

8 Guidelines for Best Practice

The practical engineer and manager is likely to want guidelines which can advise on the best and most relevant practice to follow. Several projects have produced such guidelines, either as the main objective of the project or as a subsidiary output. Guidelines have been produced for:

- the automotive industry
- management of the development process
- advanced robots
- general safety-related systems, with particular emphasis on the human factors in development
- programmable logic controllers.

8.1 *Automotive industry*

Motor vehicle manufacturers originate from an industry with a strong mechanical engineering tradition but they are increasingly having to cope with rapid growth in specialist electronic content. This is true at all levels from design, through manufacture to service, and it affects both the vehicle manufacturers and their suppliers.

In today's environment, where modern vehicles are increasing their reputation for performance, quality and value for money, succeeding in the market place is ever more dependent on the ability to employ new technology in all aspects of the business. Over the last decade the electrical feature content in vehicles has approximately doubled every 3 to 4 years, and this trend is still continuing. Increasing competition in the crowded market place creates strong competitive pressures for ever more features, quality and reliability. However this has to be achieved for less cost, weight and in a much reduced time scale.

As more and more features are added, the complexity of subsystems has multiplied and the interaction and interfaces between subsystems has reached the limit of traditional methods. Software has now become a necessary element in the deployment of new vehicle features. A further recent development is the introduction of in-vehicle serial data buses and multiplex systems for control and diagnostics functions.

Traditional electrical engineering in an automotive company was, and in some areas still is, based on component engineering, where one engineer would take total responsibility for one component. This worked well when the electrical system consisted only of a starter, alternator, headlights, wipers and a radio, but the complexity of modern vehicles demands a different approach.

In order to handle the increase in electrical features, both from a physical packaging point of view and cost, automotive engineers are using multiplex technology and software based control. Electronic modules are becoming general purpose processing elements communicating on a data bus, with the features and functions distributed around an integrated system.

The so-called electrical/electronic architecture of a vehicle must therefore encompass many elements of the component engineering approach, but pull them all together into a complete

system which has a clear structure. In order to define a proper architecture for a complex vehicle, teams of component engineers must work together with systems engineers and management to give a complete perspective on the electrical system, and this must happen before the requirements for the software and hardware are considered for individual electronic control units. The opportunity for a clean sheet design is rare, and often many of the components and other aspects of a vehicle electrical system are predetermined due to factors outside the control of the designers, eg. cost, company policy, mechanical constraints etc. The process for defining an architecture must accommodate influences such as these, even if this deviates from the ideal. Engineering a vehicle is a complex operation involving many people, each of whom often have different priorities.

The software challenge

The automotive industry has had to take on board the important differences between software and other forms of automotive engineering and components:

- software is primarily a design, with no manufacturing variation, wear, corrosion or ageing aspects;
- it has much greater capacity to contain complexity;
- it is perceived to be easy to change;
- software errors are systematic (ie built into the design), not random;
- software is intangible.

Many industries are facing this type of change: the differences between automotive applications and applications in other industry sectors are:

- production volumes are high (leading to manufacturing variation in the controlled equipment), related mechanical components are subject to wear, and maintenance levels are difficult to assure. automotive software therefore has an emphasis on data driven algorithms, parameter optimisation, adaptive control and on-board diagnostics.
- passenger car drivers receive little or no training compared with other users of computer-based products and services. Therefore automotive software requires an emphasis on failure management techniques based on the controllability of the vehicle.
- traditional automotive test environments use real vehicles and components, as well as simulations. These are available to test systems and software extensively and safely before they reach the customer.

MISRA

The MISRA project therefore set out to bring together the best available advice on how to specify, design, implement and test software in the particular context of the automotive industry.

The flagship product of the project has been the *Development Guidelines for Vehicle Based Software* [MISRA ref 10] the purpose of which is to provide assistance to the automotive industry in the creation and application within a vehicle of safe, reliable software.

The *Guidelines* cover the whole of the software lifecycle, software quality planning and a brief review of emerging technologies in the relatively short span of 72 pages. The detailed supporting information and recommendations, however, are available as separate reports [MISRA refs 1 to 9]:

- Diagnostics and Integrated Vehicle Systems
- Integrity
- Noise, EMC and Real-Time
- Software in Control Systems (the theory of linear and non-linear control systems and at the software issues involved in achieving smooth and reliable control)
- Software Metrics
- Verification and Validation
- Subcontracting of Automotive Software
- Human Factors in Software Development.

Integrity

The integrity report [MISRA *ref* 2] notes that an electronic control unit in a vehicle is required to have a high level of integrity because there are requirements that it should not cause:

- harm to humans;
- legislation to be broken;
- undue traffic disruption;
- damage to property or the environment;
- undue financial loss to either the manufacturer or the owner.

The risks associated with each of the requirements listed above should be reduced to an acceptable level by achieving the required level of integrity. Higher levels of integrity are achieved by the increased use of specialised methods and design requirements, albeit with increased cost, both of the development and of the end product.

Safety analysis

The report recommends that a preliminary safety analysis should be carried out as part of the requirements analysis phase and a more detailed analysis as the design progresses. Safety analysis is highly iterative and should be considered as a continuous process during requirements and design. In the context of on-board systems, the safety analysis should be based on a hazard analysis using categories of controllability which are defined as follows:

Uncontrollable

This relates to failures whose effects are not controllable by the vehicle occupant(s), and which are most likely to lead to extremely severe outcomes. The outcome cannot be influenced by a human response.

Difficult to Control

This relates to failures whose effects are not normally controllable by the vehicle occupant(s) but could, under favourable circumstances, be influenced by a mature human response. They are likely to lead to very severe outcomes.

Debilitating

This relates to failures whose effects are usually controllable by a sensible human response and, whilst there is a reduction in the safety margin, can usually be expected to lead to outcomes which are at worst severe.

Distracting

This relates to failures which produce operational limitations, but a normal human response will limit the outcome to no worse than minor.

Nuisance Only

This relates to failures where safety is not normally considered to be affected, and where customer satisfaction is the main consideration.

Controllability applies to the ability of the vehicle occupants, not necessarily the driver, to control the safety of the situation following a failure.

The main steps recommended in the determination of the initial Integrity Level are as follows:

- a) List all hazards which result from all the failures of the system.
- b) Assess each failure mode identified in step (a) to determine its controllability category .
- c) The failure mode with the highest associated controllability category determines the Integrity Level of the system (see Table 8.1).

Controllability Category	Acceptable Failure Rate	Integrity Level
Uncontrollable	Extremely Improbable	4
Difficult to Control	Very Remote	3
Debilitating	Remote	2
Distracting	Unlikely	1
Nuisance Only	Reasonably Possible	0

Table 8.1. System Integrity Levels

Where the Integrity Level of the system relies upon the software, this should be identified specifically in the documentation.

Once the Integrity Level of the software has been determined, an appropriate development approach should be defined. Table 8.2 (quoted from [MISRA ref 10]) summarises the MISRA guidance for each Integrity Level. For more details of the MISRA approach to integrity, see [MISRA ref 2].

Development Process	Integrity Level				
	0	1	2	3	4
Specification and design	I S O 9 0 0 1	Structured method.	Structured method supported by CASE tool.	Formal specification for those functions at this level.	Formal specification of complete system. Automated code generation (when available).
Languages and compilers		Standardised structured language	A restricted subset of a standardised structured language. Validated or tested compilers (if available).	As for 2.	Independently certified compilers with proven formal syntax and semantics (when available).
Configuration management: products		All software products. Source code.	Relationships between all software products. All tools.	As for 2.	As for 2.
Configuration management: processes		Unique identification. Product matches documentation. Access control. Authorised changes.	Control and audit changes. Confirmation process.	Automated change and build control. Automated confirmation process.	As for 3
Testing		Show fitness for purpose. Test all safety requirements. Repeatable test plan.	Black box testing.	White box module testing - defined coverage. Stress testing against deadlock. Syntactic static analysis.	100% white box module testing. 100% requirements testing. 100% integration testing. Semantic static analysis.
Verification and validation		Show tests: are suitable; have been performed; are acceptable; exercise safety features. Traceable correction.	Structured program review. Show no new faults after corrections.	Automated static analysis. Proof (argument) of safety properties. Analysis for lack of deadlock. Justify test coverage. Show tests have been suitable.	All tools to be formally validated (when available). Proof (argument) of code against specification. Proof (argument) for lack of deadlock. Show object code reflects source code
Access for assessment		Requirements and acceptance criteria. QA and product plans. Training policy. System test results.	Design documents. Software test results. Training structure.	Techniques, processes, tools. Witness testing. Adequate training. Code.	Full access to all stages and processes.

