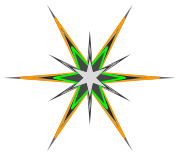


S T S A R C E S

Standards for Safety Related Complex Electronic Systems

FINAL REPORT

Safety-Related Complex Electronic Systems

Coordinator : INERIS

Partners : BIA
HSE
INRS
VTT
CETIM
INSHT – CNVM
SP
TÜV
SICK AG
JAY Electronique

Commencement date : 1st December 1997 - Completion date : 29th february 2000

European Commission – DG XII
Contract SMT 4CT97-2191

Summary

This project answers to a dedicated call for research in support to European Standardisation issued by the « Standard, Measurement and Testing » Programme. STSARCES examines the validation aspects of safety-related parts of control systems for machinery with regards to the problems encountered with modern electronic and programmable electronic technologies. This research was carried out by 11 research organisations, notified bodies and manufacturers from 6 countries of the EU through a range of related issues, including software and hardware validation, to assist in the development of pr EN 954-2 "Safety of machinery, Safety related parts of control systems, part 2 validation".

This report develops a framework for harmonized validation procedures, which should be standardised by CEN/CENELEC. The methodology is based on the overall safety lifecycle concept of a system, which is quite new in the field of the machinery sector, and covers both hardware and software. A significant part of the report deals with the software lifecycle, since it is not developed in EN 954.

The Markov modelling approach, also innovative when applied to the field of the machinery, has revealed very successful. The immense influence of the diagnostic coverage could be demonstrated and data on appropriate on-line test intervals for dedicated architectures, combined with realistic MTTF values, are provided and justified. This information provides fundamental advice for the system designer as well as hints for the persons carrying out the evaluations.

Attention has been given to prevent divergences from the requirements of the IEC 61508 since this norm has basic safety publication status. As a positive repercussion, STSARCES determines the validation methods of Programmable Electronic Systems in their uses for safety functions both in EN 954 and draft IEC 62061, a machine application standard derived from IEC 61508. It does allow defining credible and understandable links between categories (EN 954) and safety integrity levels or SILs (draft IEC 62061). This connection is indispensable during the design and development phases of control circuits for the machinery which make use of components based on the category concept, like mechanical, hydraulic, pneumatic, electro-mechanical ones, and PES better characterised by SIL concept.

An extensive presentation of the almost definitive results to ensure their wide acceptance by manufacturers has been carried out at the occasion of the International Conference on « Safety of Industrial Automated Systems », Montreal, October 1999. Thanks to its Organizing Committee, several sessions could be chaired by STSARCES members. The obtained feedback has influenced the presentation of this report, structured as a comprehensive guided tour through the lifecycle of systems, and with more deeply detailed technical contributions transferred to the annexes.

Table of contents

Figures

Tables

Annex

Annex 1	WP 1.1 : Software engineering tasks - Case tools
Annex 2	WP 1.2 : Software quality and safety requirements
Annex 3	WP 1.2 : Guide to evaluating software quality and safety requirements
Annex 4	WP 1.2 : Guide for the construction of software tests
Annex 5	WP 1.2 : Common mode faults in safety systems
Annex 6	WP 2.1 : Quantitative Analysis of Complex Electronic Systems using Fault Tree Analysis and Markov Modelling
Annex 7	WP 2.2 : Methods for fault detection
Annex 8	WP 3.1 : Safety Validation of Complex Components - Validation by Analysis
Annex 9	WP 3.2 : Validation of complex components : Intercomparison black box/white box tests
Annex 10	WP 3.3 : Safety Validation of Complex Components - Validation Tests
Annex 11	WP 4 : Applicability of IEC 61508 & EN 954. Task 1 : A study of the links and divergences between IEC 61508 and EN 954.
Annex 12	WP 4 : Task 2 : Machine Validation Exercise
Annex 13	WP 4 : Task 3 : Design Process Analysis
Annex 14	WP 5 : ASIC development and validation in safety components

Glossary

Acronyms	Definition
« / »	No requirement
« O »	Obligatory quality requirement
« R »	Recommended quality requirement
ASIC	Application specific integrated circuit
BB	Black Box
CAT	Category (according to EN 954-1)
CC	Current converter
CCF	Common cause factor (β)
CCS	Formalism used to describe parallelism semantics
CES	Complex Electronic System
CMF	Common Mode Failures
COB	Chip On Board
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
D	Drive
DNC	Direct Numerical Control
E/E/EP	Electrical/Electronic/Electronic Programmable (System) according to IEC 61508
EEPLD	Electrically Erasable Programmable Logic Device
EEPROM	Electrically Erasable Programmable Read Only Memory
EMC	Electro-Magnetic Compatibility
EN	European Norm
EPLD	Erasable Programmable Logic Device
ES	Emergency stop (actuator)
EUC	Equipment Under Control
FMEA	Failure Mode and Effect Analysis
FMECA	Failure Mode, Effect and Criticity Analysis
FPGA	Field Programmable Gates Array
FTA	Fault Tree Analysis
HCPLD	High Capacity Programmable Logic Device
HW	HardWare
IEC	International Electrotechnical Committee
IN	(Input of a) switch-off path of the drive
Ip, IP	(Input of the) switch-off path of the drive for the PED
Iw, IW	(Input of the) switch-off path of the drive for the watchdog

