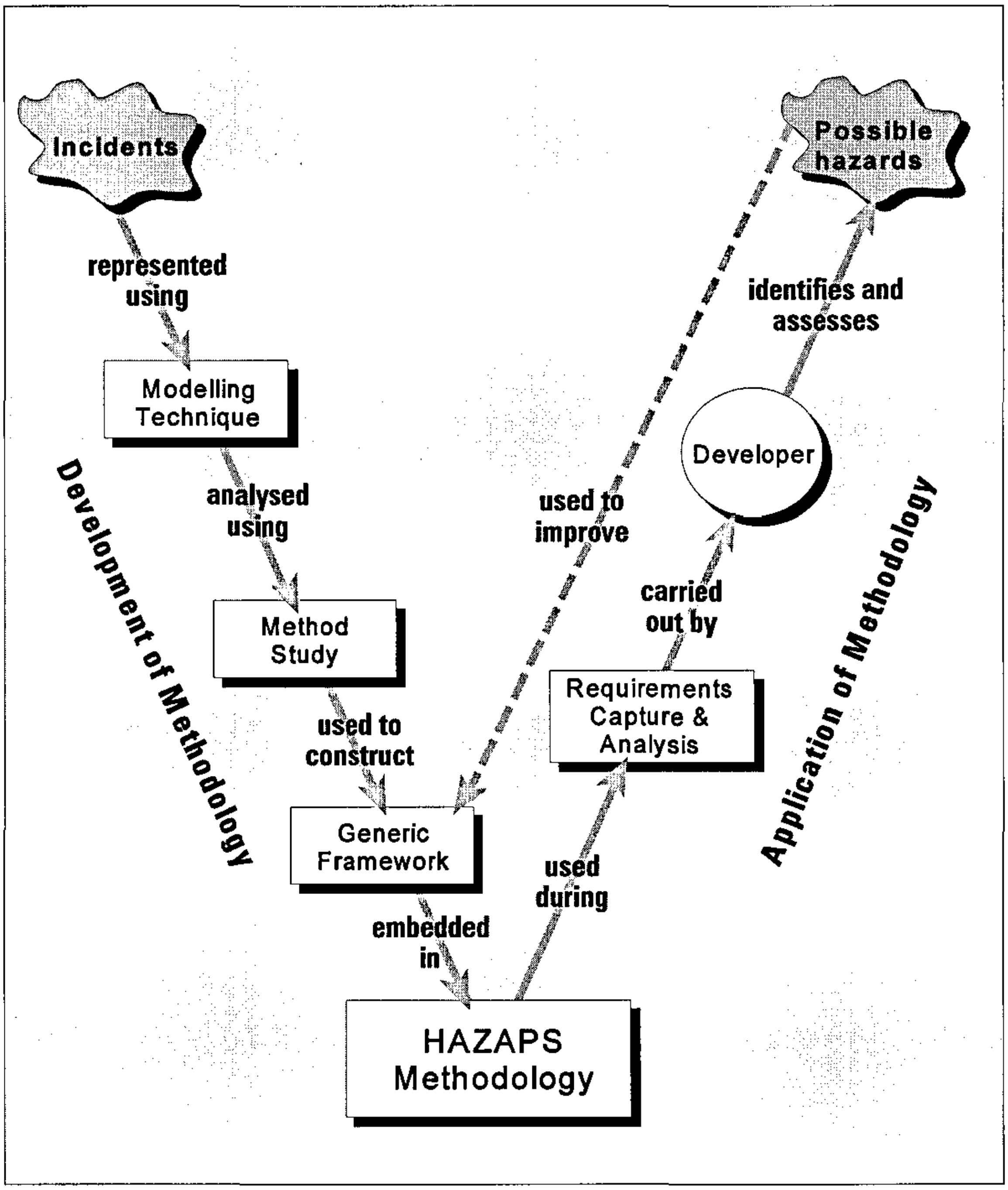


Fig. 3.1 Illustration of the strategy used to develop the hazard assessment methodology

constructed. This framework was subsequently embedded in a methodology that can be used by developers when building new systems.

Although hazards are environment dependant, the way in which the embedded programmable system contributes to hazards can be generalised. For example, the behaviour of an actuator is similar whether it is located in a chemical plant or on an aircraft. The strategy was to focus on the causal behaviour of an embedded system that can result in an incident. This involved:-

- ✦ recognising common properties and entities within an embedded system, irrespective of the application domain;
- ✦ modelling these entities;
- ✦ identifying causes of incidents;
- ✦ isolating the causes;
- ✦ constructing a framework which provides a systematic means of identifying the way in which an embedded control system can interact with its target environment (such a framework is very useful because captured knowledge, based on analysis of incidents, can be presented in a structured fashion and made accessible for future projects).

Chapter 4 describes how this framework can be embedded in a methodology which assists developers during requirements capture and analysis. The embedded framework provides a generic approach to identifying and assessing hazards in the system. This chapter describes the construction of the framework. In the next section, the modelling of incident events is described.

3.3 Modelling of incidents

The initial objective was to derive a modelling technique for embedded systems that represents the sequence of events involved in an incident. This modelling technique had to meet the following criteria:-

(a) be suitable for all incidents independent of the application domain;

(b) provide a means of understanding and interpreting incidents;

(c) facilitate the creation of a framework in which the knowledge gained could be embedded.

A number of possible options were available. Finite State Machines [Mealy, 1955] were considered, however, for each incident, it would have been necessary to have an in-depth knowledge of the system and also it was not obvious how a generic framework could be created from the analysis. Another possibility was to develop a technique based on customising Management Oversight & Risk Tree (MORT) [Johnson, 1980]. MORT itself is not specific enough for analysing incidents in terms of computer events. A MORT technique would result in a logical representation of the causes of incidents, however, it would be difficult to correlate the occurrence of different computer events and the number of resultant causes would be very large. Consequently, the resultant framework would be unsuitable for a developer building a new system. There are many other incident investigation techniques [AIChemE, 1992] but none of these fulfil the three criteria (a) to (c) above.

3.3.1 Generic events in embedded systems

To model an incident, it is essential to be able to represent discrete events. However, different domains have an infinite number of discrete events. To overcome this problem, events can be 'typed', so that any given incident can be associated with a number of generic events. As a means of identifying generic events, the following components which are generic to all embedded system architectures were selected:

- processor;
- communications link;
- sensor;
- human input device (HID);
- display;
- actuator;
- operator.

The inclusion of 'operator' in the above list might be questioned, however, the importance of considering the man-machine interface is borne out by a survey carried out by the Institute of Nuclear Power Operations (INPO). INPO analysed 180 significant incidents and identified a total of 387 root causes [INPO, 1985]. More than ***half*** of these were due to human performance problems. This human performance category was further subdivided and included: deficient procedures or documentation (43%), lack of knowledge or training (18%) and failure to follow procedure (16%).

3.3.2 Event Time Diagram (ETD) for modelling events

Any interaction within an embedded system involves one or more of the basic components above, therefore, events can be characterised by these components, and then used to analyse incidents. Each component can be associated with a functional behaviour (e.g. the operator's function is to intervene). A model (Fig. 3.2) was proposed to relate the basic components and their associated functional levels. The functional levels were subdivided into categories. For example, the input/output (I/O) level was divided into Display, Actuator, Sensor, and Human Input Device (HID). There is a hierarchy of the functional levels and these functional levels are interdependent. All components associated with the intervention level that interact with the computer, must use a component at the I/O level. Similarly, all components associated with the I/O level that interact with the computer, must involve a component at the communication level. The model can be used to establish 'what if' scenarios. For example, if an operator inputs incorrect data, what happens if the error propagates through to the inner levels? In the worst case, the operator error causes a failure at the control and processing level. This model can be used to represent incident behaviour by placing nodes and vectors on functional levels as shown in Fig 3.3.

The Event Time Diagram (ETD) is a populated functional model which models behaviour in terms of events, time, control and data flow, entities and associated functional levels. It may be viewed as a polar diagram where the angle represents time, the distance from the centre gives the functional level, and the arrows give direction of flow of information (either control or data). A node within an ETD may be described by two coordinates (c, r) where c represents the angle (entity), and r represents the radius (functional level).

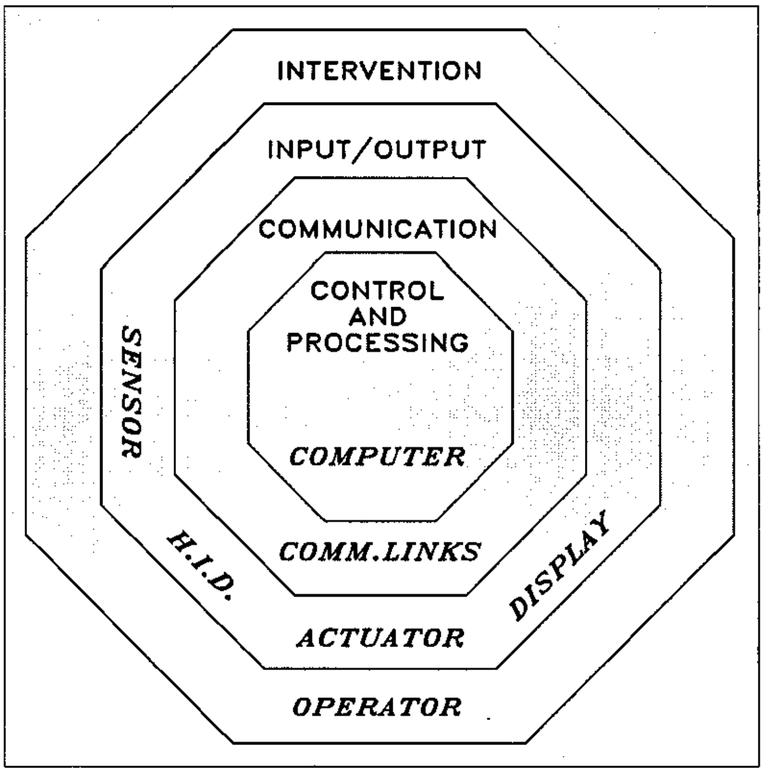


Fig. 3.2: Model showing generic components and functional levels

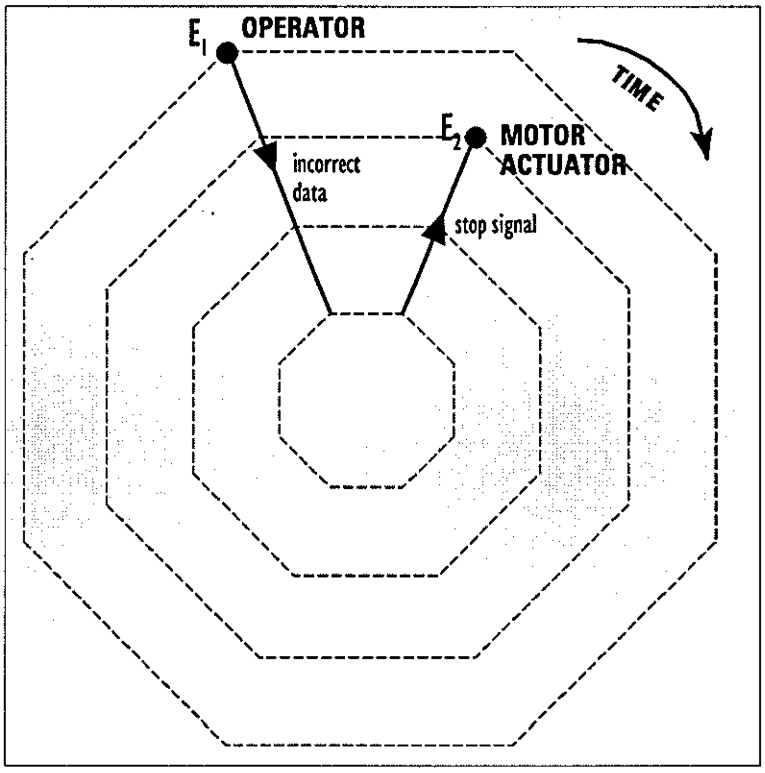


Fig. 3.3: Example of a populated Event Time Diagram (ETD)

The ETD modelling technique fulfils the criteria (a) to (c) above:-

- ✦ The ETD can be used to model any incident irrespective of the application. It is based on identifying external events and classifying these events in terms of generic components. It allows the system to be viewed from the 'outside-in', working back from observable phenomena to programmed events.
- ✦ The ETD allows the generation of scenarios which assist in understanding and interpreting the causal behaviour associated with incidents. It allows analysis both of individual events and of the interaction between events. A picture of what happened can be built up in an incremental and iterative fashion.
- ✦ The ETD is based on instantiating generic events rather than specific events, hence it is possible to construct a framework based on the knowledge acquired about these generic events and their associated safety properties.

In the next section, the use of Method Study for analysing incident events is described.

3.4 Use of Method Study for incident analysis

Having modelled the incident using the ETD, the next step was to understand and reason about the events. The approach taken was to identify why and how events within the embedded system together with other events in the environment led to the hazardous situation which preceded the incident. The objectives were to identify root causes and to incorporate these causes in a framework. This framework provided a means of assessing safety in future systems. The root causes were divided into three main categories:-

- ✦ **Specification:** failure to understand what the system is required to do (e.g. ambiguous objectives, wrong timing, inadequate control).

- **Implementation:** failure in the implementation plan (e.g. wrong devices, insufficient testing, inadequate consideration of environment).

- **Protection:** failure to identify, control or recover from hazardous situations (e.g. inadequate failure detection and recovery procedures).

The technique used for identifying root causes was Method Study, a popular problem-solving technique developed in the 1960's. Method Study was introduced as a management technique to improve methods of production so that more effective use could be made of materials, plant and equipment, and manpower. It has five stages [Currie, 1972]:-

***Select** - the work to be studied.*

***Record** - all the relevant facts of the present (or proposed) method.*

***Examine** - those facts critically and in sequence.*

***Develop** - the most practical, economic and effective method, having due regard to all contingent circumstances.*

***Install** - that method as standard practice.*

***Maintain** - that standard practice by regular routine checks.*

The power of the Method Study process lies in its third stage *Examine* which is sometimes referred to as *Critical Examination*. Table 3.1 shows a *Critical Examination* Sheet from Method Study. The examination is carried out by using two sets of questions, the first relating to the 'present facts' and the second to 'alternatives'. These questions are divided into five categories (Purpose, Place, Sequence, Person, Means).

3.4.1 Modifications to Method Study

In applying Method Study to analysis of incidents, the first four stages of the Method Study process are used in principle. The *Select* stage essentially involved the choice of an incident for analysis. The *Record* stage involved modelling the incident using the ETD. The *Critical Examination* stage identifies root causes and the *Develop* stage maps to embedding the root causes in a framework. The last two stages, *Install* and *Maintain*, are management responsibilities and therefore are not relevant for this analysis.

The Present Facts			Alternatives	
Purpose	**WHAT is achieved?**	**IS IT NECESSARY? If YES, Why?**	**What ELSE could be done?**	**What should be done?**
NOTE: What is ACHIEVED not what or how it is DONE.		Reason given may not be valid. True reason must be uncovered.	Can the achievement be ELIMINATED? Can the achievement be MODIFIED? All alternatives to the purpose should be stated including those which may require long-term investigation. The answer to this section is never "nothing"; there is always an alternative even if only the non-achievement	Helpful to divide into short-term and long-term. Under long-term can go suggestions for future research and development
Place	**WHERE is it done?**	**WHY THERE?**	**Where ELSE could it be done?**	**Where should it be done?**
The location with reference to (a) Geographical position (b) Position within the factory, plant or area (c) Detailed position under (b) When appropriate, give reference to location and distance from preceding and succeeding activities.		The reason for siting the operation there.	Consider alternatives under each heading. Can working areas be combined or distances reduced?	Where appears to be most suitable situation with present knowledge? Answer may be in relation to some other operation. Consider limitations of building design and services (steam, air) etc.
Sequence	**WHEN is it done?**	**WHY THEN?**	**When ELSE could it be done?**	**When should it be done?**
What are the previous and subsequent significant activities and what are the time factors involved?		The reason for the present sequence and time factor in the present process.	Can it be done either earlier or later in the process? If the sequence is fixed, can it be moved back to the previous operation? For example "Immediately after".	As soon as possible in the process or immediately after the previous activity.
Person	**WHO does it**	**WHY THAT PERSON?**	**Who ELSE could do it?**	**Who should do it?**
(a) Grade, e.g. unskilled worker (b) Employment, e.g. day worker (c) Name(s)		Reasons for choice under each heading.	All alternatives under each heading.	It may not be possible to select the individual without Work Measurement.
Means	**HOW is it done?**	**WHY THAT WAY?**	**How ELSE could it be done?**	**How should it be done?**
All relevant details are required of Material, Equipment and Operator engaged in the operation. Information should be tabulated as simply as possible, under the headings: (a) Materials employed, (b) Equipment employed, (c) Operator's method		The reason should be investigated for each of the tabulated items under each main heading.	Investigate all alternatives for each main heading.	Decide the alternative for each item separately and knit together at development stage. Consider safety. Consider posture and environment operator.

Table 3.1: Guide to the use of the Critical Examination Sheet (adapted from Currie, 1972)

In applying *Critical Examination* to analysis of incidents (Table 3.1), 'present facts' were related to what was observed from the raw incident data, with 'alternatives' being viewed as strategies for identifying, eliminating and controlling hazards. The Purpose, Sequence and Means categories were applied to events associated with an embedded system. Although the application of the categories Place and Person is not so obvious, Place questions are useful, for instance, for Displays, Sensors, Utilities, and Person questions could be used for assessing training of personnel, and possible requirements for automation.

3.4.2 Benefits of applying Method Study

Other workers have proposed approaches to incident analysis based on the questions What? Why? Where? When? How? The author believes that, without an underlying causal model, these questions are not sufficient to investigate the complex behaviour of embedded systems. Elliott & Owen [1968] analysed chemical plant design using Method Study. They applied it to the stages of the overall process selection. They stated ... *critical examination techniques can assist designers to produce cheaper, safer and more reliable plants* .... Their first approach was to cover *acres of paper* with questions and answers relating to the operation of the plant [Kletz, 1992]. Although this identified many potential hazards and operating problems, it was too detailed to be practical. However, after many refinements and modifications it evolved into the well-established HAZOP technique used worldwide by the chemical industry. Method Study provides a powerful analytical approach to incident analysis, in that it is focused, structured and provides a logical plan to highlight safety concerns and show where possible improvements can be made in the development of safety critical systems. The results from Method Study are very useful, however, it is a time consuming and laborious process, and, as Randall [1969] states ..... *it can reveal things invisible to the naked eye,* [but] *it can only focus on a small area at a time.* One of the major problems with Method Study is the amount and complexity of the information that can be generated. In this work, in order to limit complexity, Method Study was constrained by:-

- ✦ working backwards from incidents;
- ✦ generating general rather than specific questions related to specification, implementation plan and protective measures.

In the next section, the analysis of incidents using the ETD technique, in conjunction with Method Study, is described.

3.5 Analysis of incidents

The objective of the incident analysis was to determine why an incident occurred and what could have been done to prevent it. For a given incident, the approach was to represent the events associated with the incident and to derive a set of questions based on the underlying principles of Critical Examination, the focus being on why an incident occurred and what questions could have been asked in the first place to prevent it (i.e. what questions would have prompted the developers of the system to consider the occurrence of such an incident). These questions were generalised and used to build a framework which could be used by a developer and thereby prevent the recurrence of the same or similar incidents. The value of such a framework depends on:-

- ✦ how representative the incidents are;
- ✦ the methods used in deriving and generalising questions to build the framework;
- ✦ the effective incorporation of the framework into a development methodology.

Although industries record incidents, there is no established method of using these incidents to identify hazards at a generic level in re-engineering old systems or developing new systems, i.e. each incident is dealt with individually following its occurrence. More than 300 incidents were provided by two major organisations, one involved in the process industry, the other in avionics. The reason why two different application domains were used was to try to ensure that abstractions of safety properties were domain-independent. The format of the information received is shown in Tables 3.2 and 3.3. No user requirements, functional specifications, architecture diagrams or software code were provided for any of the incidents. This might first appear to be a disadvantage, however, it forces one to take a 'real world' view of a system and to decompose the system in a general manner thereby preventing the methods developed from becoming application-specific.

Table 3.2: An example incident from the AVIONICS INDUSTRY

A/C Type	Flight Phase	Location	Date	Occnum	Permpub
BXXXXXX	CRUISE	LXXXXX	XX XXX XX	XXXXXXXF	P
FMS malfunction in cruise at FL350 A/C nosed over lost 600ft in 5sec ---- departing altitude due to the loss of air data reference power caused by a faulty one amp circuit breaker					

Key: FMS = Flight Management System: FL = Flight Level: A/C = Aircraft

Table 3.3: An example incident from the PROCESS INDUSTRY

OBSERVED EFFECT	**ROOT CAUSE**	**CAUSE CATEGORY**
SEQ. STARTED PREMATURELY	SEQ. PERMISSIVES INCOMPLETE	APP.S/W

Key: SEQ = sequence; APP.S/W = application software

3.5.1 Procedure used to analyse incidents

The following procedure was used to analyse incidents:-

1. Study the text of the incident to determine whether it is useful (i.e. does it contain sufficient information and can it be understood without an in-depth knowledge of system).

2. List set of instantiated generic events and model on ETD. The intention is to identify the original purpose of the design, the components involved, the control and data flows, and any associated constraints.

3. Analyse the ETD, in order to understand the interactions and sequence of events. Iterate between scenarios in an attempt to identify where problems might occur. It may be necessary to make assumptions. The procedure is driven by the need to hypothesize on what did happen, what should not have happened and what could happen in a different situation. It is important to identify as many causes as possible to prevent similar incidents from recurring in the future.

4. Identify critical events and generate questions. Questions should be general rather than specific (i.e. should ignore environmental data and should abstract generic information).

3.5.2 Analysis of sample incidents

The ETD shown in Fig. 3.4 is an 'interpretation' of the incident in Table 3.2. Assumptions were made such as 'the computer made a calculation based on erroneous data' (E4). It was seen that a number of events (E1, E2 and E4) led to the final consequence (E5). E1 is the root cause, however, any of the events (E2, E3, E4) could be root causes under different

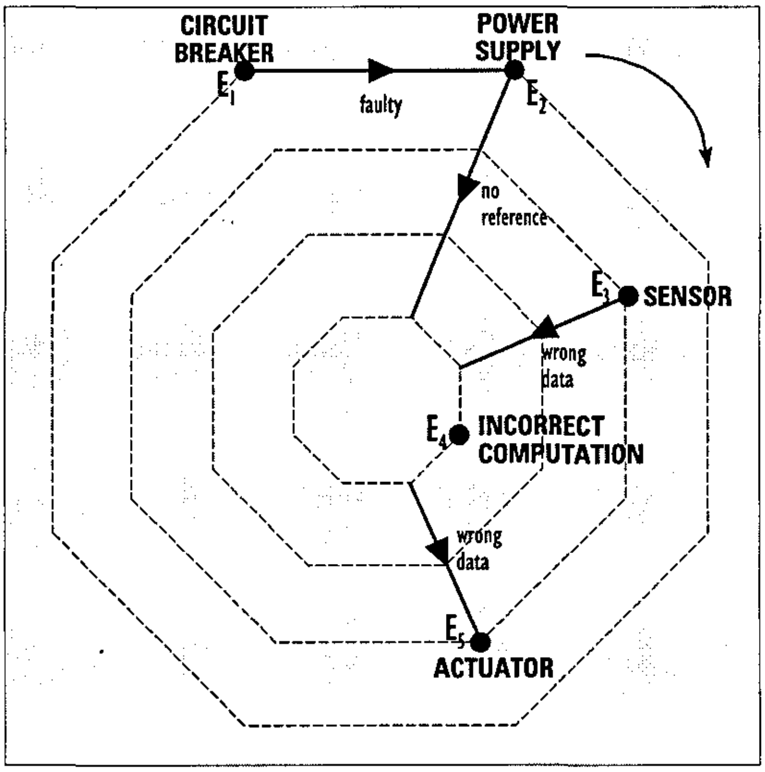


Fig. 3.4: ETD for the incident described in Table 3.2

circumstances. For example, incorrect computation (E4) may result in E5 occurring irrespectively of E1, E2 and E3. In addition, we could prevent E4 by preventing the computer from acting on erroneous information, however, this would not be sufficient as it is quite likely that the actuator requires continual updating, therefore, even if the computer recognizes erroneous data, there must be some recovery mechanism. This indicates the importance of considering all events. The ETD also allows us to analyse events on an individual basis. Even if E1, E2, E3 and E4 did not occur, E5 could still occur if, for instance, the communication link between the processor and actuator failed. The ETD provides a very effective mechanism for analysing incidents because all events are connected together via the nodes and vectors. We can start with the basic component on the node associated with the first event and trace all state changes in intermediate components through to the basic component associated with the final event. Alternatively, the state of an intermediate component can be selected and one can work backwards to a cause or forward to a consequence.

Having represented all the events related to the incident, the next step is to pose a series of questions which, had they been asked in the early in the development, would have prevented this and **similar** incidents from occurring. The questions generated are:-

- What reliability data are available for hardware components?
- How is failure of power detected?
- If power fails, how is the system placed in a safe state?
- How is the sensor to be calibrated?
- What is the range of the expected input values?
- Are multiple sensors required?
- Is a continuous self-test sequence required (e.g. to detect dramatic changes in input)?
- Is there any method of verifying or correlating output data to detect out-of-range values?

As the number of questions increased, it was found necessary to describe succinctly (by means of a 'general descriptor') the failure mechanisms associated with each incident. Consequently, all failure mechanisms could be classified under a limited number of general descriptors, the objective being to crosscheck that the main failure mechanisms for all incidents had been identified. The general descriptor for the above incident was "erroneous/corrupt operation".

There is very little information in the incident described in Table 3.3. The root cause in this case is that the sequence permissives were incomplete and the consequence was that the sequence started prematurely. However, we can still derive questions:-

- ✦ What are the preconditions for initialisation?
- ✦ How is it ensured that all preconditions have been identified?
- ✦ How is task initialised?
- ✦ How is task prevented from being initialised unintentionally?

The general descriptor is 'incorrectly initialised'.

Incidents trigger other questions which are not necessarily directly relevant to the incident under consideration. For example, in the above incident, the task started prematurely, it could equally happen that the task might end prematurely or might not end at all which leads to the questions:-

- ✦ What are the sustaining conditions for this task?
- ✦ What are the postconditions for this task?

As more incidents were analysed, more questions were generated until eventually it became difficult to generate new questions (i.e. the law of diminishing returns applied). Different problems arise as questions are generated. Some questions, features and appropriate reactions were:-

- ✦ too specific - generalise question;
- ✦ too complex - decompose into one or more questions;
- ✦ repeated - eliminate question;

- similar - if no significant difference, eliminate, otherwise create new question.

Subsequently, a framework was constructed to help ensure that the questions could be applied to a new system in a logical fashion. The construction of the framework is described in the next section.

3.6 Construction of the framework

The purpose of the framework is to assist the developer (who may not be a software specialist) to gain a deeper understanding of the system, probe the safety aspects of the system in a structured and systematic fashion, and thus strive to eliminate common safety-related faults before the design stage.

Two methods of classifying questions were used: *traditional classification*, based on classes having predefined properties; and, *clustering* [Stepp & Michalski, 1986], based on grouping entities together, formulating conceptual descriptions and consequently identifying classes.

3.6.1 Traditional classification

Predefined classes include superclasses:-

- **Specification:** questions related to understanding what is involved.
- **Implementation:** questions related to what should be considered in the implementation plan.
- **Protection:** questions related to what protective measures are to be adopted.
- **Failure_Modes:** all general descriptors derived from incidents.

3.6.2 Classification by Clustering

The clustering method is used to identify child classes of the superclasses Specification, Implementation and Protection. Each of these child classes has slots for the generic components: processor, communications, sensor, HID, display, actuator and operator (see Fig. 3.5). The clustering method is not required to identify child classes of the superclass Failure_Modes as the general descriptors *are* the child classes. The clustering approach involved grouping together sets of questions which appeared to have a common concept. The process involved refining, elaborating and iterating, both within and between different groups of questions. In some cases, one or more classes were readily identified from a single group. In other cases, it was necessary to merge groups to form a class or to reorganise questions into a different group. As the process proceeded, existing classes were replaced, subdivided, merged, and occasionally, new classes were derived by analysing existing classes. Finally, one question was derived to show the overall purpose of that particular class. Table 3.4 shows a superclass, class with associated question, and slots with associated questions. The superclasses were created so that questions could be applied in a logical sequence (i.e. Specification, Implementation, Protection and Failure_Modes). Failure_Modes provides a means of cross-checking the results from the other superclasses. The classes encourage the user to think of other questions which might also be applicable and to remove any subjective bias in the existing questions.

3.6.3 Choice of Classes

Considerable thought was given to the choice of classes. Useful guidelines have been proposed [Chillarege et al., 1992; Fleishman & Quintance, 1984] for developing classification schemes. Important criteria include:-

- ✦ classes should be defined as precisely and objectively as possible;
- ✦ classes should be distinct and mutually exclusive (orthogonality);
- ✦ classes should be simple and easy to understand (to avoid human error and confusion);
- ✦ there should be some evidence of class validity;
- ✦ the number of classes should be small so that the user can accurately resolve between them;
- ✦ the complete set of classes should attempt to cover all available data.

Table 3.5 (a to d) shows the superclasses, their classes and associated questions. The complete framework is given in Appendix 1.

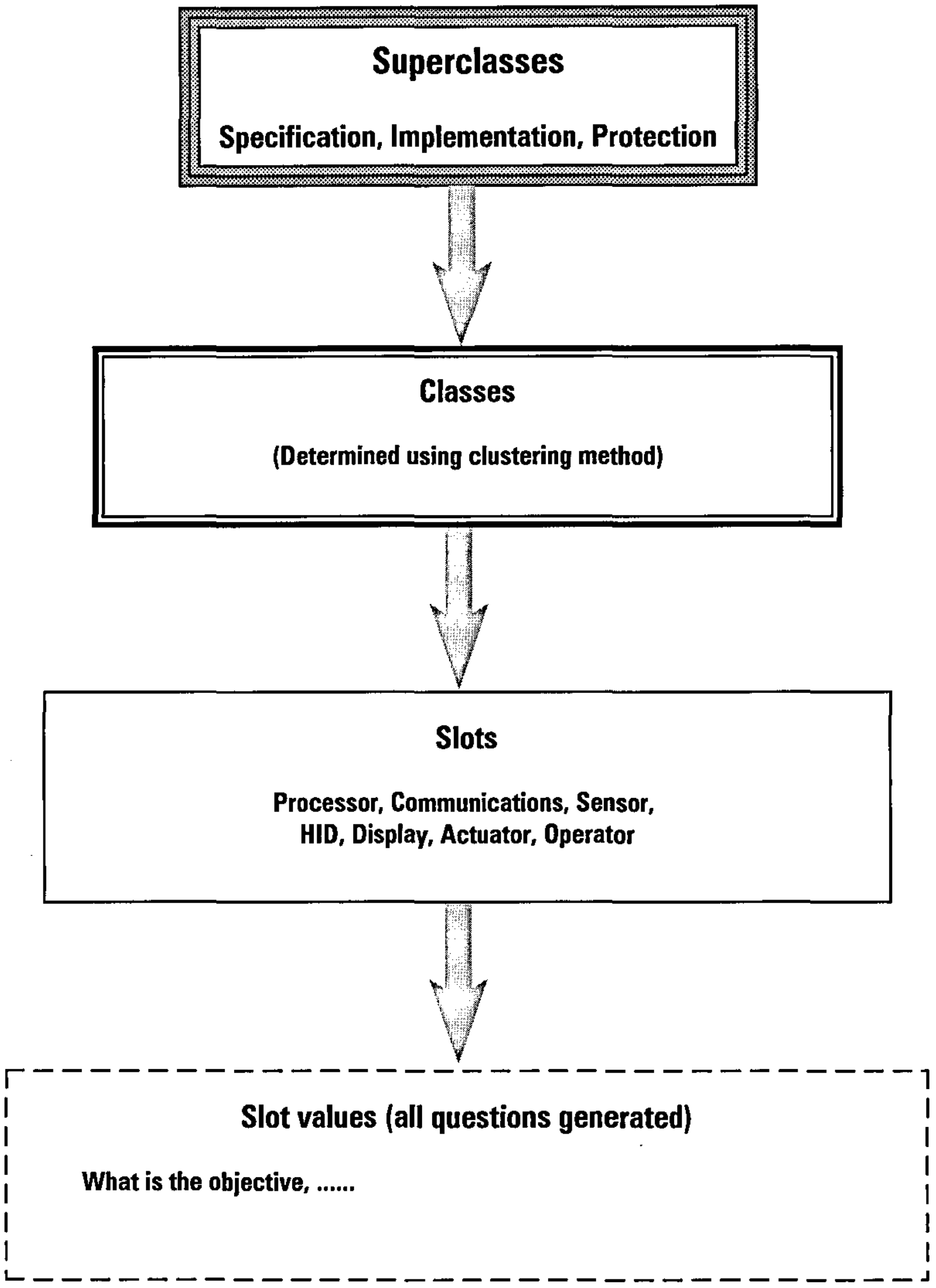

Superclasses
Specification, Implementation, Protection
Classes
(Determined using clustering method)
Slots
Processor, Communications, Sensor, HID, Display, Actuator, Operator
Slot values (all questions generated)
What is the objective, ...

Fig. 3.5: Shows the relationship between the superclasses (Specification, Implementation and Protection) and associated questions. The classes are determined using the clustering method.

Superclass : Specification

Class : Definition - *What is to be achieved?*

Slots	*Values*
Processor	What is the task?
Communications	What communication link is required?
Sensor	What state is to be monitored?
HID	What Human Input Device is required?
Display	What is to be displayed?
Actuator	What action is required?
Operator	What is the operator intervention?

Table 3.4: An example of a superclass (Specification) and one of its classes (Definition), with associated question, slots and slot values

Superclass : Specification

Class : Definition	**What is to be achieved?**
Class : Objective	**Why is it to be achieved?**
Class : Options	**How else could it be achieved?**
Class : Inputs/outputs	**What inputs and/or outputs are required?**
Class : Timing/control	**When is it to be achieved and How is it to be controlled?**
Class : Operational modes	**What operational mode/s (startup, shutdown, automatic, manual etc.) are involved?**
Class : Programmable	**Why is this task programmable?**

Table 3.5a: The superclass, Specification, its classes and associated questions

Superclass : Implementation	
Class : Selection	What device/s are required?
Class : Installation	How will the installation be carried out?
Class : Testing	How will the implementation be tested?
Class : Maintenance	What maintenance procedures are required?
Class : Environment	What effect will the environment have on this task?
Class : Utilities	What utilities (power, air, etc.) are required?