Table 8.2. Summary of requirements for software development

Human factors in safety analysis

The driver and a vehicle can interact in many different ways, and these interactions need to be considered carefully during safety assessment.

Factors to be considered include:

- human reaction times
- ease of recognition of a situation
- attentiveness
- driving experience
- risk compensation (improved safety can lead to riskier behaviour)
- subversion or over-riding of system functions
- smooth and readily perceived transfer of control from system to driver
- workload of driver, especially at the moment of transfer.

It may also be necessary to consider the effects of the range of capabilities relating to:

- vision and hearing
- mental state (for example, lack of sleep, jet lag)
- physical disability.

The *Guidelines* note that table 8.3¹, although originating in a different industrial sector, may be useful in assessing the significance of human error when interacting with a vehicle system.

Type of Human Behaviour	Human Error Probability
Extraordinary errors - those for which it is difficult to conceive how they could occur. Stress free, with powerful cues pointing to success.	10^{-5}
Errors in regularly performed, commonplace simple tasks with minimum stress.	10^{-4}
Errors of commission such as pressing the wrong button or reading the wrong display. Reasonably complex tasks, little time available, some cues necessary.	10^{-3}
Errors of omission where dependence is placed on situation and memory. Complex, unfamiliar task with little feedback and some distraction.	10^{-2}
Highly complex task, considerable stress, little time available.	10^{-1}
Process involving creative thinking, unfamiliar, complex operations where time is short and stress is high.	$1 - 10^{-1}$

Table 8.3. Probability of human error

After assigning an Integrity Level to the system the safety classification procedure provides guidance as to what development techniques and procedures should be used when designing the system, and the degree of rigour with which they should be applied. This ensures that the measures taken are necessary and sufficient for the system being designed. The use of Integrity

¹ See Comer P J and Kirwan B J, A Reliability Study of a Platform Blowdown System, Automation for Safety in Shipping and Offshore Petroleum Operations, Elsevier, ISBN 0-444-7010-1, 1986.

Levels permits a reasoned argument for assessing the degree of confidence that one should have in a system, which is commensurate with the inherent risk associated with the system function.

The recommendations and processes described by the project seek to enable a developer firstly to assess a system to determine its Integrity Level and secondly to adopt a suitable development process in order to achieve the confidence level required in the software. The intention is that the recommendations can also be used by assessors and procurers of such systems to help them determine whether the methods adopted by the system developers are suitable for the proposed application.

8.2 Management Guidelines for Developing Safety Critical Software

The MORSE project developed guidelines which provide advice and recommendations for managing the development of software for safety-critical systems. The guidelines, which are available as [MORSE *ref* D39], emphasise the importance of considering safety throughout the development process. They provide advice on what processes should be followed and on what methods and techniques should be used throughout the overall development.

The guidelines emphasise:

- the need to consider safety from the outset of a project,
- the level of the safety to be achieved by the programmable electronic system (PES) should at least match the safety level established for a similar conventional (i.e. non-programmable) system,
- until specific and detailed industry safety criteria have been established, a satisfactory standard of safety should be achieved by following the general guidance the guidelines provide combined with *sound engineering practice*.

Management Guidelines for Developing Safety Critical Software

MORSE Report D39

Contents

Introduction

Overview

Background

Safety; Safety and Reliability; Software; The advantages and disadvantages of using software;

Legal Issues

Definitions

Acknowledgements

Safety risk assessment

Risk assessment

The ALARP principle

Tolerable risk levels

Risk evaluation

Safety Case

Safety management

Types of failure

Safety integrity

Risk reduction model

Allocation of safety integrity levels; Example of the allocation of safety integrity level requirements

Strategy for reducing failures

Managing safety during system development

Safety plan

Safety records; Organisation of the design function; The use of a quality management system

Standards for developing safety-critical systems

The Standards Bodies; International Standards Bodies; Regional Standards Bodies; UK National Bodies; Standards for safety-critical software

Managing the software development

Software safety integrity levels

The software development process

Planning the development; Safety plan; Quality plan; Software definition; Software design; Software production; Design change control; Recording the development; Software maintenance

Safety management

Verification and validation in the development process; Verification; Validation; Safety validation

Soft Issues

The importance of soft issues and leadership

A leadership strategy

Cultural issues

Issues concerning individuals

Selection; Training; Motivation; Direction

Issues concerning teams

External liaison

On-going maintenance

A management model

Development techniques

Safety analysis techniques

Design considerations

Human error; Risk reduction

Quality assurance techniques

Design for assessment; Code of design factors; Coding standards; Assessment; English commentary of formal designs

Design techniques

Configuration of the safety-related system; Defensive Programming; Performance analysis; Memory management

The RAISE development method

Levels of Development; The impact of using RAISE for development; The implementation relation; Using RAISE for developing safety-related software

Verification techniques

Mathematical Proofs; Dynamic testing

Static analysis

Validation techniques

Simple scenarios; Detailed scenarios; Prototypes; Animation

August 1995, 102 pages

8.3 A Methodology for Safe Advanced Robots

Current industrial robots generally perform simple repetitive tasks especially in manufacturing environments. In such cases, the environment in which the robot operates is highly stable and structured, therefore safety precautions are well defined and interaction with humans is kept to a minimum. These robots are pre-programmed to perform actions unless interrupted by human intervention.

Robotic applications are becoming increasingly more advanced and robots have to be more sensitive to changes in their environment. This provides a tremendous technical challenge due to the need to manage the application complexity as well as find technological solutions that provide effective control over the robot behaviour in all environmental conditions.

Whilst there is currently no agreed international definition of what constitutes an advanced robot, the characteristics which make an advanced robotic system different from other conventional systems containing programmable electronic are that they:

- possess some degree of autonomy and their tasks are not well specified,
- need to respond intelligently to uncertain or incomplete data to accurately sense and interpret their environment,
- are generally complex, real time systems operating in uncertain environments and this makes predicting the behaviour of the robot difficult.
- are designed to interact with their environment and as such have poorly defined boundaries.
- may perform complex tasks which often need close interaction with humans to provide adequate control,
- may operate in hazardous environments and this may demand extremely high system reliability depending on the degree of autonomy involved.
- may have no fail-safe states and require on-going monitoring to ensure that correct operation, or adequate safety functioning, is maintained.

In addition, there are some distinctive features about the development of advanced robots in that they:

- may be based on novel design architecture strategies.
- involve various prototyping or concept proving stages.
- involve many different types of technology.
- may be composed of many components (reused or new), each with some variation in its development history, the technologies used and its design integrity.

The main safety considerations for advanced robots are therefore:

1. *Environment.* The most critical aspect of robot safety is specifying the robot's environment and identifying the safety-related interfaces between the robot and elements of the environment. The interactions between an advanced robot and the environment become more complex when the robot is more advanced. This means that an advanced robot's system boundaries are more difficult to identify. The robot's environment will therefore need to be analysed in depth.
2. *Design Integrity.* Design architectures with proven integrity need to be developed, especially for real time sensor and control systems, to help overcome potential re-use and integration problems. This will enable sufficient confidence in the operation and safety of these areas of the system.

3. *Safety Requirements*. As a result of the uncertainties surrounding the interactions between the robot and the environment, it may be difficult to define complete and verifiable safety requirements.

There are currently no specific standards for considering the safety of advanced robots. However, there are many safety life cycles in use today, e.g. IEC 1508. These safety lifecycles are generic in that they can be applied to any application and they contain a number of high level processes.

Typically, a generic safety lifecycle involves conducting the following processes:

- 1) Safety planning.
- 2) Hazard identification and analysis.
- 3) Safety requirements specification.
- 4) Safety risk assessment.
- 5) Safety verification.
- 6) Safety case generation.

These processes can apply to any development stage, although the safety techniques used in their application are dependent on the level of system detail available. These safety techniques identify hazards, their effects and causes, and provide a measure of the associated safety risk. This information contributes to the Safety Case evidence.

The role of a safety lifecycle in system development is an important influence on the achievement of assurance that a system meets defined safety requirements.