Acronyms	Definition
M	Motor
MCM	Multi Chip Module
MTTF, $MTTF_d$	Mean time to dangerous failure
PDF	(Average) probability of a dangerous failure per hour
PED	Programmable Electronic Device
PES	Programmable Electronic System
PFD	(Average) probability of failure on demand
PHA	Preliminary Hazards Analysis
PLC	Programmable logic controller
PN	Petri Network
RAM	Random Access Memory
RC	Relay circuit
ROM	Read Only Memory
S	general sensor
SADT	Structured Analysis Design Technique
SART	Structured Analysis Real Time
SIL	Safety Integrity Level according to IEC 61508
SRCES	Safety Related Complex Electronic System
SRCPES	Safety Related Complex Programmable Electronic System
Statecharts	Specification method based on transition systems
SW	SoftWare
WB	White Box
WD	Watchdog
Z	Formal specification language based on the Zermelo theory of sets
ß	Common cause factor (CCF)

1. INTRODUCTION

1.1. Objective

STSARCES (Standards for Safety-Related Complex Electronic Systems) was funded by the European Commission in answer to a dedicated call for proposals for research in support to European standardisation, initiated by CEN/CENELEC, and issued by the « Standard, Measurementt and Testing » Programme in 1996, to examine the validation[1] aspects of safety-related parts of control systems for machinery with regard to the problems encountered with modern electronic and programmable electronic technologies. This research was focused upon the development, or completion, of knowledge regarding validation techniques for both hardware and software elements of a machine control system in the context of the Machinery Directive (98/37/EC) and its implementing Regulations.

This research was carried out by a partnership of eleven organisations through completion of five work-packages (WPs) on a range of related issues, including software and hardware validation. The objective was to assist in the development of an emerging standard, prEN 954-2 'Safety of machinery – Safety-related parts of control systems – Part 2. Validation' by production of an document which describes proposed harmonised validation methods.

Pr EN 954-2 is a draft European standard that provides details of the measures and techniques that should be applied in order to validate the safety-related parts for all technologies applied of control systems for machinery. This proposed standard relates to safety-related parts designed in accordance with the general principles set out in EN 954-1 : 1996 'Safety of machinery – Safety-related parts of control systems – Part 1. General principles for design'.

A constraint imposed upon any validation methods developed from this research programme was that attention should be given to prevent divergence from the requirements of IEC 61508 'Functional safety of electrical/electronic/programmable electronic safety-related systems'. This was necessary since IEC 61508 has basic safety publication status and its principles may be preferable to those of EN 954 for electrotechnical aspects of safety-related complex electronic systems for machinery control.

INERIS, coordinator of the STSARCES project, and the following organisations participated in the research programme :

- INERIS (Institut National de l'Environnement Industriel et des Risques, of France)
- BIA (Berufsgenossenschaftliches Institut fur Arbeitssicherheit, of Germany)

[1] Validation is the activity of demonstrating that the safety-related parts of the control system under consideration, before or after installation, meets in all respects the safety and functional performance requirements specified for that safety-related control system.

- HSE (Health & Safety Executive, of United Kingdom)
- INRS (Institut National de Recherche et de Sécurite, of France)
- VTT (Technical Research Centre, of Finland)
- CETIM (Centre Technique des Industries Mecaniques, of France)
- INSHT (Instituto Nacional de Seguridad e Higiene en el Trabajo, of Spain)
- JAY (Jay Electronique SA, of France)
- SP (Swedish National Testing and Research Institute, of Sweden)
- TUV (TUV Product Service GMBH, of Germany)
- SICK AG (SICK AG Safety Systems Division, of Germany)

The research programme work-packages were assigned as :

- Work-package 1 : Software safety (leader – INRS)
  - WP 1.1 Software engineering tasks : CASE tools (CETIM)
  - WP 1.2 Tools for software faults avoidance (INRS)
- Work-package 2 : Hardware safety (leader – BIA)
  - WP 2.1 Quantitative analysis (BIA)
  - WP 2.2 Methods for fault detection (SP)
- Work-package 3 : Safety validation of complex components (leader – VTT)
  - WP 3.1 Validation by analysis (VTT)
  - WP 3.2 Intercomparison white-box/black-box tests (INSHT)
  - WP 3.3 Validation tests (TÜV)
- Work-package 4 : Link between the EN 954 and IEC 61508 standards (leader – HSE)
- Work-package 5 : Innovative technologies and designs (leader – INERIS)
  - Operational partners : Industrial JAY and test-house INERIS

1.2. What are complex electronic systems ?

The Machinery Directive (98/37/EC), which covers components that are separately supplied to fulfil a safety function, and EN 292 : 1991 ‘Safety of machinery – Basic concepts, general principles for design’ are, in general, based upon established practices in machine control system design, such as guard or power interlocking, where personnel may access hazardous areas for tasks such as setting, tool changing and maintenance. These safeguards are commonly designed and implemented at the machine after its basic control system design has been completed. This retrospective application of safeguards was (and remains) a practicable solution whenever there was an adequate degree of independence from the machines control system.

However, this approach to machinery safety has been shown to be less viable with the emergence of electronic and programmable electronic solutions (generically referred to as ‘complex electronic systems’ for the STSARCES Project) which have to be more closely integrated within the design of a machines control system. The safety-related control systems that implement these solutions often comprise a range of devices/components and electrical/electronic technologies.

These complex electronic systems may be characterised as machine control systems in which :

- the failure mode of at least one constituent device or component is not well defined ; or
- the behaviour of the device or component under fault conditions cannot be completely determined ; or
- there is insufficient dependable failure data (from field experience) to support claims for rates of failure for detected and undetected dangerous failures of the device or component.