Table 3.5b: The superclass, Implementation, its classes and associated questions

Superclass : Protection	
Class : Failure_Detection	How will any failures be detected?
Class : Interlocks	How are hazardous events prevented?
Class : Trips	How will the system be shut down if a hazard is identified?
Class : Security	How will breaches of security be prevented?
Class : Fault _Recovery	What fault recovery procedures are associated with this task?
Class : Verification	How will the Fail Safe/Protection features be verified?

Table 3.5c: The superclass, Protection, its classes and associated questions

Superclass : Failure_Modes

- Class : Not_Initialized
- Class : Incorrectly_Initialized
- Class : Incorrectly_Executed
- Class : Not_Terminated
- Class : Incorrectly_Terminated
- Class : Erroneous/corrupt_Operation
- Class : No_Input/output
- Class : Incorrect_Input/output
- Class : Lockup
- Class : Too_Fast
- Class : Too_Slow
- Class : Defective_Hardware
- Class : Failure_Not_Detected

Table 3.5d: The superclass, Failure_Modes and its classes

3.7 Conclusions

Incident analysis was carried out in order to develop a framework that provided a unified and coherent method of analysing possible safety-critical failures. The framework derived can be used to drive the process of identifying and assessing safety issues early in the development of a system. The construction of the framework involved the following steps:-

1. Identify common characteristics of incidents by introducing generic events.

2. Develop a modelling technique (the ETD) to represent incident events so that these events and their interrelationships can be analysed.

3. Using the principles of Method Study, derive sets of questions based on the analysis of incidents.

4. Incorporate the resultant questions in a framework thus making use of the experiential knowledge gained.

The ETD modelling technique can be used to unify the causes of incidents and to assist in classifying them. Method Study has been modified for safety assessment. The 'Record' stage is replaced with the ETD modelling technique. The 'Critical Examination' stage of classical Method Study is shown in Table 3.1 and modified Method Study is shown in Table 3.5 (a-d). Both Methods are similar in that there is still a logical sequence of applying questions and moving from the general to the specific. However, the modified Method Study, uses more detailed classification with less emphasis on alternatives.

Parallels can be drawn between classical hazard evaluation procedures (HAZOP & FMEA) and the derived framework. Two new terms are introduced Embedded-System-HAZOP (ES-HAZOP) and Embedded-System-FMEA (ES-FMEA).

✦ ES-HAZOP: ES-HAZOP is based on using classes of the superclasses Specification, Implementation and Protection. Using these classes in conjunction with the ETD technique is similar to using classical HAZOP and an engineering line diagram. The classes are similar to *guidewords* in HAZOP although they are more general because they are used at a higher level of abstraction, i.e. Requirements stage rather than Design stage.

✦ ES-FMEA: ES-FMEA is based on the application of the superclass Failure_Modes. It is more specific than the use of FMEA because failure modes are specified and only investigated for typed events. Also, there is a parallel between the superclass, Failure_Modes and Action Error Analysis (AEA) in that failure modes are specified for operator actions.

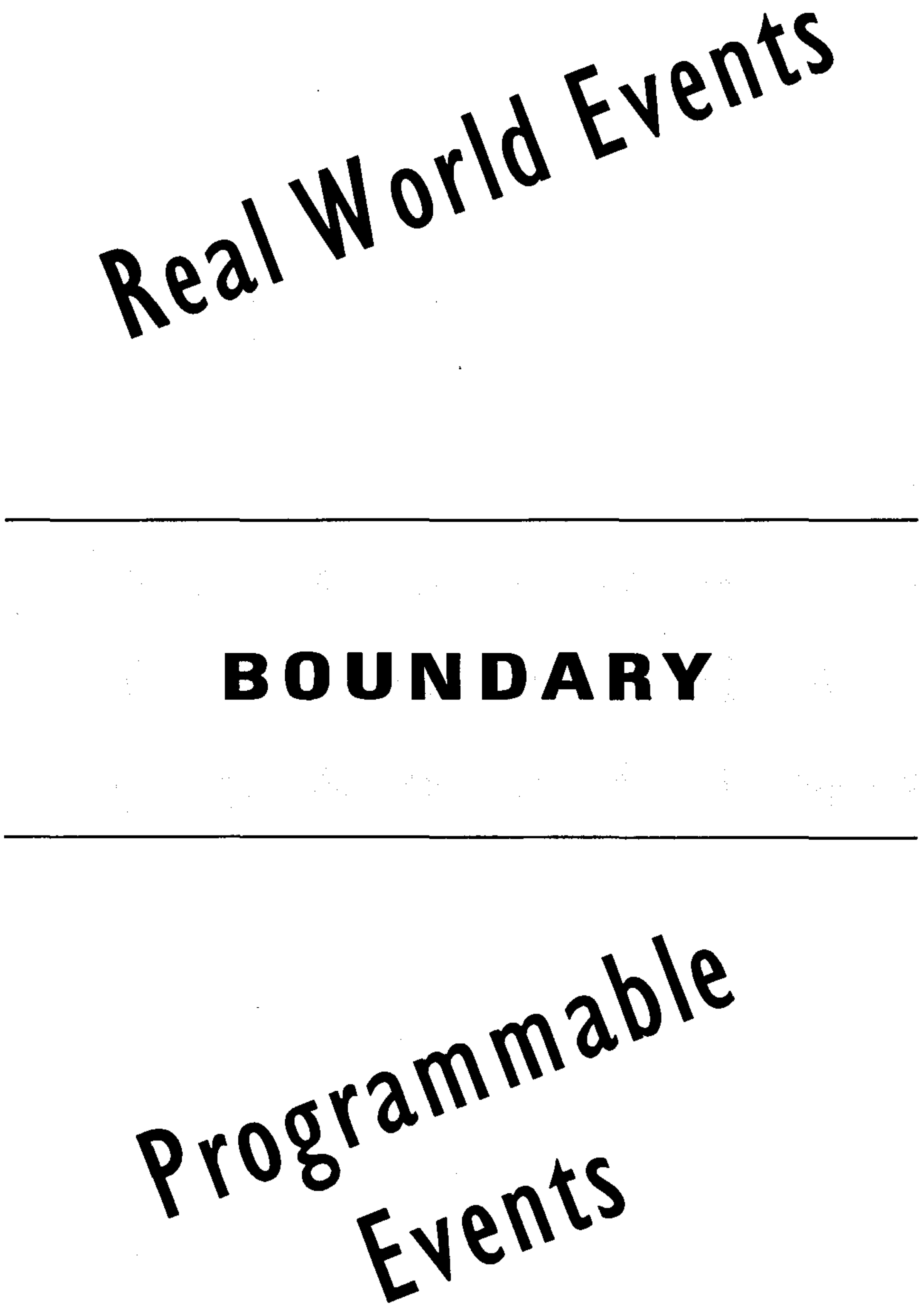
The framework is generic, therefore, it can be used in the development of embedded systems irrespective of the application domain. It focuses on safety issues and is based on 'real world' data.

CHAPTER 4 :
The HAZAPS Methodology

In order to develop a method for 'assessing' (i.e. identifying those aspects of the system related to safety and determining the hazards and contributors to these hazards, with the final aim of eliminating or reducing the risks associated with these hazards) safety-critical systems, the basic principles underlying the system safety programme must be clear. According to MIL-STD-882C [1993] the principal objective of a system safety programme *... is to make sure safety, consistent with mission requirements, is included in technology development and designed into systems, subsystems, equipment, facilities and their interfaces and operation.* It is important to take an overall view and ensure that all entities (human, hardware, software and environmental concerns) are integrated and interact safely. In order to construct safety requirements, the environment should be examined and potential hazards identified and traced back from the environment, through the system interface, to the software. Fig. 4.1 shows the boundary between a generalised embedded system and its operational environment. Events can be classed as 'real world' and 'programmable'. Real world events in the environment can be hazardous; programmable events in the embedded system can contribute to hazards. When developing safety-critical systems, requirements elicitation is very complex since it is necessary both to capture the intended behaviour of systems and to integrate diverse constraints (requiring expertise from different disciplines) based on the intended behaviour. Both a software engineering approach and a safety engineering approach should be used at the requirements phase and all participants should be involved in analysing potential hazards of the proposed system.

The underlying strategy of any hazard analysis technique is based on searching for hazardous events, their causes and consequences. The efficiency of a search strategy depends on how the search is initiated and on how the search mechanism constrains the search space. When analysing programmable systems, the search involves both programmable events and 'real-world' events.

Fig. 4.1: The boundary between a generalised embedded system and its operational environment



ALASKA
USDA
Wildlife Fo
EPSON

There are several approaches to identifying hazards, including:-

- ✦ Starting with a top level hazard and working backwards through all world events, and finally down to programmable events.
- ✦ Starting with a programmable state and working forwards until top level hazards are reached.

Neither of the above approaches is feasible because the search would become intractable, therefore, HAZAPS uses a combination of both approaches, working backwards to the embedded system boundary to identify safety concerns, and working forward from the programmable events to the system boundary. Further division of the searches related to programmable events depends on how the intended behaviour of the proposed system is specified. Whatever combination of search strategies is used for hazard analysis, it is essential that the causes can be traced backwards into the programmable system and that consequences can be traced forward from the programmable system into the environment.

It is important to carry out hazard analysis as early as possible in the requirements phase as it targets where future safety efforts should be focused and it is also easier to incorporate design changes at this stage. The act of carrying out hazard analysis in itself is useful because it gives a different perspective on the system design. The effectiveness of a technique depends on the availability of domain expertise. No technique can guarantee completeness.

4.1 Safety requirements

Once hazards have been identified, it is necessary to construct safety requirements and incorporate them into the system design in order to eliminate or reduce the risk of these hazards. Safety can be viewed as constraints imposed on the functional requirements of the system. Laprie [1993] considers safety (and security) in terms of what *should not* happen as well as what *should* happen, this in turn leads to additional functions that the system *should* fulfil in order to reduce the likelihood of what should not happen. However, determining these 'additional functions' can be difficult, especially if the potential hazards are expressed in a very general way (e.g. 'the solvent used in the coating process is flammable'). Another problem is that the 'additional functions' can conflict with other functions. For example, a safety function may be in conflict with a reliability function or a cost function. A method is therefore required to refine the 'additional functions' and resolve conflicts with other functions.

Whatever method is used, the potential hazards must be eliminated or reduced in risk by constructing implementable safety requirements and these should be as specific and unambiguous as possible for the design stage. The objectives of any method must be:-

- ✦ to trace the potential hazards back through to the environment system interface;
- ✦ to determine the software safety requirements;
- ✦ to construct strategies which can be used to implement these requirements.

McDermid [1994] suggests the general approach of a 'goal strategy' for safety cases where a 'goal' is an objective to be achieved and each 'goal' can be represented in a hierarchical tree structure. Each goal has an associated *context, strategy* and *solution*. De Lemos et al. [1995] analysed the safety requirements of a process control system. The system was partitioned into environment, plant, plant interface, and control system. Safety strategies for the plant were produced for each of the identified potential hazards (a safety strategy was defined as *a scheme to maintain a safety constraint - more specifically, ... a set of conditions imposed on controllable factors over the physical process*). The safety strategies were firstly refined in terms of the plant interface and, secondly in terms of the control system. A safety requirements analysis was then produced and this provided the basis for developing the software. Both of the above approaches use a system model for safety analysis. This system model is based on either a goal strategy or the partitioning of the system into entities. In the next section the HAZAPS system model is discussed.

4.2 HAZAPS system model

The objective of HAZAPS is to assist the developer (at the requirements stage of the software life cycle) to determine potential hazards and assess these hazards. The HAZAPS system model is shown in Fig. 4.2. The methodology can be described as a top-down strategy using multiple levels of abstraction. The four levels of the model are:-

Level 1:	partitioning of the system into safety-critical subsystems.
Level 2:	assigning safety requirements to the subsystem.
Level 3:	implementing the requirements by expressing requirements in the form of generic tasks.
Level 4:	assessing the system by analysing tasks, using predefined criteria.

4.2.1 Derivation of the system model

The derivation of the system model is based on the abstraction and refinement of Level 3 above. This level is based on generic tasks. When analysing incidents (Chapter 3), generic events were used to determine what happened after the system was built. When developing a new system the term 'generic task' is used to express the desired operational behaviour. The modelling techniques and associated assessment framework (described in Chapter 3) developed for generic events is applicable to generic tasks and forms the basis of Levels 3 and 4. Levels 3 and 4 are based on generic knowledge and hence are application-independent. The generic knowledge is related only to the embedded system and not to its target environment. Levels 1 and 2 were introduced as a means of using this generic knowledge when developing a safety-critical system for a specific application. A safety requirement is viewed as a set of instantiated generic tasks and the safety requirements in turn are mapped to a subsystem. In essence, the system model as a whole, allows us to

	Level	Method	
		Software	Safety
LEVEL 1	System ↓ partition into Subsystems	Functional & Object Oriented	Preliminary Hazard Analysis (PHA)
LEVEL 2	Subsystem ↓ assigned Requirements		Fault Tree Analysis (FTA)
LEVEL 3	Requirement ↓ achieved by Tasks	Operational paradigm & Event Time Diagram (ETD)	Task Analysis (TA)
LEVEL 4	Task ↓ assessed by Properties	Scenario Analysis	ES-HAZOP & ES-FMEA

Fig. 4.2: HAZAPS system model and its relationship to safety and software methods

assess the safety of the total system by analysing the behaviour of instantiated generic tasks. Ideally identified generic safety requirements should be reused, however, because safety requirements are so closely interleaved with the system and the environment, this is not possible. So instead, generic tasks are used which can be related to the components of embedded systems in any application.

4.2.2 Application of the system model

The HAZAPS methodology consists of four stages each of which maps to one of the four levels above. It is an iterative and incremental process, in that, development at any subsequent stage may result in changes in previous stages. A number of software modelling and safety engineering methods are used as shown in Fig. 4.2. A case study (described in Chapter 6), based on a rotary screen line printing machine is used to illustrate how the different software and safety methods are integrated in the HAZAPS methodology. In the following sections, the various stages of the HAZAPS methodology are described.

4.3 Stage 1: Identifying safety-critical subsystems

The objective of the first stage is to subdivide the system and to identify the subsystems which are safety-critical. Requirement specifications together with available design schematics are used. Partitioning a system into subsystems is always a difficult problem. It is assumed that some partitioning of a system will already have been carried out before the HAZAPS methodology is applied. Domain analysis is used to assist in identifying and understanding subsystems. Neighbors [1984] defines domain analysis as:-

> *.... an attempt to identify the objects, operations and relationships between what domain experts perceive to be important about the domain.*

This very useful definition indicates the underlying concepts of domain analysis, namely:-

- an identification technique - in HAZAPS, two identification techniques are used for domain analysis, Preliminary Hazard Analysis (PHA) and coarse grain Object Oriented Analysis (OOA);

- domain expertise - experts must be used when building safety-critical systems and it must be possible for them to communicate this knowledge to partners in the system development;
- focusing on a particular aspect - the term 'important' can be replaced by 'safety-critical' where the emphasis is on those properties of the system and environment related to safety.

Domain analysis has the following benefits:-

- it helps to obtain a better understanding of the system and to identify safety concerns;
- it provides a focus for the rest of the safety analysis;
- safety-critical concerns identified in a specific application domain can be reused.

4.3.1 Preliminary Hazard Analysis (PHA)

The purpose of PHA is to identify safety-critical concerns and provide an initial assessment of the hazards. *Ad hoc* methods have been proposed by Kirwan [1994], namely:-

- *By determining the various hazards associated with the intended system's materials inventory (e.g. dangerous chemicals, radioactive substances, etc.).*
- *By reviewing previous incident/accident experience to see what types of incidental/accidental events have occurred.*
- *By using the judgement of an experienced assessor.*
- *By reviewing hazards identified in other similar plant.*

More structured methods for PHA include:-

- Different types of Checklists can be used. A general Checklist is given by Hammer [1980] which contains classes of hazards (e.g. mechanical, explosive) which are further subdivided into more specific hazards (e.g. rotating equipment, explosive gas). AIChemE [1985] describes a number of checklists for use in the chemical industry.

✦ Creative Checklist HAZOP (a variant of classical HAZOP) was developed to address two needs. Firstly, the need for a study that can be carried out earlier in the design, when only a limited amount of information is available and, secondly, the need for a study that can examine adverse interactions between units of the plant, and between the units of the plant and the environment [AIChemE, 1985]. The difference between Creative Checklist HAZOP and classical HAZOP is that, in Creative Checklist HAZOP, (i) the accompanying model used for analysis is in block format (e.g. a process to be carried out) rather than an engineering line diagram or a piping and instrumentation diagram, and (ii) a checklist (e.g. fire, toxicity, radioactivity) is used.

There are other hazard identification techniques, however, the emphasis here is on preliminary analysis, in other words, a first cut at the problem.

4.3.2 Object Oriented Analysis (OOA)

The purpose of this 'coarse' OOA is to identify classes of objects which are related to hazards and hence group them into one or more classes to form a subsystem. Ideally, when identifying subsystems, the objectives are loose coupling between subsystems and strong coupling within subsystems. This emphasis on coupling is similar to that described by Myers [1975] for software modules.

There are several approaches to Object Oriented analysis, however, the emphasis is generally on the analysis of the object model rather than on identifying the objects in the first place. Booch [1991] mentions briefly a number of approaches to identifying classes and objects. Possible sources of object classes include tangible things, roles, events, interactions, structure, devices, locations. Rumbaugh et al. [1991] suggest using the requirement statement from which nouns are abstracted to form tentative classes and subsequently using a refinement procedure to eliminate spurious classes.

4.3.3 Procedure for identifying safety-critical subsystems

1. Draw top level block diagram showing main processes and avoiding too much detail. Briefly describe the processes.
2. Identify top level hazards using prior histories, domain expertise, checklists, regulations, standards, text books. Determine chemical, physical, biological and ergonomic hazards. If necessary, the identified safety processes can then be modelled using Data Flow Diagrams (DFDs) which have been used by others [Edwards, 1993] to model material and energy flows.
3. Coarse OOA is used to refine and abstract information from processes to identify subsystems. Candidate objects are based on devices, materials, events. Scanning the requirement specifications and design schematics and highlighting entities is useful.

In summary, PHA allows us to investigate the safety-related behaviour and OOA allows us to associate entities with this behaviour. Combining both techniques provides a clear definition of subsystems, their boundaries and associated hazards. The results may identify particularly dangerous subsystems and result in their elimination/substitution or in changes to the proposed design requirements. In the next section the construction of safety requirements for the identified safety-critical subsystems is described.

4.4 Stage 2 : Constructing safety requirements

The purpose of this stage is to construct safety requirements for the subsystem based on the associated hazards. Each of the hazards is investigated using a fault tree (an example fault tree is shown in Fig. 4.3) which is used to determine the combination of failures and conditions that could cause the particular hazard to occur. The importance of Fault Tree Analysis (FTA) as a method of safety analysis is illustrated by its extensive use in many industries (chemical, military, avionics, nuclear).

4.4.1 Fault tree construction

Many workers have described the underlying logic of fault trees and how they can be analysed, however, few have described the expertise required to construct fault trees. Fussell et al. [1974] give the following reasons why it is extremely difficult to generate high quality fault trees:-

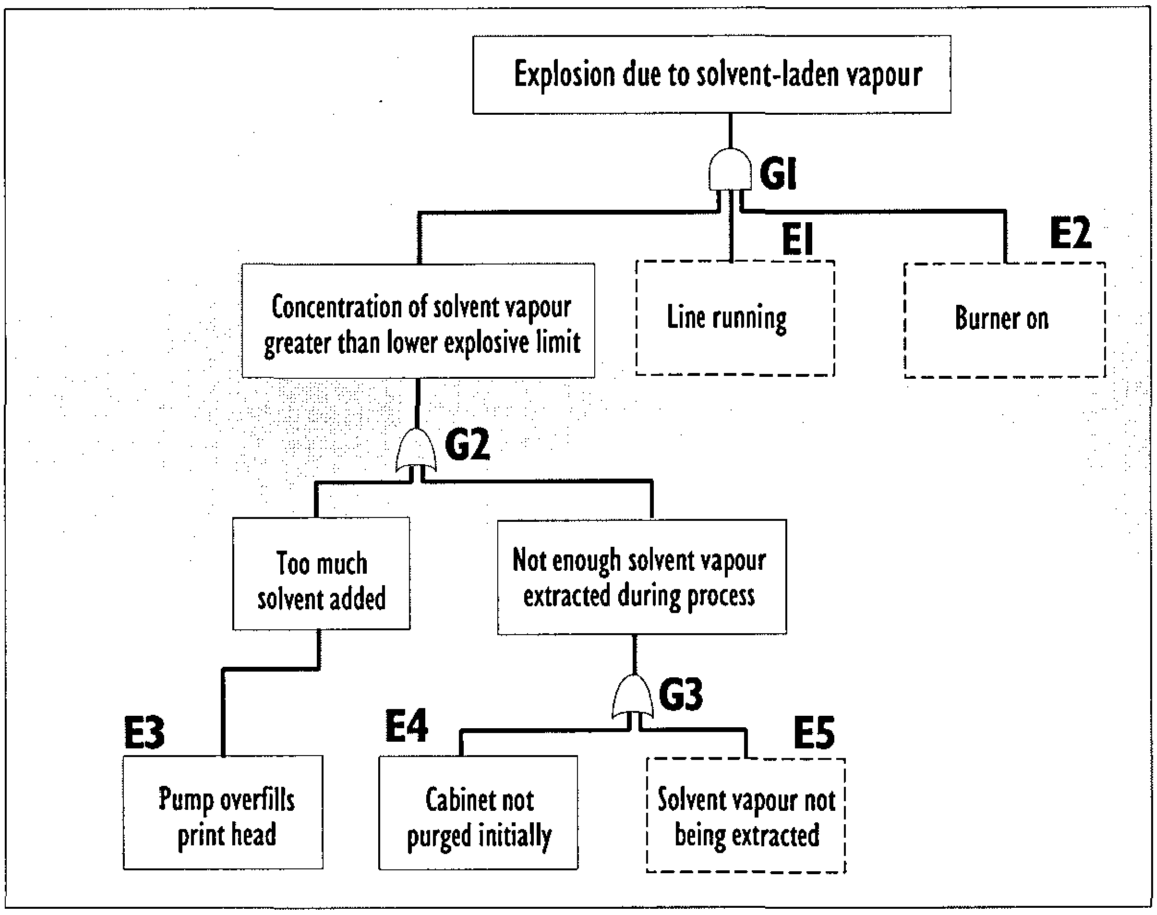

Explosion due to solvent-laden vapour
GI
EI
E2
Concentration of solvent vapour greater than lower explosive limit
Line running
Burner on
G2
Too much solvent added
Not enough solvent vapour extracted during process
G3
E3
E4
E5
Pump overfills print head
Cabinet not purged initially
Solvent vapour not being extracted

Fig: 4.3: Fault tree showing possible causes of an explosion due to solvent-laden vapour in a print station. E1 and E2 are not faults but indicate the state of the subsystem at a given time.

1 The exercise requires a group of analysts who can generate and analyse fault trees.

2 This group of analysts must

(a) be intimately familiar with the system being analysed (involving long sessions with designers and operators of the system in order to understand the functionality of the system);

(b) understand the basic physics, chemistry and economics which describe the system performance;

(c) have sufficient time to analyse the system.

Assuming that the above requirements are fulfilled, the resultant fault tree can provide an excellent logical model of the mechanisms by which a system might fail.

4.4.2 Fault tree analysis

The 'cut set' algorithm is used to identify safety requirements [Fussell & Vesely, 1972]. A 'cut set' is ... *a set of basic events whose occurrence causes the top event to occur*. The basis of the algorithm is a matrix which is used to re-express gates and events in terms of sets of events. The algorithm starts by the selection of the gate at the top of the tree. OR gates result in an increased number of rows and AND gates result in an increased number of elements per row. The algorithm terminates when all gates have been expanded. An illustration of the algorithm is given using Fig. 4.3 where gates are prefixed by G and basic events are prefixed by E.

```
Starting at G1 ⇒
        E1    E2    G2
Expanding G2 ⇒
        E1    E2    E3
        E1    E2    G3
Expanding G3 ⇒
        E1    E2    E3
        E1    E2          E4
        E1    E2                E5
```

Both E1 and E2 occur in all cut sets, therefore, in order to eliminate hazards, either E1 or E2 may be prevented. However, in this case both E1 and E2 are normal operating events therefore E3, E4 and E5 must be negated so predicate statements based on this fault tree may be expressed as

```
E1   AND  E2   AND  (NOT) E3 ............................ (1)
E1   AND  E2   AND  (NOT) E4 ............................ (2)
E1   AND  E2   AND  (NOT) E5 ............................ (3)
```

4.4.3 Procedure for constructing safety requirements

1. Select one of the subsystems identified using domain analysis in the previous stage of HAZAPS.
2. Choose a top event for the fault tree.
3. Identify possible causes in general terms (i.e. independent of system component characteristics).
4. Identify functional failures specific to the plant.
5. Continue to identify sublevels of functional failures until resultant causes can be related to the embedded system interface.
6. Use the cutset algorithm to identify safety requirements.

Stage 2 in Fig. 4.4 shows one of the safety requirements based on predicate statement (3).

The more thorough the domain analysis, the easier it is to carry out this procedure. Requirement documents and design schematics can also be used as a source of information. The objective of the fault tree is not to determine all possible causes but just top level hazards so that they can be investigated in terms of intended behaviour of the system. It can be viewed as full 'breadth' search and a 'depth' search which stops when causes are identified that can be investigated in terms of the interface between environment and the system. There may be conflict between existing functional requirements and the newly created safety requirements as a consequence of which the original requirements may have to be changed. The results from this stage include a set of safety requirements which must be modelled and assessed. This is discussed in the following sections.

Fig. 4.4: SAMPLE OUTPUT FOR HAZAPS

Stage 1

Subsystem

Print Station
- Hazard - Explosion
- Component - Solvent

Stage 2

Safety Requirement

If no solvent vapour is being exhausted from the drying cabinet while the line is running and the burner is switched on, the machine must be shut down and placed in a safe state.
This requirement is represented by the dashed boxes in the Fault Tree (Fig. 4.3).

Stage 3

Operational Tasks

- Flow sensor indicates no exhaust flow
- Line sensor detects line is running
- Burners are switched off
- Nips are forced to open
- Line is kept running
- Alarm is activated
- Warning is shown on Display (The ETD is shown in Fig. 4.5)

Stage 4

Assessment (partial sample)

Specification

Options:

Q: What other way could this task be accomplished?

R: A sensor which directly measures solvent concentration or one which measures airflow

Timing/Control:

Q: How often does this state have to be scanned?

R: Scan time should be less than a second.

Implementation

Maintenance:

Q: What maintenance procedures are required for this task?

R: The condition and operation of the sensor is checked manually on a regular basis because of its importance and the corrosive environment in which it is placed.

Protection

Failure_Detection:

Q: What alarms are associated with this task?

R: A siren and alarm message on a display

Fault_Recovery:

Q: What fault recovery procedures are associated with this task?

R: Manually test operation of exhaust fan and sensor, purge dryer and measure concentration of solvent in dryer.

Failure_Modes

Incorrectly_Terminated:

Q: What if task is incorrectly terminated?

R: If operator presses emergency stop, this would leave coated material in dryer which may ignite. To prevent this, the siren should be unique to this task, indicating that no operator intervention is required.

Too_Slow:

Q: What if signal is too slow

R: If the response of the sensor was too slow, this would cause a build up of solvent vapour in dryer and possible subsequent explosion.

4.5 Stage 3: Transforming safety requirements into operational tasks

The objective of this stage is to transform safety requirements into operational tasks. The basic challenge underlying the development of all software systems is to transform models from requirements to source code. The transformations can be classed as operational or transformational paradigms. From the Requirements Engineering viewpoint, the major difference between the paradigms is the output from the requirements analysis phase; in the case of the operational paradigm [Zave, 1984], the output is an operational specification rather than a requirements specification. The advantages of using an operational specification for embedded software systems are that it can be used:-

- ✦ to examine how the system is supposed to behave at the embedded system boundary;
- ✦ to determine constraints on functional behaviour;
- ✦ as the basis of a prototype which allows different disciplines to analyse the system from their own perspective;
- ✦ to formulate test plans.

As with all requirements engineering techniques, the difficulty in using the operational paradigm is in capturing the requirements in the first place. In contrast, with HAZAPS the safety requirements are already expressed in a logical format from analysis of the fault tree (i.e. output from stage 2 of HAZAPS). Consequently, HAZAPS safety requirements are a useful input for an operational specification. These safety requirements must be decomposed into operational tasks and modelled.

4.5.1 Transforming safety requirements

The safety requirements have to be transformed into operational tasks. Two techniques are used: a technique based on the principles of Task Analysis and requirements parsing.

4.5.1.1 Task analysis

Task Analysis was originally used to identify how people could be trained to perform particular tasks. Task Analysis has been applied to systems after they have been built; to evaluate proposed designs; and, to capture requirements for new systems. Recently, it has been used in what appear to be vastly different areas:-

- the activity of a processor in the development of real time embedded systems [Ward & Mellor, 1985], the focus being on determining data and timing constraints;

- the interaction of the human with the system [Johnson, 1992], the focus being on explicitly investigating the activities and cognitive process of the operator(s).

In HAZAPS both processor activity (i.e. in general terms what the computer must do) and human-computer interaction play a vital part in task synthesis (see Section 3.3.1 for the reasons why it is important to include the human aspect in safety analysis).

Useful prompts for synthesizing tasks have been described by Kirwan & Ainsworth [1992], these are shown in Table 4.1. Once application-specific tasks have been identified, they can be further transformed into sets of generic tasks (processor, communications, sensor, human input device, display, actuator, operator). The origin of these generic tasks is described in Section 3.3.1.

4.5.1.2 Requirements parsing

The requirements parsing technique involves examining each requirement to ensure that it is ... *defined unambiguously by a complete set of attributes (e.g. initiation of an action, source of an action, the action, the object of the action, constraints)* [Peng & Wallace, 1993]. Requirements parsing is generally used as an error detection mechanism in the requirements phase of the life cycle, however, it is useful in HAZAPS as it assists in transforming tasks from the safety requirements derived using fault tree logic.

Table 4.1: A taxonomy of descriptive decomposition categories which have been used in various studies [from Kirwan& Ainsworth, 1992]

Description of task	**Complexity/task complexity**
Description	Task difficulty
Type of activity/behaviour	Task criticality
Task/action verb	Amount of attention required
Function/purpose	**Performance of the task**
Sequence of activity	Performance
Requirements for undertaking task	Time taken/starting time
Initiating cue/event	Required speed
Information	Required accuracy
Skills/training required	Criterion of response adequacy
Personnel requirements/manning	**Other activities**
Hardware features	Subtasks
Location	Communications
Controls used	Co-ordination requirements
Displays used	Concurrent tasks
Critical values	**Outputs from the task**
Job aids required	Output
Nature of the task	Feedback
Actions required	**Consequences/problems**
Decisions required	Likely/typical errors
Responses required	Errors made/problems
	Error consequences
	Adverse conditions/hazards

4.5.2 Modelling safety requirements

Once safety requirements have been expressed in terms of operational tasks, it is necessary to model these tasks. Most methodologies for developing software systems depend on modelling techniques. These modelling techniques can be classified as 'structured', 'functional' or 'behavioural' as shown in Table 4.2.

Table 4.2: Classification of modelling techniques

	Structured	**Functional**	**Behavioural**
Rumbaugh et al. [1991]	Object	Function	Dynamic
Ward & Mellor [1985]	Data	Process	Dynamic
Davis [1993]	Object	Function	States

Ideally all three modelling techniques are used and integrated. Although the ETD technique (derivation described in Section 3.3.2) may appear at first glance to model the behavioural view only (i.e. it is a representation of operational tasks: an example is shown in Fig. 4.5), it is founded on information based on both a structured and a functional view. In Stage 1 in HAZAPS, both a functional (in terms of processes) and a structured (OO analysis) view is used. In Stage 2, the method of applying FTA could be viewed as functional decomposition, the major difference being that in FTA the emphasis is on finding failure 'paths' rather than success 'paths'. The notation of the ETD does integrate these different viewpoints:-

- ✦ nodes are based on objects;
- ✦ the radial dimension reflecting the functional view;
- ✦ the angular dimension is related to dynamics.

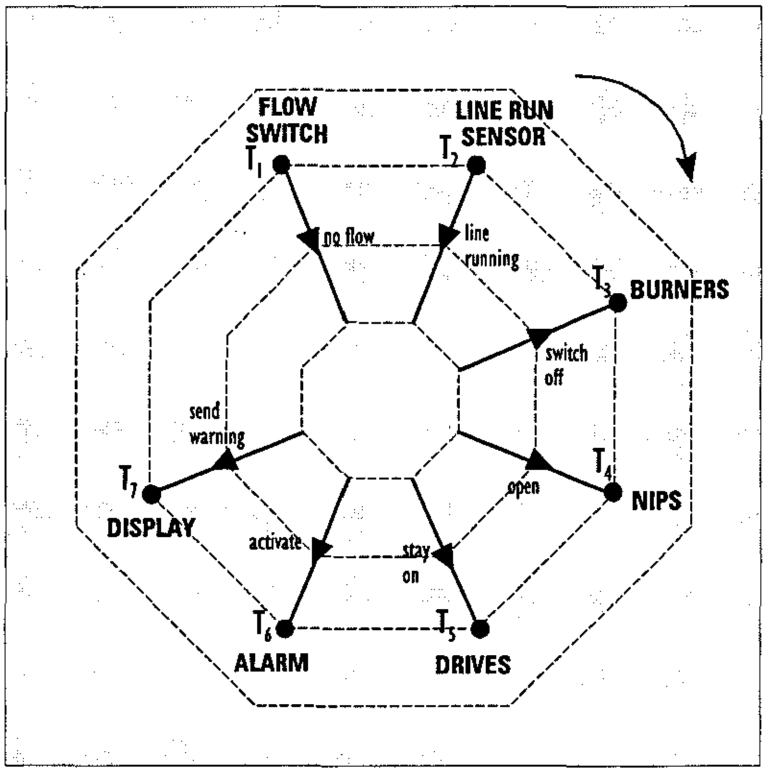


Fig. 4.5: An Event Time Diagram (ETD) for a safety requirement

There is an apparent limitation with the ETD in that the maximum number of tasks that can be drawn on the ETD is eight. However, a major objective of the ETD is to permit analysis of ALL the tasks associated with a specified safety requirement. Even eight tasks can be difficult to interrelate at any one time. Experimental psychologists have suggested that the maximum number of tasks an individual can cope with is 'around seven' [Miller, 1956]. In SADT (see Section 2.2.2.2), a well-established methodology, the maximum number of activities that can be analysed at any one time is six.