The two key purposes of the safety processes within a life cycle are:

- 1) The safety requirements must be adequately specified.
- 2) The safety requirements must be shown to be met.

Overall, the safety lifecycle processes must rigorously address the above aspects of safety requirements. Despite the importance of the safety life cycle, there are a number of inherent weaknesses in its implementation which cause difficulties when evaluating the safety of an advanced robot. These are:

1. Initial hazard identification and various safety analyses can be subjective processes and are dependent on an understanding of the robot system under development and its environment.
2. Current approaches to safety analysis only consider the environment informally, possibly resulting in an incomplete understanding of the whole system.
3. Safety behaviour specification and its analysis is not normally included in safety requirements or Safety Case evidence.
4. Safety requirements are currently specified in a number of different ways, often causing ambiguity.
5. There can be difficulties in assessing the risk associated with hazards.
6. The results of safety analyses are not adequately integrated with design activities.
7. The completeness of the safety considerations for a system is difficult to demonstrate.
8. A Safety Case is often compiled at the end of a project rather than built up progressively throughout the project.

These weaknesses need to be addressed by creating new processes and techniques to enhance the traditional safety lifecycle. Closer integration between the enhanced safety lifecycle processes

and other development life cycle processes is required to achieve intrinsic safety within design activities.

The real need is to:

- 1) Define and apply a formal approach to safety management.
- 2) Enhance current safety processes and techniques to rigorously address the analysis of complex innovative systems such as advanced robots.

ROBUST Safety Methodology

The ROBUST Safety Methodology is set within a Total Development Framework which relates it to the development activities via the concept of a generic lifecycle model. The Generic Development Life Cycle (GDLC) is based on a typical series of development stages and is used to show how the methodology is applied to each stage. The description of how to apply the safety processes to a particular Generic Development Life Cycle stage is then applicable to the equivalent project stage.

This involves accounting for any interaction between the safety processes and development activities and the associated information flows across the ROBUST Development Interface. This interface has been designed to ensure that the methodology may be properly integrated into project standards and development practices.

The Generic Development Life Cycle stages are:

- Requirements Specification
- System Specification
- System Design
- Subsystem Design
- Integration and System Test

Figure 8.1 shows a high level view of the ROBUST safety processes. This diagram also re-emphasises that there is a general connection between the overall development activities and the safety processes through the development interface.

The safety processes are:

- 1) *Safety Requirements Capture* This process identifies and defines the safety requirements. The aim is to concentrate on ensuring that a valid, complete and consistent safety requirements specification is achieved as early in development as possible. This is based on a thorough understanding of the robot system within a defined operational environment and employs environment and system modelling techniques. These safety requirements are refined for each level of design.

2) *Hazard Identification and Analysis* This process identifies hazards and analyses their possible effects and causes. ROBUST has developed an environment modelling and a system modelling approach which is used to enhance the understanding of the application. The use of models in conjunction with traditional safety analysis techniques ensures that hazards are

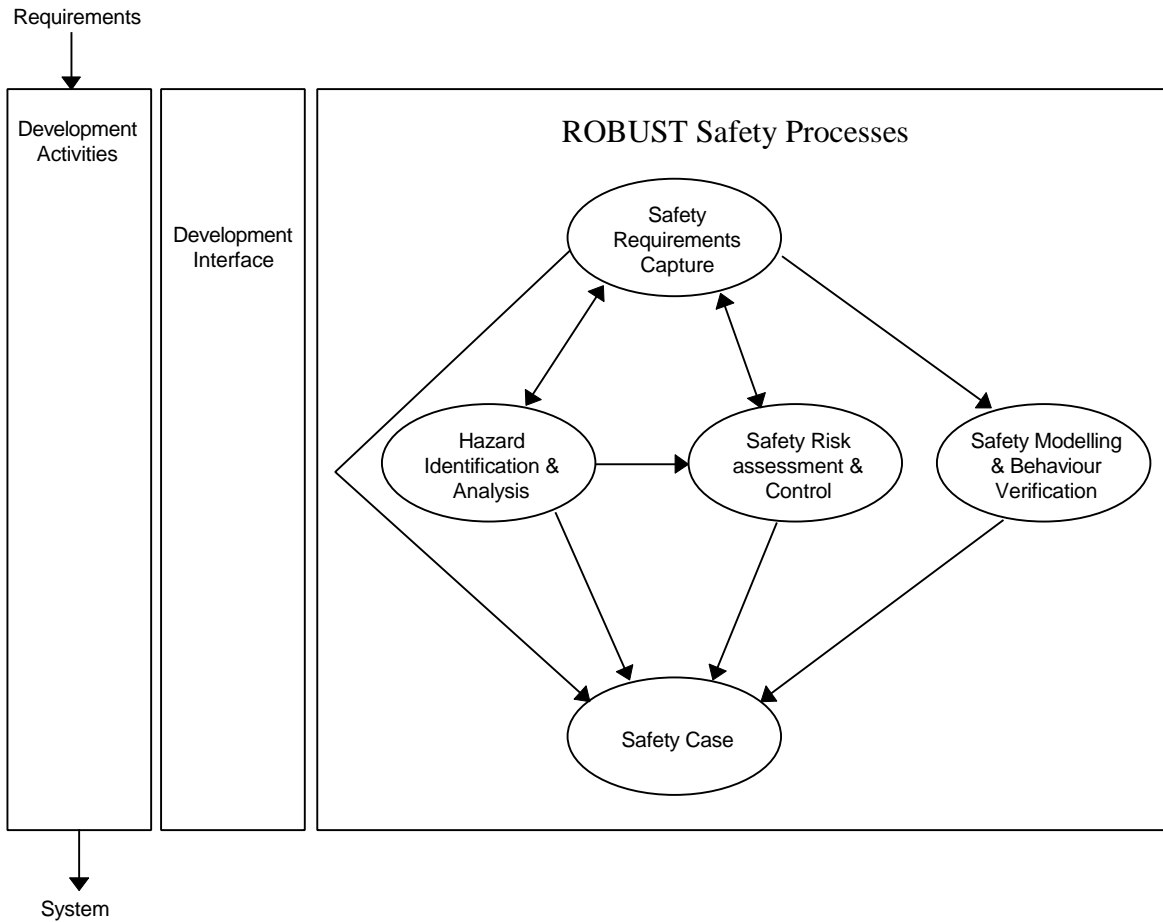


Figure 8.1. The ROBUST Safety Processes

identified together with their causes and effects. This approach is invaluable in complex operational applications such as in advanced robotics.

3) *Safety Risk Assessment and Control* This process assesses the safety risks associated with a proposed design solution. It assesses and controls safety risks in a balanced manner throughout development. Risk reduction measures may be suggested as an input to further development activities. These measures may propose further safety functions to minimise the safety risks.

4) *Safety Modelling and Behaviour Verification* This process defines, verifies and reasons whether the desirable safety behaviour has been achieved. This is a new safety life cycle process which initially derives the desirable safety behaviour from the safety requirement process. Rigorous modelling techniques are encouraged, but less rigorous systematic approaches may yield benefits simply through reasoning about safety behaviours. A rigorous safety model may be created to represent the safety behaviours. These safety models are used with previously developed system models to check that the proposed system design exhibits such desired safety behaviours. The definition of safety behaviour and the associated safety models are refined throughout development. The notion is that the safety behaviour of each system component should be defined to demonstrate its contribution to the safety of the system.