An example of a ‘complex’ electronic systems are presence detection, speed and motion control schemes at a numerically controlled (NC) machine. This may involve a programmable electronic based machine controller that performs designated safety and non-safety functions.

This controller may be capable of processing the input signals received from motion control devices (or sensors) installed adjacent to the dangerous rotating or moving shafts and transmitting an output signal to actuating devices, such as a power drive system (which may be a complex electronic system in its own right) in order to reduce (or stop) the speed or motion to a safe level.

1.3. Problems to solve

Complex electronic and programmable electronic devices and components, such as large scale (LSI) and very large scale integrated (VLSI) circuits, application specific integrated circuits (ASICs), programmable logic controllers (PLCs), microcontrollers, etc, are increasingly being used in safety-related functions implemented by machine control systems.

The safety performance of such devices and components, whether as individual parts or in combination as a complete safety-related system, has been found difficult to establish in practice. This is primarily a result of the fundamental characteristics of a complex electronic system, which make it difficult to establish that its final implementation satisfies the necessary functional or safety performance requirements by testing a machine.

Consequently, it has been found that testing has to be supplemented by analysis of the design of the complex safety-realed electronic system used in machinery control to properly evaluate the safety performance of its hardware and software elements. There are a variety of measures and techniques for this design analysis based on quantitative and qualitative methodologies which may be used by machinery manufacturers and test houses.

Most established techniques and measures, such as fault tree and failure mode and effects analyses, have merit when used in combination with conventional testing philosophies for complex safety-related electronic systems. The difficulty for practitioners is in determining which measures may be suitable for particular machinery applications and in achieving consistency in their use.

These difficulties have to some extent been addressed by existing and emerging standards considered by the STSARCES Project.

1.4. EN 954-1 & IEC 61508

The STSARCES Project has included a comparison of the methodologies and requirements of two standards : IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems) and EN 954 (Safety of machinery - Safety related parts of control systems). This was carried out to determine how different are the requirements of these two standards for complex electronic and programmable electronic technologies when applied to machinery control systems.

Both standards propose a structured approach towards the design of safety-related control systems but differ in that EN 954 is designed to address all types of control system technologies whilst draft IEC 61508 has been primarily (but not exclusively) designed to apply to electrical, electronic and programmable electronic (referred to as E/E/PE) based safety-related control systems. The standards require that the safety-related functions of the control system are classified: IEC 61508 requires that the safety-related functions performed by a machine's control system be allocated a safety integrity level whilst EN 954 uses the concept of safety performance and places the system into one of five categories. There is a significant difference in the way that the safety integrity levels and categories are derived and defined and it is the problems that this difference causes that have been considered, especially as the two classifications were compared with a view to developing a strategy to link them.

IEC 61508 uses a safety lifecycle approach to ensure that the design of a safety control system is systematically carried out. This lifecycle approach is examined in the project to establish whether it would be suitable for the design of machinery control systems.

1.5. View of test houses

Certified safety-related CES are on the market today also in the machinery sector. Despite today there is no harmonised validation procedure for CES the following remarks can be made :

- The use of CES for safety-related control systems has for some time also been the advanced state of the art for the machine sector.
- There have been no accidents with the certified machines and safety devices which are attributable to programmable electronic technology in control unit.
- In view of the large number of different applications in practical use and the encouragingly low accident rate, it can be stated that CES have also proven themselves as a safety technology in mechanical engineering and, furthermore, that they often permit entirely new protection concepts.
- The expenditure for development on the part of the manufacturer and for validation by a testing body are usually different than those for classical control technologies. The challenge is to assure safety as well as high availability despite of high complexity. The greater flexibility, the lower production costs, the frequently lower maintenance requirement and the greater reliability compensate for the extra costs.
- Almost all manufacturers involved in certification processes had to make conceptual changes during development because, in some cases, serious weaknesses became apparent in the course of validation.
- Often only manufacturers involved in certification procedures with many years of experience in classical safety control systems had the sufficient know-how to develop

acceptable CES solutions. Other manufacturers who were tackling safety applications for the first time were frequently unable to produce acceptable solutions.

- The idea of specifying validation procedures for PES in standards so rigidly that every body and every person always comes to the same result in every part of the validation appears to be exaggerated in view of the complexity of the subject being dealt with. Even IEC 61508, with its more than 300 pages, prepared by selected international experts, is no guarantee for that. There is a technical limit to standardisation here which leads to the serious question of whether we should not be satisfied with a framework for a harmonised validation procedure.

As can be seen by these remarks European test houses need to give constructive answers to the validation of safety-related CES. Validation procedures are developed and there are several test houses who certify safety-related CES according to the machinery directive. There is still a lack of harmonisation between the different test houses working in that field. This final report of the European Project ”Standards for Safety-Related Complex Electronic Systems” will develop a framework for a harmonised validation procedure which should be standardised by CEN/CENELEC.

2. ANALYSIS OF PRESENT SITUATION

2.1. Increasing use of CES for safety applications

At the end of the 70s/beginning of the 80s, electronic systems were introduced into mechanical engineering. At first, they were used only in functions not related to safety. However, attempts were made to use such technology for safety-critical applications. This intention met with the better resistance of practically all established safety experts. It was not surprising : the electronic techniques met none of the safety criteria and principles of classic control technology which had been successfully applied in the past. A comparison shows just how big the problem was :

- Inherent safety, in other words, safety guaranteed by the design of individual components, cannot be achieved with electronic systems.
- It is virtually impossible to exclude the possibility of failures with physical causes, as is the case with electromechanical circuits.
- The nature of possible defects and their consequences are usually known in the case of electromechanical components; they are mostly unknown for complex integrated circuits.
- Programmable systems are highly complex. We have to accept – and this is something new – that these systems are no longer fully testable.
- If modifications are made, there is a relatively high risk – compared with conventional technology – that serious defects are integrated without being noticed.
- Electro-Magnetic Compatibility (EMC) is no problem in electromechanical components; there is extreme sensibility, however, in PES.
- Several other proven safety methods such as grounding control circuits, starting up by logic „high“ make little sense or are not practicable in electronic systems.