4.5.3 Procedure for transforming and modelling safety requirements

1. Select one of the safety requirements identified in Stage 2.

2. Formulate a description of the safety requirement using prompt list shown in Table 4.1, and requirements parsing. The objective is to express safety requirements in terms of actions, timings, post and pre-conditions, devices and attributes.

3. Extract list of generic tasks (processor, communications, sensor, human input device, display, actuator, operator) from description of safety requirement.

4. Model on ETD. This may result in reordering of task list or introduction of new tasks.

Application of this procedure may result in the creation of new safety requirements. In summary, a safety requirement is modelled using the ETD (example shown in Fig. 4.5) by transforming the requirement into a set of generic tasks (see Fig 4.4), i.e. a set of events required to fulfil a safety requirement is a set of instantiated generic tasks.

4.6 Stage 4: Assessing safety requirements

Having identified and modelled safety requirements, the final step is to assess these requirements. The evaluation is driven by scenario analysis using ES-HAZOP and ES-FMEA.

4.6.1 Scenario analysis

The importance of scenario analysis has been highlighted in a number of different areas: human dependability [Atkins, 1990], task analysis [Kirwan & Ainsworth, 1992] and software requirement inspections [Porter et al., 1995]. Scenario analysis has been incorporated in OOA by Jacobson et al. [1993] in their methodology, Object Oriented Software Engineering (OOSE). The term *Use Case* is used in OOSE, a scenario can be described as an instance of a *Use Case*. Scenario analysis is becoming accepted as a vital part of 'front end' OOA as shown by the introduction of *Use Cases* into the new unified methodology of Rumbaugh and Booch [Rumbaugh, 1996]. Gough et al. [1995] give a clear description of the benefits of using scenario analysis during requirement engineering when they state:-

It is not only the requirements engineer, but also the stakeholder, who benefit from scenarios, with an improvement in the communication of ideas, especially in the process of elicitation and validation of requirements.

Scenario analysis is clearly a useful technique but there are problems in applying it. They are as follows:-

- ✦ obtaining in-depth domain knowledge;
- ✦ selecting the right level of abstraction;
- ✦ identifying underlying concepts;
- ✦ generating meaningful scenarios;
- ✦ representing the scenarios;
- ✦ providing an efficient method of analysing scenarios.

These difficulties are particularly pertinent to embedded safety-critical systems because of:-

- ✦ the number of different disciplines involved in the development of a system;
- ✦ the distributed nature of embedded systems;
- ✦ the vast number of possible scenarios;
- ✦ the problem of systematically evaluating each of the scenarios.

4.6.2 Using ES-HAZOP and ES-FMEA to assess safety requirements

Having carried out Stages 1, 2 and 3 of the HAZAPS methodology, subsystems and associated safety concerns have been clearly identified. The safety concerns have been expressed as safety requirements. The safety requirements have subsequently been translated into sets of generic tasks (consisting of a single object and action) which are modelled using the ETD. The ETD is an ideal basis for scenario analysis, where a scenario is based on predicted behaviour of an individual task or the interaction between one or more tasks. It provides a common frame of reference for all disciplines and focuses on the operational behaviour at the embedded system boundary. It overcomes both general and specific difficulties of scenario analysis described above with one exception the ETD does not on its own provide a systematic method of evaluating scenarios. However, the ETD technique can be combined with ES-HAZOP and ES-FMEA to overcome this difficulty. ES-HAZOP is based on the causal superclasses Specification, Implementation and Protection. Each superclass has subclasses, and each subclass has one or more associated questions dependent on task type. ES-FMEA is based on one causal superclass, Failure_Modes. Failure_Modes has subclasses based on general failure modes derived from incidents. The derivation of ES-HAZOP and ES-FMEA is described in Chapter 3.

Scenarios are, by definition, context-dependent which makes them difficult to evaluate in a generalised fashion. ES-HAZOP and ES-FMEA are context-independent but can be used to evaluate application-specific scenarios which are formulated in terms of generic tasks. The questions associated with the ES-HAZOP and ES-FMEA procedures can be applied to different scenarios. The questions:-

- ✦ are dependent on the type of generic task (processor, communications, sensor, human input device, display, actuator, operator);
- ✦ assist in identifying causal chains in behaviour which otherwise would be extremely difficult to identify and analyse;
- ✦ encourage the developers to probe deeply into the proposed design;
- ✦ are classified into causal classes;
- ✦ are arranged in a filter-like fashion (i.e. questions get more specific as the analysis progresses);
- ✦ are based on 'real' incident data.

(A complete list of questions is given in Appendix 1).

4.6.3 Procedure for assessing safety requirements

1. Select safety requirement and associated ETD for evaluation.

2. Identify critical tasks using ETD. It is useful to focus on significant parameters or control of same, e.g. initiating conditions.

3. Apply ES-HAZOP by applying in sequence each of the associated questions (based on task type). Check if subclass prompts to any other questions that might be relevant. Record responses and required actions. Typical actions described include test cases, modifications to requirements, new requirements, requests for further information.

4. Apply ES-FMEA. Follow procedure as for ES-HAZOP. ES-FMEA provides a cross-check of ES-HAZOP and also allows more focused evaluation of the causes and consequences of critical tasks.

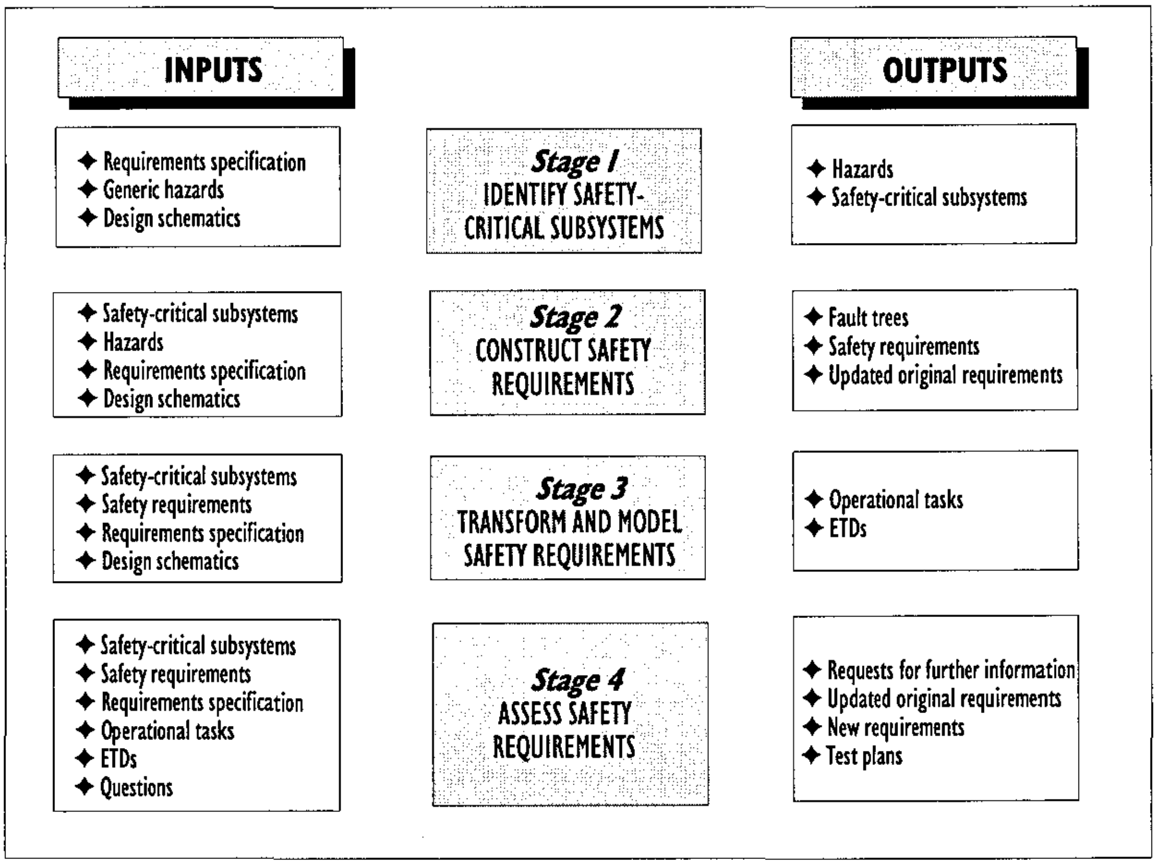
A partial assessment is shown in Fig. 4.4.

4.7 Conclusions

At the end of applying the methodology, the safety-critical subsystems have been identified with their associated safety requirements, hazards, components, operational tasks, ETDs and recommendations. Use is made of the requirements specifications at all stages of the process, this in itself provides a way of checking the functional requirements related to the safety-critical subsystems. It helps to identify any inconsistencies, incompleteness or ambiguities. Fig. 4.6 shows an overall view of the process.

HAZAPS is based on identifying, specifying and assessing safety requirements. It differs from a design or test methodology in that the emphasis is on what should NOT happen rather that what should happen. Top level hazards are analysed in terms of low level tasks of the embedded system. The major challenge in assessing safety in embedded systems is relating real world events that are hazardous to programmable events. The HAZAPS approach is to integrate a number of models, techniques and associated procedures. New models (System and ETD) and new techniques (ES-HAZOP and ES-FMEA) are introduced. Existing techniques (domain, fault tree, task and scenario analyses) have been tailored specifically for assessing safety in embedded systems. Overall the objective was to develop a holistic approach which:-

- captured in a systematic way the different types of knowledge required;
- focused on the interface boundary of the embedded system where major problems are known to occur;
- bridged the gap between the perspectives of different disciplines;
- unified different software and safety methods so that the results can be fed smoothly from one method to the other.



INPUTS
OUTPUTS
✦ Requirements specification ✦ Generic hazards ✦ Design schematics
Stage 1 IDENTIFY SAFETY-CRITICAL SUBSYSTEMS
✦ Hazards ✦ Safety-critical subsystems
✦ Safety-critical subsystems ✦ Hazards ✦ Requirements specification ✦ Design schematics
Stage 2 CONSTRUCT SAFETY REQUIREMENTS
✦ Fault trees ✦ Safety requirements ✦ Updated original requirements
✦ Safety-critical subsystems ✦ Safety requirements ✦ Requirements specification ✦ Design schematics
Stage 3 TRANSFORM AND MODEL SAFETY REQUIREMENTS
✦ Operational tasks ✦ ETDs
✦ Safety-critical subsystems ✦ Safety requirements ✦ Requirements specification ✦ Operational tasks ✦ ETDs ✦ Questions
Stage 4 ASSESS SAFETY REQUIREMENTS
✦ Requests for further information ✦ Updated original requirements ✦ New requirements ✦ Test plans

Fig. 4.6: Overview of the HAZAPS process

CHAPTER 5 :
HAZAPS TOOL

In this chapter the HAZAPS tool is described. It supports some of the key processes involved in the HAZAPS methodology, namely, constructing, modelling and assessing safety requirements. The requirements of the tool are to:

- Assist in identifying safety requirements by providing facilities to view hazards lists, design schematics, fault trees, system and/or software requirements.

- Allow browsing and editing of safety requirements so that task synthesis can be carried out, the objective being to identify the generic tasks required to fulfil these requirements.

- Automatically generate Event Time Diagrams (ETDs) based on identified tasks and provide facilities for subsequent labelling of the ETDs.

- Assist in the assessment of safety-critical tasks by displaying relevant questions for a selected task and allowing responses and required actions associated with a particular task to be stored.

- Provide comprehensive and flexible report generation allowing selection of one or more requirements and display any combination of associated tasks, ETDs, required actions, etc. Reports can be output to display or printer (Postscript file or ASCII text).

- Provide facilities for the system administrator to edit/update the question library and associated information.

Fig. 5.1 shows a conceptual view of the HAZAPS tool. The Knowledge Base contains both generic knowledge (keywords/questions library, incident information and hazards) and

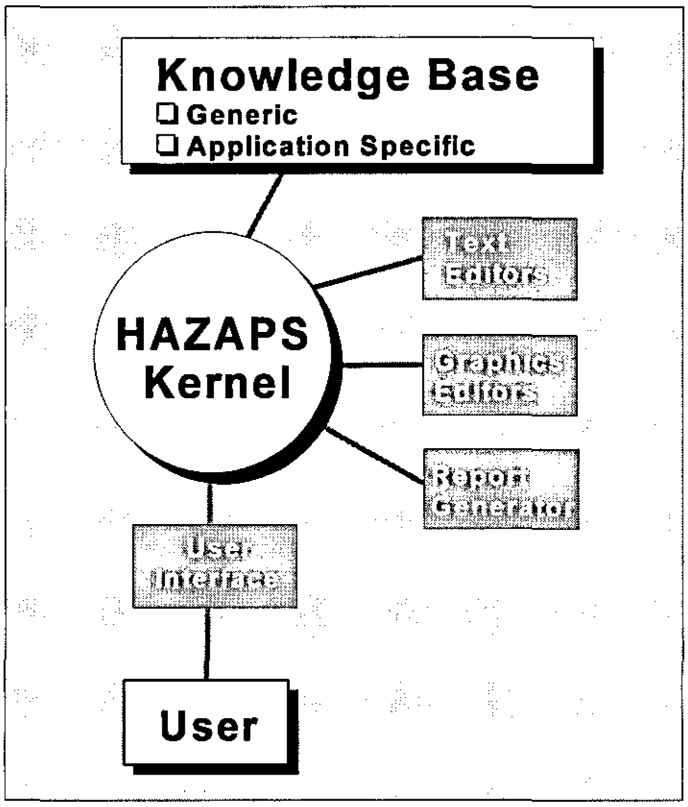


Fig. 5.1: Conceptual view of the HAZAPS tool

application specific knowledge (information specifically related to the ongoing project). The text editors (Requirements, Tasks and Questions) are provided for editing both the generic and the application specific knowledge. Graphics editors are used to model safety requirements via the ETD representation and to view design schematics and fault trees. The report does not exist as an independent entity but is compiled by the Report Generator each time it is invoked from information stored in the Knowledge Base. The HAZAPS Kernel controls the flow of information between the editors and the Knowledge Base. It converts the information from each of the editors into the relevant format and transfers it to the Knowledge Base. The creation and deletion of entities is controlled by the HAZAPS Kernel via the text and graphics editors. The Kernel maintains the integrity of the Knowledge Base by checking the operations made by the editors.

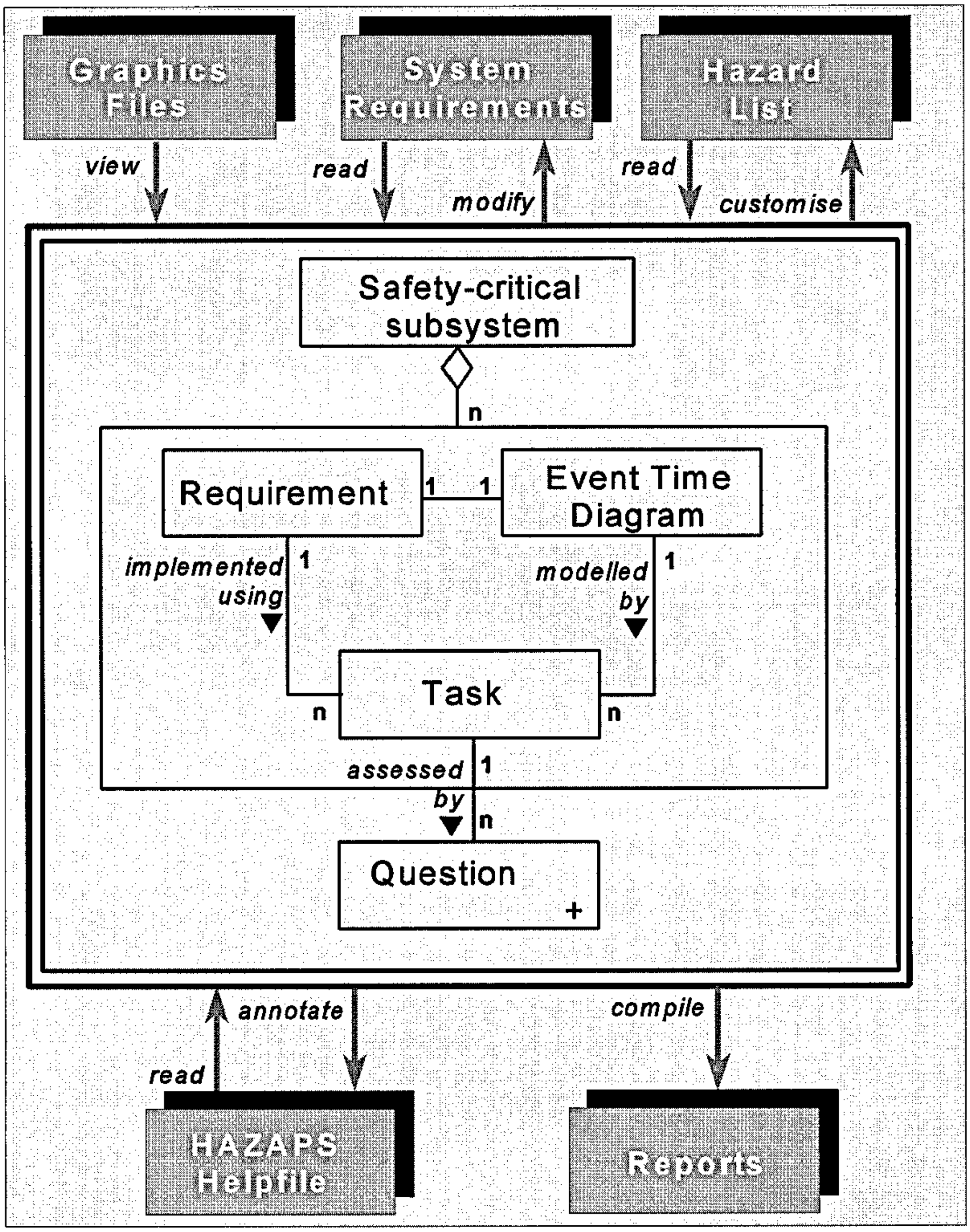


Fig. 5.2: External entities and class diagram for HAZAPS tool

5.1 Design

The strategy used in the development of the HAZAPS tool was based on identifying key abstractions. There are two general types of abstraction (see Fig. 5.2).

1. External file entities, including graphics files (e.g. fault trees and process, mechanical and electrical diagrams); system requirements (e.g. customer requirements, guidelines and standards); hazard lists (e.g. sector, industrial or application-specific lists); and the help file which provides assistance on how to use the tool.

2. Object classes (in doubled bordered box in Fig. 5.2). The notation used follows the new 'Unified Method' of Rumbaugh [1996]. An instance of a safety-critical subsystem is referred to as a 'project'; each project has a number of associated requirements and ETDs; each requirement has an associated ETD and is implemented using a number of tasks; each task is assessed using a number of questions.

The QUESTION class shown in Fig. 5.2 is a superclass; its subclasses are shown in Fig. 5.3. Questions associated with generic tasks are instances of the root classes shown in Fig 5.3. This generalisation of classes allows a user, with no programming experience, to add new questions associated with a question and a generic task. When the system is initialised, a project is selected. Instances of other classes may be created at run-time. At startup, instances of the QUESTION class always exist, however, instances of other classes (i.e. REQUIREMENT, ETD and TASK) do not exist unless they were already created in a previous project and that project is reloaded. Other classes used within the HAZAPS tool (e.g. frames, text boxes etc.) are defined within the development environment. Facilities are provided to read from or write to the slots of these predefined classes and to use their associated methods.

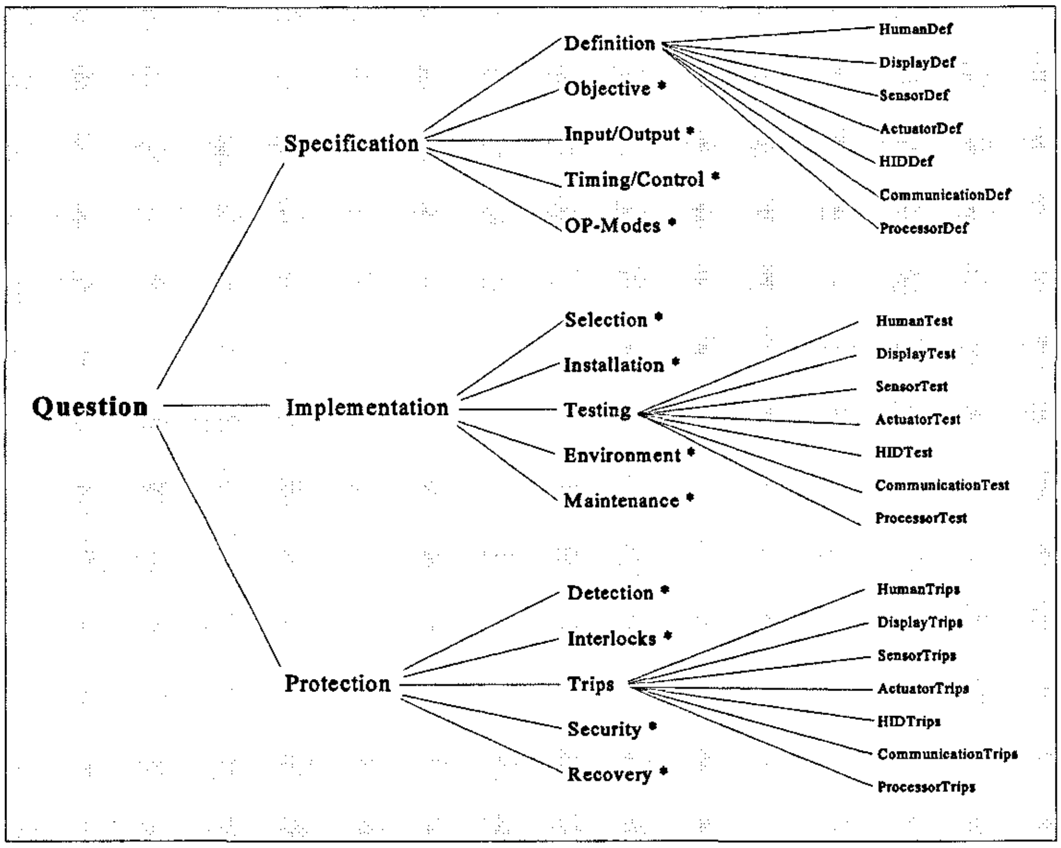


Fig. 5.3: The subclasses of the QUESTION class. * = for simplicity, root classes not displayed

The associations between classes REQUIREMENT, ETD, TASK and QUESTION are implemented using simple naming conventions and pointers.

- The link between a requirement instance and an ETD instance is based on the identifying number of the requirement and ETD e.g. Requirement No.X is associated with ETD No.X.

- The association between a requirement instance and task instances is established by mapping every instance of a requirement to a task class, consequently, all instances of a TASK class are associated with a particular requirement instance.

- The link between a task instance and question instances is implemented by using slots within the task instance. These slots contain pointers to associated question instances.

The templates for each of the classes developed for the HAZAPS tool are shown in Fig. 5.4. The notation used follows the new 'Unified Method' of Rumbaugh [1996]. The template consists of three compartments: (1) name of class, (2) slots, and (3) operations.

REQUIREMENT
description
CreateRequirement
DeleteRequirement
DisplayReqDescription
UpdateReqDescription

QUESTION
question_description
question_info
CreateQuestion
DeleteQuestion
DisplayQuestionDes
DisplayQuestionInfo
UpdateQuestionDes
UpdateQuestionInfo

TASK
description
type
specification_questions
implementation_questions
protection_questions
specification_responses
implementation_responses
protection_responses
specification_actions
implementation_actions
protection_actions
CreateTask
DeleteTask
DisplayTaskDescription
DisplayTaskType
DisplayAssocQuestions
DisplayAssocResponses
DisplayAssocActions
UpdateTaskDescription
UpdateTaskType
UpdateAssocQuestions
UpdateAssocResponses
UpdateAssocActions
RespondDeleteReq

ETD
node_label
arrow_label
node_coords
line_coords
arrow_coords
flow_direction
CreateETD
DrawETD
DisplayLabels
RespondDeleteReq
RespondEditTask
RespondUpdateType

KEY

Name of Class
Slots
Operations

Fig. 5.4: Templates for each of the classes developed for the HAZAPS tool

5.1.1 Slots

Each class contains slot(s) for free text. The slot for instances of the REQUIREMENT and TASK classes contains a description of the requirement or task. The ETD class has slots *node_label* and *arrow_label* containing strings that correspond to task instances. Each instance of a QUESTION class has slots *question_description* (containing the question itself) and *question_info* (containing information associated with the question).

The *type* slot of the TASK class can have one of seven values (Human, Display, Sensor, Actuator, HID, Communication, Processor). This determines how the task will be modelled and assessed. Following the *type* slot, there are nine slots relating a task instance to question instances. The first three of these nine slots contain lists of pointers to groups of question instances. The next three slots are for responses to questions, and the final three slots are for actions associated with the relevant questions.

The ETD template has slots, *node_coords*, *line_coords* and *arrow_coords*, containing Real numbers which are used to draw the ETD automatically. The *flow_direction* slot can only have a value of 1 (indicating flow to the processor) or 0 (indicating flow from the processor).

5.1.2 Operations

Common operations for all classes include Create, Delete, Display and Update. The Create and Delete operations refer to the creation and deletion of instances, the Display and Update operations refer to reading from, or writing to, slots.

The operations *UpdateAssocQuestions*, *UpdateAssocResponses*, and *UpdateAssocActions* of the TASK class depend on the value of the *type* slot and on the group (i.e. specification, implementation or protection) selected by the user. If a requirement is deleted, all its associated tasks must also be deleted. This is handled by the operation *RespondDeleteReq*.

The operations *DrawETD* and *DisplayLabels* of the ETD class are used for displaying an ETD on the screen. The process is as follows:-

- ✦ drawing the node is dependent on the task-identifying number and the task-type (using the *type* slot from the task instance);
- ✦ drawing a line from the node;
- ✦ drawing an arrow on the line dependent on flow direction;
- ✦ labelling the node;
- ✦ labelling the arrow.

The last three operations in the ETD template are used to respond to situations where a requirement is deleted, a task is deleted/inserted or a task type is changed. These operations maintain consistency of the ETD model for both the requirement instance and the task instances. For example, if task No.X is deleted, all other tasks shown on the ETD whose task identifying number is greater than X must be moved back 40°.

5.2 Implementation

The tool was developed for a Windows environment using an expert system shell, CLIPS [NASA, 1994] and a graphical user interface, wxCLIPS [Smart, 1996]. CLIPS provides procedural and object oriented facilities and wxCLIPS supports the creation of window frames, menus, textboxes etc. All code written in CLIPS is supported by wxCLIPS. Other software used includes HlpMATIC [Ghag, 1994] to generate Windows help files and AZ Icon Edit [AZIcon, 1994] to create Windows icons. The combination of packages provides an excellent prototyping environment.

The approach used in developing the software was based on the levels of abstraction as described by Dijkstra [1968] and expanded by Myers [1976]. The main benefits of layering are clarity, and ease of maintenance, testing and consistency checking. For the HAZAPS tool the idea of layering was combined with object oriented classes. The module is the basic mechanism for organising the software in the HAZAPS tool. The modules are divided into two types - 'declarative' and 'procedural'. The declarative module is used to declare classes and contains no procedural statements. The procedural modules contain no class structure but control the execution of the program. Fig. 5.5 is a 'module-dependency' diagram for the HAZAPS tool. The boxes represent modules (boxes representing procedural modules have an upper compartment giving the name of the module and a lower compartment showing how the module can interact with the different classes). The diagram shows the different levels:

Level 1 (Base Level) contains the declarative modules associated with the classes

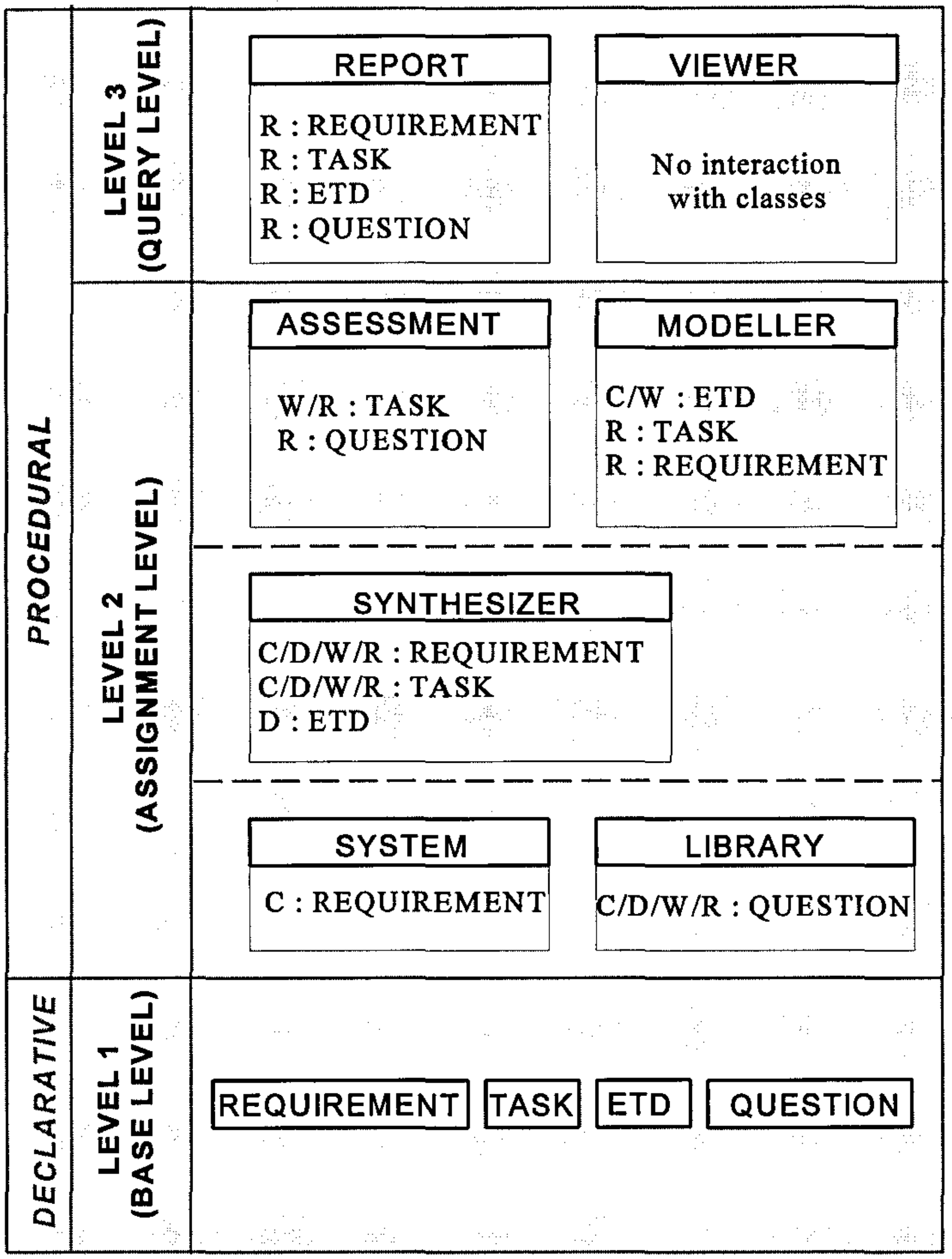
Level 2 (Assignment Level) includes modules for the construction, modelling and assessment of safety requirements. Level 2 has three Sub-levels:

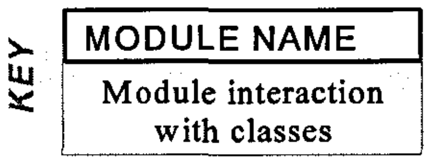
- ✦ **SYSTEM** (for capturing safety requirements) and **LIBRARY** (for editing the question library) modules;
- ✦ **SYNTHESIZER** module for synthesising generic tasks;
- ✦ **MODELLER** module (for modelling and identifying safety critical tasks) and **ASSESSMENT** module (for assessing the safety of these tasks).

Level 3 (Query Level) includes the **REPORT** module and the **VIEWER** module for viewing designs and fault trees.

A module consists of

- ✦ Declarations - all variables used in the module;
- ✦ Functions - simple building blocks and operations e.g. deleting all instances of a class;
- ✦ Specification of frame and associated devices e.g. defining size of windows, where text boxes, or menus are placed;
- ✦ Procedures - all other components associated with the module e.g. responding to events such as menu selection, combining functions and interface calls to other modules.





C : create instance of class
D : delete instance of class
W : modify slot content of instance
R : read slot content of instance

Fig. 5.5: Module dependency diagram

As the prototype was being developed the following useful guidelines evolved:-

- Changes in higher level modules should not affect lower level modules;
- Deletion of instances should be performed on one level/sublevel only;
- Creation of instances should be performed on one level/sublevel only (with one exception - for practical reasons, creation of requirements instances is allowed both in the **SYSTEM** and **SYNTHESIZER** modules);
- Where possible, avoid the changing of slot values at more than one level/sublevel;
- Place no restriction on reading information from slots from any level.

5.3 Operation

wxCLIPS [Smart, 1996] provides facilities to create frames, each with an optional menu bar. Within a frame, one or more subwindows can be created. Subwindows can be panels, text subwindows, or canvasses, where,

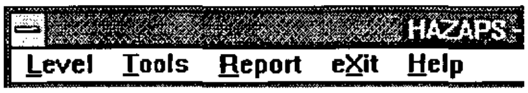
- Panels contain such items as buttons, choice boxes, list boxes, text boxes;
- Text subwindows are used for displaying and editing text files;
- Canvasses are used for drawing graphics.