5) *Safety Case* This process collates the safety evidence and arguments derived from all the information generated within the remainder of the ROBUST Safety Processes. This is an enhanced process as it demands safety arguments at all life cycle stages. Therefore the generation of a Safety Case will apply for each proposed design throughout the lifecycle. Each Safety Case will be reviewed as a part of the management process within other development

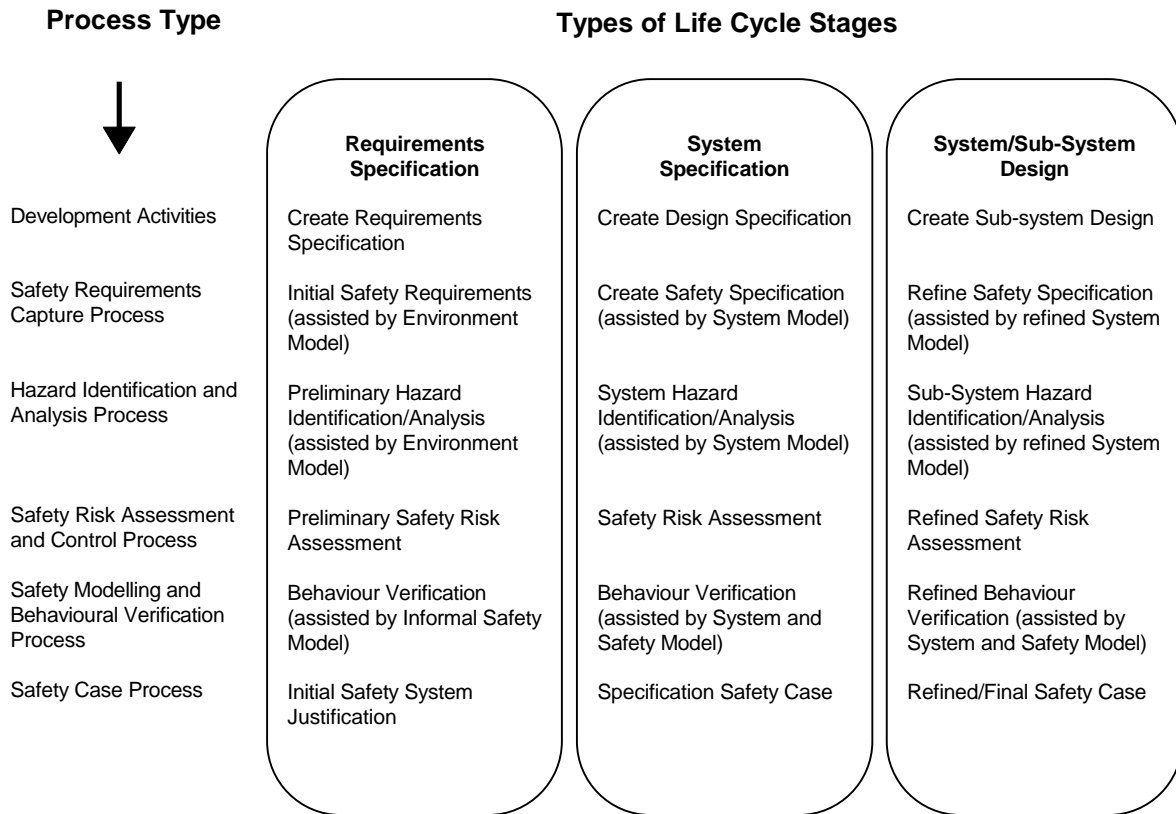


Figure 8.2. ROBUST Safety Methodology Overview

activities.

The essence of the ROBUST Safety Methodology is to apply the safety processes in a series of method steps for each life cycle stage. These are illustrated in Figure 8.2. This shows a simplified view of the Generic Development Life Cycle stages into three groups where the System Design and Subsystem Design stages have been combined.

To be satisfactory for this purpose, a modelling technique must

- 1) capture the different types of interactions between systems and their environments .
- 2) express the deterministic and behavioural aspect of systems.
- 3) capture and analyse the information about system states and their transitions.

Various modelling approaches could be adopted in order to gather the information needed to optimise the consideration of safety. Different analytical techniques are available to study different, yet related, aspects of a system's behaviour. Examples are models which capture system states, functions, objects, data flow, control logic and failure behaviours.

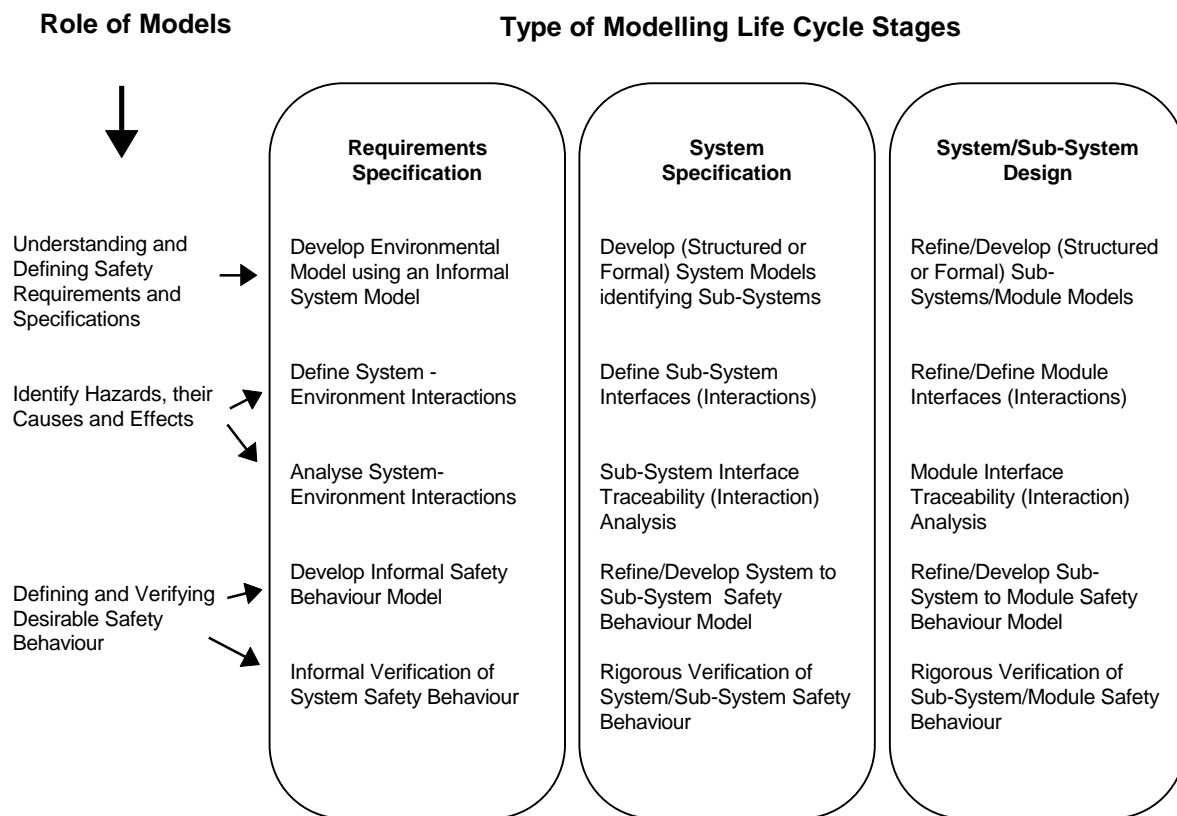


Figure 8.3. Role of Models within the ROBUST Safety Methodology

The essence of the ROBUST modelling approach is to focus initially on system states to address the safety behaviours of systems. Subsequently object-oriented concepts are used in studying system environments and the associated interactions with the system.

This focus on states is intuitive as the notion of unsafe states is directly linked with the identification of hazards, i.e. hazardous states, as derived from other safety analyses. Continued state analyses of system designs may also lead to identifying the potential causes of hazards. Hence state models provide powerful analysis capabilities, and through the use of tools, can offer useful verification and animation support.

The ROBUST Safety Methodology does not exclude the use of other modelling approaches, e.g. capturing data flows and control logic, as these can be adopted in carrying out the ROBUST Safety Processes.

There are many types of modelling activities that may be undertaken. This may involve a combination of static and dynamic models employing analytical, animation, virtual reality and simulation techniques. Furthermore different modelling techniques capture different aspects of a system, as highlighted in explaining the modelling criteria above.

In order to support the safety process, ROBUST has identified three types of models which enable the use of analytical and animation techniques. These models may be implemented using a variety of techniques, providing different degrees of rigour and support for validation and verification activities within the safety processes. The three types of models are:

1. Environment Model This describes a system's environment, i.e. the external objects that interact with a system, to systematically define its elements and the relationships with a "black-box" representation of the system and can be used to study external interactions and accident scenarios. This model has been implemented using a specific object oriented technique which uses a systematic approach to understanding the system and its interactions with a defined environment.
2. System Model This describes a system itself emphasising its state properties which are used to define all system functions and behaviours. Other system properties such as timing, control and data usage can be considered. This model has been implemented by deriving a State Model, which is subsequently expressed as a mathematical Logical Model. The logical models can be implemented using available software tools which demonstrate how the logical model can be used in verification and validation analyses and provide animation facilities.
3. Safety Model This describes the safety behaviours, i.e. the desirable meaning and implementation of safety for the system, in order to verify that they are consistent with the system behaviours derived from the system model.