These problems were solved by national and international standardisation papers which introduced basic requirements especially during design and testing of CES to overcome these difficulties. All these papers are based on a safety life cycle. The following three examples will illustrate how the situation developed during the past ten years.

Presumably, paper-cutting machines were the first machines to use computers for safety functions at the beginning of the 80s. Paper-cutting machines, which are used in large numbers in paper-processing factories, are especially dangerous machines. Fingers and hands can be seriously injured by the press cross-head and blade if a cut unexpectedly occurs as a result of a malfunction or if integrated protective devices (electro-sensitive device and two-hand control) fail.

This posed a problem because there was a lack of practical experience in industrial applications and suitable assessment methods. In cooperation with a German test house, the first machine manufacturer decided to develop the control in diverse redundancy where one channel was the computer, another was hard-wired in CMOS logic. In the course of over 15 years and several generations of machines, almost all manufacturers of large paper-cutting machines now use computer control systems. State-of-the-art technology for category 4 controls (according to EN 954-1) means : diverse or homogeneous redundancy with fail-safe comparator, two-hand control and lightcurtain.

The great need of industry forced the national standardisation body in Germany to create a general standard for safety-related computer control, see DIN V VDE 0801[1]. The technical content of this national standard has later been brought into the European, see EN954-1[2] and international standardisation, see IEC 61508[3].

Electro-sensitive protective equipment has been used to ensure the safety of machines and potentially dangerous areas for almost 30 years. So far, protective field geometry has always been unchangeable: computers were not used until around 1992. At the beginning of the 90s several light curtains have been manufactured which contained microcontrollers in homogeneous redundancy. As a basis for certification the German draft DIN V VDE 0801 was used. This standard is based on DIN V 19250[4] which builds up a hierarchical system of eight risk reduction levels. These first light curtains using CES were certified according to level 5 of DIN V VDE 19250 and could be used as electro-sensitive protective devices for power presses.

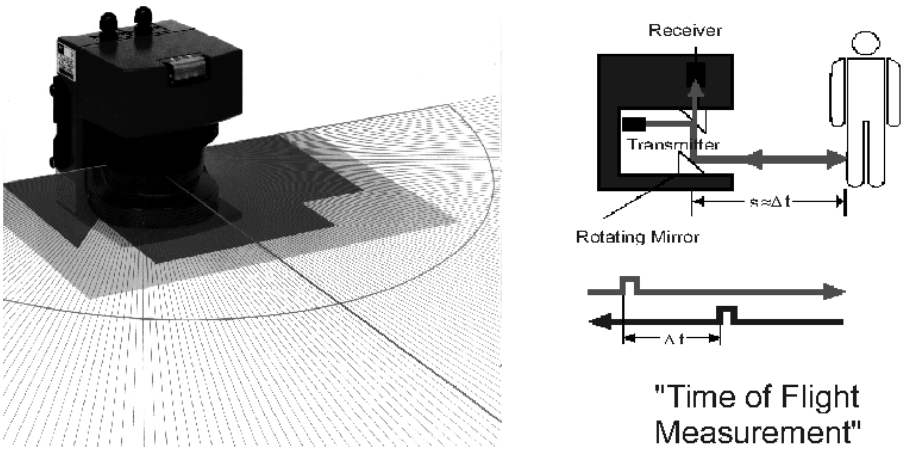


Figure 1 : Laser scanner for area protection

In the last 2-3 years, there has been a real "quantum leap" in these systems. Not only microprocessors but the applied physical principle are revolutionising the previous safety philosophy of electro-sensitive protective equipment. In the past, for example, the safety function of electro-sensitive protective devices was actuated exclusively through interruption of the beam of light.

In the illustrated example **(**Figure 1**)** of the "laser scanner", the human body in the danger area is detected through the reflection of an infrared beam of light. A rotating beam of light rapidly scans the danger area and transmits an image of the area to the computer. To determine the person's exact position, the running time of the light from the equipment to the person and back is measured. The protective field geometry can be adjusted via the software. This makes its use very flexible.

The electro-sensitive scanner depicted here is implemented in Category 3 according to EN 954-1. The architecture of the system is single-channelled with numerous self-tests and additional monitoring devices[5].

The first company to introduce such a system on the market took 15 man years in a development period of 3 years. Because of the many fundamental questions which had to be answered, the effort to validate the first system by the test house took 1.5 man-years. Today more than sixty thousand computer controlled electro-sensitive devices are used in all kinds of applications. An accident, caused by technical faults in the system, was not reported since now.

Machining centres are numerically controlled and have facilities to enable tools to be changed automatically from a magazine or similar storage unit in accordance with the machining programme. The most dangerous situation is an unexpected movement, start or speed acceleration when the worker observes the process while the protective guards are still open. To avoid this risk the machining centres in the past required to open and close the protective guards very often in the setting mode.

With a new safety concept, based on computers, the worker can operate in the setting mode by open protective guards and observe and estimate the machine´s movements. In the new approach a safe monitoring was integrated in a diverse redundant computer. The computers primarily responsible for the non-safety functions of the machine are part of the diverse redundant architecture[6].