Seven frames (Figs. 5.6 to 5.12) facilitate user interaction with the tool. Each frame maps to one of the seven procedural modules shown in Fig. 5.5. Calling of the modules is controlled by the user via these frames. The four *main frames* , their associated modules and their mapping to the stages of the HAZAPS process, are shown in Table 5.1. Frame names were chosen to be meaningful to the user; module names were chosen to be consistent with the overall design and development of the tool. The three *ancillary frames* do not map specifically to any stage in the HAZAPS methodology (see Table 5.2). The **Report Generator** and **Graphics Viewer** may be used at any stage in the process, and the **Library Editor** frame is for use by the administrator for maintenance purposes only.

Table 5.1: Mapping of modules and *main frames* to HAZAPS stages

Methodology	Tool: MODULE	Tool: Frame
Stage 1 + *2*: Identification of safety-critical subsystems and construction of safety requirements	**SYSTEM**	**System Level**
Stage 3: Transforming and modelling of safety requirements	**SYNTHESIZER** **MODELLER**	**Requirement level** **ETD Editor**
Stage 4: Assessment of safety requirements	**ASSESSMENT**	**Task Level**

Frame design provides a commonality of code for the designer and a clear and simple presentation for the user. Most frame menu bars have common features



Level	Allows transfer between frames (e.g. move from **Requirement Level** to **Task Level**).
Tools	Includes facilities to call a hazards list (in help file format) or **Graphics Viewer.**
Report	Provides access to **Report Generator.** Available on all *main frames*.
Help	Calls the help file. Available on all *main frames*.

Table 5.2: ***Ancillary frames*** **and their associated modules**

Frame	MODULE
Graphics Viewer	**VIEWER**
Report Generator	**REPORT**
Library Editor	**LIBRARY**

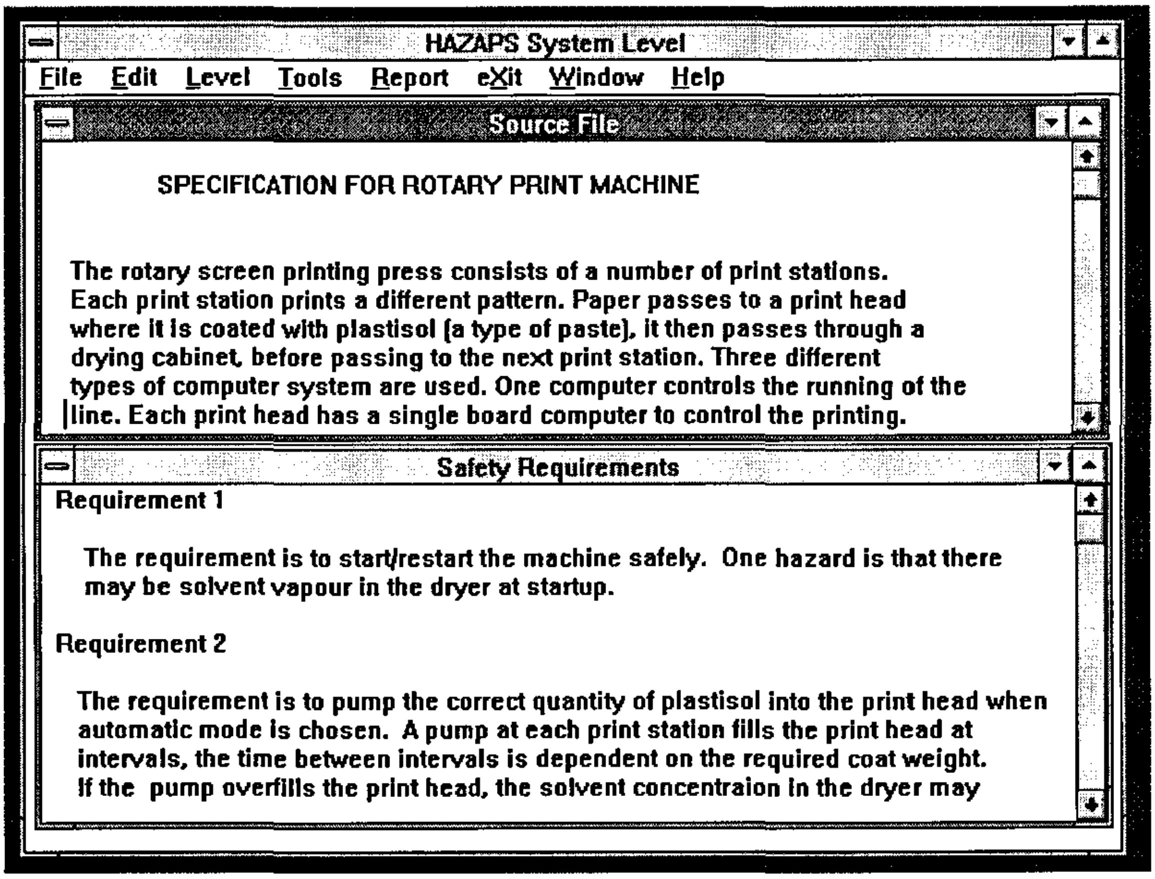
5.3.1 System Level

The **System Level** Frame, used to identify safety requirements (see Fig. 5.6), consists of two text subwindows: an upper window for loading and displaying the *Source File* (an ASCII text file), and, a lower window for entering safety requirements. The procedure is as follows:

System Level Icon

1. If starting a new project load the *Source File* (options include appending to an existing *Source File* or clearing the *Source File* subwindow). If an assessment has already been carried out then data can be reloaded, edited and saved and both the *Source File* and the safety requirements displayed in their respective subwindows.

2. Capture and identify safety concerns by scrolling through the *Source File*. Assistance is provided via the Tools menu in the form of a Hazards List and **Graphics Viewer** (to view design schematics or fault trees).



SPECIFICATION FOR ROTARY PRINT MACHINE

The rotary screen printing press consists of a number of print stations. Each print station prints a different pattern. Paper passes to a print head where it is coated with plastisol [a type of paste], it then passes through a drying cabinet, before passing to the next print station. Three different types of computer system are used. One computer controls the running of the line. Each print head has a single board computer to control the printing.

Requirement 1

The requirement is to start/restart the machine safely. One hazard is that there may be solvent vapour in the dryer at startup.

Requirement 2

The requirement is to pump the correct quantity of plastisol into the print head when automatic mode is chosen. A pump at each print station fills the print head at intervals, the time between intervals is dependent on the required coat weight. If the pump overfills the print head, the solvent concentraion in the dryer may

Fig. 5.6 System Level Frame

3. Candidate safety requirements are entered into the lower subwindow by typing or pasting. The word 'Requirement' must be included as a header to each requirement.

4. Once all safety requirements have been identified, they can be edited and rearranged in a logical order.

5. Store all Requirements (i.e. create instances of the REQUIREMENTS class) by clicking on the File menu and selecting Process. Once Process has been selected, Requirements can no longer be edited at this Level but should be edited individually at the **Requirement Level.**

5.3.2 *Requirement Level*

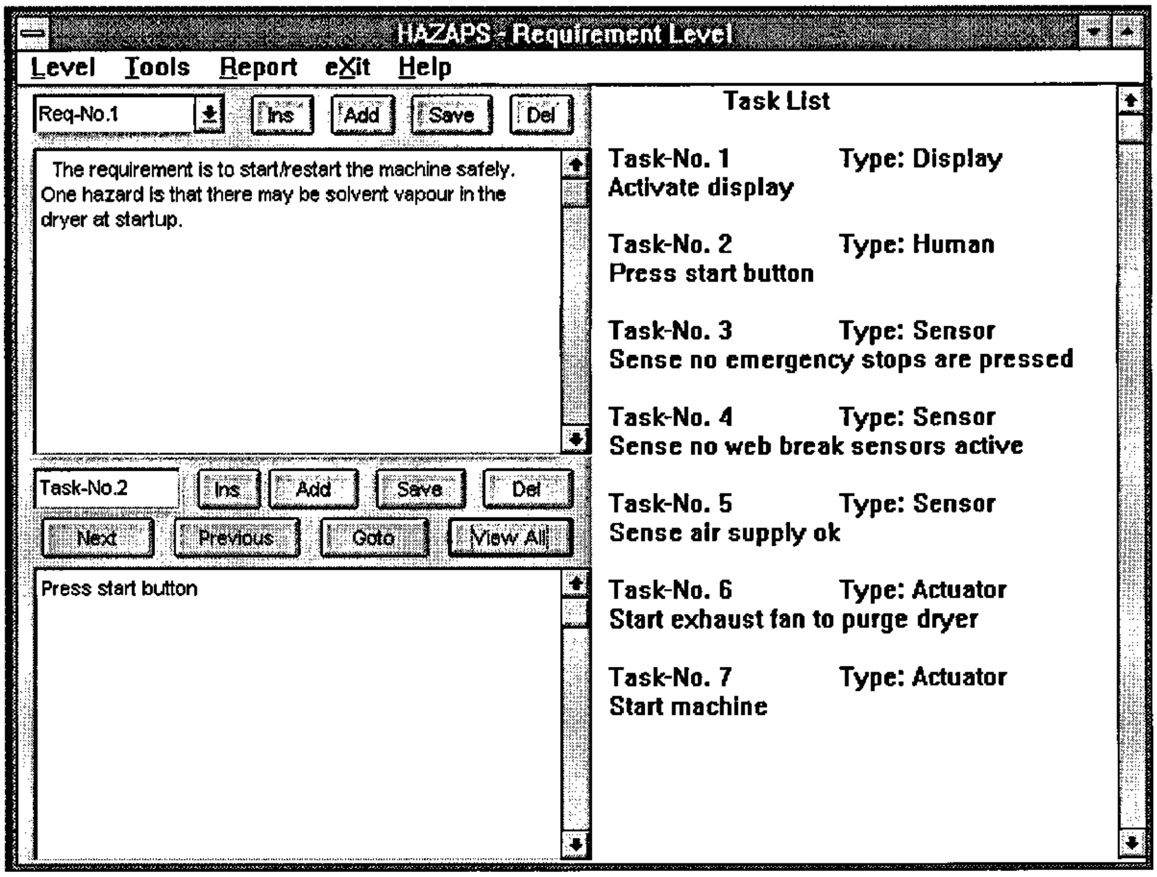
The objective of this frame is to assist in synthesising generic tasks to fulfil the safety requirements (see Fig. 5.7). There are three subwindows made up of two panels and one text subwindow. One panel displays the description of the selected requirement, the other displays the description of the selected task associated with the chosen requirement. The text subwindow shows a list of tasks associated with the chosen requirement. The procedure is as follows.

Requirement Level Icon

1. Select requirement for which tasks need to be constructed. Options are provided to a) insert a new requirement at the highlighted requirement position, b) append a new requirement to the existing list, c) edit a requirement description, and d) delete a requirement (i.e. delete a requirement instance).

2. Select task and enter description of task in lower left hand panel. Options are available to insert, append, edit and delete a task. Other options include a) move forward to next task, b) move backward to previous task, c) go to a particular task, and d) view all tasks associated with the highlighted requirement.

3. Store the task and its description (i.e. create task instance and fill its description slot) by pressing the Save button. This pulls up a choice box with seven options (Human, Display, Sensor, Actuator, HID, Communication, Processor). Select an appropriate type for the task.


HAZAPS - Requirement Level
Level Tools Report eXit Help
Req-No.1
Ins Add Save Del
Task-No.2
Ins Add Save Del
Next Previous Goto View All

The requirement is to start/restart the machine safely. One hazard is that there may be solvent vapour in the dryer at startup.

Press start button

Task List

Task-No. 1 Type: Display
Activate display

Task-No. 2 Type: Human
Press start button

Task-No. 3 Type: Sensor
Sense no emergency stops are pressed

Task-No. 4 Type: Sensor
Sense no web break sensors active

Task-No. 5 Type: Sensor
Sense air supply ok

Task-No. 6 Type: Actuator
Start exhaust fan to purge dryer

Task-No. 7 Type: Actuator
Start machine

Fig. 5.7 Requirement Level Frame

4. Repeat steps 2 and 3 until all tasks associated with the chosen requirement have been determined.

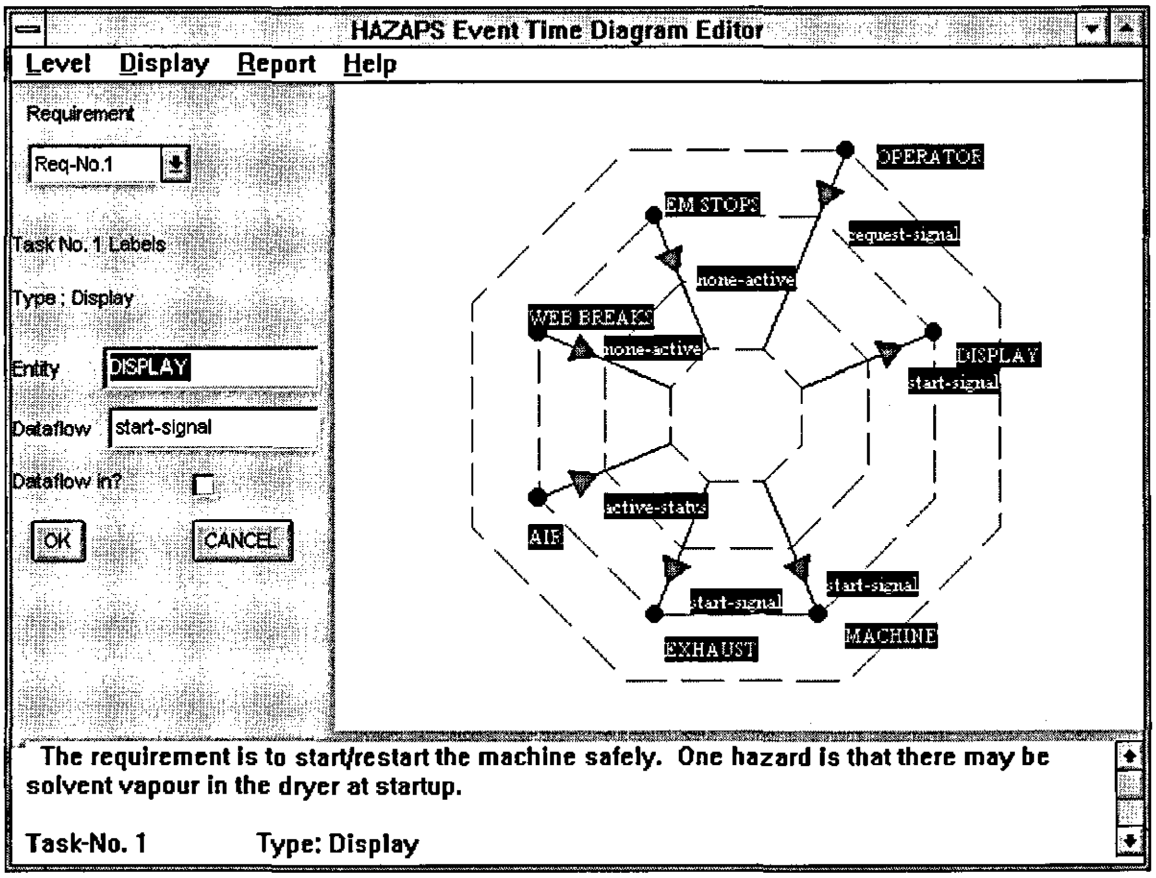
5. Repeat steps 1 to 4 until all requirements have analysed.

5.3.3 Event Time Diagram (ETD) Editor

The purpose of the **ETD Editor** is to model safety requirements (see Fig. 5.8). It has three subwindows; a canvas for drawing the ETD, a dialog subwindow for selection of requirement and labelling entities of associated tasks, and, a text subwindow at the bottom of the display for browsing the requirement and tasks descriptions. The procedure is as follows:

ETD Editor Icon

1. Select Requirement from Choice Box. Requirement description and associated task appear in text subwindow at the bottom of the display.



Requirement: Req-No.1

Task No. 1 Labels

Type : Display

Entity: DISPLAY

Dataflow: start-signal

Dataflow in? ☐

The requirement is to start/restart the machine safely. One hazard is that there may be solvent vapour in the dryer at startup.

Task-No. 1 Type: Display

Fig. 5.8: ETD Frame (n.b. time increases anti-clockwise)

2. Select a specific task by clicking on the mouse-sensitive canvas at one of the vertices of the ETD template (i.e. select a particular task to be modelled). This action pulls up a dialog subwindow on the left of the canvas.

3. Provide the requested information in the dialog subwindow. The text subwindow is available to browse through relevant task information. When all information has been entered, exit the dialog subwindow by clicking OK. Subsequently the task vector, node, direction arrow and associated text are displayed on the ETD template.

4. Repeat Steps 2 & 3 until all tasks are displayed on the ETD template. To maintain consistency between task generation and modelling, tasks cannot be deleted at this level: this must be done at the **Requirement level.** Labelling and direction of flow may be edited at this level, the display may be tidied up by using the Refresh facility which is available via Display on the menu bar.

5. Repeat Steps 1 to 4 until all requirements have been modelled.

5.3.4 Task Level

At the **Task Level** safety critical tasks associated with requirements are assessed (see Fig. 5.9). It consists of three choice boxes and four text boxes. The choice boxes are used for selection of Requirement, Task and Group. Two text boxes are used to display information for the user, one displays the description of the selected task, the other displays relevant questions associated with the task. Two other text boxes are provided for the user to input information, one to respond to questions, the other to enable the user to specify further actions (e.g. requests for further information and requirement changes). The procedure is as follows.

Task Level Icon

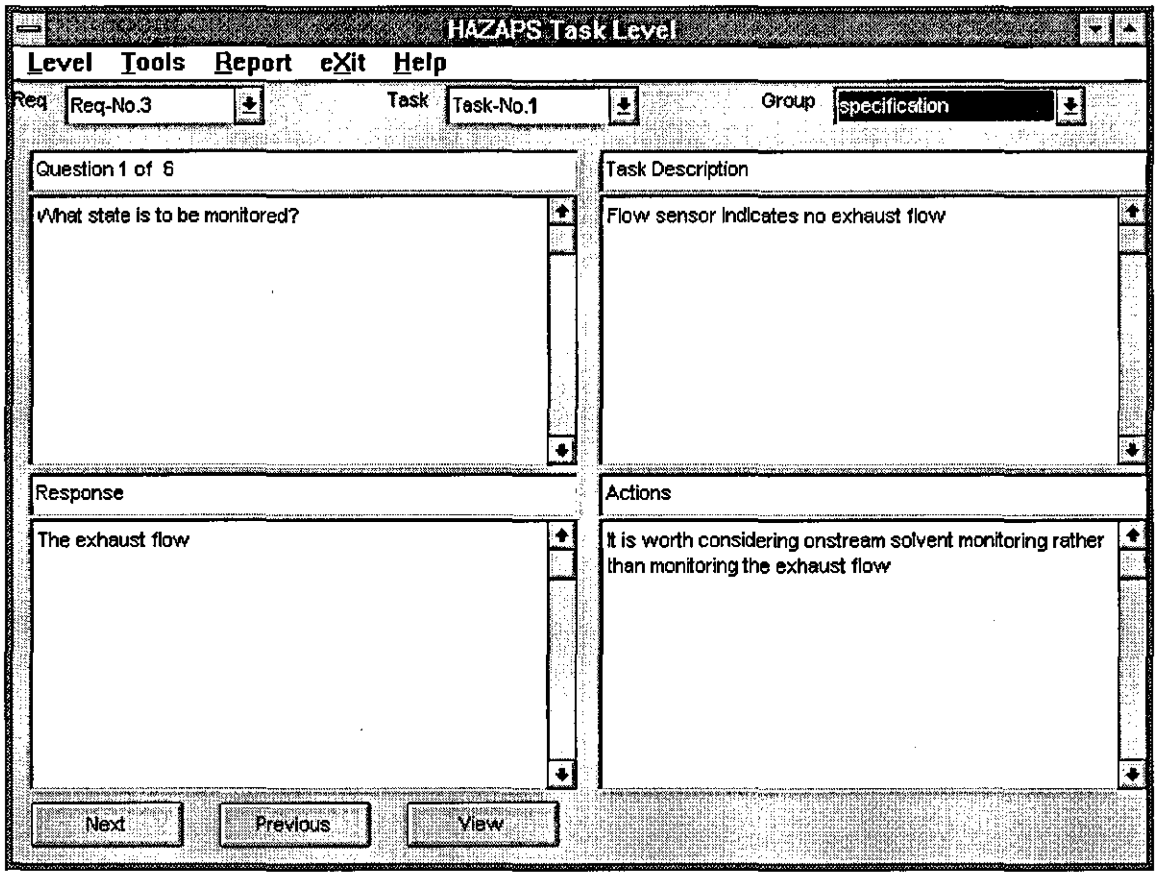
1. Select a requirement. This action populates the task Choice Box with tasks associated with the highlighted requirement.

2. Select a task. The task description is displayed in the top right text box and the Group Choice Box is initialised.

3. Select a group (i.e. Specification, Implementation or Protection). The question Text Box at the top left hand side displays the number of relevant questions available and the first question. Further information can be obtained about the relevant questions by pressing the View button. This action invokes the Library Browser which displays the list of questions under consideration and associated information.

4. Enter a response and actions in the two lower text boxes. Save responses and actions by pressing the Next button which automatically displays the next question if one exists.

5. Repeat steps 3 and 4 for each Group.


HAZAPS Task Level

Level Tools Report eXit Help

Req: Req-No.3 Task: Task-No.1 Group: specification

Question 1 of 6

What state is to be monitored?

Task Description

Flow sensor indicates no exhaust flow

Response

The exhaust flow

Actions

It is worth considering onstream solvent monitoring rather than monitoring the exhaust flow

Next Previous View

Fig. 5.9 Task Level Frame

6. Repeat Steps 2 to 5 for each critical Task associated with the selected Requirement.

7. Repeat Steps 1 to 6 for all Requirements.

5.3.5 Graphics Viewer

The **Graphics Viewer** is for viewing design schematics and fault trees (see Fig. 5.10) It consists of one canvas on which a graphic can be displayed. The procedure is as follows:-

Graphics Viewer Icon

1. Click on the File menu. Select Load file. A file selector subwindow displays all files with .BMP extensions in the default directory.

2. Choose a file to load. The graphic is displayed on the canvas.

3. To replace the file displayed with a different file select Clear from the File menu and repeat steps 1 and 2.

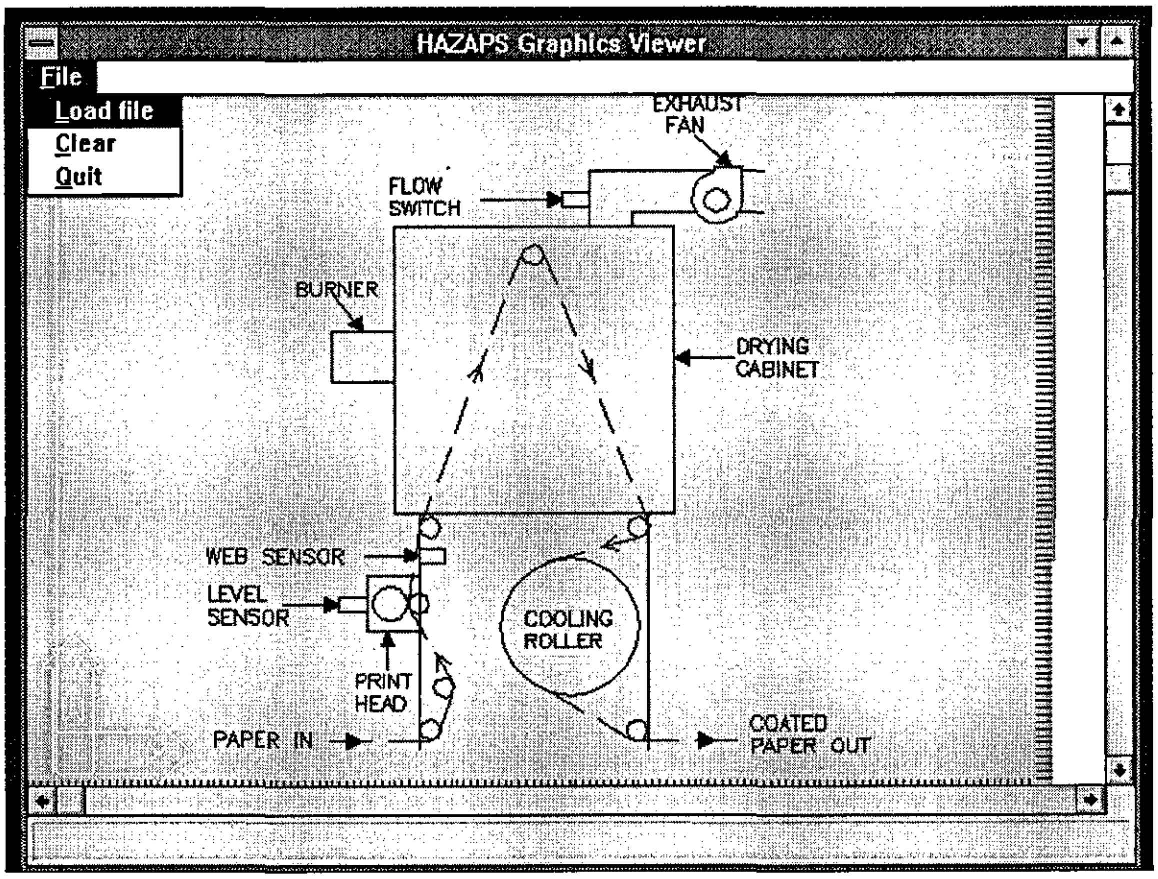


Fig. 5.10: Graphics Viewer Frame

4. To exit the **Graphics Viewer** and return to previous frame select Quit from the File menu.

5.3.6 Library Editor

The **Library Editor** allows customisation of questions and associated information used for assessing critical tasks (see Fig. 5.11). The frame includes a List Box and two Text Boxes. The List Box displays existing questions. The two Text Boxes are provided for editing questions and associated information. The procedure is as follows:

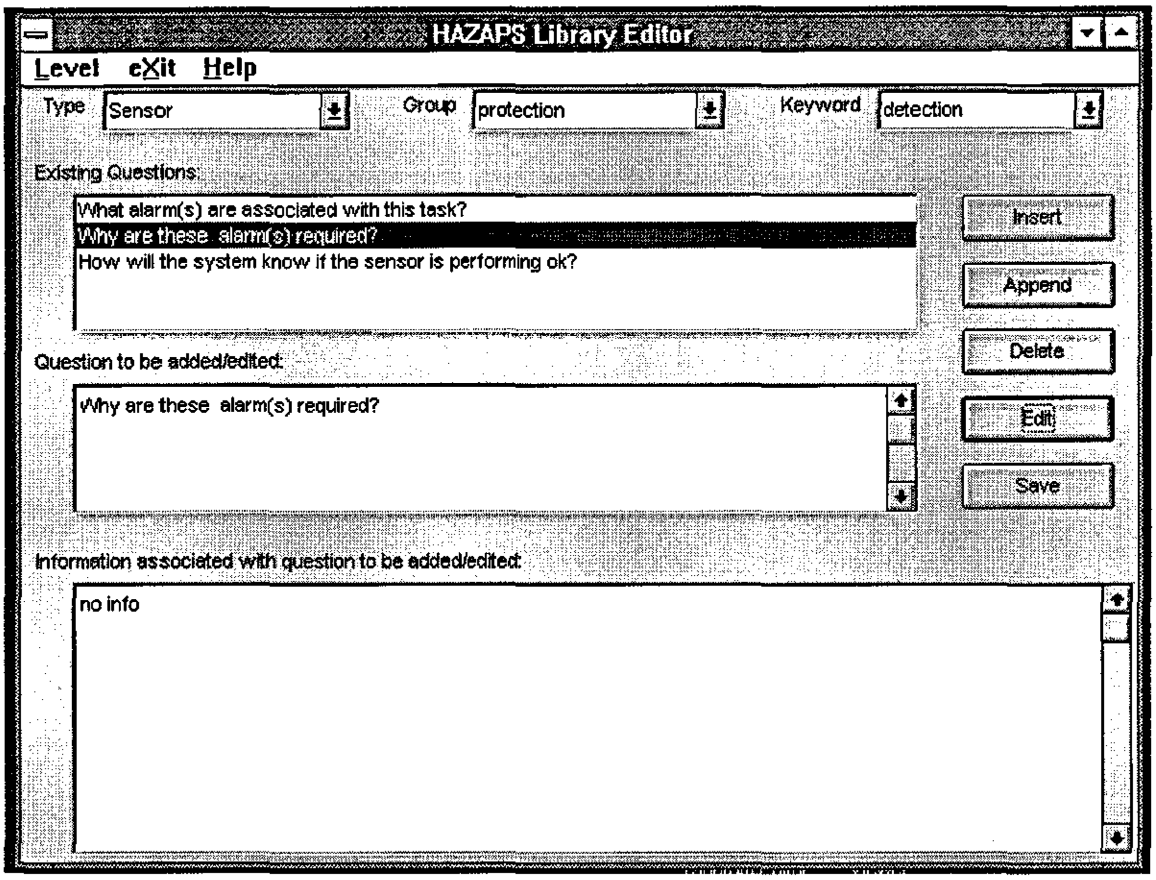
Library Editor Icon

1. Select Task Type for question.

2. Select Group (i.e. Specification, Implementation or Protection) to which question belongs. As a consequence of this action, Keywords belonging to this Group are passed to the Keyword Choice box.

3. Select Keyword. Existing questions associated with the highlighted Type, Group and Keyword are displayed in the List Box.

4. Select one of 4 options :-
   a) highlight a question and delete it ;
   b) edit a question and its associated information by highlighting question in List Box and pressing Edit button;
   c) insert a question by highlighting in the List box where a question is to be positioned and pressing Insert button. 'Nil' then appears in the List Box signifying that a vacant position is available in the List Box (i.e. a new question instance has been created), click on this position and enter new question;
   d) append to the existing question list by pressing Append button, 'nil' appears at the last position in the list, click on this position and enter new question.



Type: Sensor
Group: protection
Keyword: detection

Existing Questions:
What alarm(s) are associated with this task?
Why are these alarm(s) required?
How will the system know if the sensor is performing ok?

Question to be added/edited:
Why are these alarm(s) required?

Information associated with question to be added/edited:
no info

Fig. 5.11: Library Editor Frame

5. If necessary, edit the question and associated information in the relevant Text Boxes. Store all changes by pressing Save button.

6. To return to System Level select eXit from menu bar. Under eXit, there is an option to permanently overwrite the database file for future projects.

5.3.7 Report Generator

The **Report Generator** is used to view and/or print data at any stage during the assessment process (see Fig. 5.12). The Report is not static but continually updates as the user enters information to the system. It has one dialog subwindow. The procedure is as follows:

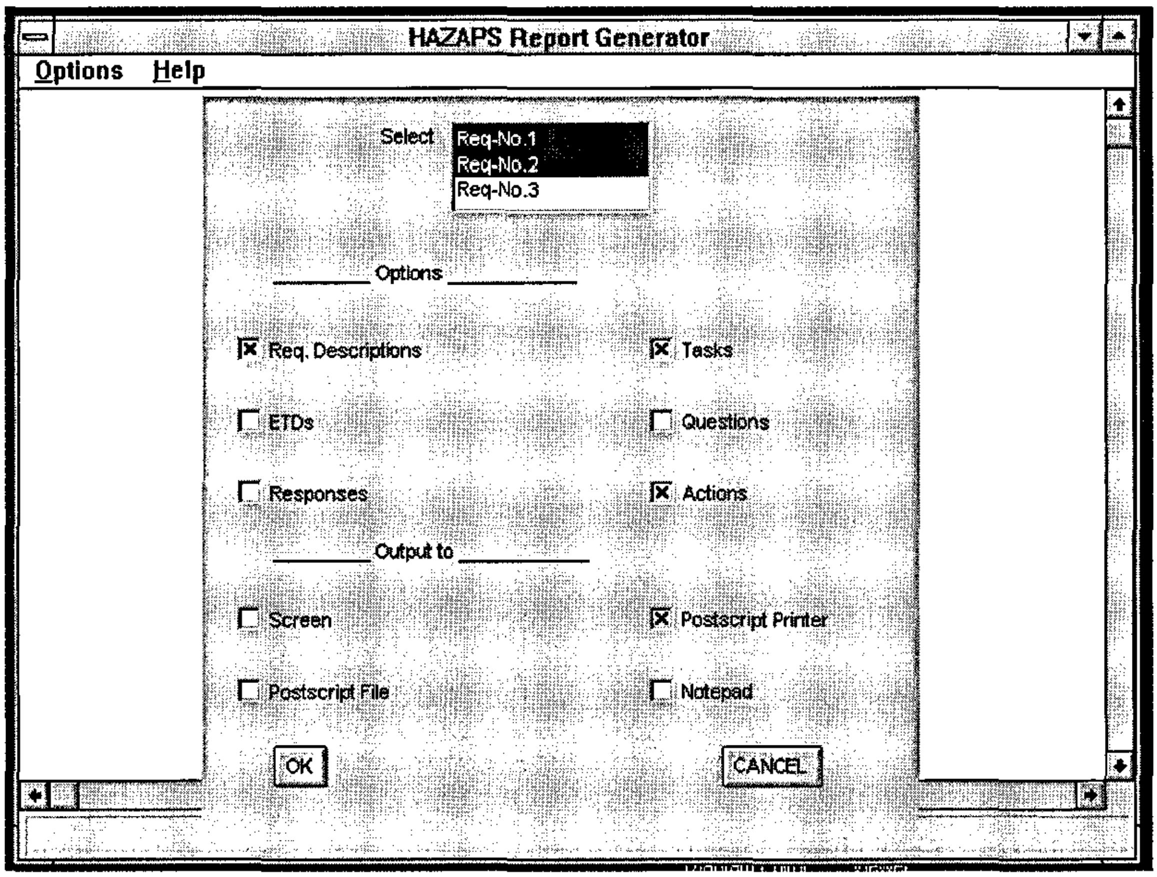
Report Generator Icon

1. Select Options from the menu bar. Click on Start and a dialog subwindow is displayed.

2. Select one or more Requirements.

3. Select the items (Requirement Description, Tasks, ETDs, Questions, Responses and Actions or any combination of these options) required for the report.