The ROBUST Safety Methodology Handbook has been created to assist with the development of safe advanced robots. Its primary function is to make the results of the ROBUST research programme available in a convenient and usable form to the robot community. The handbook details the techniques developed during the project and includes examples of the novel techniques being applied to robot development projects.

Target readers include:

- developers systems
- integrators
- procurers
- users
- regulators
- robot and safety standards authorities.

The safety behaviours can be expressed in terms of safe or unsafe states, again expressible in a mathematical logical form.

In summary, ROBUST has used both state and object based modelling approaches to develop an intuitive and practical style of modelling to address safety.

8.4 A Code of Practice for the human dimension

The PRICES project has produced a Code of Practice for the production of safety-critical systems. The Code of Practice [PRICES *ref* D29] and its Justification [PRICES *ref* D30] are aimed at the producers of programmable electronic systems embodying conventional Information Technology components and/or Programmable Logic Controllers (PLCs) across all industry sectors.

The Code of Practice seeks to balance safety issues, especially integrity, with practical concerns, particularly the productivity of the underlying process. It places particular emphasis on the

human factors component of the process and on how one might use methods and tools in conjunction with an appropriate appreciation of the human element.

Underpinning the drafting of the Code of Practice has been research into current process models, the tools and methods used, and the psychology and sociology of the human participants in the process. This work has involved significant investigation into current practice and what can be deduced about real benefits, both in terms of safety and productivity, from current practice.

The Code of Practice takes account of key standards such as IEC 1508 and ISO 9000. In particular, it assumes that users have a quality management system in place, without which benefits from the Code of Practice would be very limited. The Code of Practice is intended to supplement ISO 9000-3, in that it addresses specifically:

- the low volume, highly complex creative work of producing safety-related software systems,
- the purchaser's side of the equation,
- requirements issues,
- guidance on techniques, methods and human-centred issues.

The Code of Practice has been validated in two ways:

- By exposure of early drafts to a number of Technical Advisory Partners for their comments. The Technical Advisory Partners represent a cross section of industry sectors and comprise procurers, vendors and regulators of safety-critical systems.
- By applying a late draft to a number of Case Studies and incorporating the lessons learnt into the final version.

The Code of Practice is accompanied by an extensive Justification [PRICES *ref* 30] containing the rationale for the advice provided in the Code of Practice and justifying the broad view of human error taken in drafting the Code of Practice. It is a central tenet of the PRICES project that the majority of the safety-related software errors stem from:

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

Errors of this type are often inserted early in the process and discovered late, entailing costly rework. They arise primarily from misunderstandings between client and requirements analyst; misunderstandings or lack of communication between system and software teams; and communication breakdown within teams. Findings such as these, suggest ways of minimising the introduction of safety-related software errors, which form the basis for the approach taken in the Code of Practice.

8.5 Guidelines for Programmable Logic Controllers

The advent of IEC 1508 (see section 2.3), the new standard for the functional safety of programmable electronic systems, is leading suppliers to take a new approach to the demonstration and justification of safety performance. Existing practices in software and system development are having to be enhanced to meet the requirements for risk assessment and safety analysis, and a development approach which matches the rigour of the software design and test process with the level of risk in the application or process under control.

There are problems to be answered in establishing the level of change required by the new safety standard. These include:

- if the requirements of IEC 1508 are followed, will software and electronics be safer?
- how close are existing software development practices to those required by IEC 1508?
- what immediate changes to current practice are required to enable compliance with the main tenets of IEC 1508?
- what will be the cost of compliance to IEC 1508?

In the process industry, where Programmable Logic Controllers (PLCs) are often used to implement safety protection systems, conventional software engineering techniques are often not applicable. PLC programming languages (e.g. ladder diagram or function blocks) are special purpose industrial languages designed to be understood by electrical engineers rather than software engineers. The design and test criteria for software using conventional languages such as Pascal or C are often not applicable. Not only are the relevant design and test techniques different but the associated terminology is different. Much of the research work and software engineering literature is based on the use of conventional programming languages and consequently the guidance on software for safety applications is often not relevant for PLC systems.

To provide guidance on the development of safe application software for PLCs and to answer some of the questions posed about the impact of the newer safety standards, the SEMSPLC project was instigated. The project had a specific aim of producing guidelines for developing safe PLC application software in line with the requirements of IEC 1508 and a wider aim of sharing some of the ideas and approaches from both conventional software engineering and the PLC engineering sector. The SEMSPLC Guidelines [SEMSPLC *ref* 1] have been published by the IEE.

An introduction to the SEMSPLC Guidelines and the findings from a demonstrator project using the Guidelines are presented in the rest of this section. The results from the demonstrator project may provide some insights as to the answers to questions on the impact of IEC 1508 which were posed above.

Some of the other areas of work in the project are discussed following the section on the Guidelines

The SEMSPLC Guidelines

For the purpose of the SEMSPLC Guidelines, the IEC 1131-1¹ definition of a programmable controller is used. This defines a PLC as:

“a digitally operating electronic system, designed for use in an industrial environment, which uses a programmable memory for the internal storage of user-oriented instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic, to control, through digital or analogue inputs, various types of machines or processes. Both the PLC and its associated peripherals are designed so that they can be easily integrated into an industrial control system and easily used in all their intended functions.”

¹ International Electrotechnical Commission, IEC 1131-3 : Programmable controllers Part 3. Programming languages, IEC, 1993.

The scope of the SEMSPLC Guidelines was limited to the development of PLC application software. The development of PLC kernel software and PLC programming tools was excluded, as was the development of the overall system in which the PLC operates, since inclusion of these areas would have resulted in too broad a scope.

IEC 1508 defines four different safety integrity levels (SILs), SIL 1 being the lowest safety categorisation and SIL 4 being the highest safety categorisation. The techniques described in the SEMSPLC Guidelines are considered appropriate for SIL 1 and SIL 2 systems. Extra actions for the development of safety-related PLC systems of higher safety integrity levels are recommended within the Guidelines, although the extent to which they enable SIL 3 and 4 to be achieved is still very much open to debate.

Results of the Demonstrator Trials

Two full-scale demonstrator projects were carried out to establish whether the recommendations contained in the Guidelines had achieved their objectives. One demonstrator project had the objective of implementing a fire and gas detection and protection system, the other had the objective of implementing a combined heat and power simulation display system. For each demonstrator project, two development teams were set up: one to carry out the project following current PLC application software development practices (standard method) and the other to carry out the project following the recommendations contained in an early version¹ of the Guidelines (SEMSPLC method). For each demonstrator project, a ‘dummy’ client was appointed to be responsible for formulation of requirements and acceptance of the final results from both development teams. For the SEMSPLC teams, a training course was provided on the types of methods which would be needed for the SEMSPLC approach.

The evaluation of the success of the recommendations contained in the Guidelines in achieving their objectives during the demonstrator projects was based largely on recordings of the number of errors found at each phase of the project, the lifecycle phase to which those errors could be attributed, whether the error would result in failure which could affect the plant hazards and the amount of effort taken in achieving each phase of the lifecycle. However, the procedures recommended in the Guidelines themselves also contributed to the evaluation of the quality of the work undertaken in the different streams of the demonstrator projects. In particular, a review of the software safety rationale was carried out, which led both to a comparison between the quality of the verification and validation activities carried out in the two parts of each project, as well as to recommendations for changes to the collection of software safety rationale evidence in the Guidelines themselves.

Metrics analysis

The results of the quantified metrics collection process are shown in tables 8.4 to 8.9. From tables 8.4 and 8.5 it is noted that three times as many defects in the requirements were found during the initial phases of the fire and gas system project using the Guidelines as compared with the conventional approach, and that almost no unsafe defects were detected as being introduced during the design and coding phases using the Guidelines. However, the results also show that more ‘safe’ defects were detected as being introduced during the design and coding stages of the project using the SEMSPLC approach. Clearly, one explanation would be that the testing processes used in the SEMSPLC approach are more thorough and therefore more errors

¹ The Guidelines have since been revised in light of the results of this work.

were detected, however this would not account for the difference between the profiles of the statistics for the detection of ‘safe’ and ‘unsafe’ failures within the SEMSPLC method results. Another possible explanation is that the focus of the SEMSPLC method on safety aspects may be such that special emphasis is given to removal of such errors, for example by the use of Failure Modes and Effects Analysis. If this is the case, then adaptations of the SEMSPLC method to focus also on errors which lead to loss of functionality, as well as loss of safety, might help to improve the quality of the overall software development process.