If the automatic motion is controlled in this safe way the user can move inside defined areas. Any deviation in space or velocity is detected by the diverse redundant adjustable speed power drive system, realised by safe software. In a highly flexible way the machine can be adapted to the work of the user and not vice versa. The safety functions realised by the integrated monitoring are safely reduced speed, safe operational stop, safe standstill, safe limit switch (by software) and safe position switches (by software).

2.2. Basis for the validation of CES

As mentioned in the previous paragraph safety-related CES today are certified according to EN 954 in conjunction with national specifications, like DIN V VDE 0801. Some authorities also have used draft international standard IEC 61508 for the certification of safety-related programmable logic controllers[7]. In most cases a certificate has EN 954-1, national specifications and sometimes IEC 61508 mentioned as test requirements. Nevertheless today there is no internationally used procedure for the validation of safety-related CES in the machinery sector and practice between test-houses in Europe. Some general remarks can be made on the basis of the procedures used today :

- Validation at the end of development only, when the product is complete, is no longer possible. As the development process itself is an essential subject of the validation, it is advisable to involve the validating body in advance in order to agree on the documents to be submitted.

- The specification of the safety-related CES is of key importance and needs to be inspected by the certifying authority. One of the important inputs of the specification is the required risk reduction of the CES when used in a specific safety function. If a CES is manufactured without a specific safety function in mind, the minimum achievable risk reduction of the CES, as a subsystem, has to be specified.

- During all phases of the product life cycle analytical and testing measures are necessary to achieve a product which is robust against random and systematic failures. These measures have to be taken mainly by the CES manufacturer and they increase the effort of development dramatically.

- Software is getting more and more important for the safety integrity of modern CES. A comprehensive understanding of the safety-related software is essential for the CES validation. The simpler the hardware looks like the more complex the integrated software can be.

- The installer and user of the CES needs sufficient information and also ergonomic software for a safe installation and use of safety-related CES. This documentation and the software are a very important part of the validation process.

- The modification of CES is a very critical process and has to be planed during the first design. The certifying authorities have to be involved into the entire maintenance process of safety-related CES.

The following chapter will describe a harmonised procedure for the validation of safety-related CES while chapter 4 considers the applicability of EN 954 and IEC 61508 to the machinery sector.

3. ACHIEVING SAFETY BY FOLLOWING THE LIFE CYCLE

3.1. Introduction

3.1.1. The Overall Safety Lifecycle

The complexity of current electronic systems is such that it is impossible for the analyst to validate the safety level of a CES system by carrying out only activities when the design process is well underway or even completed.

This new approach, called the lifecycle, was not so important for electromechanical or low integrated electronic technologies which were generally used until now in the safety of machinery.

To deal with safety correctly, it is now necessary to have an overall view of the different stages that mark out the "life" of an application, from the definition of the limits of this application through at least the safety validation (It is often difficult for the machinery to deal with the maintenance and the final decommissioning of the elements). These stages are generally grouped within a safety lifecycle at global application level (Cf. IEC 61508-1). This lays down the activities necessary to implement the different means for risk reduction, namely CES systems, but also safety-related systems based on other technologies and external risk reduction facilities.

Three broad parts in the overall safety lifecycle can be distinguished.

[1] The first stages, based on the definition of the concept and a hazard and a risk analysis at global application level in particular, lead to allocating safety requirements to the different means of risk reduction. Two reference systems can potentially be employed to carry out hazard and risk analyses : EN 954 (in conjunction with EN 1050[8]) and IEC 61508 (see chapter 4 for the choice of a reference system).

[2] Then comes the lifecycle inherent to the development of the CES system(s) employed, but also to safety-related systems based on other technologies and external risk reduction facilities.

[3] These two first parts are completed by the phases of installation and overall safety validation, operation and possible modifications to the E/E/PES (with, where appropriate, a return to the adequate life cycle phase).

3.1.2. The E/E/PES Safety LifeCycle

In this section, the content of the lifecycle concerning the development of CES Systems will be examined. This cycle starts at the specification of the necessary and sufficient conditions for the realisation of such systems and finishes when all the CES safety-related systems are no longer available for use.

The principle phases that make up the lifecycle are the following :

- Specification : Based on the allocation made in the overall safety lifecycle, this phase specifies the safety functions carried out by the CES as well as the performance in terms of Safety Integrated Level (IEC 61508) or of category (EN 954), the choice of one or the other being oriented by considering the elements given in chapter 4.

- Architecture : This phase determines the hardware (simple channel, dual or triple redundancy, diagnostic coverage, etc.) and software (use of diversity, etc.) architecture adopted to develop the CES(s), see chapter 3.3.

- Design and development : This phase is intended to design and produce the hardware and software of the CES system(s) in order to respect the requirements laid down in the specification phase. Safety validation planning of the CES in question must take place alongside this phase. Section 3.1.3 looks at the specific life cycle of the software.

- Validation : The validation is intended to ensure that the requirements laid down in the specification phase are indeed respected.

So as not be judge and judged, the interventions of the analyst are carried out primarily at CES specification and validation level. Sections 3.2 to 3.5 deal with each of these phases.

3.1.3. The Software LifeCycle

The software life cycle is a subset of the CES lifecycle which concerns the activities occurring during a period of time that starts when software is designed and ends when the software is permanently disused.

The unrolling of the activities is defined in relation to the particularities of the project , for example, complexity of the system under consideration and the software, end-use of previously developed products, production of a prototype or availability of testing materials. The organisation of these activities can be represented by different, distinct lifecycles, defined according to the type of project in question.