4. Select one or more output devices. Options include Screen, Postscript Printer, Postscript file, Windows Notepad, or any combination of these options. If the report has already been displayed on screen and a printout is subsequently required, a postscript file can be obtained via the submenu Print to a postscript file under Options on the menu bar.


HAZAPS Report Generator

Options Help

Select: [✓ Req-No.1] [✓ Req-No.2] Req-No.3

Options

[✓ Req. Descriptions] [✓ Tasks]

ETDs Questions

Responses [✓ Actions]

Output to

Screen [✓ Postscript Printer]

Postscript File Notepad

OK CANCEL

Fig. 5.12: Report Generator Frame

5.4 Conclusions

The combination of CLIPS and wxCLIPS provided a very powerful environment for prototyping the HAZAPS tool. The main difficulty encountered was in identifying the key abstractions and their associations (ie. Class structures and external entities). Combining the principles of layering and object oriented classes allowed the tool to evolve in a gradual and easily-maintainable fashion.

The tool has proved very useful in supporting the HAZAPS methodology. The advantages being

- ✦ Automatic generation of ETDs;
- ✦ Flexibility of the Report Generator;
- ✦ Ability to customise the questions library and hence build experience into the tool;
- ✦ Extendability is possible because the tool has been developed in a structured and logical fashion.

CHAPTER 6 : Applications of HAZAPS

In this chapter HAZAPS is applied to different systems. The objectives of applying HAZAPS are to:-

- ✦ show typical results;
- ✦ demonstrate the benefits;
- ✦ identify any major difficulties;
- ✦ show how HAZAPS can be applied to different application domains.

Three applications of HAZAPS are described: (i) an illustrative example based on a simple aircraft navigation task, (ii) a case study based on a rotary screen line printing press, and (iii) a case study based on a water treatment plant. In the case of the aircraft example, modelling and assessment (Stages 3 & 4 of HAZAPS) of a single requirement is briefly described. For the two case studies, all four stages of HAZAPS were used and several requirements were considered in detail. The rotary screen printing press has been tested and is operational. The water treatment plant has not yet passed the design stage. The assessment of both these case studies was based primarily on information obtained from real functional requirement specifications and design drawings. In the following sections the three applications of HAZAPS are described.

6.1 Modelling and assessing an avionics safety requirement

This example of an aircraft navigation requirement illustrates how stages 3 and 4 of the HAZAPS process can be used to model and assess safety requirements. The example shows the use of the ETD technique and the ES-HAZOP and ES-FMEA procedures.

6.1.1 *Modelling the safety requirement*

The safety requirement is *Aircraft shall maintain a safe heading*. The following operational tasks are associated with this requirement:-

- Pilot feeds heading into the system;
- System determines aircraft's present position via a sensor;
- Processor calculates new heading for aircraft;
- System issues update to display;
- Pilot confirms this update;
- Aircraft is steered in correct direction.

Fig. 6.1 shows how the above tasks are modelled using the ETD technique. There are several possible scenarios that might lead to a hazardous situation, including (i) Pilot feeds in wrong heading and subsequently accepts the issued update; (ii) Pilot incorrectly confirms 'bad' update and the aircraft is steered off-course. These scenarios initially prompt investigation of pilot interventions and updating of the display console.

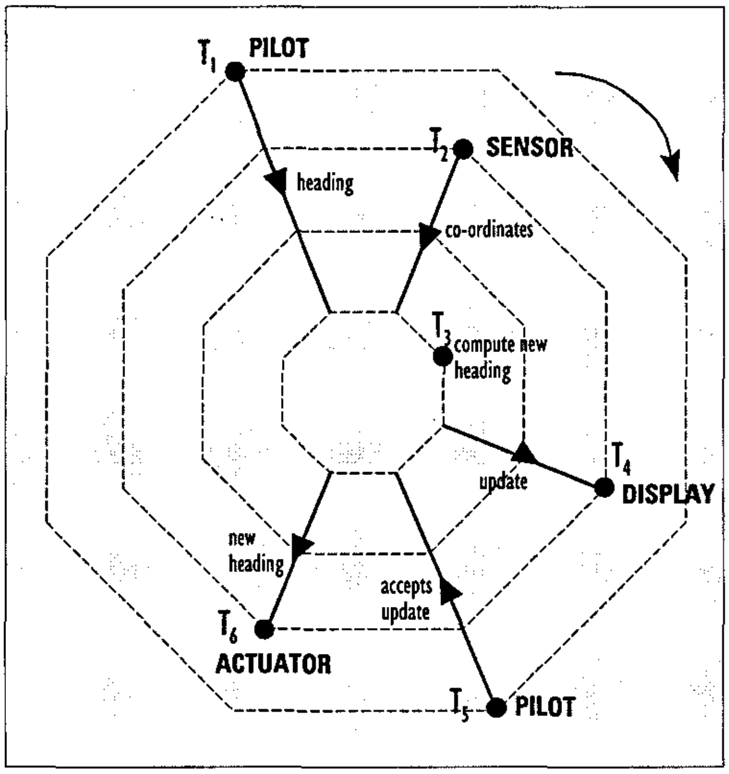


Fig. 6.1: ETD for the aircraft navigation requirement

6.1.2 Assessing the safety requirement

The ES-HAZOP and ES-FMEA procedures (cf. Appendix 1) provide the basis for assessing requirements. The operational tasks were first categorised in terms of generic tasks and then the ES-HAZOP and ES-FMEA procedures were applied. From the above scenarios, the pilot interventions and updating of the display were considered important. The pilot interventions,

- ✦ Pilot feeds heading into system
- ✦ Pilot confirms this update

are classed as Operator-type tasks, and the updating of the display,

- ✦ System issues update to display

is classed as a Display-type task.

Table 6.1 shows a subset of questions for each of the tasks which can be applied from each of the Superclasses. The purpose of the questions is to help clarify safety considerations and reduce or eliminate any associated hazards by revising functional requirements or implementing safety control strategies. In the following case studies, responses to questions are considered and it is shown how they facilitate an in-depth analysis of the design.

6.2 Case Study 1 - Print Machine

The first case study is based on a rotary screen line printing machine. The development of such a machine requires a multi-disciplinary approach, bringing together expertise in process engineering, logic control, variable speed drives and mechanical engineering. Each stage of the HAZAPS process is applied in turn.

Table 6.1: Subset of assessment questions for each of the tasks

	Superclass	Class	Slot/Task	Question
ES-HAZOP	**Specification**	Timing/control	Operator	*When is this intervention required?*
			Display	*How often does this information need to be updated?*
	Implementation	Environment	Operator	*Is this intervention easy to perform in a stressful situation?*
			Display	*Where can the display be positioned for most effective use by the operator?*
	Protection	Fault_Recovery	Operator	*Is it clear what the operator has to do in an emergency situation?*
			Display	*What emergency procedures are associated with this task?*
ES-FMEA	**Failure_Modes**	Not_Initialised	Operator	*What if operator fails to carry out task?*
			Display	*What if information is not shown on display?*

6.2.1 *Identification of safety-critical subsystems*

The objective of this stage is to understand the processes associated with the printing machine, determine underlying concepts and identify subsystems for analysis. The rotary screen printing press consists of a number of print stations. Each print station prints a different pattern. Material flow through the printing press is shown in Fig. 6.2. Paper is automatically unwound from a reeler and passes to a print head where it is coated with

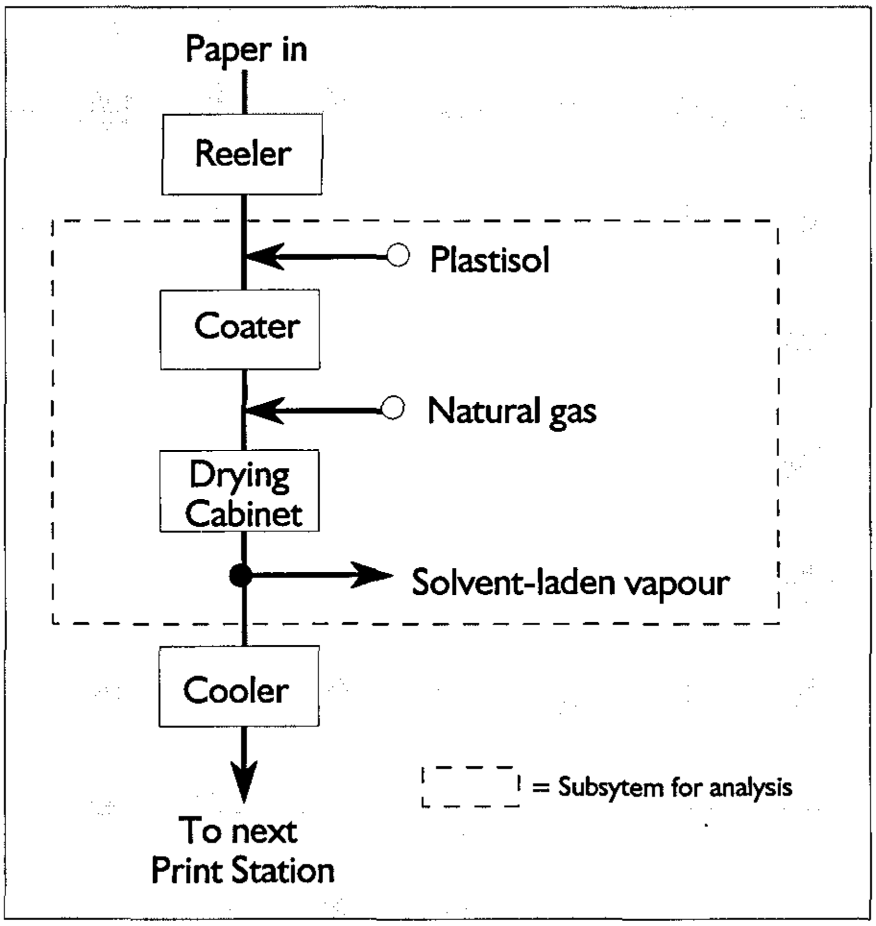


Fig. 6.2: Material flow through rotary screen line printing press

plastisol. It then passes through a drying cabinet and down to a cooling roller, before passing to the next print station. Three different computer systems are used. One computer controls the running of the line. Each print head has a single board computer to control the printing and each burner in the drying cabinet has an associated proportional, integral and derivative controller.

6.2.1.1 Major hazards

Major sources of hazards associated with the printing process include:-

- solvent-laden vapour which evaporates as the coated material is dried;
- large steel rollers rotating at high speed which pull the web (i.e. the material passing through the machine) through the printing press;
- gas burners in the drying cabinet which are used to dry the coated paper.

The case study focuses on monitoring and controlling the solvent concentration in the dryer (dashed box in Fig. 6.2).

6.2.1.2 System Components

Components of the system under consideration (Fig. 6.3) can be divided into 2 object classes, Chemicals and Equipment.

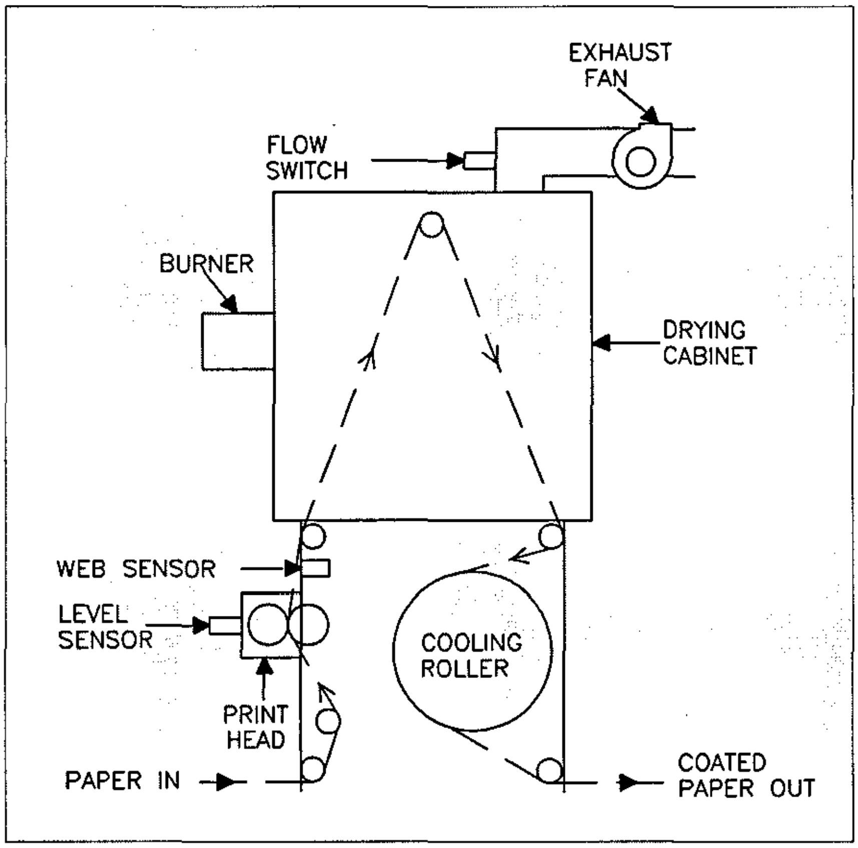


Fig. 6.3: One of the print stations in the screen line printing press

Chemicals

- Plastisol contains a number of organic chemicals including plastiser and solvent: the solvent is emitted during the drying process.
- Natural gas is used by the burners in the drying cabinet.

Equipment

The Equipment class is divided into three subclasses: Process, Sensors and Actuators.

Process

- A *print head* consisting of a screen (a hollow cylindrical container into which the plastisol is pumped) and backing roller. The term *nip* is used to describe the gap between screen and backing roller.
- A *pump* fills the screen with plastisol.
- *Pneumatic rams* press the backing roller against the screen.
- A *drying cabinet* contains a burner which heats and gels paper and plastisol.
- A *fan* removes solvent vapour from the dryer.
- A *cooling roller* cools the coated paper and helps set up the web path for the next print station.

Sensors

- The *line sensor* senses whether the web is moving through the machine.
- The *level sensor* measures the amount of plastisol in the screen.
- The *air pressure sensor* senses whether or not the pneumatic system is operational.
- The *web sensor* determines whether the web is continuous.
- The *nip pressure sensor* determines whether the nip is open or closed.
- The *flow sensor* determines whether there is flow through the exhaust.

The system has other sensors (e.g. encoders, tachometer, load cells and thermocouples) but these are not relevant to the present analysis.

Actuators

- ✦ *Drive motors* are used for backing and cooling rollers;
- ✦ *Solenoids* for pneumatic rams;
- ✦ *Relays* for exhaust fan, burner and plastisol pump.

Having identified a subsystem, its components and top level hazards, the next step is to construct safety requirements related to this subsystem.

6.2.2 Construction of safety requirements for print station

The objective is to construct safety requirements for the print station related to the solvent concentration in the dryer. The use and control of flammable solvents in printing inks and plastisols is a well known source of hazards in the printing industry. Each drying cabinet in the machine has explosion panels, however, the machine user is responsible for ensuring that

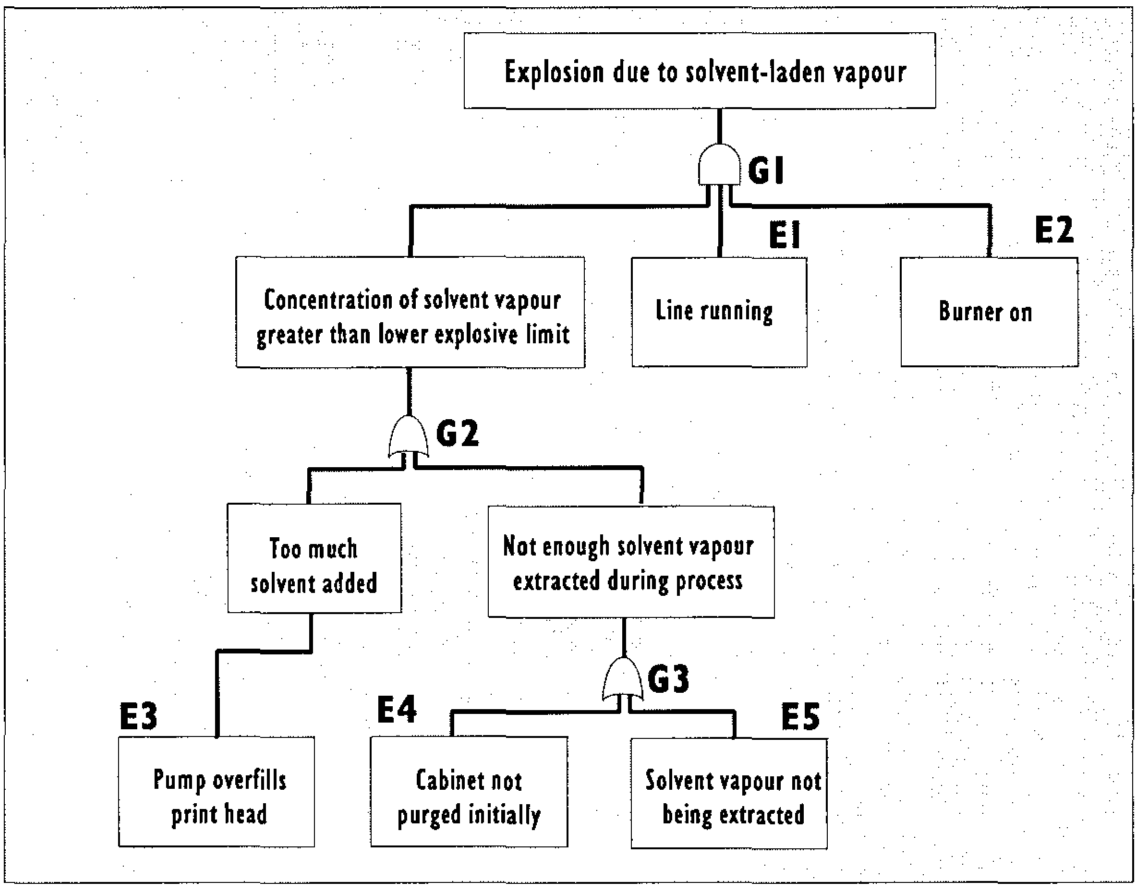

Explosion due to solvent-laden vapour
G1
E1
E2
Concentration of solvent vapour greater than lower explosive limit
Line running
Burner on
G2
Too much solvent added
Not enough solvent vapour extracted during process
G3
E3
E4
E5
Pump overfills print head
Cabinet not purged initially
Solvent vapour not being extracted

Fig. 6.4 Fault tree for Print Station. E1 and E2 are not faults but indicate the state of the subsystem at a given time.

flammable solvent levels in the circulating air and exhaust air associated with each cabinet are safe as defined in the HSE regulations [HSE, 1981]. The emphasis here is on identifying contributors to these hazards when a programmable system is used. The fault tree shown in Fig. 6.4 is used (see Section 4.4.2 for derivation of predicate statements).

The predicate statements were

E1 AND E2 AND (NOT) E3 1
E1 AND E2 AND (NOT) E4 2
E1 AND E2 AND (NOT) E5 3

The following safety requirements were constructed using the above predicate statements.

Requirement No. 1 (based on predicate statement 1)

The pump at the print station must not overfill the print head because the solvent concentration in the dryer will increase and may increase beyond the specified limit.

Requirement No. 2 (based on predicate statement 2)

Before the machine is started or restarted, the drying cabinet must be purged initially.

Requirement No. 3 (based on predicate statement 3)

If no solvent vapour is being exhausted from the drying cabinet while the line is running and the burner is switched on, the machine must be shut down and placed in a safe state.

Having constructed safety requirements, the next step is to transform the requirements into operational tasks and assess them.

6.2.3 Modelling and assessing safety requirements for print station

The safety requirements are assessed in turn.

6.2.3.1 Requirement No. 1 - Print station

Description: *The pump at the print station must not overfill the print head because the solvent concentration in the dryer will increase and may increase beyond the specified limit.*

Operational tasks:

- Operator selects automatic mode for pump operation
- Line sensor indicates line is running
- Nip sensor indicates that nip is closed
- Level sensor indicates low plastisol
- Processor determines pump 'on time' t, based on line speed and required coat weight
- The pump is switched on for time t

ETD: The ETD is shown in Fig. 6.5. The pump can operate in automatic or manual mode. Automatic mode is only operative when the line is running and the nip is closed. The pump is switched on when the level detector detects a low plastisol level in the print head. The pump fills the print head at intervals, the time between intervals is dependent on the required coat weight.

Critical tasks: The safety aspect of this requirement ultimately depends on the switching on of pump for time *t*.

Analysis of ACTUATOR task: The pump is switched on for time *t*.

Specification

Definition:

Q: What action is required?

R: To turn on pump for a 'set' time

Objective:

Q: Why is this action required?

R: To fill print head with required amount of plastisol

Options:

Q: What other way could this task be accomplished?

R: Rather than using the level sensor to trigger switch on of pump, a detector which directly measures coat weight could be used.

Action 1: Consider using an on-line coat weight monitor as there would be immediate feedback on excessive coating. This would be useful for safety, from an economic point of view and for better consistency in coating. The difficulty of using such a device is that the coat weight profile varies across the paper for a given pattern and there are also variations due to inconsistencies in batch production (i.e. continual recalibration would be required).

Inputs/Outputs:

Q: What are inputs/outputs for this task?

R: The pump solenoid receives an energise signal from the main system

Q: What parameters are associated with this task?

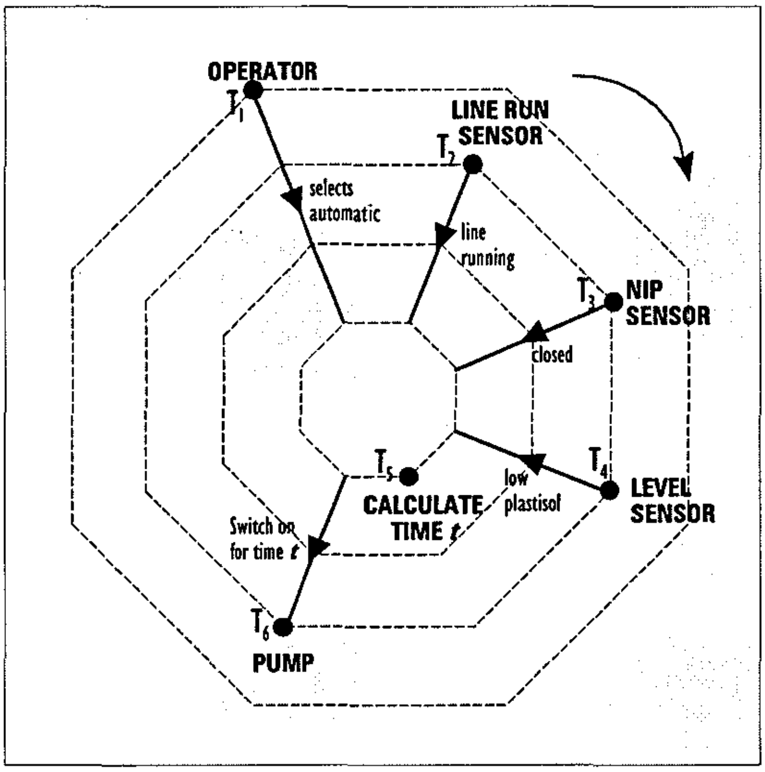


Fig. 6.5: ETD for Requirement No. 1 - Print Station

R: 'Line speed' and 'pump factor'. 'Pump factor' is set by operator. It is dependent on coat weight required.

Q: What is the range of this output signal?

R: On/off only. No associated range.

Timing/Control:

Q: How is this task to be initialised?

R: Level probe senses that the plastisol level is low

Action 2: Check calibration of level sensor as the operation of the pump is dependent on the sensitivity of the level sensor.

Q: How is this task to be terminated?

R: The task is terminated after a time 't' set by the main programmable system which de-energises the pump solenoid.

Operational_Modes:

Q: What relationship does this task have to manual mode?

R: In manual mode the pump action is interlocked only to the level sensor, the pump action is independent of line speed and status of nip.

Implementation

Selection:

Q: What actuator will be used?

R: A solenoid valve

Installation:

Q: How will this device be interfaced to the system

R: The solenoid valve is energised via a signal from an electromechanical relay. One input of the relay is connected to programmable system via a digital output

Testing:

Q: How will the implementation be tested?

R: The switch on time of pump is dependent on coat weights. The coat weight is related to the pattern being printed on the paper.

Action 3: Trials are required to determine maximum and minimum pump on times.

Q: What reliability data is available on hardware items?

R: There is no reliability data available at present.

Action 4: Obtain hardware reliability data on pump and associated devices.

Protection

Failure_Detection:

Q: How will the system know if any of the hardware devices associated with this task have failed?

R: There are no directly associated detection mechanisms as there is no automated feedback from the coating operation. Detection is dependent on the operator

Trips:

Q: What trips are associated with this task?

R: There are no trips associated with this task

Action 5: Consider implementing trip in case of pump remaining on unintentionally.

Security:

Q: What parameters associated with this task can be modified by the operator?

R: The 'pump factor' can be modified.

Q: Why can these parameters be modified by the operator?

R: There are many different coat weights required and there are always inconsistencies within batches.

Action 6: If operator selects and sets up pump on time, he/she may need training on how to do this.

Failure_Modes

Lockup:

Q: What if this task locks up?

R: This would occur if there was no signal to deactivate pump which would cause the plastisol to overfill print head and result in excessive coating. To prevent this, a maximum on time for pump is incorporated in the software which, if exceeded, trips power to pump via a totally independent route.

6.2.3.2 Requirement No. 2 - Print station

Description: *Before the machine is started or restarted, the drying cabinet must be purged initially.*

Operational tasks:

- ✦ Display is activated
- ✦ Operator presses start button
- ✦ Sensors indicate no emergency stops pressed
- ✦ Web break sensors indicate no web breaks
- ✦ Air sensor detects air supply OK
- ✦ Exhaust fan is started to purge dryer
- ✦ Machine is started

ETD: The ETD is shown in Fig. 6.6. Emergency stop buttons are located at the print station so that the operator can stop the machine at any time. Web break sensors are used to detect any break in the material. The air supply is used to open and close nips. The exhaust fan is used to remove solvent vapour from the drying cabinet. The digitally controlled drives synchronize the passage of material through the print heads.

Critical tasks: The display task is very important as it allows the operator to monitor the startup conditions and to intervene if necessary.

Analysis of DISPLAY task: Display is activated.

Specification

Definition:

Q: What is to be displayed?

R: Information about states on the machine.

Objective:

Q: Why does information need to be displayed?

R: To permit operator to monitor startup conditions and to allow manual intervention if necessary.

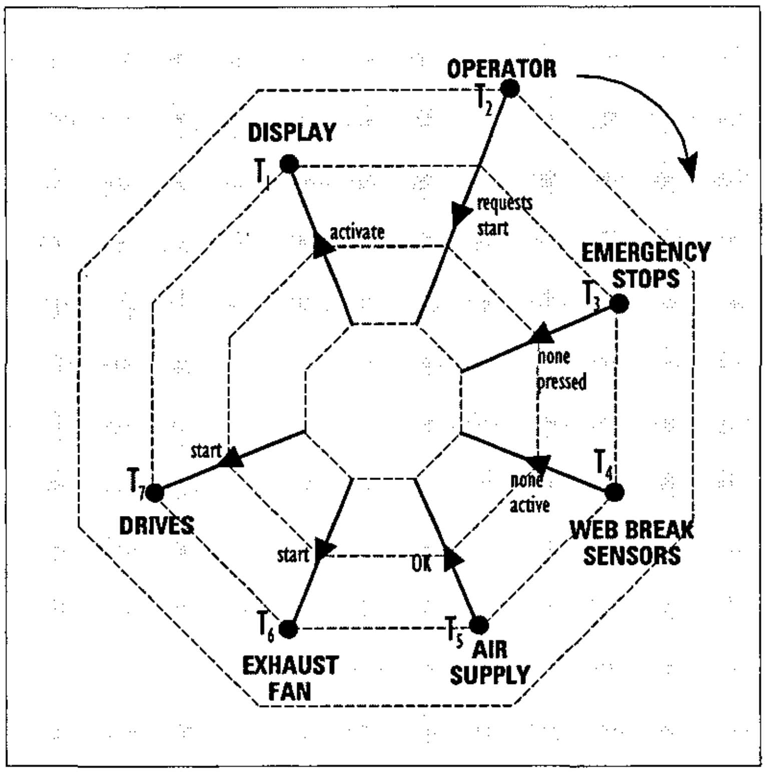


Fig. 6.6: ETD for Requirement No.2 - Print Station

Inputs/Outputs:

Q: What are the inputs/outputs for this task?

R: Status of line (pre-startup, starting, running, normal shutdown, emergency shutdown), emergency push buttons, web break sensors, air supply.

Q: What is the format for displaying information?

R: Icons for identifiers, states in alphanumerics, any unexpected states to flash in red.

Timing/Control:

Q: How often does this information need to be updated?

R: Every 2 seconds

Action 7: Check why 2 seconds is chosen as the update interval for the display.

Operational_Modes:

Q: What relationship does this task have to normal shutdown?

R: The display is also used in the shutdown sequence to allow the operator to monitor a safe shutdown

Q: What relationship does this task have to an emergency shutdown?

R: The display (in conjunction with an audible alarm) is also used in the emergency shutdown sequence to warn operator and to permit intervention if necessary.

Implementation

Selection:

Q: What type of display will be used?

R: A VDU

Installation:

Q: If this display is used to show other information, how is information unique to this task distinguished?

R: The display format is paged, at startup the default is the startup page, similarly for normal shutdown and emergency shutdown. At other times, the page is pre-startup, normal running or the operator selects the page with the information required.

Environment:

Q: What particular aspects of the environment may affect the operation of this task?

R: The environment is 'dirty' and hot. The print station will have an industrial type transparent window. The air is solvent laden requiring a flame proof enclosure for the VDU.

Maintenance:

Q: What maintenance procedures are required for this task?

R: A software test module will be used to test display and inputs from various sensors. This will also prove useful for testing operation of sensors

Action 8: Find out if this test module can be executed when the machine is operational. If it can, how is it ensured that it does not interfere with normal operation.

Protection

Failure_Detection:

Q: How will failure of display task be detected

R: When display is operating correctly a number in the top right hand side of the display will continually update. Failure of this number to update will indicate display task has failed.

Action 9: Test idea of number in top right hand of display. Normally, the number will be continuously updating, however, on the rare occasion that it does not update will this be obvious to the operator?

Fault_Recovery:

Q: What fault recovery procedures are associated with this task.

R: Where possible any warnings given on display will have an associated help page which will give operator information on what he should do.

Action 10: Give list of all warnings to be shown on display and describe associated recovery procedures.

Failure_Modes

Incorrectly_Initialised:

Q: What if task is incorrectly initialised?

R: This could occur if there are no default displays. To prevent this, if there is power to the VDU, it always shows some display i.e. startup, normal shutdown, emergency shutdown or if not in any of previous modes then pre-startup, running mode or some page selected by operator.

6.2.3.3 Requirement No. 3 - Print station

Description: *If no solvent vapour is being exhausted from the drying cabinet while the line is running and the burner is switched on, the machine must be shut down and placed in a safe state.*

Operational tasks:

- Flow sensor indicates no exhaust flow
- Line sensor detects line is running
- Burners are switched off
- Nips are forced to open
- Line is kept running
- Alarm is activated
- Warning is shown on Display

***ETD*:** The ETD is shown in Fig. 6.7. If there is no flow through the exhaust and the line is running then the burner is switched off to remove heat source from drying cabinet. The nip is forced open to stop coating of paper and the line is kept running to remove paper from the drying cabinet and prevent it from igniting. Siren is activated and warning is displayed.

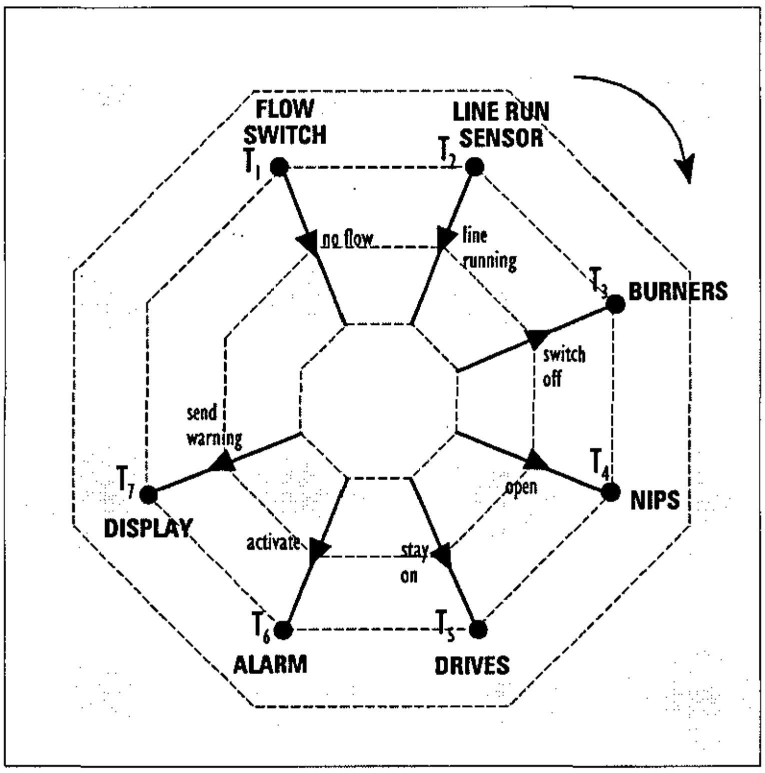


Fig. 6.7: ETD for Requirement No.3 - Print Station

Critical tasks: The operation of the flow switch is critical. Incorrect operation or logic associated with this device could lead to catastrophic events.

Analysis of SENSOR task: Flow sensor indicates no exhaust flow.

Specification

Definition:

Q: What state is to be monitored?

R: The exhaust flow

Objective:

Q: Why does this state need to be monitored?

R: To prevent build up of solvent concentration in drying cabinet

Options:

Q: What other way could this task be accomplished?

R: A sensor which directly measures solvent concentration or one which measures airflow.

Action 11: Consider directly measuring solvent concentration rather than monitoring the exhaust flow. Firstly, it measures the required parameter directly and, secondly, continuous feedback from direct monitoring could detect an unpredicted rise in solvent concentration. The major disadvantages of this are cost and number of monitors required.