Standard Method (non-COP)					
	User Reqmts	Functional Design	Detailed Design	Coding	Total
Functional Design	7				7
Detailed Design	2	4			6
Coding	3	0	21		24
Testing	0	3	3	12	18
Total	12	7	24	12	55

SEMSPLC method						
	User Reqmts	Reqmts Spec	Design Spec	Detailed Design Spec	Code	Total
Reqmts	26					26
Design	1	1				2
Detailed Design	2	1	6			9
Coding	0	0	21	12		33
Testing	0	0	6	16	24	46
Total	29	2	33	28	24	116

Table 8.4 Distribution of safe defects discovered during the Fire & Gas Project

Standard Method (non-COP)					
	User Reqmts	Functional Design	Detailed Design	Coding	Total
Functional Design	3				3
Detailed Design	0	3			3
Coding	1	0	3		4
Testing	3	7	4	13	27
Total	7	10	7	13	37

SEMSPLC method						
	User Reqmts	Reqmts Spec	Design Spec	Detailed Design Spec	Code	Total
Reqmts	7					7
Design	0	0				0
Detailed Design	0	0	0			0
Coding	0	0	0	0		0
Testing	0	0	1	0	5	6
Total	7	0	1	0	5	13

Table 8.5: Distribution of unsafe defects discovered during the Fire & Gas Project

In table 8.6, the relative effort and duration associated with the two fire and gas system development streams is given. It is noticeable that the effort in the early stages is almost three times higher for the SEMSPLC method as compared to the conventional approach. However, slightly less time was spent in testing. Although the SEMSPLC testing method was more thorough, and hence more time consuming than the standard testing method, the standard testing method took longer to complete due to the number of errors found during the testing and the subsequent necessity to repeat tests. A significant proportion of the effort associated with the SEMSPLC development stream has also been attributed to familiarisation with the specialised safety analysis techniques that are required to be employed when following the SEMSPLC method. Unfortunately, in the time allowed and under the conditions of the demonstrator projects, it was not possible to evaluate whether similar savings in time and effort would be achieved during installation and maintenance.

	Total Man Days		Duration	
	'Standard' Method	SEMSPLC	'Standard' Method	SEMSPLC
Requirements	12	40	11	18
Design	48	87.5	20	19
Coding	26	57	7	12
Testing	39	31	8.5	6.5
Total	125	215.5	46.5	55.5

Table 8.6 Man Days Effort and Duration for the Fire & Gas Project

A look at the distribution of error detection over the project lifecycle for the combined heat and power simulation display project (tables 8.7 and 8.8) shows again that a large number of defects were discovered by the SEMSPLC team during the software requirements definition phase of the development. A comparison of the total effort spent by the two teams working on this project (table 8.9) shows again that the team following the SEMSPLC approach spent significantly more than the team following the conventional approach. However, a large proportion of this

difference may be accounted for when the amount of supporting documentation produced by the two teams is compared. For example, the team working to the standard method produced no formal design documentation and no test specifications. Some of the difference may also be attributed to the additional system hazard analysis carried out by the SEMSPLC team at the start of the project and to the necessary familiarisation effort involved in adopting the SEMSPLC method.

Standard Method (non-COP)					
	Reqmts Spec	Design	Coding & Integration	Test	Total
Reqmts	1				1
Design	2	0			2
Coding	2	3	2		7
Testing		1	0	0	1
Total	5	4	2	0	11

SEMSPLC Method							
	User Reqmts	Software Reqmts	High Level Design	Low Level Design	Coding	Test	Total
Reqmts	12	17					29
Design	0	6	8	0			14
Coding	0	2	0	0	5		7
Testing	0	2	1	0	1	0	4
Total	12	27	9	0	6	0	54

Table 8.7 Distribution of safe defects discovered during the CHP Simulator Project

Standard Method (non-COP)					
	Reqmts Spec	Design	Coding & Integration	Test	Total
Reqmts	2				2
Design	0	0			0
Coding	6	6	5		17
Testing	1	0	2	0	3
Total	9	6	7	0	22

SEMSPLC Method							
	User Reqmts	Software Reqmts	High Level Design	Low Level Design	Coding	Test	Total
Reqmts	11	4					15
Design	0	3	2	1			6
Coding	1	2	0	0	2		5
Testing	0	2	0	1	0	0	3
Total	12	11	2	2	2	0	29

Table 8.8 Distribution of unsafe defects discovered during the CHP Simulator Project

	Total Man Days		Duration	
	'Standard' Method	SEMSPLC	'Standard' Method	SEMSPLC
Requirements	5	41.4	54	20
Design	17.4	56.3	50	30
Coding	5.5	20.7	4	32
Testing	7	33.1	5	32
Total	34.9	151.5	113	114

Table 8.9 Man Days and Duration for the CHP Simulator Project

The error detection metrics were analysed further using the technique developed during the work on human errors. The results are given below.

Using the SEMSPLC method the design phases introduced significantly more errors than the code phase. This is different from the standard method and not as expected. It is probable that this is caused by the introduction of the distinction between software design and detailed design, which appears to have caused confusion because of its inappropriateness for such a small project.

Defects are frequently not found in the appropriate testing phase; sometimes earlier than expected but more often in a later phase than expected. This is probably due to difficulty in focusing the tests on the output of the corresponding earlier phase (i.e. requirements, design or coding) and the fact that testing one area often uncovers errors elsewhere.

The introduction of the software requirements phase in the SEMSPLC method has been very effective (in one demonstrator project) in detecting errors earlier than in the standard method. The other demonstrator project metrics did not distinguish user requirements from software requirements so no conclusion could be reached.

Using the SEMSPLC method produced a significantly lower percentage of unsafe defects than the standard method: 10% instead of 40% for one demonstrator, and 35% instead of 67% for the other.

Requirements test coverage

Further work on the assessment of the adequacy of the software safety rationale for the fire and gas system led to an assessment of the test coverage for the safety requirements (Table 8.10). This work confirmed that the test coverage achieved using the SEMSPLC method was greater, even though less time was spent on the activity.

	SEMSPLC team	NON-SEMSPLC team
Total safety requirements	80	80
Fully traceable to tests	46	37
Not traceable to tests	4	29
Partially traceable to tests	30	14

Table 8.10 Comparison of the Traceability of Safety Requirements to Test Specifications

A demonstrator project team's comments

During the demonstrator projects, the metrics collection process was supplemented by the provision of log books for use by the engineers involved in the projects to record subjective impressions as to the strengths and weaknesses of the development process¹. The following comments are based largely on extracts from the log books from the fire and gas system demonstrator project.

The two main software design techniques that were selected for this particular project were State Transition Diagrams (STD) and Control and Data Flow Diagrams (CDFD). The CDFDs were found to describe the software's functionality successfully, whilst the STDs succinctly defined the functionality at module level, which easily translated into PLC code. With both techniques initial unfamiliarity was the most difficult aspect, although both gained support during the course of the demonstrator project.

An initial impression of the Guidelines may be 'daunting', when confronted also with the normal commercial pressures of running a project, specifically, timescales and budgets. It was

¹ As described in the early version of the Guidelines used for the demonstrator projects.

found that the quantity of documentation required by the Guidelines was very large compared to the standard method - 18 documents compared to 5. Whilst the documentation did provide higher levels of confidence in the safety integrity of the software, when coupled with the verification activities required, it proved an obstacle to project progression and it was possible to lose sight of the end goals of the project. Production of so many outputs was found laborious and boring by the project team and reduced the impetus and enthusiasm normally associated with this type of work. Difficulties that were encountered in the main software design phases were related mostly to unfamiliarity with the specialist techniques and understanding exactly what output was required at each phase.

Unfamiliarity and lack of clarity in the early version of the Guidelines in general affected the quality of the design reviews where the same question was often asked, “the requirements of the Guidelines are defined and described but what do they actually mean?”

After an initial coming to terms with the more stringent techniques required by the Guidelines, and the large amounts of documentation, the actual project implementation differed in only minor ways from the standard implementation required, the main differences being in the verification activities, such as the safety analysis. These are the areas which it is expected that users will have the most difficulty coming to terms with. It was suggested that more guidance and clarity should be provided in these areas, particularly in the content and objectives of the software safety rationale.