3.1.3.1.The software V-lifecycle

The V-shaped cycle is chosen for its simplicity and because it is adapted to software products. A test activity is placed directly across from a development activity to ensure that the specified elements have been correctly achieved.

This lifecycle corresponds to a continuous development activity, with no or infrequent deliveries of intermediate products and with no significant changes to the specifications.

This V-shaped representation is illustrated in the following schema :

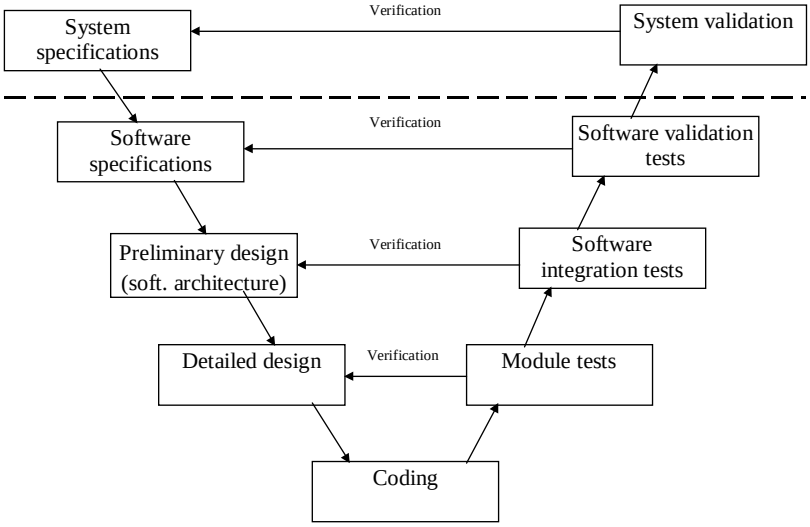


Figure 2 : The software V-lifecycle

Software definition usually includes the following sequential development activities :

- Specification : activity that consists in describing the expected software functionalities, and that takes into account inputs and any applicable constraints.
- Design : activity of constructing the software architecture (usually called "preliminary design"), which allows the implementation of software specifications and the construction of detailed algorithms (activity usually referred to as "detailed design"). The source code can then be constructed without any further information.
- Coding : activity of producing the source code, in languages such as assembly language or C, which can then be accepted by an assembler or compiler for the production of executable instructions for use by the target processor.
- Executable code production: activity in which the different code components are grouped (by link editing especially) into a form that allows the generation of an executable code for loading into the target system.

These definition activities are complemented by software verification activities, the goal of which is to verify that the software product satisfies specified requirements at each development steps and to detect any errors that might have been introduced during the software development.

The principal activity of any software verification is testing. Other activities such as review or analysis (for example rereading the code) are possible components of software verification.

Software testing activities generally include different phases that correspond to different development activities :

- Module tests at the level of each individual module can be used to demonstrate that the module carries out the specified function, and only this function. Different types of module tests can be found, including logical tests (error search, verification of correctness of the interconnections of the different branches, search for abnormal behaviour) and calculation tests (verification of calculation results, performance and exactness of algorithms). Calculation tests typically include data tests within specification limits as well as outside these limits (abnormal state), at the specified limits, and at algorithmic singularities. Abnormal behaviour tests (outside boundary values, algorithmic singularities, errors) are generally referred to as robustness tests.

- Software integration tests are used to demonstrate that the functional units made up of an assembly of modules operate correctly. This type of test is principally concerned with the verification of the interconnections between modules, data circulation, dynamic aspects and the sequencing of expected events. They typically include tests on inter-modular connections, dynamic aspects, the sequencing of expected events, and the rerun of operations in case of interruption.

- Validation tests are to check that the software installed in the hardware satisfies the functional specifications, especially by verifying hardware/software interfaces, general performance levels, real-time operation, general functions, and the use and allocation of resources.

3.1.3.2.Other software lifecycles : the Incremental lifecycle

There are other safety lifecycle which were not expressed earlier like incremental lifecycle which primarily concerns those projects which have not been accurately defined.

The exploratory lifecycle concerned projects that deal with the development of a product concept that relies on innovative technologies or on technologies which have not yet been totally mastered.

More details can be found in Annexes 1 to 5.

3.1.4. Lifecycle Requirements

The goal of the following lifecycle requirements[2] is to obtain a formalised description of the organisation of the development and, in particular, of the different technical tasks that the development is composed of. This description promotes improved planning of the development activities, and more thought going into the optimal timeline for this development.

- The software development lifecycle should be specified and documented (e.g in a Software Quality Plan). The lifecycle should include all the technical activities and phases necessary and sufficient for software development.
- Each phase of the lifecycle should be divided into its elementary tasks and should include a description of :
  - inputs (documents, standards etc.),
  - outputs (documents produced, analytical reports, etc.),
  - activities to be carried out,
  - verifications to be performed (analyses, tests, etc.).
- The necessary documentation should be established at each phase of the lifecycle to facilitate verification and validation, and the software safety requirements should be traceable and capable of being verified at each stage of the process (traceability matrix for each definition document).

  This will avoid a situation where the only available documentation is the source code because the documents that should have been prepared were not (deadlines too tight, project manager transferred to another contract, etc.).
- The analyst should be able to carry out the evaluation of software conformity to the present requirements by conducting any audits or expertises deemed useful during the different software development phases.

[2] These requirements are not unique to the software life cycle and can therefore be applied to the design of the different sub-assemblies of an CES.