Inputs/Outputs:

Q: What are the input(s)/output(s)for this task?

R: Output signal from flow sensor indicating no exhaust flow

Q: Over what range is the signal to be monitored?

R: Output signal on/off, no associated range

Timing/Control:

Q: How often does this state have to be scanned?

R: Scan time should be less than a second.

Operational_Modes:

Q: What relationship does this task have to startup?

R: The flow switch is associated with a purge sequence which must complete before startup.

Implementation:

Selection:

Q: What sensor will be used?

R: An air flow failure detector which has an embedded mercury switch.

Installation:

Q: How will this device be interfaced to the system

R: The mercury switch is connected to an electromechanical relay. One output of the relay is connected to programmable system via a digital input.

Q: How will this device be calibrated?

R: The device is supplied by the manufacturer calibrated, it will be tuned when the machine is being commissioned

Q: Where will this device be positioned?

R: In the exhaust ducting, not near any disturbances e.g. fresh air dampers.

Action 12: Need to ensure correct calibration of the airflow switch.

Environment:

Q: What particular aspects of the environment may affect the operation of this task?

R: The environment is hot and laden with plastisol fumes. The device works on the principle of difference in pressure inside and outside the exhaust duct. Clean air outside the duct activates the device.

Action 13: The dependence on a single device for monitoring exhaust flow is questionable. Check how difficult it would be to incorporate another sensor (better if sensor operates on a different physical principle).

Maintenance:

Q: What maintenance procedures are required for this task?

R: The condition and operation of the sensor is checked manually on a regular basis because of its importance and the corrosive environment in which it is placed.

Action 14: A good maintenance procedure for the airflow switch is essential, because of its importance and the 'dirty' environment. The device should be easily accessible for the same reasons.

Protection

Failure_Detection:

Q: What alarms are associated with this task?

R: A siren and alarm message on a display

Q: Why are these alarms required?

R: To warn operator that the sensor has indicated no flow in the exhaust, and to check that the machine is shutdown safely.

Q: How will the system know if this task has failed?

R: There will be a test sequence pre-startup which will check the sensor indicates 'off' state with the exhaust fan off indicating no exhaust flow and on state with the fan on to indicate there is an exhaust flow

Interlocks:

Q: Are there any postconditions associated with the completion of this task which can be checked to ensure that the task has executed successfully and on time?

R: Must be checked manually by operator.

Trips:

Q: What trips are associated with this task?

R: Nip is forced off and burner is switched off

Q: Why are these trips required?

R: The nip is opened to stop coating, the burner is switched off to remove heat source

Fault_Recovery:

Q: What fault recovery procedures are associated with this task?

R: Manually test operation of exhaust fan and sensor, purge dryer and measure concentration of solvent in dryer.

Failure_Modes

Incorrectly_Terminated:

Q: What if task is incorrectly terminated?

R: This would occur if operator presses emergency stop, this would leave coated material in dryer which may ignite. To prevent this, the siren should be unique to this task, indicating that no operator intervention is required.

Action 15: Ensure that the alarm siren is readily identifiable as being associated with 'dangerous' level of solvent vapour.

Too_Slow:

Q: What if signal is too slow?

R: If the response of the sensor was too slow, this would cause a build up of solvent vapour in dryer and possible subsequent explosion.

Action 16: Hardware reliability information is required to make sure the airflow switch is robust enough for the number of operations and environment. Verify that the response of sensor is adequate.

6.2.4 Summary of assessment of print station

The assessment focuses on one top level hazard (i.e. solvent vapour concentration in the dryer). Three safety requirements were identified and only one task associated with each requirement was assessed. A small number of all possible questions were used for assessment purposes. Sixteen actions are listed as a result of the assessment, these are shown in Table 6.2. Actions are related to: (i) requests for further information; (ii) checking of design decisions; (iii) suggestions for alternative designs; (iv) maintenance proposals; and, (v) test plans. Major concerns are the measurement of critical parameters and the environmental conditions. Some critical parameters are measured indirectly; it is proposed that these parameters should be measured directly (Actions 1 and 11). Also, the dependence on a single device to measure a critical parameter is questioned (Action 13). The environmental conditions are particularly nasty and therefore require attention (Actions 4, 12, 14, and 16).

Table 6.2: Actions based on assessment of print station

Action 1:	Consider using an on-line coat weight monitor as there would be immediate feedback on excessive coating. This would be useful for safety, from an economic point of view and for better consistency in coating. The difficulty of using such a device is that the coat weight profile varies across the paper for a given pattern and there are also variations due to inconsistencies in batch production (i.e. continual recalibration would be required).
Action 2:	Check calibration of level sensor as the operation of the pump is dependent on the sensitivity of the level sensor.
Action 3:	Trials are required to determine maximum and minimum pump on times.
Action 4:	Obtain hardware reliability data on pump and associated devices.
Action 5:	Consider implementing trip in case of pump remaining on unintentionally.
Action 6:	If operator selects and sets up pump on time, he/she may need training on how to do this.
Action 7:	Check why 2 seconds is chosen as the update interval for the display.
Action 8:	Find out if this test module can be executed when the machine is operational. If it can, how is it ensured that it does not interfere with normal operation.
Action 9:	Test idea of number in top right hand of display. Normally, the number will be continuously updating, however, on the rare occasion that it does not update will this be obvious to the operator?
Action 10:	Give list of all warnings to be shown on display and describe associated recovery procedures.
Action 11:	Consider using on-stream solvent monitoring rather than monitoring the exhaust flow. Firstly, it measures the required parameter directly and, secondly, continuous feedback from on-stream monitoring could detect a unpredicted rise in solvent concentration. The major disadvantages of this are cost and number required.
Action 12:	Need to ensure correct calibration of the airflow switch.
Action 13:	The dependence on a single device for monitoring exhaust flow is questionable. Check how difficult it would be to incorporate another sensor (better if sensor operates on a different physical principle).
Action 14:	A good maintenance procedure for the airflow switch is essential, because of its importance and the 'dirty' environment. The device should be easily accessible for the same reasons.
Action 15:	Ensure that the alarm siren is readily identifiable as being associated with 'dangerous' level of solvent vapour.
Action 16:	Hardware reliability information is required to make sure the airflow switch is robust enough for the number of operations and environment. Verify that the response of sensor is adequate.

6.3 Case Study 2 - Water treatment plant

This case study is based on preliminary design work done on a water treatment plant. In addition to water treatment experience, the design requires expertise in chemical engineering, instrumentation and software engineering.

6.3.1 Identification of safety-critical subsystems

The objective of this stage is to understand the processes associated with the water treatment plant, determine underlying concepts and identify subsystems for analysis. Water treatment involves several different processes as shown in Fig. 6.8. The first stage, DAF (Dissolved Air Flotation) is used to remove finely divided particles such as clays and colouring matter that may be held in suspension in the water. $FeCl_3$ assists this process and acts as a flocculating agent. H_2SO_4 reduces pH and promotes the separating out of fine particles. Cl_2 is added as a prechlorination agent to remove micro-organisms. Once the suspended content in the water is reduced to consistent normal proportions, further clarification is effected by passing water into the RGS (Rapid Gravity Sand) filters. NaOH is added prior to RGS to raise the pH and help remove iron. The GAC (Granular Activated Carbon) removes traces of chemicals and dissolved organic substances. In the final stage, disinfection, Cl_2 is used as the sterilising agent. Following disinfection, a number of chemicals are added: an SO_2 solution is used to reduce residual Cl in the water; NaOH is used adjust the pH; and, $NaSiO_3$ is used to reduce the effects of corrosion in metallic mains pipes.

6.3.1.1 Major hazards

Major hazards associated with treated water include the presence of:-

- ✦ micro-organisms;
- ✦ pesticides;
- ✦ fertilisers;
- ✦ metals.

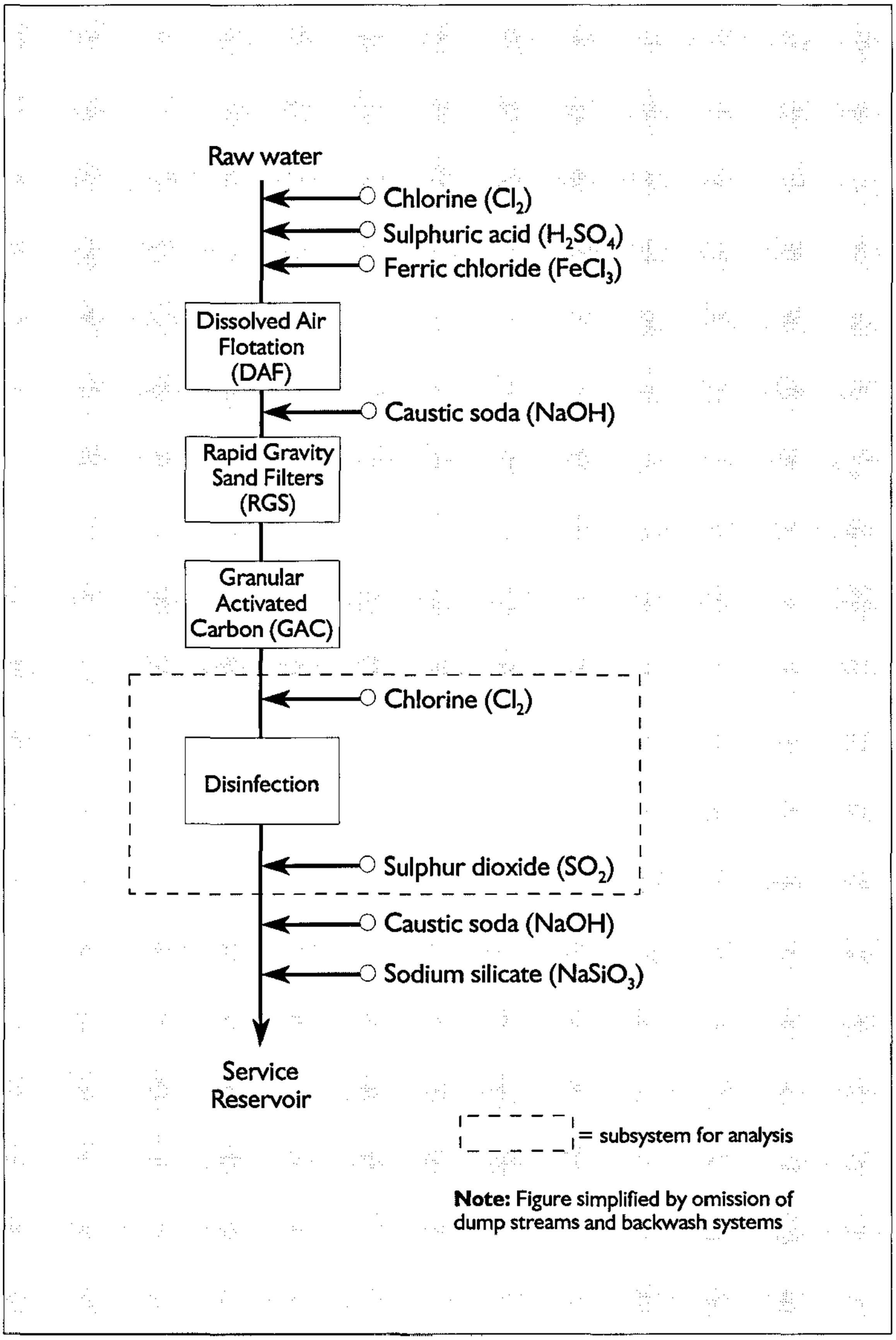

Raw water
Chlorine (Cl_2)
Sulphuric acid (H_2SO_4)
Ferric chloride ($FeCl_3$)
Dissolved Air Flotation (DAF)
Caustic soda (NaOH)
Rapid Gravity Sand Filters (RGS)
Granular Activated Carbon (GAC)
Chlorine (Cl_2)
Disinfection
Sulphur dioxide (SO_2)
Caustic soda (NaOH)
Sodium silicate ($NaSiO_3$)
Service Reservoir
= subsystem for analysis
Note: Figure simplified by omission of dump streams and backwash systems

Fig. 6.8: Material flow through water treatment plant

The focus of this study is on the disinfection system (Fig. 6.8, dashed box), that is, the reduction or removal of micro-organisms. Other processes prior to the disinfection system assist in the removal of pesticides, fertilisers and metals. Only those parts of the Cl_2 and SO_2 dosing systems which interface with the disinfection system are considered in this analysis.

6.3.1.2 System Components

Components of the system under consideration (Fig. 6.9) can be divided into 2 object classes, Chemicals and Equipment.

Chemicals

- Cl_2 is used to remove micro-organisms from the water. It is highly toxic.
- SO_2 (a colourless gas with a choking penetrating smell) is used to reduce any residual Cl in the treated water to a preset level.

Equipment

The Equipment class is subdivided into Process, Sensors and Actuators.

Process

- *Inlet, outlet* and *dump valves* control water flow in the disinfection system.
- *Mixers* add a Cl_2 and SO_2 solution to the process water.
- *Chlorinators* and *sulphonators* mix Cl_2 and SO_2 gases respectively with a water supply (termed the 'motive water supply' as distinct from the 'process water supply' which refers to the water being treated) to form a solution.
- The *contact tank* provides adequate contact time between the Cl_2 solution and the process water.

Sensors

- The *electromagnetic flowmeter* measures flow of water into the disinfection system.
- The *composition sensors* determine concentration of residual Cl in process water.
- The *pressure sensors* detect flow of Cl_2, SO_2 and motive water supply.

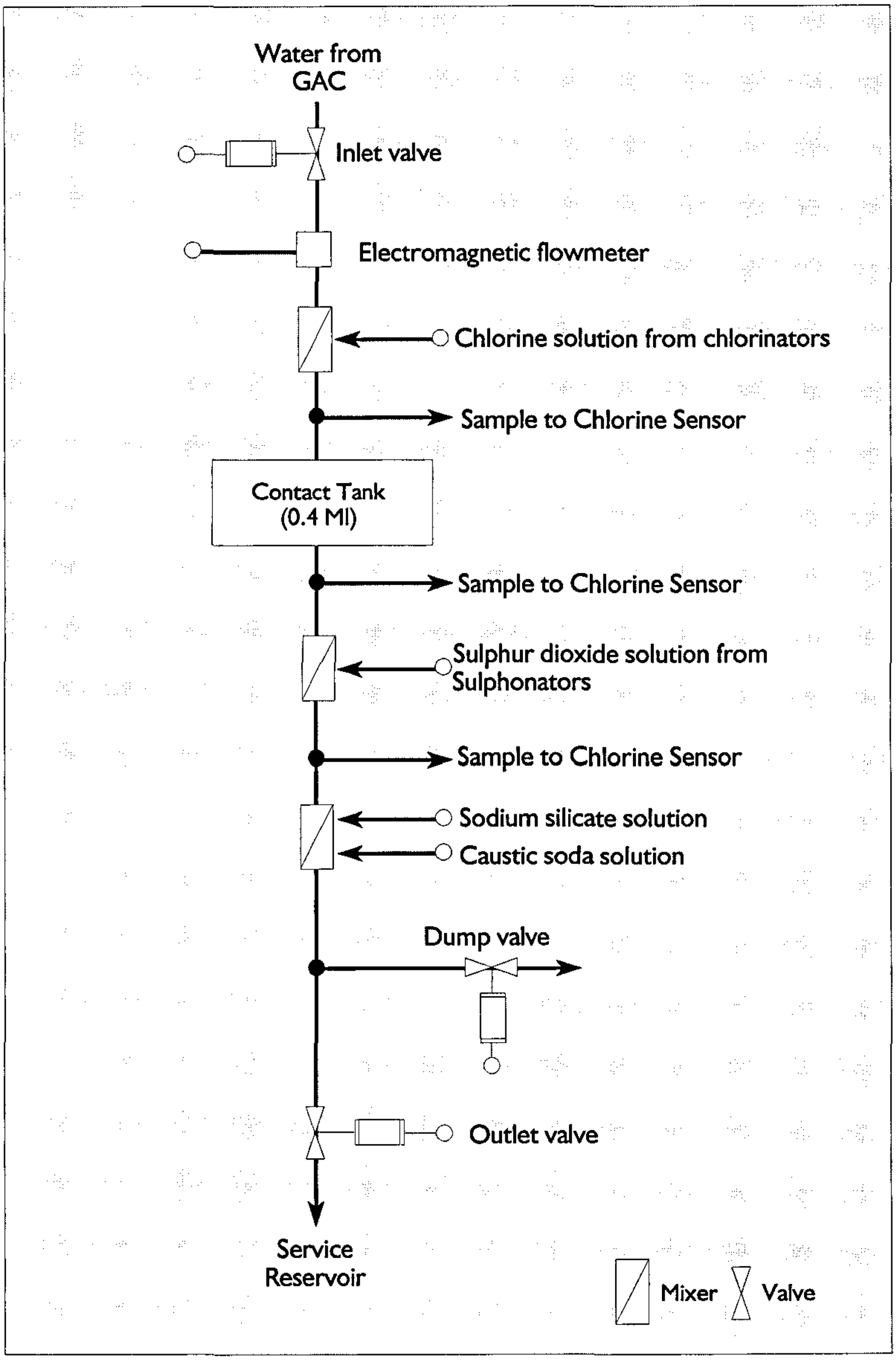

Water from GAC
Inlet valve
Electromagnetic flowmeter
Chlorine solution from chlorinators
Sample to Chlorine Sensor
Contact Tank (0.4 Ml)
Sample to Chlorine Sensor
Sulphur dioxide solution from Sulphonators
Sample to Chlorine Sensor
Sodium silicate solution
Caustic soda solution
Dump valve
Outlet valve
Service Reservoir
Mixer
Valve

Fig 6.9: Disinfection system in water treatment plant

Actuators

- *Motorised valves* are used for inlet, outlet, dump, chlorinators and sulphonators.

Having identified a subsystem, its components and top level hazards, the next step is to construct safety requirements related to this subsystem.

6.3.2 Construction of safety requirements for disinfection system

The objective is to construct safety requirements related to the disinfection system. The addition of Cl_2 is vital as not only does it improve odour and taste (provided correct dose is added) but it assists in removing micro-organisms. A Cl_2 solution is added to process water before the contact tank and is subsequently measured three times: before the contact tank, after the contact tank and after addition of SO_2 solution. If the water stream after the contact tank contains too much Cl, it can be attenuated using SO_2 solution. It is important that the final treated water has the correct quantity of Cl present. Too much Cl can facilitate the formation of 'nasty' organic compounds and too little can result in growth of micro-organisms as the water passes through the distribution network. The associated fault tree is shown in Fig.6.10 where gates are prefixed by G and base events are prefixed by E.

```
Identifying cut sets
      E1    G1
Expanding G1⇒
      E1    G2
      E1    G3
Expanding G2 ⇒
      E1    E2    G4
      E1    G3
Expanding G4 ⇒
      E1    E2    E3
      E1    E2         E4
      E1    G3
Expanding G3 ⇒
      E1    E2    E3
      E1    E2         E4
      E1                 E5
      E1    G5
```

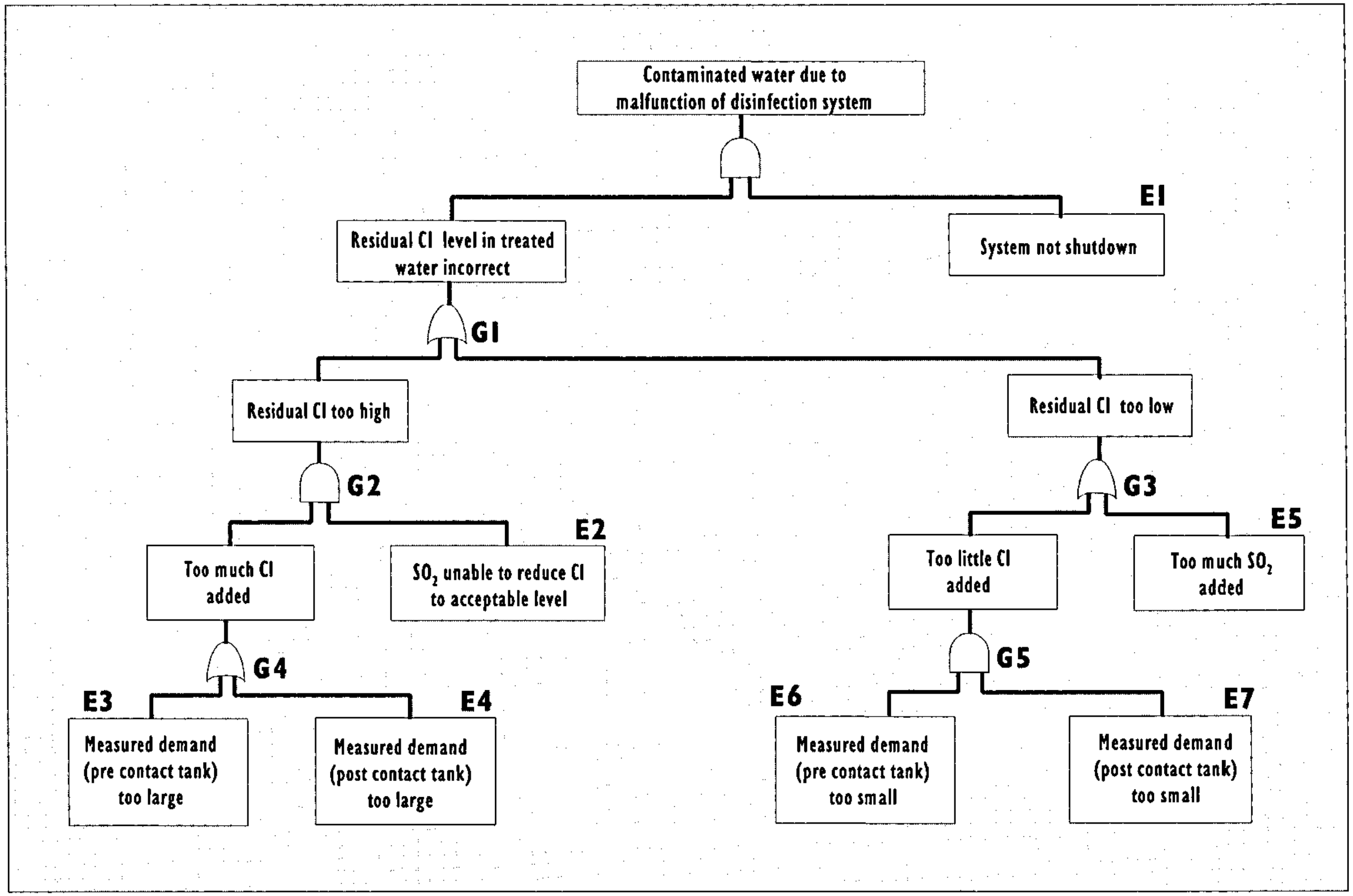

Contaminated water due to malfunction of disinfection system
E1
System not shutdown
Residual Cl level in treated water incorrect
G1
Residual Cl too high
Residual Cl too low
G2
G3
E2
E5
Too much Cl added
SO$_2$ unable to reduce Cl to acceptable level
Too little Cl added
Too much SO$_2$ added
G4
G5
E3
E4
E6
E7
Measured demand (pre contact tank) too large
Measured demand (post contact tank) too large
Measured demand (pre contact tank) too small
Measured demand (post contact tank) too small

Fig. 6.10: Fault tree for disinfection system.E1 is not a fault condition but indicates the condition of the subsystem a given time.

Expanding G5 ⇒

```
E1    E2    E3 ....................................... 4
E1    E2        E4 ................................... 5
E1               E5 .................................. 6
E1                     E6    E7 ...................... 7
```

E1 is common to all cut sets. Under normal operating conditions, E1 is always true (i.e. cannot negate E1). The disinfection system must be shut down if any of the following predicate statements are true (based on cut sets 4, 5, 6, 7 above)

```
E2    AND  E3 ........................................ 8
E2    AND  E4 ........................................ 9
E5 ................................................... 10
E6    AND  E7 ........................................ 11
```

These predicate statements can be divided into two sets to form the following safety requirements:

Requirement No.1

Requirement No.1 is based on predicate statements 8, 9 and 10 which are all related to the addition of SO_2 solution.Each predicate includes either E2 (i.e. SO_2 solution unable to reduce Cl) or E5 (i.e. too much SO_2 added). These predicate statements are used to construct the Requirement No. 1.

If, after addition of SO_2 solution, the measured Cl in the treated water is outside set limits, then the disinfection system must be isolated.

Requirement No. 2

Requirement No. 2 is based on predicate statement 11 and is dependent on the addition of Cl_2 solution. This predicate statement is used to construct Requirement No.2

The correct quantity of Cl_2 solution must be added as indicated by sensor measurements either before or after the contact tank.

Having constructed safety requirements, the next step is to convert the requirements to operational tasks and assess them.

6.3.3 Modelling and assessing safety requirements for disinfection system

The safety requirements are assessed in turn.

6.3.3.1 Requirement No. 1 - Disinfection system

Description: *If, after addition of SO_2 solution, the measured Cl in the treated water is outside set limits than the disinfection system must be isolated.*

Operational tasks

- Sensor measures residual Cl level
- Processor determines that the residual Cl level is outside set limits
- Inlet valve to the disinfection system is closed
- Electromagnetic flowmeter indicates no flow
- Outlet valve from disinfection system is closed
- Audible alarm is activated
- Messages shown on display

ETD: The ETD is shown in Fig. 6.11. The residual Cl sensor is positioned after the SO_2 mixer and it is used to determine the residual Cl in the final treated water. The processor checks if the measured residual Cl is within and upper and lower limit. If the residual Cl level is outside set limits then the inlet valve is closed preventing further flow of water into the disinfection system. The electromagnetic flowmeter is positioned after the inlet valve and indicates the amount of water flowing into the disinfection system. When the flowmeter indicates zero flow the outlet valve is closed, preventing water from entering the service reservoir. The operator is alerted to the situation by an audible alarm and by messages displayed on the control console.

Critical tasks: The processor task which determines residual Cl level is outside set limits is critical. Failure of this task may have very serious consequences.

Analysis of PROCESSOR task: Processor determines that the residual Cl level is outside set limits.

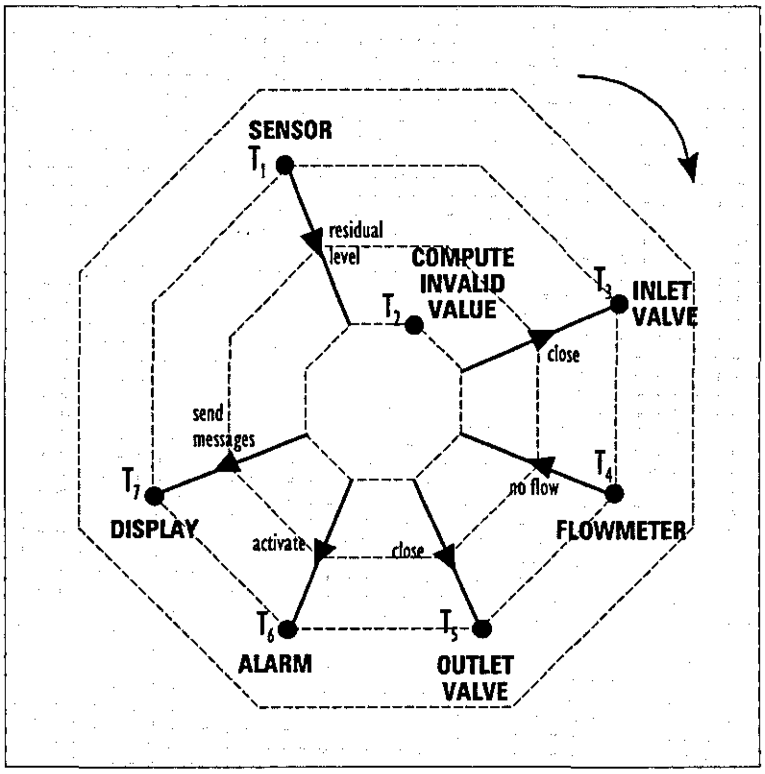


Fig. 6.11: ETD for Requirement No. 1 - Disinfection system

Specification

Definition:

Q: What is the task?

R: To determine whether residual Cl is within preset limits

Objective:

Q: Why is task required?

R: To determine whether the disinfection system should be taken off line.

Inputs/Outputs:

Q: What calculations and models are required?

R: The processor receives an analog value from a Cl sensor. This analog value is proportional to the residual Cl in the process water. The processor converts the analog value to a digital value which must lie between present upper and lower limits.

Timing/Control:

Q: How is this task to be initialised?

R: Sensor value is polled at intervals by the processor which subsequently calculates residual Cl.

Action 1: Check polling time.

Operational_Modes:

Q: What relationship does this task have to startup?

R: Task is prohibited during startup of the plant.

Q: What relationship does this task have to an emergency shutdown?

R: The task may initiate a shutdown of the disinfection system.

Implementation

Selection:

Q: What proprietary hardware will be used?

R: A special purpose residual Cl sensor will be used.

Installation:

Q: How will this task be implemented?

R: Sample water will be pumped from the process stream to the Cl sensor which will transmit results to the processor.

Q: If settings are to be altered on proprietary or customised boards, how will it be ensured that these settings are done correctly?

R: These settings are vital for safe treatment of water.

Action 2: Obtain full details relating to setting of parameters for Cl sensor.

Testing:

Q: How will integrity of proprietary hardware be checked?

R: Initially, the Cl sensor will be calibrated and checked with an established Cl analyzer in the laboratory. Before installation, it will be tested on an existing plant which operates under similar field conditions.

Maintenance:

Q: What maintenance procedures are required for this task?

R: None specified.

Action 3: Samples need to be taken directly from process water by operator, checked in the laboratory and compared with those logged by processor. All conditions are logged when/if disinfection system shuts down. Patterns need to be identified (e.g. Cl_2 and SO_2 injection rates, source of raw water, turbidity, pH).

Protection

Failure_Detection:

Q: What alarms are associated with this task?

R: An audible alarm in control room and display messages 'Treated water Cl level outside permitted range' and 'Plant tripped by disinfection system'.

Q: Why are the alarms required?

R: Once the disinfection system is shutdown, all water within the disinfection system must be dumped. Problems should be sorted as soon as possible otherwise plant has to be shutdown.

Q: How is it detected if this task fails to receive inputs or updates?

R: Once task initialised, it continually expects sensor to give non-zero values lying within preset limits. If it fails to receive a value, it assumes that the residual Cl level is outside preset limits.

Trips:

Q: What trips are associated with this task?

R: Inlet valve and outlet valve are closed.

Q: Why are these trips required?

R: To isolate the disinfection system from the rest of the plant.

Fault_Recovery:

Q: What fault recovery procedures are associated with this task?

R: None as yet specified in functional specification.

Action 4: Instigate fault recovery procedures.

Failure_Modes

Incorrectly_Initialised:

Q: What if task is incorrectly executed?

R: The task could execute incorrectly if the processor determined falsely that the residual Cl level is outside the upper/lower limits, or the processor determines falsely that the residual Cl level is within the upper/lower limits.

Action 5: An algorithm is required which tracks the variation in the residual Cl level. Perhaps this could be based on permutations of increasing and decreasing Cl_2 and SO_2 injection rates.

6.3.3.2 Requirement No. 2 - Disinfection system

***Description:** The correct quantity of Cl solution must be added as indicated by sensor measurements, either before or after the contact tank.*

Operational tasks

- Electromagnetic flowmeter measures flow rate
- Operator selects chlorinator
- Pressure switch indicates motive water supply ok
- Pressure switch indicates Cl_2 supply ok
- Sensor pre-contact tank measures level of residual Cl
- Cl_2 injection rate adjusted
- Sensor post-contact tank measures level of residual Cl
- Cl_2 injection rate is adjusted

ETD: The ETD is shown in Fig. 6.12. The electromagnetic flowmeter is located at the inlet to the disinfection system. Once the flowmeter indicates that water is entering the disinfection system, the operator can select one of two chlorinators via a local control panel. The motive water supply is used in conjunction with the Cl_2 gas injectors to form a Cl solution which is passed to the mixer in the disinfection system. The level of residual Cl is measured both before and after the water enters the contact tank. The measurements are used to control the Cl_2 injection rate.

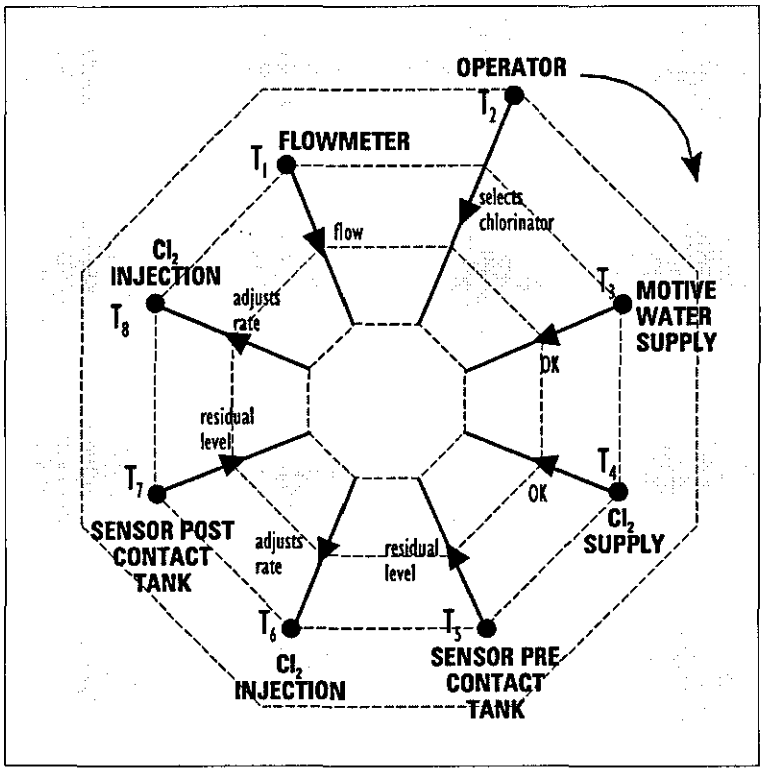


Fig. 6.12: ETD for Requirement No. 2 - Disinfection system

Critical tasks: The operator task is chosen for analysis as it is the only operator intervention in the sequence and it assists the analysis of the interaction between the disinfection and Cl dosing system.