Overall, it was felt that previous training and experience of the Guidelines is as important as previous PLC programming skills, perhaps by these means the cost and timescales associated with using the Guidelines could come within acceptable limits.

Changes made to the Guidelines since the end of the demonstrator projects have incorporated changes resulting from these comments and also from other sources. These have already resulted in significant improvements in both clarity and user friendliness.

8.5.1 Safety Analysis

During the first phase of the project, techniques which could be used to identify hazards and evaluate safety were investigated and their applicability to PLC application software development was considered. This work found that:

- safety analysis techniques should be used during all phases of the software development process
- combinations of techniques, such as Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA), should be used to maximise the number of hazards and failure modes identified
- generic fault tree constructs could be defined for a limited set of software control constructs to assist the identification of specific failure modes for a given piece of software
- there was a need for guidance on how to apply hazard analysis techniques during the PLC application software development process in a way that would integrate the hazard analysis process with the software design and verification processes.

A second strand of work undertaken during the first phase of the project identified the steps considered necessary to assure and demonstrate the safety of a software system. The steps identified were to:

- define a safety target for the software system, such as a tolerable probability of an unsafe failure
- design the software system to meet the defined safety target, employing the recommended hazard analysis techniques during the design and verification activities
- evaluate the resulting software system to establish whether the safety target has been achieved.

Incorporation of safety analysis recommendations in the Guidelines

During the second phase of the project, the findings of the two strands of work carried out during the first phase were incorporated in the Guidelines. There, guidance is given on the safety analysis activities that should be carried out during each phase of the PLC application software development lifecycle and how the results of these activities should be used to support the claim that the delivered PLC application software would meet the required safety integrity level. The main components to be included as evidence to support the safety argument in a safety rationale are outlined.

Evaluation of the safety analysis recommendations

The success of the recommendations contained in the Guidelines on the use of safety analysis techniques was evaluated during the two full-scale demonstrator projects. An analysis of the distribution of errors over the software development lifecycles of the demonstrator projects showed that use of the recommended safety analysis techniques was successful in detecting a large number of unsafe defects in the user's statement of requirements. It was found also that, in the case of the one demonstrator project in which the recommended safety analysis techniques had been used throughout the software development lifecycle, far fewer unsafe defects were introduced by the team using the Guidelines during the software design and coding phases of the lifecycle compared with the team using a conventional application software development method.

The two companies that carried out the demonstrator projects were impressed by these results and have indicated intentions to incorporate the safety analysis recommendations in their own company work instructions. One of the project sponsors has already begun to apply the techniques to a design assessment of a large process plant.

Difficulties were however experienced by the demonstrator project teams in always managing to link the results of the safety analysis activities to the software design and testing processes. Although the recommended safety analysis techniques were employed, there was only some evidence that the software design and the software test plans were influenced as a result. Also, it was evident that difficulties were experienced in analysing the results of the safety analysis activities in order to show that the developed application software would meet the specified safety integrity target. These difficulties were partially attributed to a weakness in the Guidelines in clearly communicating the objectives of the safety analysis activities during each phase of the software development process. It has been suggested that the identified weakness in the Guidelines should be addressed by providing illustrative examples of the use of the recommended safety analysis techniques.

8.5.2 *Human Errors*

An initial study of the published literature on human errors showed that, while an extensive body of work existed, most of it related to humans performing physical actions such as operating machinery and was not immediately apparent as being applicable to the software development process. However, a model was developed, by considering each phase in the development process as an individual human action, which demonstrated how errors are introduced and propagated through the development lifecycle. An amalgam of existing work on error classification systems was adopted and amended to be consistent with this model of error propagation also. A further outcome of this initial study was the development of a simple algorithm which assisted in focusing attention on those categories of errors where the error reduction effort could be applied most cost effectively.

To determine the starting point for error reduction and the scale of the problem a survey of errors in actual PLC development projects was undertaken using the classification system and error model described above. This extensive survey, covering 778 errors, was also backed up by interviews with PLC development engineers. The results, when analysed by the type of error, led to the following conclusions:

- the distribution of errors by error type is independent of whether errors are safe or unsafe, the project organisation, the details of the PLC system or the personnel involved
- the distribution of errors by error type is not very consistent, probably due to the difficulty of precisely categorising each error
- human errors are by far the most significant, but too much reliance should not be placed on the details of the error type distribution
- it appears likely that the most probable causes of undetected errors are incorrect assumptions and mistakes
- there is a significant difference between the actuality and the perception of the distribution of errors by error type, therefore any proposed improvements to our development methods should be based on reliable metrics rather than opinion.

The above conclusions are believed to have general applicability since they result from the nature of humans and how they work, but the results of the source phase versus detection phase analysis seem to be of a different nature in that this is dependent on the structure of the development lifecycle process. The consideration of the results of this analysis led to the development of a useful new technique for the assessment of the efficacy of the development process. This technique comprises the following steps:

- an ideal source versus detection phase error distribution is deduced theoretically from an examination of the development lifecycle method
- metrics are collected from an actual development using this development lifecycle method
- these metrics are compared with the ideal situation to reveal specific deviations from this expected distribution
- the specific nature of these deviations can give useful insight into the ways in which the development method is failing and hence how it can be improved.

This technique was used successfully in the assessment of the demonstrator project results.

8.5.3 Languages

Because the importance of technology transfer had been recognised in the project proposal, the language work was initially approached from the perspective of computer languages in general rather than specifically PLC languages. Two parallel strands of work were undertaken which led to the development of a set of criteria for establishing the suitability of a language for the production of safety-critical software. The first of these two strands was original work undertaken by members of the project, starting with a blank sheet and using their language expertise and common sense, and resulted in a draft set of criteria. The safe language criteria are reported in detail in [SEMSPLC ref 5]. The second strand was a literature survey, which was used to validate the draft criteria. The degree to which the two strands confirmed and complemented each other was very gratifying and led us to adopt the following agreed set of criteria:

- the language syntax and semantics should have a close fit to the application domain
- the language should be fully and unambiguously defined, and should be the subject of a widely accepted standard
- the language should support and encourage modularisation or partitioning
- the language should be readable
- the language should support the tracing of requirements through design to code
- the language should have stringent rules which are checkable either at compile-time or at run-time
- the language should produce well structured and statically analysable programs
- the language should enable the production of deterministic programs.

Implications of SEMSPLC on IEC 1131-3

As far as this language work was concerned the timing of the SEMSPLC project was fortuitous in that it coincided with the development by the IEC of the PLC language standard (IEC 1131-3¹). Thus the project was able to assess the emerging IEC standard against the above established criteria and the conclusion was reached that the languages (as then defined) failed to meet most if not all of the criteria. This was not particularly unexpected as the languages were not designed with safety specifically in mind. It was felt that these failings in the IEC languages could be overcome in three ways:

- minor revisions to the existing standard to remove ambiguity and to improve clarity and rigour
- development of coding standards to advise the programmer how to use the languages safely
- development of a subset / extended language which would satisfy all or most of the SEMSPLC criteria.

The first approach was covered by feeding back comments into the standardisation process through membership of PLCopen. While these changes alone are not sufficient to produce a safe language, they are a necessary step and will in any event produce an improved standard.

To cover the second approach a generic coding standard was developed which could be adapted in a manner specific to the PLC language and an individual project's requirements.

¹ International Electrotechnical Commission, IEC 1131-3 : Programmable Controllers, Part 3. Programming languages, IEC, 1993.

Work on the third approach was been started, but to produce a well defined subset for each of the IEC languages would have required more effort than was available. The completion of the subset could form part of a follow-on project.

8.5.4 Testing

In mainstream computing, it is generally accepted that systematic testing must be carried out on safety-critical systems. Various techniques have been widely used for over 20 years, increasingly supported by tools which assist in automating the testing process. Methods such as structural coverage are now mandated by standards such as DO178B¹ for civil aviation software. Although originally used mainly for safety-critical code, these techniques are now gaining wider acceptance in the software community as automated tools have made them available and easily usable for most common languages.

However, for PLC application software, this background of testing capability and knowledge does not exist. One reason for this is that for most PLC systems, it is virtually impossible to use any of the systematic techniques which are commonplace for mainstream computer software. In the first place, existing PLC languages do not lend themselves to these techniques: additionally, existing PLC languages are machine-specific, making investment by independent tool manufacturers not worthwhile. A thorough literature search carried out at the beginning of the project failed to find even one reference to any type of systematic or white-box testing of PLC application software.