All technical aspects of the software lifecycle processes are subject to evaluation by the analyst.

The analyst must be allowed to consult all verification reports (tests, analyses, etc.) and all technical documents used during software development.

The intervention of the analyst at the specification phase is preferable to an a posteriori intervention since it should limit the impact of any decisions made. On the other hand, financial and human aspects of the project are not subject to evaluation.

It is in the interest of the applicant to provide proof of all activities carried out during software development.

The analyst should have all the necessary elements at his or her disposal in order to formulate an opinion. Subcontracted software should not be left out of the evaluation procedure and should comply with the same documentation’s requirements.

3.2. Specification

3.2.1. Specification procedure for safety software

Today, software programs are characterised by their complexity and size which are such that it seems overly optimistic to strive to avoid all faults they contain, by means of their construction and by testing at the end of development. In order to master software development, it is absolutely necessary that a procedure adapted to this technology be established.

During the life span of a software program, one of the most delicate steps to be accomplished is the translation of needs into specifications. The drafting of specifications and the possibility of evaluating these specifications are very important advantages, in particular for safety software. This is confirmed by a study run by HSE concerning the primary causes of failures, based on 34 incidents, which shows the primordial proportion (44.1%) caused by poor specifications (Figure 3).

It is estimated that repairing an error in specifications requires approximately 20 times more effort if it is only detected when used, and sometimes even more. In fact, the consequences of software failures are not limited to repairs. In certain cases, human lives may be involved.

Developing systems is often accompanied by quality requirements, which are certainly necessary, but insufficient in the case of safety systems. Quality requirements must be considered along with safety requirements.

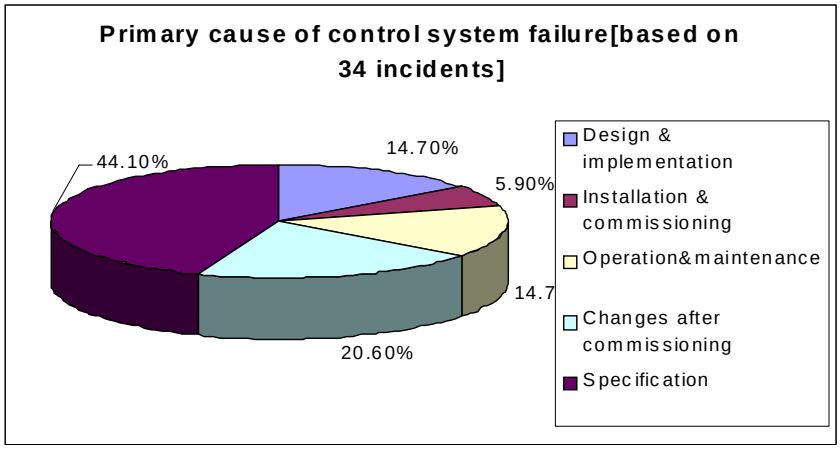


Figure 3 : Major portion attributed to specifications in the causes of failures

Safety software specifications procedures lie within the framework of a system operation safety approach which makes it possible to master the risks of the system in its environment. It is a complete methodological approach which must be planned and systematic in order to identify, analyse and control the risks throughout the life cycle of the system, in particular throughout the life cycle of the software, with the intention to anticipate and reduce incidents.

While traditional quality procedures implement means which aim towards a system exempt from faults, the operation safety procedure implements other means in order to aim towards a system exempt from failures (Figure 4).

The zero fault objective is a lure which must be completely abandoned when faced with strict safety requirements. It is unrealistic to strive to eliminate all faults, which leads to tremendous efforts far above ordinary economic rules and beyond the present limits of software production technology and the state of the art.

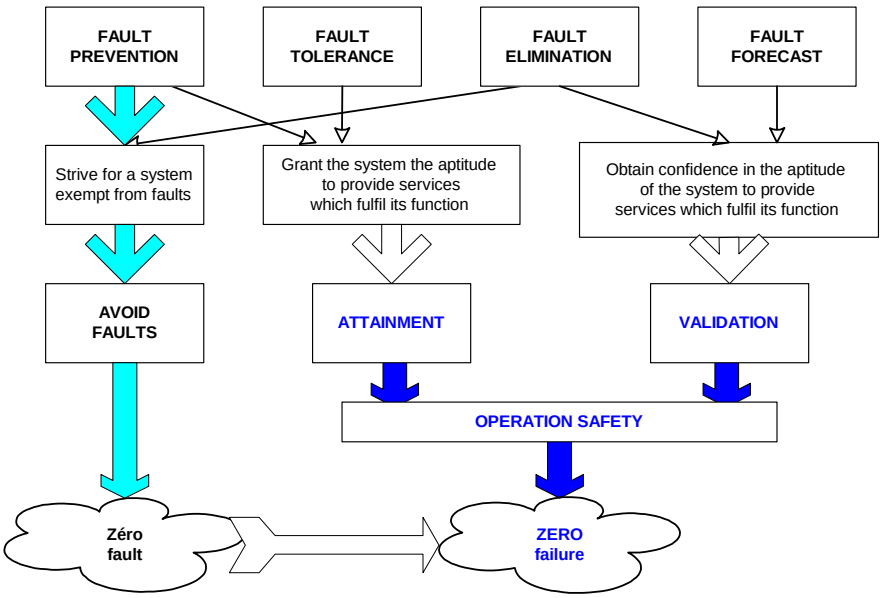


Figure 4 : Operation safety means

3.2.2. Specification methods

In the 1980's, Mr. Gérald WEINBERG, a specialist in programming and in software engineering, declared that "if buildings were built like programs are written, the first flight of birds would destroy everything".