Analysis of OPERATOR task: Operator selects chlorinator.

Specification

Definition:

Q: What is the operator intervention?

R: To initialise Cl dosing system so that Cl_2 is injected into process water.

Objective:

Q: Why is this operator intervention necessary?

R: The Cl_2 injection system is in itself hazardous and needs to be monitored carefully and only activated when necessary.

Inputs/Outputs:

Q: What form does this operator input have?

R: The operator selects one of two chlorinators via a switch unit.

Timing/Control:

Q: How are parameters initialized or re-initialised?

R: Each chlorinator has a metering device which adjusts the rate of Cl_2 injection. The injection rate is proportional to the flow of process water into the disinfection system. This proportional relationship is determined by a preset value. This preset value is trimmed by measurements of residual Cl both before and after the contact tank.

Operational_Modes:

Q: What relationship does this task have to an emergency shutdown?

R: If the disinfection system is tripped, flow to disinfection system will be stopped and this will automatically shut down chlorination dosing system.

Action 6: Cross check the emergency procedure for shutdown of the Cl_2 dosing system.

Implementation

Selection:

Q: What proprietary hardware will be used?

R: Special purpose metering devices for chlorination.

Installation:

Q: How will this task be implemented?

R: A mimic display is provided which shows chlorination streams, motive water supply streams and preset injection rate. Local switch panel allows selection of stream.

Q: Does operator need training?

R: All personnel involved with Cl_2 injection system need to be trained.

Action 7: Set up training plan for users; they have to be aware of the importance of the chlorination system and need to know how to monitor and control the process.

Testing:

Q: How will the task be checked against the functional specification?

R: Cl_2 injection can be monitored by checking the value obtained from the residual Cl sensor located before the contact tank.

Action 8: Set up test procedure. Under normal operation, selection of a chlorination stream has no effect unless the following conditions hold: (i) flowmeter signals non-zero flow into disinfection system (ii) motive water supply OK and (iii) Cl_2 supply OK. If any of these conditions is false and then no Cl_2 is injected into the process stream.

Environment:

Q: What particular aspects of the environment may affect the operation of this task?

R: There are several environmental factors to be considered which may affect the effectiveness of the Cl_2 injection e.g. pH, turbidity of water entering disinfection system, and properties of the raw water entering the plant (raw water may be supplied from one of 3 different sources).

Action 9: Implement plan for micro-organism sampling of treatment water to validate effectiveness of Cl_2 dosing.

Maintenance:

Q. What maintenance procedures are required for this task?

R: The Cl_2 dosing system has two chlorinators. In normal operation, one is in service and the other is on standby. For maintenance purposes, the chlorinator on standby can be taken offline and serviced while the system is still operational.

Q: How will maintenance procedures associated with this task affect normal operation?

R: If the chlorinator in service fails and the other chlorinator is offline being serviced, then the disinfection system must be shut down (i.e. automatic changeover of chlorinators is not possible).

Action 10: Incorporate requirement to disable automatic changeover if either chlorinator is being serviced.

Protection

Failure_Detection:

Q: What alarms are associated with this task?

R: Chlorination System Gas Meter failed. Motive water system failed.

Q: Why are these alarms required?

R: To signal that Cl_2 dosing system has failed.

Q: What are the alarm conditions/set values?

R: Measured pressure outside set limits for gas metering and motive water supply.

Interlocks:

Q: Are there any post-conditions associated with the completion of this task which can be checked to ensure that the task has executed successfully and on time?

R: Monitor value from residual Cl sensor located before contact tank.

Security:

Q: Can the sequence associated with this task be modified?

R: If a chlorinator is being serviced, only one chlorinator is available for selection.

Failure_Modes

Not_Initialised:

Q: What if operator fails to carry out task?

R: If a chlorinator is not selected, no Cl_2 will be injected into the process water in the disinfection system.

Action 11: In the disinfection process, the residual Cl level is measured three times. Check if relationship can be established between these three values.

6.3.4 Summary of assessment of disinfection system

The assessment focuses on one top level hazard (i.e. residual Cl in treated water). Two safety requirements were identified and only one task associated with each requirement was assessed. A small number of all possible questions were used for assessment purposes.

Eleven actions were listed as a result of the assessment, these are shown in Table 6.3. There is extensive use of instrumentation to measure critical parameters, however, better use could be made of the measurements obtained; in particular, the relationships and interaction between different parameters measured should be established (Actions 5 and 11). The functional specification is incomplete, this is illustrated by the large number of actions related to implementation (Actions 2, 3, 7, 8, 9 and 10). It is vital to crosscheck the emergency procedure for the Cl_2 dosing system (Action 6).

Table 6.3: Actions based on assessment of disinfection system

Action 1:	Check polling time.
Action 2:	Obtain full details relating to setting of parameters for Cl sensor.
Action 3:	Samples need to be taken directly from process water by operator, checked in the laboratory and compared with those logged by processor. All conditions are logged when/if disinfection system shuts down. Patterns need to be identified (e.g. Cl_2 and SO_2 injection rates, source of raw water, turbidity, pH).
Action 4:	Instigate fault recovery procedures.
Action 5:	An algorithm is required which tracks the variation in the residual Cl level. Perhaps this could be based on permutations of increasing and decreasing Cl_2 and SO_2 injection rates.
Action 6:	Cross check the emergency procedure for shutdown of the Cl dosing system
Action 7:	Set up training plan for users; they have to be aware of the importance of the chlorination system and need to know how to monitor and control the process.
Action 8:	Set up test procedure. Under normal operation, selection of a chlorination stream has no effect unless the following conditions hold: (i) flowmeter signals non-zero flow into disinfection system (ii) motive water supply OK and (iii) Cl_2 supply OK. If any of these conditions is false and then no Cl_2 is injected into the process stream.
Action 9:	Implement plan for micro-organism sampling of treatment water to validate effectiveness of Cl_2 dosing.
Action 10:	Incorporate requirement to disable automatic changeover if either chlorinator is being serviced.
Action 11:	In the disinfection process, the residual Cl level is measured three times. Check if relationship can be established between these three values.

6.4 Conclusions

The case studies demonstrate that HAZAPS provides a means of systematically assessing the safety aspects of a system, irrespective of the application domain. The application of the methodology is relatively straightforward, the major difficulty is in identifying and understanding the safety critical subsystems in the first place. There is a definite need for effective participation by different disciplines involved in the development of the system. Fault tree construction assists in targeting the functions of the system which can contribute to top level hazards. Construction of the ETDs provides an effective means of establishing, understanding and analysing operational tasks.

Answering the sets of questions forces the user to consider different safety aspects. The actions resulting from an assessment illustrate the 'total system' view of HAZAPS. There are actions related to environmental, human, hardware and software aspects of the system.

Although the primary objective of HAZAPS is to 'assess' (see p.66, first paragraph for definition of the term 'assess') the safety aspects of a system, it also complements and enhances the development of a system by identifying ambiguities, inconsistencies and incompleteness in the functional specification.

CHAPTER 7 :
Conclusions and further work

HAZAPS, a unique methodology for integrating hazard evaluation procedures and requirements engineering, makes a novel contribution to the field of re-use of incident data to permit feedback into the design of safety-critical systems. It is unique in that it demonstrates how:-

✦ non-functional requirements associated with safety can be captured from diverse disciplines;

✦ safety requirements can be formulated, logically represented, and analysed;

✦ safety considerations associated with the total system can be assessed.

To derive the methodology, new concepts, models, methods (and associated procedures) and a computer tool have been developed.

7.1 Novelty and benefits of HAZAPS

The following sections describe the benefits of HAZAPS and demonstrate its novelty.

7.1.1 Generic tasks allow a system-wide approach

A fundamental concept in HAZAPS is the 'generic task' which, applied in a novel way, permits the capturing of different types of knowledge and reduces uncertainty associated with development of a safety-critical system. A generic task can be defined as 'an activity carried out by the operator or any device associated with the target system which controls and/or responds to changes in the environment'. The importance of the generification of tasks has been discussed by Johnson et al. [1988] who used generic tasks to analyse data based on the observation of people carrying out tasks. **HAZAPS uses generic tasks in a novel way, as generic tasks permit the integrated analysis of not only the human aspect but also the hardware, software and the environmental aspects.**

7.1.2 The system model integrates software and safety engineering methods

A major problem in assessing the safety of programmable systems is in integrating safety engineering and software engineering methods. In HAZAPS, the system model (p.71) has been developed for this purpose. The system model effectively consolidates safety engineering and software engineering methods. Of the various approaches to integrating software engineering and safety engineering methods discussed in Section 2.3.3, only a few included a general system model (most had either no system model or an application-specific system model). In those that did have a general system model it was difficult to determine where to start/stop applying a particular technique or how the output from one technique fed to another technique. **The HAZAPS system model (p.71) is novel in that it permits different techniques to be used at different levels of abstraction and combines the strength of both deductive and inductive safety techniques** (i.e. it allows one to work backwards to the embedded system boundary to identify safety concerns, and work forwards from the programmable events to the system boundary). FTA permits tracing of the causes of hazards back to the embedded system boundary, and the ES-HAZOP and ES-FMEA procedures facilitate the identification of programmable events which may lead to hazardous states at the embedded system boundary.

7.1.3 An operational approach is used

An operational view is required to understand how failure in a system may lead to hazardous states. The approach in HAZAPS differs from the operational specification methods discussed in Section 2.2.3. **HAZAPS is novel in that it uses a combination of FTA and Task Analysis to synthesise operational tasks, and has an associated model, the ETD**

which provides an effective way of establishing, understanding and analysing tasks related to the high level fault tree representation of hazards.

7.1.4 The assessment framework is generic

The framework (Appendix 1) which is the basis of the ES-HAZOP and ES-FMEA procedures, provides a unique method of analysing safety requirements in a systematic and exploratory manner. Although using sets of questions for assessing the safety of a system is common practice, combining models of causation and sets of questions based on incident analysis, is rare. Hurst & Radcliffe [1994] developed an audit technique based on an incident classification scheme to investigate major hazards on offshore plant. However, no sample questions or answers were provided. It can only be assumed that the technique proved useful. Their assessment technique is for management use, considerations include: system climate, organisation and management, and communication and feedback. **HAZAPS is novel in that the sets of questions which are applicable to any embedded programming system, can be used for analysing low level tasks in conjunction with the ETD.** In combination with the HAZAPS tool, the framework provides a novel and very effective method not only for assessing the system under consideration, but for accumulating safety strategies for future systems.

7.1.5 The usefulness of HAZAPS has been demonstrated

Two case studies have demonstrated that HAZAPS is effective at assessing the safety of an embedded system. HAZAPS has been tested by two companies which subsequently provided reports (one written, one verbal). The reports made general comments regarding improving and adding features to the tool (e.g. addition of a designer's sketch pad). Both companies regarded HAZAPS as a sound methodology for safety-critical systems and one suggested that it would also be a useful tool for the development of non-safety-critical systems. **HAZAPS is novel in that no competing methodology exists.**

Fault tree construction assists in targeting the functions of the system which can contribute to top level hazards. Construction of the ETDs provides an effective means of establishing, understanding and analysing operational tasks. Answering the sets of questions forces the user to consider different safety aspects. The actions resulting from the application of the methodology illustrate the 'total system' view of HAZAPS. There are actions related to environmental, human, hardware and software aspects of the system. HAZAPS also complements and enhances the development of a system by identifying ambiguities, inconsistencies and incompleteness in the functional specification.

7.2 Limitations of HAZAPS

7.2.1 Domain analysis is difficult

Domain analysis is always difficult but this is particularly true in the case of safety-critical embedded systems. Lessons were learnt from the case studies. The author had in-depth experience of the rotary printing press but none of the water treatment plant. For the water treatment plant case study, three people were interviewed: the software programme manager responsible for the design of the automatic control system of the water treatment plant; a chemical engineer with limited experience of water treatment; and, a microbiologist with experience of testing water quality. What was surprising was how little knowledge each had of the others' expertise. The requirements engineer has to be flexible to reconcile the different viewpoints and understand the basic principles involved in the What and Why of the system. Also, the procedure for Stage 1 of HAZAPS requires a top level diagram and associated description, and this information could not be abstracted from the documentation provided for either case study. Further work is required on the domain analysis.

7.2.2 The assessment framework has not been proved complete

Completeness was discussed in Sections 2.3.2 and 2.4.2.3 in relation to both safety engineering methods (where it was mentioned how different techniques were needed to support each other) and the integration of safety and software engineering techniques. The term 'completeness' has been described formally as ... *the degree to which full implementation of required function has been achieved* [Pressman, 1994]. Here the term 'completeness' is used in relation to the framework; the assumption is that, if all hazards have been identified in the environment, then, for the framework to be complete, it must identify all possible contributions to the hazards associated with the embedded system. Arguments for claiming that the framework is complete include: the framework is based on a large number of incidents; there is in built redundancy in that ES-FMEA procedure cross checks the ES-HAZOP procedure; and, the framework has proved effective in two case studies. Counter arguments include: analysis of a large number of incidents and two case studies cannot prove completeness; the derivation of questions based on incidents could itself be flawed; and, the effectiveness of ES-FMEA in cross checking ES-HAZOP has not been verified.

Applying HAZAPS retrospectively to computer-related incidents which have occurred would not be sufficient to prove completeness because of the exploratory nature of the framework (i.e. after an incident it can always be assumed incorrectly that a particular question would have readily identified one of the events in the sequence which led to the incident). To validate the framework, HAZAPS should be applied to several more systems and these systems should be monitored throughout their lifecycle.

7.3 Further work

Recommendations for further work to extend and refine HAZAPS include:

✦ Indirect knowledge elicitation methods, such as card sort and ladder grid, could be tailored specifically for embedded systems and used to improve the domain analysis technique.

✦ It would be useful to carry out more case studies across different application domains and within the same application domain with the objectives of (a) customising HAZAPS and establishing generic safety requirements for particular application domains; (b) validating the question framework; and, (c) empirically identifying general strategies to reduce or eliminate risk (specific strategies have already emerged from the two case studies carried out which could be useful for future assessments of embedded systems: see p.133, Table 6.2 and p.148, Table 6.3).

✦ Formalisation and simulation of the ETD technique would be advantageous. HAZAPS allows abstraction of safety concerns in a systematic and effective way. The addition of a formal method (e.g. Finite State Machines, temporal logic) would complement HAZAPS by assisting in the detailed design and checking of the final specification. It is important that this formal method should be transparent to all but the software engineers on the project, in other words, all non-software specialists should be able to participate in the assessment of a system without having to acquire expertise in a formal mathematical method. Simulation of the ETD would enrich the notation in that scenarios could be executed and interpreted by different participants involved in system development.

✦ HAZAPS already has overlap with MIL-STD882C and it is essential to the future of HAZAPS to investigate how it could be linked to safety critical standards and used to support these standards. Information gleaned from standards could be used to strengthen HAZAPS and make it an even more powerful tool.

References

ABBOTT, A.: 'Cluster relaunch looks to minisatellites'. *Nature*, 1996, **382**, (6587), pp.102.

AIChemE: 'Guidelines for hazard evaluation procedures' (Center for Chemical Process Safety of the American Institute of Chemical Engineers, 1985)

AIChemE: 'Guidelines for investigating chemical process incidents' (Center for Chemical Process Safety of the American Institute of Chemical Engineers, 1992).

AIChemE: 'Guidelines for safe automation of chemical processes' (Center for Chemical Process Safety of the American Institute of Chemical Engineers, 1993).

AIChemE: 'Guidelines for technical management of chemical process safety' (Center for Chemical Process Safety of the American Institute of Chemical Engineers, 1989).

ALFORD, M.W.: 'A requirements engineering methodology for real-time processing requirements', *IEEE Trans. Soft. Eng.*, 1977, **SE-3**, (1), pp.60

AMELAN, R.: 'Software testing blamed for Ariane failure'. *Nature*, 1996, **382**, (6590), pp.386.

ANDRIOLE, S.J. and FREEMAN, P.A.: 'Software systems engineering: the case for a new discipline'. *Softw. Eng. J.*, 1993, **5**, pp.165-179.

ARTHUR, C.: 'Ambulance computer system was too complicated'. *New Scientist*, November 14, 1992, pp.7.

ATKINS, R.K.: 'Human dependability requirements, scope and implementation at the European Space Agency'. Proc. Annual Reliability and Maintainability Symposium, 1990 (IEEE, 1990) pp.85-89.

AZIcon: 'AZIcon Editor' (AZ Computer Innovations, Glendale, Arizona, USA, 1994).

BARLOW, P.R. and SMITH, D.J: 'Programmable safety related systems in the gas industry' *in* 'Computers and Safety', IEE Conference 8-10 Nov., 1990, Cardiff, UK, pp.28-29.

BELL, T.E., BIXLER, D.C. and DYER, M.E.: 'An extendable approach to computer-aided software requirements engineering'. *IEEE Trans. Soft. Eng.*, 1977, **SE-3**, (1), pp.49

BENYON, D. and SKIDMORE, S.: 'Towards a tool kit for the systems analyst'. *The Computer Journal*, 1987, **30**, (1), pp.2-7.

BOOCH, G.: 'Objected oriented design with applications' (The Benjamin/Cummings Publishing Co., Inc, Redwood, CA, USA, 1991).

BORNING, A.: 'Computer systems reliability and nuclear war'. *Commun. ACM*, 1987, **30**, (2), pp.112-131.

BOWEN, J. and STAVRIDOU, V.: 'Safety-critical systems, formal methods and standards'. *Softw. Eng. J.*, 1993, **7**, pp.189-209.

BRADLEY, E.A.: 'Determination of human error patterns: the use of published results of official enquiries into system failures'. *Qual. & Reliab. Engng. Int.,* 1995, **11**, pp.411-427.

BURNS, D.J., and PITBLADO, R.M.: 'A modified Hazop methodology for safety-critical system assessment' *in* REDMILL, F. and ANDERSON, T. (Eds.): 'Directions in Safety-critical Systems' (Springer-Verlag, 1993), pp.232-245.

CANNING, A.A.: 'The assessment of PES's in safety related applications: an update' *in* 'Computers and Safety', IEE Conference Publication No.314, 1990, pp.67-69.

CHILLAREGE, R., BHANDARI, I.S., CHAAR, J.K., HALLIDAY, M.J. MOEBUS, D.S., RAY, B.K. and WONG, M-Y.: 'Orthogonal defect classification - a concept for in-process measurements'. *IEEE Trans. Softw. Eng.* 1992, **18**, (11), pp.943-955.

CHUDLEIGH, M., BERRIDGE, C., BUTLER, J., MAY, R. and POOLE, I.: 'SADLI: functional programming in a safety-critical application'. *in* REDMILL, F. and ANDERSON, T.: 'Safety-critical systems: the convergence of high tech and human factors', Proc. 4th Safety-critical Systems Symposium, Leeds, UK, 6-8 February 1996 (Springer Verlag, 1996), pp.223-242.

CHUDLEIGH, M.F. and CLARE. J.N.: 'The benefits of SUSI: safety analysis of user system interaction' *in* GÓRSKI, J. (Ed.): 'Safety, security and reliability of computer based systems', 1993 (SAFECOMP'93) Symp., Poznań-Kiekrz, Poland, 27-29 October 1993, (Springer-Verlag, 1993), pp.124-132.

CIA: 'A guide to hazard and operability studies'. (Chemical Industry Safety and Health Council of the Chemical Industries Association, 1977).

COAD, P. and YOURDON, E.: 'Object-oriented analysis' (Prentice-Hall, Englewood Cliffs, N.J., 1991).

CRAIGEN, D., GERHART, S. and RALSTON, T.: 'Case study: traffic alert and collision - avoidance system'. *IEEE Soft.*, 1994, pp.35-39.

CURRIE, R.M. (revised by FARADAY, J.E.): 'Work Study'. (British Institute of Management, 3rd. ed., 1972).

CUTHILL, B.B.: 'Applicability of object-oriented design methods and C++ to safety-critical systems' *in* 'Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop' (NUREG/CP-0136) NIST Special publication 500-216, 1993, pp 163-191.

DAVIS, A.M.: 'A comparison of techniques for the specification of external system behavior'. *Commun. ACM*, 1988, **31**, (9), 1098-1115.

DAVIS, A.M.: 'Software requirements, objects functions and states' (Prentice-Hall International, Englewood Cliffs, New Jersey, 1993).

DE PANFILIS, S.: 'Fault avoidance through a development environment adopting prototyping' *in* LINDBERG, J.F.: 'Safety of computer control systems 1991 (SAFECOMP'91)' Safety, Security and Reliability of Computer Based Systems, Proc. IFAC Symp., Trondheim, Norway, 30 Oct - 1 Nov. 1991, (Pergamon Press, 1991), pp.125-130.

DIJKSTRA, E.W.: 'The structure of the THE-multiprogramming system'. *Commun. ACM*, 1968, **11**, (5), pp.341-346.

EDWARDS, K.: 'Real-time structured methods: systems analysis' (Wiley series in Software engineering practice, John Wiley & Sons, Chichester, England, 1993).

ELLIOTT, D.M. and OWEN, J.M.: 'Critical examination in process design'. *The Chemical Engineer*, November 1968, 377-383.

EMBLEY, D.W., JACKSON, R.B. and WOODFIELD, S.N.: 'OO systems analysis: is it or isn't it?'. IEEE Software, July 1995, p.19-33.

ESA: 'Software reliability modelling study: quantitative and qualitative reliability analyses of systems including software' (Aerosystems International Ltd., 1991).

FENELON, P. KELLY, T.P. and MCDERMID, J.A.: 'Safety cases for software application reuse' *in* RABE, G. (Ed.): 'Computer safety, reliability and security 1995' (SAFECOMP'95) Symp., Belgirate, Italy, 11-13 October 1995, (Springer-Verlag, 1995), pp.419-436.

FINK, R., OPPERT, S., COLLISON, P., COOKE, G., DHANJAL, S., LESAN, H. and SHAW, R.: 'Data management in clinical laboratory information systems' *in* REDMILL, F. and ANDERSON, T. (Eds.): 'Directions in safety-critical systems' (Springer-Verlag, 1993), pp.84-95.

FINKELSTEIN, A. and SOMMERVILLE, I.: 'Viewpoints in requirements engineering'. *Soft. Eng. J.*, 1996, **11**, (1), pp.2-4.

FLEISHMAN, E.A. & QUINTANCE, M.K.: 'Taxonomies of Human Performance' (Academic Press, Florida, USA, 1984).

FUSSELL, J.B. and VESELY, W.E.: 'A new methodology for obtaining cut sets'. *American Nuclear Soc. Trans.*, 1972, **15**, (1), pp.262-263.

FUSSELL, J.B., POWERS, G.J. and BENNETTS, R.G.: 'Fault trees - a state of the art discussion'. *IEEE Trans. Reliab.*, 1974, **R-23**, (1), pp.51-55.

GHAG, H.S.: 'HIpMATIC' (H.S. Ghag, Wolverhampton, West Midlands, WV10 OTE, 1994)

GÓRSKI, J.: 'Extending safety analysis techniques with formal semantics' *in* REDMILL, F. and ANDERSON, T. (Eds.): Proc. 2nd Safety-critical Systems Symposium, Birmingham, UK, 8-10 February 1994. (Springer-Verlag, 1994), pp.147-163.

GOUGH, P.A., FODEMSKI, F.T., HIGGINS, S.A. and RAY, S.J.: 'Scenarios - an industrial case study and hypermedia enhancements'. *ICRE '95*, 1995, pp.10-17.

GRADY R: 'Practical results from measuring software quality'. *Commun. ACM*, 1993, **36**, pp.62-68.

HAMMER, W.: 'Product safety management and engineering' *in* FABRYCKY, W.J. and MIZE, J.H.: Prentice Hall International Series in Industrial and Systems Engineering (Prentice Hall, 1980).

HAREL, D: 'Statecharts: a visual formalisation for complex systems', *Science of Computer Programming*, 1987, **8**, pp231-274.

HERBSLEB, J.D., KLEIN, H., OLSON, G.M., BURNNER, H., OLSON, J.S. and HARDING, J.: 'Object-oriented analysis and design in software project teams'. *Human-Computer Interaction*, 1995, **10**, pp.249-292.

HOBLEY, K.M. and JESTY, P.H.: 'Analysis and assessment of advanced road transport telematic systems' *in* RABE, G. (Ed.): 'Computer safety, reliability and security 1995' (SAFECOMP'95) Symp., Belgirate, Italy, 11-13 October 1995, (Springer-Verlag, 1995), pp.252-266.

HSE: 'Evaporating and other ovens' (Her Majesty's Stationery Office, London, 1981).

HSE: 'HSE guidelines for programmable electronic systems: generic aspects. Part 2: General technical guidelines' (Health & Safety Executive/Her Majesty's Stationery Office, London, 1987).

HSE: 'Out of control: why control systems go wrong and how to prevent failure' (Health & Safety Executive/Her Majesty's Stationery Office, 1995).

HURST, N.W. and RADCLIFFE, K.: 'Development and application of a structured audit technique for the assessment of safety management systems (STATAS)' *in* Hazards XII, European Advances in Process Safety. Vol. IChemE Symposium Series No.134. (IChemE, 1994), pp.315-339.

IBM: 'Business systems planning - information systems planning guide'. GE20-0527-1. I.B.M. Technical Publications.

IEC: 'Functional safety : safety related systems'. International Electrotechnical Commission, Commission, 65, IEC Draft Standard 1508, 1995.

IEE: 'Software in safety-related systems: a report prepared by a joint project team', Institution of Electrical Engineers and the British Computer Society, London, 1989.

IEE: 'Safety-related systems: a professional brief for the engineer' IEE Report, September 1992 (Institution of Electrical Engineers, London, 1992).

INPO: 'An analysis of root causes in 1983-84 significant event reports' (INPO - Institute of Nuclear Power Operations, Atlanta, Georgia, 1985).

IRWIN, A.: 'Cluster scientists set to sue'. *Times Higher Education Supplement*, 26 July 1996.

JACKSON, M. and ZAVE, P.: 'Domain descriptions' *in* Proc. IEEE International Symposium on Requirements Engineering, San Diego, USA, 4-6 January 1993 (IEEE, 1993), pp.56.

JACOBSON, I., CHRISTERSON, M., JONSSON, P. and ÖVERGAARD, G.: 'Object-oriented software engineering: a Use Case driven approach' (Addison-Wesley Publishing Co., Wokingham, England, 1993).

JAFFE, M.S., LEVESON, N.G., HEIMDAHL, M.P.E. and MELHART, B.E.: 'Software requirements analysis for real-time process-control systems'. *IEEE Trans. Softw. Eng.*, 1991, 17, (3), 241-258.

JARKE, M. BUBENKO, J., ROLLAND, C., SUTCLIFFE, A. and VASSILIOU, Y.: 'Theories underlying requirements engineering: an overview of nature at genesis *in* Proc. IEEE International Symposium on Requirements Engineering, San Diego, USA, 4-6 January 1993 (1993), pp.19-31.

JENTSCH, F.G.: 'Problems of systematic safety assessment: lessons learned from aircraft accidents' *in* WISE, J.A., HOPKIN, V.D. and STAGER, P.: 'Verification and validation of complex systems: human factors issues' Procs. NATO Advanced Study Institute on Verification and Validation of Complex and Integrated Human-Machine Systems (Springer-Verlag, Berlin, Germany, 1993), pp.251-259.

JESTY, P.H., LAMB, D.A., BUCKLEY, T.F. and WEST, M.M.: 'Choice of design methodologies for demanding high integrity industrial control systems *in* Proc. 3rd International Conference on Software Engineering for Real Time Systems, Cirencester, UK, 16-18 September 1991 (IEE, 1991), pp.16-21.

JIROTKA, M. and GOGUEN, J.: 'Introduction' *in* JIROTKA, M. and GOGUEN, J. (Eds.): Requirements Engineering: Social and Technical Issues (Academic Press, 1994), pp.1-13.

JOHNSON, P.: 'Human-computer interaction: psychology task analysis and software engineering' (McGraw-Hill International (UK) Ltd., 1992).

JOHNSON, P., JOHNSON, H., WADDINGTON, R. and SHOULS, A.: 'Task-related knowledge structures: analysis, modelling and application'. in JONES, D.M. and WINDER, R. (Eds.): Proceedings of the Fourth Conference of the British Computer Society Human-Computer Interaction Specials Group, University of Manchester, 5-9 Sept 1988. People and computers IV. 1988, pp.35-62.

JOHNSON, W.G.: 'MORT safety assurance systems'. (Marcel Dekker Inc., New York, 1980).

JONES, P.G.: 'Computers in chemical plant - a need for safety awareness' *in* GIBSON, N. (Ed.): 'Hazards XI: New Directions in Process Safety. Vol. IChemE Symposium Series No.124. (IChemE, 1991), pp.289-297.

JONES, D: 'Nomenclature for hazard and risk assessment in the process industries. 2nd ed. IChemE, 1992. p.3.

KEDZIERSKI, B.: 'Communication and management support in system development environments' *in* GREIF, I. (Ed.): Computer-Supported Cooperative Work (Morgan Kaufman, 1988).

KERSHAW J: 'The special problems of military systems'. *Microprocessors and Microsystems,* 1993, **17**, pp.25-30.

KIRWAN, B. and AINSWORTH, L.K. (Eds.): 'A guide to task analysis' (Taylor & Francis Ltd., London, 1992).

KIRWAN, B.: 'A guide to practical human reliability assessment' (Taylor & Francis Ltd., London, 1994).

KLEIN, W.E. and LALI, V.R.: 'Model-OA wind turbine generator: failure modes & effects analysis' *in* Proc. Annual Reliability and Maintainability Symposium, 1990.
KLETZ, T.A.: 'HAZOP and HAZAN - Identifying and assessing process industry hazards'. 3rd edition (Institution of Chemical Engineers, Rugby, UK, 1992).

KLETZ, T.A.: 'Some incidents that have occurred, mainly in computer-controlled process plants' *in* KLETZ, T., CHUNG, P.W.H, BROOMFIELD, E.J. and SHEN-ORR, C.: 'Computer control and human error' (IChemE, Rugby, UK, 1995), pp.3-44.

KOLB, J. and ROSS, S.S. (Eds.): Product Safety and Liability: a Desk Reference (McGraw Hill, 1980).

LAPRIE, J-C.: 'Dependability: from concepts to limits' *in* GÓRSKI, J. (Ed.): Safety, Security and Reliability of Computer Based Systems, 1993 (SAFECOMP'93) Symp., Poznań-Kiekrz, Poland, 27-29 October 1993, (Springer-Verlag, 1993), pp.157-168.

LEMOS, R. de, SAEED, A. and ANDERSON, T.: 'Analyzing safety requirements for process-control systems'. *IEEE Software,* 1995, (5), pp.42-53.

LEVESON, N.G. and STOLZY, J.L.: 'Safety analysis of Ada programs using fault trees'. *IEEE Trans. Reliab.*, 1983, **R-32**, (5), pp.479-484.

LEVESON, N.G.: 'Safeware: system safety and computers' (Addison-Wesley, 1995).

LEVESON, N.G.: 'Software safety in embedded computer systems'. *Commun. ACM*, 1991, **34**, (2), pp.34-26.

LINDLAND, O.D., SINDRE, G. and SOLVBERG, A.: 'Understanding quality in conceptual modelling'. *IEEE Softw.* March 1994, pp.42-50.

LUTZ, R.R.: 'Targeting safety-related errors during software requirements analysis'. *SIGSOFT Softw. Eng. Notes*, 1993, **18**, (5), pp.99-106.

MAIER, T.: 'FMEA and FTA to support safe design of embedded software in safety-critical systems' *in* Proc. 12th Annual CSR Workshop/1st Annual Encress Conference on Safety and Reliability of Software Based Systems, 1995.

MALCOLM, B.:'The UK safety critical systems research programme'. *Reliab. Eng. Syst. Saf.*, 1994, **43**, pp.233-244.

MARTIN, J.N.T.: 'On mapping real systems'. *J. Appl. Sys. Anal.*, 7, April, 1980, pp.151-156.

McDERMID, J.A.: 'Support for safety cases and safety arguments using SAM'. *Reliab. Engng. Sys. Saf.*, 1994, **43**, pp.111-127.

McDERMID, J.A. : 'Software hazard and safety analysis: opportunities and challenges. *in* REDMILL, F. and ANDERSON, T.: Safety-critical systems: the convergence of high tech and human factors. Proc. 4th Safety-critical Systems Symposium, Leeds, UK, 6-8 February 1996 (Springer Verlag, 1996), pp.209-222.

McHUGH, J.: 'The role of formal specifications' *in* Proc. Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP-0136)' NIST Special Publication 500-216, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, Maryland, 1993, pp.139-147.

MEALY, G.H.: 'A method for synthesizing sequential circuits'. *Bell System Tech. Journal*, 1955, **34**, pp.1045-1079.

MILLER, G.: 'The magical number seven, plus or minus two: some limits on our capatcity for processing information'. *The Psychological Review*, 1956, **63**, (2), pp.86.

MIL-STD-882C 'Military standard: system safety program requirements'. MIL-STD-882C, Department of Defense, Washington DC 20301, USA, 19 January 1993.

MoD 00-55: 'The procurement of safety-critical software in defence equipment'. Interim Defence Standard 00-55, Issue 2, Ministry of Defence, Directorate of Standardisation, Kentigern House, 65 Brown Street, Glasgow G2 8EX, UK, 5 April 1991.

MoD 00-56:'Safety management requirements for defence systems containing programmable electronics' Draft Defence Standard 00-56, Ministry of Defence, Directorate of Standardisation, Kentigern House, 65 Brown Street, Glasgow G2 8EX, UK, February 1993.

MoD 00-58: Draft interim DEF-STAN 00-58. 'A Guideline for HAZOP Studies on Systems which include a Programmable Electronic System'. U.K. Ministry of Defence 1995.

MOJDEHBAKHSH, R., TSAI, W-T. and KIRANI, S.: 'Retrofitting software safety in an implantable medical device'. *IEEE Softw.*, 1994, Jan., pp.41-50.

MONTAGUE, D.F.: 'Process risk evaluation - what method to use?'. *Reliab. Eng. Syst. Saf.*, 1990, **29**, pp.27-53.

MYERS, G.J.: 'Reliable software through composite design' (Mason/Charter Publishers Inc., 1975).

MYERS, G.J.: 'Software reliability' (John Wiley & Sons, New York, 1976).