Investigation into testing methods for PLC software

The original intention was to develop prototype software tools, similar to those used in mainstream computing which would carry out, in at least a partially automated manner, those testing techniques which were thought relevant to PLC application software. Some of the work on PLC testing is described in [SEMSPLC refs 10 and 11].

The first step was to discover which techniques were relevant to PLC application software. Most PLC application software written in the UK is written in a ladder logic language particular to the PLC hardware being used. A “standard” ladder logic program consists of a set of instructions which is continually executed in the form of an infinite loop while the system is switched on. Each instruction is executed on every scan of the program. Most instructions use the values of Boolean-valued switches to set other switches, although numeric values and computation are also handled. Some “control structure”, as occurs in mainstream computing languages, is present but this is not a commonly used feature of ladder logic.

PLC application software is different from mainstream computing software and it was discovered that adequate testing methods also need to be different. For example, statement coverage is a basic mainstream computing technique which involves the execution of every statement in a program. It is often difficult and time-consuming to achieve full coverage, but the equivalent in PLC application code is often guaranteed simply by turning on the machine. Similarly, to test every path through a mainstream program is held to be a “holy grail” of

¹ Software considerations in airborne systems and equipment certification. Prepared by RTCA, Inc. SC-167/EUROCAE WG-12, RTCA/DO 178 B, Washington, USA, Dec 1992.

structural testing, but many ladder logic programs have only one path, making the achievement of this trivial.

PLC application software is essentially different because the information and complexity is not in control flow decision making but in the register values. Similarly, testing methods covering control flow techniques are relatively useless and more useful testing methods are those based on testing register values. Full statement coverage is still necessary however, although achieving it may well be trivial.

An early testing success was achieved using mutation testing on a small piece of code written and tested as an example of ladder logic as it could be applied to a simple traffic lights control program. This found an error which was rather obscure, not having been picked up by normal testing, but which could have existed in live operation. By flicking a switch off and on at the “wrong” times, the four traffic lights all became green simultaneously. Without detailed analysis, it is unlikely that this error would have been found.

Full mutation testing is very expensive and time-consuming, and hence impractical under most circumstances for any sizeable software. This technique could be described as the “holy grail” of register testing. In a similar manner to attempting to solve the well known problems of structural testing when full path testing is not feasible, other methods have been designed which attempt to maximise the value of register testing. One way of concentrating on register values is to test that they take both the values true and false wherever they are used in the code. When used as a decision, e.g.

```
if T4_02.DN then
```

this is covered by branch testing but structural testing does not consider the values in a statement such as:

```
B3_20 := N11_01_03 and not N11_03_02 and not B3_50;
```

The main technique proposed is similar to branch condition testing. In the above example, the register values N11_01_03, N11_03_02 and B3_50 must each take the values true and false in the statement. There are more comprehensive means of testing these register values, for example that all combinations of the register values have occurred (similar to branch condition combination testing), but due to the often very long expressions containing the registers this was thought not to be worthwhile for this code. A refinement of the latter technique, known as modified condition/decision testing, has recently been proposed and is worth investigating further. It promises to achieve a significantly higher level of testing than branch condition testing for some extra work.

Having realised that the PLC environment did not lend itself to the implementation of software tools, it was decided to build a system to translate PLC application code into a language which could be analysed. This was done, and two variants (from Mitsubishi and Allen-Bradley) were translated, although some hand tuning was required. The translation system is very flexible, and the PLC application code can be translated into a number of different representations including Ada, C and Structured Text, an IEC 1131 language.

Evaluation of the recommended testing techniques

In order to test the PLC application software produced during the demonstrator projects code, Ada was used as a target language, as good test and analysis tools were known to be available. Statement coverage, branch coverage and branch condition coverage were measured, in Ada. Although the first two did not have a one-to-one mapping to the original code, their joint coverage was necessary to achieve the equivalent of electricity flow along every part of every rung of the ladder. Added to this, the branch condition coverage measure gives an excellent validation technique.

Code produced by one of the demonstrator project teams was tested by running their test data through the equivalent Ada program (about 8000 lines) in the order given in their test documentation. This translated into 205 test data sets, achieving final coverage measures of 96% for statements, 95% for branches and 88% for branch conditions. Of the 205 tests, 24 increased statement coverage, 29 increased branch coverage and 131 increased branch condition coverage. This implies that the latter measure is more finely tuned to this type of software.

8.5.5 *Timing*

Timing constraints are a constituent of most control systems, and in most safety-related systems the ability to meet specified times will affect safe operation. This could be indirect, for example often the system performance will be affected by timing issues, and failure to achieve the required control performance may again impact on system safety.

In most projects, only one or two types of timing issue are considered. It became clear during the SEMSPLC project that not only were there many more types of basic timing requirements that can be specified, but also that these spawned a number of particular issues within PLC-controlled equipment, which need careful thought and design. Work on timing issues for PLCs can be found in [SEMSPLC *refes* 12, 13 and 15].

Some of these issues could be identified against a particular phase of the software life-cycle, and are therefore included in the text of the Guidelines as appropriate. But many of them needed to be considered throughout the various phases of the life-cycle, and might need to be re-visited several times. For this reason they are grouped together in an Appendix at the end of the Guidelines.

A fundamental point, sometimes missed, is that the PLC is part of a system, and any “response time” must include the effects of sensors, links, and actuators, as well as the PLC.

The finite scan-period of a PLC is small enough to capture and control most events in industrial processes, but there are several situations where this scan-period must be taken into account. Short-lived events, shorter than the scan time, should be given special consideration due to the potential for aliasing problems; hardware solutions may be required. Sampling of analogue quantities is quite common, since many PLCs now incorporate extensive arithmetical processing functions. In this situation, the loop gain and the ability of the system to provide a response will depend on the way that the sampling is programmed in, and also on the scan time itself, which in turn may well be governed by the size of the program and any associated A/D conversion hardware.

The general guidance here is to fix the scan rate at a defined value if at all possible. Ideally, an automated tool would be used to predict a guaranteed Worst Case Execution Time, and the rate

would be set slightly slower. This is not yet possible with PLCs, but the Guidelines do recommend run-time execution checks during the Software Detailed Design phase (sometimes known as Unit Test), possibly via an emulator. In the highest integrity applications, if critical reliance is placed on timing then such predictions will have to be made with assurance, which at the present state of the art means extended run-time testing.

With some PLCs I/O or logical operations, if intensive, can hamper the proper timing operation and a degree of manual scheduling of the various operations is needed.

Timing recommendations

A table included in the Appendix to the Guidelines gives advice in the form of a list of “do’s and don’ts” with brief examples. These support the general principles discussed above, and some particular pieces of advice for PLCs has been singled out below.

Realistic specification	There is no point in over-engineering on response time. But failure of the complete system to deliver a timely response can affect safety.
Multitasking	Not recommended in the Guidelines unless backed up by Worst Case Execution Time Analysis and a scheduling model. There can be difficulties in allocating task priorities, variable updates etc. when time slicing.
Analyse/calculate/measure times	It is advisable to predict timing performance from several different viewpoints. Analysis can be performed using manufacturer’s information, any function block constraints, and by determining just how data propagates from inputs through program scans to outputs. Actual measurements during execution are also vital.
Communication times	It is not usually possible to guarantee the timing performance of communications links. Any safety-related timing or performance should not be made conditional on the timings assumed for such links.
Unbounded execution time	Any programmable feature which has an indeterminate or unpredictable execution time must be avoided. This includes loops which are not bounded, and recursion.

Operator interface	Operator inputs can be fleeting in nature, for instance when a keyboard is used. A timely “feedback”, confirming input, provides assurance that the system is continuing to work correctly. If an operator needs to respond to an output within a set time, the system must be able to take alternative action on time-out (due to operator absence, for example).
Defaults and cascade effects	Timers used in cascade can multiply errors, since each one will be correct only to within 1 scan period. On start-up, a default value will be necessary for those timers waiting for definition from communications data.
Self-test times	Monitoring of the self-test operations performed by safety sub-systems can prevent them being out of action, unknown, for an unexpected length of time.
Defined times	When interfacing to plant, there are often limits on the time interval between outputs, which can affect plant safety, between inputs, which if abnormal can be used as a diagnostic aid, or response time from input to output, which is often related to the detection or avoidance of a hazard.