NASA: 'CLIPS' (Software Technology Branch, NASA, Lyndon B. Johnson Space Center, USA, 1994)

NEIGHBORS, J.: 'The Draco approach to constructing software from reusable components'. *IEEE Trans. Soft. Eng.*, 1984, **SE-10**, (9), pp.564-573

NEUMANN, P.G.: 'Computer related risks' (ACM Press, New York, 1995).

NOLAN, D.P.: 'Application of HAZOP and What-If safety reviews to the petroleum, petrochemical and chemical industries' (Noyes Publications, 1994).

NTSB: Reports of the National Transportation Safety Board (distributors: National Technical Information Service (NTIS), Springfield VA 22151, USA).

OZOG, H.: 'Hazard identification, analysis and control'. *Chem. Eng.*, Feb. 1985, pp.161-170.

PARNAS, D.L., SCHOUWEN, J. VAN and KWAN, S.P.: 'Evaluation of safety-critical software'. *Commun. ACM*, 1990, **33**, (6), pp. 636-648.

PENG, W.W. and WALLACE, D.R.: 'Software error analysis'. NIST Special Publication 500-209, Computer Systems Technology, National Institute of Standards and Technology, Gaithersburg, Maryland, 1993.

PORTER, A.A., VOTTA, L.G., and BASILI, V.R.: 'Comparing detection methods for software requirements inspections: a replicated experiment'. *IEEE Trans. Soft. Eng.*, 1995, **21**, (6), pp.563-575.

PRESSMAN, R.S.: 'Software Engineering: A Practitioners Approach'. European Edition. (McGraw Hill, 1994).

PROROK, J., BÜHRER, K. AMMANN, U. and VIT, K.: 'Design and planning in the development of safety-critical software with ADA' *in* FREY, H.H. (Ed.): 'Safety of computer control systems 1992' (SAFECOMP'92), Computer Systems in Safety-critical Applications, Proc. IFAC Symp., Zürich, Switzerland, 28-30 October 1992 (Pergamon Press, 1992) pp.75-80.

PROSSER, J.: 'Airbus A320' *in* 'In-flight magazine for Excalibur Airways', Summer 1993 edition. (Dennis Fairey & Assocs. Ltd., 1993), pp.24-26.

RAF: 'Critical Human Factors Incident Reporting Programme (CHIRP)' Sponsored by the CAA (RAF Institute of Aviation Medicine: Farnborough, Hants., UK, 1989)

RANDALL, P.E.: 'Introduction to work study and organization and methods' (Butterworths, 1969) pp.140.

RAVN, A.P., RISHCEL, J. and STAVRIDOU, V.: 'Provably correct safety-critical software' *in* DANIELS, B.K. (Ed.): 'Safety, security and reliability related computers for the 1990s' (SAFECOMP'90), Proc. IFAC Symp., Gatwick, UK, 30 October-2 November 1990, (Pergamon Press, 1990), pp.13-18.

REDMILL, F.: 'Software in safety-critical applications - a review of current issues' *in* REDMILL, F. and ANDERSON, T. (Eds.): 'Safety-critical systems' (Chapman & Hall, 1993) pp.3-15.

REIFER, D.J.: 'Software failure modes and effects analysis'. *IEEE Trans. Reliab.*, 1979, **R-28**, (3), pp.247-249.

REUNANEN, M. and HEIKILÄ, J.: 'A method for considering safety and reliability in automation design' *in* LINDBERG, J.F. (Ed.): 'Safety of computer control systems 1991' (SAFECOMP'91), Safety, Security and Reliability of Computer Based Systems, Proc. IFAC Symp., Trondheim, Norway, 30 Oct - 1 Nov 1991, (Pergamon Press, 1991), pp.107-112.

ROBINSON, B.W.: 'Application of Hazard and Operability Studies to a wide range of industries and activities'. *Qual. & Reliab. Engng. Int.*, 1995, **11**, pp.399-402.

ROSNESS, R.: 'Limits to analysis and verification' *in* WISE, J.A., HOPKIN, V.D. and STAGER, P. (Eds.): Proc. NATO Advanced Study Institute on Verification and Validation of Complex and Integrated Human-Machine Systems, Vimeiro, Portugal, 6-17 July 1992 (Springer-Verlag, 1993), pp.181-191.

ROSS, D.T.: 'Structured analysis (SA): a language for communicating ideas'. *IEEE Trans. Softw. Engng.*, 1977, SE-3, (1), 16-28.

RTCA DO-178B: 'Software considerations in airborne systems and equipment certification'. DO-178B, RTCA Inc., Suite 1020, 1140 Connecticut Avenue NW, Washington CD 20036, USA, 1 Dec 1992.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. and LORENSON, W.: 'Object-oriented modelling and design' (Prentice-Hall, 1991).

RUMBAUGH, J.: 'To form a more perfect union: unifying the OMT and Booch methods'. *J. Object Oriented Programming*, Jan 1996, p.14-18.

RUSHBY, J.: 'Critical system properties: survey and taxonomy'. *Reliab. Eng. Syst. Saf.*, 1994, **43**, pp.189-219.

RUSHBY, J.: 'Formal methods and their role in the certification of critical systems' *in* Proc. 12th Annual CSR Workshop/1st Annual Encress Conference on Safety and Reliability of Software Based Systems, 1995.

RUSHTON, A.G.: 'Hazard and operability study of offshore installations - a survey of variations in practice' *in* Hazards XII, European Advances in Process Safety. Vol. IChemE Symposium Series No.134. (IChemE, 1994), pp.341-350.

SAEED, A., DE LEMOS, R. and ANDERSON, T.: 'Safety analysis for requirements specifications: methods and techniques' *in* RABE, G. (Ed.): 'Computer safety, reliability and security 1995' (SAFECOMP'95) Symp., Belgirate, Italy, 11-13 October 1995, (Springer-Verlag, 1995), pp.27-41.

SCHOITSCH, E., DITTRICH, S., GRASEGGER, D., KROPFITSCH, D., ERB, A., FRITZ, P. and KOPP, H.: 'The Elektra testbed: architecture of a real-time test environment for high safety and reliability requirements' *in* DANIELS, B.K. (Ed.): 'Safety, security and reliability related computers for the 1990s' (SAFECOMP'90), Proc. IFAC Symp., Gatwick, UK, 30 October-2 November 1990, (Pergamon Press, 1990), pp.59-65.

SEWARD, D.W., MARGRAVE, F.W., SOMMERVILLE, I. and KOTONYA, G.: 'Safe systems for mobile robots: the Safe-SAM project' *in* REDMILL, F. and ANDERSON, T.: 'Achievement and assurance of safety', Proc. Safety-critical Systems Symposium, Brighton, UK, 7-9 February 1995, (Springer-Verlag, 1995), pp.153-170.

SHEBALIN, P.V., SON, S.H. and CHANG, S-Y.: 'An approach to software safety analysis in a distributed real-time system' *in* COMPASS '88. (IEEE, 1988) pp.29-43.

SMART, J.: 'wxCLIPS' (Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, 1996).

SMITH, D.J. and WOOD, K.B.: 'Engineering quality software' (Elsevier Applied Science, 2nd ed., 1989).

SOMMERVILLE, I., RODDEN, T., SAWYER, P., BENTLEY, R. and TWIDALE, M.: 'Integrating ethnography into the requirements engineering process' *in* Proc. IEEE International Symposium on Requirements Engineering, San Diego, USA, 4-6 January 1993 (1993), pp.165-173.

STEPP, R. and MICHALSKI, R.: 'Conceptual clustering of structured objects: a goal-oriented approach'. *Artificial Intelligence*, 1986, **28**, pp.53.

SUOKAS, J. and ROUHIAINEN, V.: 'Quality control in safety and risk analyses'. *J. Loss Prev. Process Ind.*, 2, April 1989, 67-77.

TAYLOR, J.R.: 'Use of "lessons learned" for in-depth hazards analysis'. *J. Loss Prev. Proc. Ind.*, 1989, **112**, pp.15-17.

TAYLOR, J.R.: 'Hazard identification' Chapter 3, 'Risk Analysis for Process Plant, Pipelines and Transport' (E & FN Spon, 1994A).

TAYLOR, J.R.: 'Developing safety cases for command and control systems' *in* REDMILL, F. and ANDERSON, T. (Eds.): 'Technology and assessment of safety-critical systems'. Proc. 2nd Safety-critical Systems Symposium, Birmingham, UK, 8-10 February 1994 (Springer-Verlag, 1994B), pp.69-78.

UMPHRESS, D.A. and MARCH, S.G.: 'Object-oriented requirements analysis'. *J. of Object-Oriented Programming*, 1991, pp.35-40.

WALLACE, D.R. and IPPOLITO, L.M.: 'A framework for the development and assurance of high integrity software', NIST Special Publication 500-223, December 1994, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, Maryland, 1993.

WARD, P.T.: 'The transformation schema: an extension of the data flow diagram to represent control and timing'. *IEEE Trans. Softw. Eng.*, 1986, **SE-12** (2), 198-210.

WARD, P.T. and MELLOR, S.J.: 'Structured development for real-time systems: Vols.1-3: Implementation modelling techniques' (Yourdon Computing Series, Yourdon Press, Englewood Cliffs, New Jersey, 1985).

WELLS, G. and WARDMAN, M.: 'Risk and safety reviews' *in* REDMILL, F. and ANDERSON, T. (Eds.): Proc. 2nd Safety-critical Systems Symposium, Birmingham, UK, 8-10 February 1994. (Springer-Verlag, 1994), pp.128-146.

WHALLEY, S.P.: 'Minimising the cause of human error' *in* LIBBERTON, G.P. (Ed.): Proc. 10th Advances in Reliability Technology Symposium, Bradford, UK, 6-8 April 1988 (Elsevier, 1988), pp.114-128.

YADAV, S.B., BRAVOCO, R.R., CHATFIELD, A.T. and RAJUKMAR, T.M.: 'Comparison of analysis techniques for information requirements determination'. *Commun. ACM*, 1988, **31**, (9), pp.1090-1097.

YIH, S, CHIN-FENG, F. and, SHIRAZI, B.: 'Anatomy of safety-critical computing problems'. *Reliab. Eng. Syst. Saf.*, 1995, **50**, (1), pp.69-78.

YOUNT, L.J.: 'Generic fault-tolerance techniques for critical avionics systems' *in* Proc. of AIAA Guidance and Control Conference, Snowmass, CO, 1985, pp.1-5.

ZAVE, P. & YEH, R.T: 'Executable requirements for embedded systems' *in* GEHANI, N. and McGETTRICK, A.D.: 'Software specification techniques (Addison-Wesley, 1986), pp.341-360.

ZAVE, P.: 'An insider's evaluation of PAISLey'. *IEEE Trans. Soft. Eng.*, 1991, **17**, (3), pp.212-216.

ZAVE, P.: 'An operational approach to requirements specification for embedded systems' *in* GEHANI, N. and McGETTRICK, A.D.: 'Software specification techniques' (Addison-Wesley, 1986), pp.131-169.

ZAVE, P.: 'The operational versus the conventional approach to software development'. *Commun. ACM,* 1984, **27**, (2), pp.104-118.

Appendix 1
Assessment Framework

Superclass : Specification

Class : Definition - **What** is to be achieved?

Slots	***Values***
Processor	What is the task?
Comms	What communication link is required?
Sensor	What state is to be monitored?
HID	What Human Input Device is required?
Display	What is to be displayed?
Actuator	What action is required?
Operator	What is the operator intervention?

Class : Objective - **Why** is it to be achieved?

Slots	***Values***
Processor	Why is this task required?
Comms	Why is this communication link required?
Sensor	Why does this state need to be monitored?
HID	Why is this Human Input Device required?
Display	Why does this information need to be displayed?
Actuator	Why is this action required?
Operator	Why is this operator intervention necessary?

Class : Options - **How** else could it be achieved?

Slots	***Values***
Processor	What other way could this task be accomplished?
Comms	What type/s of communication protocols would be suitable?
Sensor	What alternative states, methods would be suitable?
HID	What types of *HID* would be suitable?
Display	What would be the best way of attracting the operator's attention?
Actuator	What type/s of transducers would be suitable?
Operator	What would be the best way of allowing the operator to intervene?

Class : Inputs/outputs - **What** inputs and/or outputs are required?

Slots	***Values***
Processor	What are the inputs/outputs for this task?
Processor	What calculations and models are required and how will these be verified?
Processor	What parameters are associated with this task?
Processor	Are there any transfers of data to files?
Processor	Are there any P. . M/. AM transfers?
Comms	What is the approximate distance between the two communicating devices?
Comms	What is the format of the information to be transmitted on this communication link?
Sensor	. ver what range is the signal to be monitored?
HID	What ranges are associated with this input device?
Display	Is it status or control information?
Display	What is the format for displaying information?
Actuator	What is the range of this output signal?
Operator	What form does this operator input have?

Superclass : Specification Cont./....

Class : Timing/control - **When** is it to be achieved and **How** is it to be controlled?

Slots	***Values***
Processor	How is this task to be initialized?
Processor	How will data be transferred to/from tasks?
Processor	How are parameters initialized or reinitialized?
Processor	How is this task to be terminated?
Processor	What are the preconditions for termination?
Processor	What are the preconditions for initialization?
Comms	Is a synchronous or asynchronous protocol required and, if so, why?
Comms	How is this communication link controlled?
Comms	Is this a bidirectional communication link?
Comms	What response time is required?
Comms	How fast does the data need to be transmitted?
Comms	What terminates communication on this link?
Comms	What initiates communication on this link?
Sensor	When does this state have to be measured?
Sensor	How often does this state have to be scanned?
Sensor	If multiple sensors are to be used to monitor a state, what strategy will be adopted? What variations are due to the positioning of these devices? Will these variations remain constant with time?
Sensor	How fast does the response have to be?
HID	How often will this input facility be used?
HID	When will this input facility be used?
Display	How often does this information need to be updated?
Actuator	What response time is required?
Actuator	How will the system know that this action is required?
Actuator	If multiple actuators are required for a given task, what strategy is used to ensure that actuators work together?
Actuator	How frequently will this output be used?
Actuator	When is this action required?
Operator	When is this intervention required?

Class : Operational_Modes - **What** operational mode/s (startup, shutdown, automatic, manual etc.) are involved?

Slots	***Values***
Processor	What relationship does this task have to startup?
Processor	What relationship does this task have to normal shutdown?
Processor	What relationship does this task have to emergency shutdown?
Processor	What relationship does this task have to automatic mode?
Processor	What relationship does this task have to manual mode?
Operator	How will a smooth changeover from manual to automatic be achieved?

Class : Programmable - **Why** is this task programmable?

Slots	***Values***
Processor	Should this task be programmable or can it be hardwired?
Sensor	Should access to this state be programmable or can it be hardwired?
HID	Should this input be programmable or can it be hardwired?
Display	Is a programmable display required?
Actuator	Should this output be programmable or can it be hardwired?
Operator	Should this intervention be carried out by the programmable system or by hardwiring?

Superclass : Implementation

Class : Selection - **What** device/s are required?

Slots	***Values***
Processor	What customized boards will be used?
Processor	What proprietary hardware will be used?
Processor	What storage requirements are necessary?
Comms	What communications protocol will be used?
Comms	What interface will be used (e.g. RS232, Current loop)?
Sensor	What sensor will be used?
HID	What type of *HID* will be used?
Display	What type of display will be used?
Actuator	What actuator will be used?

Class : Installation - **How** will the installation be carried out?

Slots	***Values***
Processor	How will this task be implemented?
Processor	If settings need to be altered on proprietary or customized boards, how will it be ensured that these settings are done correctly?
Comms	What status indications are required?
Comms	Is provision required for modification or expansion at a later stage?
Sensor/ HID/ Display/ Actuator	How will this device be installed?
Sensor/ HID/ Display/ Actuator	How will this device be interfaced to the system?
Sensor/ HID/ Display/ Actuator	How will this device be calibrated?
Sensor/ HID/ Display	Where will this device be positioned?
Sensor	Is position representative of state being measured?
HID	If other devices are to be used in conjunction with this device, what is the best logical layout?
HID	Is the purpose of this device clear to the operator?
Display	If this display is used to show other information how is information unique to this task distinguished?
Operator	Does operator need training?

Class : Testing - **How** will the implementation be tested?

Slots	***Values***
Processor	How will the implementation be tested?
Processor	How do you know if these requirements are sufficient?
Processor	How will customized boards be tested?
Processor	How will the integrity of proprietary hardware be checked?
Processor	For hardwired connections, how is the logic to be tested?
Processor	How will the task be checked against the functional specification?
Processor	What reliability data are available on hardware items?

Superclass : Implementation Cont/....

Class : Environment - **What** effect will the environment have on this task?

Slots	***Values***
Processor	What particular aspects of the environment may affect the operation of this task?
Comms	Is the environment noisy requiring screened or fibre optic cable?
Sensor/HID/Display/Actuator	Is this device robust enough for environment and number of operations?
Sensor/HID/Display/Actuator	Is electromagnetic protection required for this device?
Sensor/HID/Actuator	Is noise filtering/rejection required for this device?
Sensor/HID/Display/Actuator	What particular aspects of the environment may affect the operation of this task?
Display	Where can this display be placed for most effective use by the operator?
Operator	Is this intervention easy to perform even in a stressful situation?

Class : Maintenance - **What** maintenance procedures are required?

Slots	***Values***
Processor	What maintenance procedures are required for this task?
Processor	How will maintenance procedures associated with this task affect normal operation?

Class : Utilities - **What** utilities (power, air, etc.) are required?

Slots	***Values***
Processor	What utilities are required for this task?

Superclass : Protection

Class : Failure_Detection - **How** will any failures be detected?

Slots	***Values***
Processor	What alarm(s) are associated with this task?
Processor	Why are these alarm(s) required?
Processor	What are the alarm conditions/set values?
Processor	How does the system know if this task has failed?
Processor	How are erroneous/invalid inputs or updates to this task detected?
Processor	How are erroneous/invalid outputs or updates from this task detected?
Processor	How is it detected if this task fails to receive inputs or updates?
Processor	How is it detected if this task fails to transmit outputs or updates?
Processor	How is it detected if an associated task fails to execute or executes incorrectly?
Comms	What error indications are associated with this communication link?
Comms	Is error detection required?
Sensor	Is a continuous self-test sequence required?
Actuator	Is there any method of verifying or correlating output data to detect out of range values?
Sensor/HID/Actuator	How will the system know if any of the hardware devices associated with this task have failed?
HID	How will the system know if the *HID* has given an invalid or erroneous signal?
Display	How will failure of display task be detected?

Superclass : Protection Cont./.....

Class : Interlocks - **How** are hazardous events prevented?

Slots	***Values***
Processor	How is it ensured that all preconditions have been identified?
Processor	Do the preconditions hold only when task is required to execute? If not, how is unintended initialization of this task prevented?
Processor	Do the preconditions hold only when task is required to terminate? If not, how is unintended termination of this task prevented?
Processor	Are there any sustaining conditions associated with this task?
Processor	Are there any postconditions associated with the completion of this task which can be checked to ensure that the task has executed successfully and on time?
HID	If this input facility is only to be used when the machine is in a certain state, how is its use prevented in other states?
Operator	Is the operator intervention only available at certain times during the process, if so, how can the operator be prevented from intervening at other times?
Operator	What prevents the operator from ignoring displays/alarms etc?

Class : Trips - **How** will the system be shut down if a hazard is identified?

Slots	***Values***
Processor	What trips are associated with this task?
Processor	Why are these trips required?
Processor	What are the associated trip conditions/values?
Processor	How are these trips implemented?
Processor	What trip recovery procedures are associated with this task?
Operator	What trips are associated with this intervention?
Sensor/HID/Actuator	What trips are associated with this task?
Sensor/HID/Actuator	What values are trips set at?

Class : Security - **How** will breaches of security be prevented?

Slots	***Values***
Processor	What security measures are required with this task?
Processor	Can the sequence associated with this task be modified?
Processor	What parameters associated with this task can be modified by the operator?
Processor	Why can these parameters be modified by the operator?
Processor	Can the hardware or hardware settings associated with this task be modified, if so why is this necessary?
Processor	Can associated alarms be disabled, and if so why?
Processor	How can operator be prevented from illegally modifying software?
Processor	How can the operator be prevented from illegally modifying hardware?

Class : Fault_Recovery - **What** fault recovery procedures are associated with this task?

Slots	***Values***
Processor	What fault recovery procedures are associated with this task?
Processor	What emergency procedures are associated with this task?
Processor	What error correction mechanisms are required?
Processor	What procedures are followed if parameters have to be reinitialized during processes?
Display	If alarm under what conditions information, can it be made clear to operator what action is required, what has gone wrong?
Operator	Is it clear what the operator has to do in emergency situations?

Superclass : Protection Cont./....

Class : Verification - **How** will the Fail Safe/Protection features be verified?

Slots	***Values***
Processor	How will failure detection mechanisms be verified?
Processor	How will interlock mechanisms be verified?
Processor	How will trips be verified?
Processor	How will security measures be verified?
Processor	How will fault recovery emergency procedures be verified?

Superclass : Failure_Modes

Class : Not_Initialized

Slots	***Values***
Processor	Task not initialized
Operator	Operator fails to carry out tasks
Display	What if information is not shown on display?

Class : Incorrectly_Initialized

Slots	***Values***
Processor	Task incorrectly initialized
Sensor/HID/Display/Actuator	No default settings
Sensor/HID/Display/Actuator	Uncalibrated
Operator	Operator interferes at wrong time in process
Operator	Operator thinks there is a failure when there is none

Class : Incorrectly_Executed

Slots	***Values***
Processor	Task incorrectly executed
Sensor/HID/Display/Actuator	Incorrect value
Sensor/HID/Display/Actuator	Hunting
Display	Too complex or too much information
Operator	Operator carries out task wrongly

Class : Not_Terminated

Slots	***Values***
Processor	Task not terminated

Class : Incorrectly_Terminated

Slots	***Values***
Processor	Task incorrectly terminated
Operator	Operator stops task at wrong time

Superclass : Failure_Modes Cont./....

Class : Erroneous/corrupt_Operation

Slots	***Values***
Processor	Erroneous or corrupt operation
Sensor/HID/Display/Actuator	Invalid or corrupt signal
Sensor/HID/Display/Actuator	Noisy signal
Display	Incompatible
Display	Inconsistent
Operator	Erroneous input by operator

Class : No_Input/output

Slots	***Values***
Processor	No input/output to task
Comms	Communication device does not receive/transmit signal
Sensor/HID	Device does not transmit signal
Display/Actuator	Device does not receive signal

Class : Incorrect_Input/output

Slots	***Values***
Processor	Incorrect input/output to task
Sensor/HID/Actuator	Signal too high
Sensor/HID/Actuator	Signal too low

Class : Lockup

Slots	***Values***
Processor	Task locks up
Comms	Communications link lock up
Comms	Communications link time out
Sensor/HID/Actuator	Frozen/jammed
Display	Display not updated

Class : Too_Fast

Slots	***Values***
Processor	Task operates too fast
Sensor/HID/Actuator	Signal too fast
Display	Display updated too fast
Operator	Operator action too fast

Class : Too_Slow

Slots	***Values***
Processor	Task operates too slow
Sensor/HID/Actuator	Signal too slow
Display	Display updated too slow
Operator	Operator action too slow

Superclass : Failure_Modes Cont./....

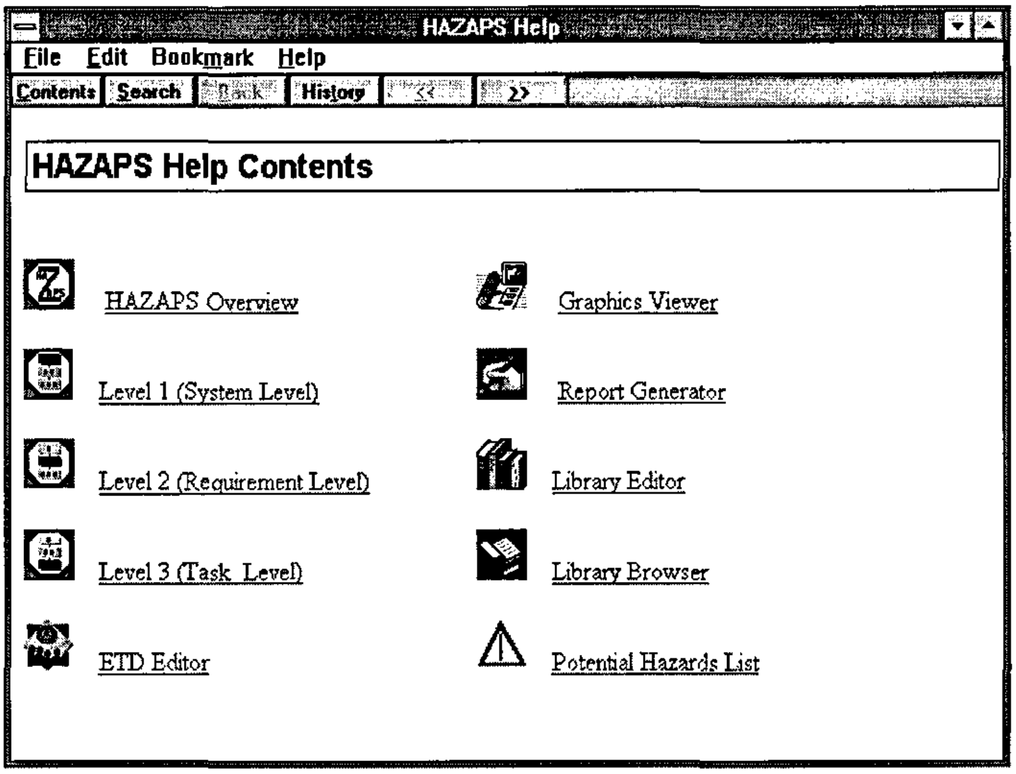
Class : Defective_Hardware

Slots	***Values***
Processor	Processor
Processor	Storage devices
Processor	Wrong cards
Processor	Digital/analogue I/O/communication/counter timer/other cards
Processor	Defective device/s
Processor	Physically detached link/s

Class : Failure_Not_Detected

Slots	***Values***
Processor	Failure to detect failure

Appendix 2



HAZAPS Help Contents

HAZAPS Overview
Graphics Viewer
Level 1 (System Level)
Report Generator
Level 2 (Requirement Level)
Library Editor
Level 3 (Task Level)
Library Browser
ETD Editor
Potential Hazards List

HAZAPS Overview

HAZAPS is a tool to aid the developer in assessing hazards in programmable systems. It has three main levels

1. **System Level**

2. **Requirement Level**

3. **Task Level**

HAZAPS also incorporates a new graphical technique, the Event Time Diagram (ETD) to aid the developer in understanding and analyzing tasks and their interaction.

Various tools are provided - a **Report Generator**, **Graphics Viewer**, **Library Editor**, **Potential Hazards List**.

HAZAPS System Level

System
Level Icon

DESCRIPTION

Purpose
At the System Level safety requirements are constructed using the software requirements specification, a **Potential Hazards List** and available design schematics.

Layout
The System Level has two windows, the upper one for the specification **Source File**, the lower for Safety Requirements.

Tools
A number of tools (e.g. **Potential Hazards List**, **Graphics Viewer**,**Library Editor**) and cut and paste facilities are provided.

Actions
At this Level, a **Source File** can be retrieved, edited, saved and cleared; assessments can be loaded, saved and cleared; requirements are processed (i.e. loaded into system).

PROCEDURE

1 Retrieve the **Source File** into the upper window and, using the **Graphics Viewer** and **Potential Hazards List**, construct safety requirements and place in lower window. Use the lower window to decompose the specification into a number of requirements.

2 Enter the requirements in the lower window. **Format note**: the word 'Requirement' must be included as a header to each requirement.

3 Process the Safety Requirements (i.e. load into system).

4 Move to **Requirement Level**

HAZAPS Requirement Level

Requirement
Level Icon

DESCRIPTION

Purpose
At the Requirement Level, Requirements are decomposed into Tasks and Tasks are classified according to **Task Type.**

Layout
There are three panels: top left to select Requirement for decomposition; bottom left to add Tasks and RHS to view all Tasks associated with a particular Requirement.

Tools
An **ETD Editor** (useful for understanding and analyzing Tasks and their interaction, and in identifying critical tasks) is provided at this Level. Access to the **Graphics Viewer** and **Potential Hazards List** is also provided.

Actions
Requirements and Tasks can be inserted, appended, saved and deleted at this Level.

PROCEDURE

1 Select a Requirement (top left panel).

2 Enter a Task (bottom left panel)

3 Save the Task - when a Task is saved, the developer is prompted to select a **Task Type**.

4 Continue until all Requirements have been decomposed into Tasks. At any stage it is possible to scroll through Tasks and view list of Tasks associated with a particular Requirement (RHS).

5 Move to **ETD Editor**.

HAZAPS Task Level

Task Level Icon

DESCRIPTION

Purpose
At the Task Level, Tasks are assessed by posing a number of questions to the developer. The developer responds to the questions and enters any actions required.

Layout
The Task Level has three choice boxes (to select Requirement, Task and Group) and four panels. The top left panel displays the questions, top right displays the Task Description, bottom left is for entry of Response to question, and bottom right for optional entry of Actions.

Tools
Tools available at this Level include the **Library Browser**, **Report Generator**, **Graphics Viewer** and **ETD Editor**.

Actions
The developer answers questions and can input actions associated with each question. These can be edited and saved at this Level.

PROCEDURE

1 Select a Requirement, an associated Task and an assessment **Group**

2 The system will return a number of questions based on the Task Type which was selected for this Task at Level 2, the Task description is also displayed. The **Library Browser** can be used for viewing all associated questions.

3 Respond to the sequence of questions and enter any necessary actions saving each in turn.

4 If output is required, go to the **Report Generator**.

HAZAPS ETD Editor

ETD Editor Icon

DESCRIPTION

Purpose
The **ETD** is a new graphical technique to help the developer analyse Tasks and their interaction and to identify Critical Tasks. It models behaviour in terms of events, time, control and data flow, entities and associated functional levels. It may be viewed as a polar diagram where the angle respresents time, the distance from the centre gives the functional level, and the arrows give direction of flow of information (either control or data).

Layout
There are two windows, the top main window displays the **ETD** and the bottom window is a browser displaying Requirement and Tasks descriptions.

Actions
The Safety Requirement is modelled and analysed.

PROCEDURE

1. Select Requirement from Choice Box

2. Click on **ETD** template at one of the vertices.

3. A form is displayed on the left - fill in form and press 'OK'. Task appears on **ETD** template.

4. Repeat for all tasks

5. Move to **Task Level**

Graphics Viewer Icon

HAZAPS Graphics Viewer

The Graphics Viewer allows any figure (e.g. design schematics) with **.BMP** format to be viewed within the system.

Report Generator Icon

HAZAPS Report Generator

DESCRIPTION

Purpose
The Report Generator is accessible at all Levels and can be used to view progress at any stage. It is not static but continually updates as the user enters information to the system. The user can select various options for a Report from the system, including a full report, and reports on sections/sub-sections of the work.

Layout
The Report Generator has one window. Options for the type of Report required are available from the menu bar 'Options'. A Dialog Window offers a selection of items that can be included in the Report and output options (see below).

Actions
The developer selects Requirement(s) and chooses item(s) to be included in the Report and selects output device(s).

PROCEDURE

1 Click on 'Options', then 'Start' - a Dialog Box will be displayed

2 Select Requirement(s) (top of Choice Box)

3 Select the items required for the report. Choices include

- Requirement Descriptions
- Tasks
- ETDs
- Questions
- Responses
- Actions

4 Select desired output device(s)

- Output displayed on Screen

- Output to Postscript Printer
- Output to Postscript (.ps) file
- Output to Windows 'Notepad'

5 There is an option to print the Report after it has already been output to the Screen. This option is available via 'Options' on Menu Bar.

Library Editor Icon

HAZAPS Library Editor

DESCRIPTION

Purpose
The Library Editor allows customisation of the Questions which are used for assessing Tasks at the **Task Level.**

Layout
The Library Editor has three choice boxes (to select **Type**, **Group** and **Keyword**) and three
windows. The top window displays existing Questions, the middle window is for entering or editing Questions, and the bottom window is for entering or editing information associated with Questions.

Actions
The Library Editor allows the user to edit questions and associated information.

PROCEDURE

1 Select a **Task Type**, a **Group** and an associated **Keyword**

2 Depress the appropriate button on the RHS (Insert, Append, Delete, Edit).

3. For 'Insert' or 'Append' the word 'nil' appears in the top window at the position where the question will be added. Highlight 'nil' and then depress the Edit button. For 'Edit', highlight the relevant question and depress Edit button. The question appears in the middle window.

3 Type the new question or edit existing question in the middle window. Type additional information in the bottom window.

4 When operation complete, press 'Save' button on RHS

Note: There is an option on the menu bar under 'Exit' to permanently change the generic library for all future assessments.

Library
Browser
Icon

HAZAPS Library Browser

DESCRIPTION

Purpose
The Library Browser allows the user to view questions and associated information directly from the **Task Level**.

Layout
The Library Browser has two windows, the upper one for Questions and the lower for Associated Information.

PROCEDURE

At the **Task Level** depress the View button at the bottom of the screen. To exit Browser, depress 'OK' button.

Potential
Hazards
Icon

Potential Hazards List

To provide assistance for identifying top level hazards, the Potential Hazards List has been adopted from W. Hammer *Product safety management and engineering*, Prentice Hall, N.J., 1980.

An important feature of this facility is that it can be customised by the user.

Appendix 3
Related Publications

Broomfield EJ, Chung PWH (1994) "A hazard identification methodology for programmable systems". In: "Risk Management and Critical Protective Systems", Safety & Reliability Conference, Altrincham, Cheshire, 12-13 October 1994. SaRS Ltd.

Broomfield EJ, Chung PWH (1994) "Hazard identification in programmable systems - A methodology and case study". Applied Computing Review 2:7-14.

Broomfield EJ, Chung PWH (1995) "Using incident analysis to derive a methodology for assessing safety in programmable systems". In: Proceedings of the Safety-critical Systems Symposium: Achievement and Assurance of Safety, Brighton, pp.223-239.

Kletz T, Chung PWH, **Broomfield EJ**, Shen-Orr, C (1995) "Computer control and human error". IChemE, Rugby, UK.

Broomfield EJ, Chung PWH (1997) "Safety assessment and the software requirements specification". Reliability Engineering and System Safety 55:295-309.