The situation has evolved since then. Methods and Software Engineering Workshop (SEW) tools are offered to developers, in order to help them and accompany them in their work. The table below provides a non-exhaustive list of a few specification methods and their main characteristics.

These methods may be used for all types of software. The safety requirements at the software specification level are taken into account by the operation safety methods.

Name	Place in the life cycle and purpose of the method	Observations
PN Pétri Network	Specification Development	Method based on transition systems, using tokens and spaces. It makes it possible to demonstrate properties such as non-blocking, vivacity or equity of a set of co-operating processes. It is often used to specify parallelism and synchronisation.
Statecharts	Specification Development	Specification method based on transition systems.
SADT (Structured Analysis Design Technique)	Specification, design Development	Graphic specification method. It uses boxes to represent data or activities and arrows to represent flows between data or these activities. It is sometimes designated as a semi-formal design method and is often used in industry.
SART (Structured Analysis Real Time)	Specification Development	Real time extension proposed for the structured S.A. method of E. Yourdon and T. de Marco. One of the most widely used structured software analysis methods for real time applications.
Z	Specification, design Development	Formal specification language based on the Zermelo theory of sets. It makes it possible to express functional conditions of the problem to be translated into set notation.
LDS	Specification Development	Specification and functional description language. It is subject to a CCITT standard.
CCS	Specification Development	Formalism used to describe parallelism semantics. It is based on process algebra and remains very abstract and impossible to be used to make useful conclusions.
CSP	Specification Development	Presents the same characteristics as CCS.

Table 1 : Specification methods

3.2.3. Case tools for safety software specifications

To our knowledge, no SEW combines specification tools and operation safety tools for software specification, while taking into account safety requirements. Nevertheless, there is a whole store of software specification software programs on the market which implement formal methods (VDM methods, B methods, etc.) and semi-formal methods (SART methods, SADT methods, etc.), as well as software programs which make it possible to create FMEA and arborescent failure charts.

Tool	Designer	Platform	Method
ATELIER B	STERIA, GEC-ALTHOM	UNIX (HP-UX-LINUX-SOLARIS)	Formal B method
AXIOM/SYS	STG	Microsoft (Windows 3.xx, Windows 95, Windows NT)	SART
DESIGN IDEF	METASOFTWARE	Microsoft (Windows 3.xx, Windows 95, Windows NT)	IDEF0 and IDEFIX
ENVISION	FUTURE TECH.SYSTEMS	Microsoft (Windows 3.xx, Windows 95, Windows NT)	SART, SADT, UML
OBJECT GEODE	VERILOG	UNIX (AIX, HP, UX, SOLARIS, SUN/OS)	OMT, SDL, MSC
ORCHIS	TNI	Microsoft and UNIX	IDEF0
STP-SE	AONIX	Microsoft and UNIX	SART
SYNCCHARTS	SIMILOG	UNIX	Esterel
TEAMWORK	CAYENNE	Microsoft, UNIX	SART

Table 2 : Case tools for safety software specifications

Formal methods are more than a tool for representation ; they are also a technique for drafting specifications which restrains the designer to make abstractions and finally results in a better anlysis and an enhanced degree of modelisation of the specification problems. It is sometimes even possible to make simulations.

Use of a formal method requires considerable investments in time and training. Formal methods are a significant step forward for the development and evaluation of critical software.

Software Specifications	Level	
	1	2
Software specifications should take the following points into account : • safety functions with a quantitative description of the performance criteria (precision, exactness) and temporal constraints (response time), all with tolerances or margins when possible, • non safety functions with a quantitative description of the performance criteria (precision, exactness) and temporal constraints (response time), all with tolerances or margins when possible, • system configuration or architecture, • instructions relevant to hardware safety integrity (programmable electronic systems, sensors, actuators, etc.), • instructions relevant to software integrity and the safety of safety functions, • constraints related to memory capacity and system response time, • operator and equipment interfaces, • instructions for software self-monitoring and for hardware monitoring carried out by the software, • instructions that allow all the safety functions to be checked while the system is working (on-line testing). *The instructions for monitoring, developed taking safety objectives and operating constraints (duration of continuous operation ,etc.) into account, can include devices such as watch dogs, CPU load monitoring, feedback of output to input for software self-monitoring. For hardware monitoring, CPU and memory monitoring, etc.* *Instructions for safety function verification: for example, the possibility of periodically verifying the correct operation of safety devices should be included in the specifications.*	O	O
Functional requirements should be specified for each functional mode. The transition from one mode to the other should be specified *Functional modes can include:* • *nominal modes,* • *one or more degraded modes.* *The objective is to specify the behaviour in all situations in order to avoid unexpected behaviours in non-nominal contexts.*	O	O

Table 3 : Software specifications

Notation :

The requirements discussed in the present document are organised into two software requirement levels (1,2) according to the criticity of the functions ensured by the software. Level 2 corresponds to the highest requirements for the software considered in this document. The level associated with a given function depends on a risk analysis of the entire system.

The level can be used to establish a list of elementary requirements for the software under consideration. Three degrees of importance can be defined to help decisions whether or not it is necessary to consider a given requirement as a function of the level of criticity:

- "**O**" (**O**bligatory quality requirement) : this requirement should be applied systematically to the software in question.
- "**R**" (**R**ecommended quality requirement) : the application of this requirement is recommended but not automatically imposed.
- **"/"** (no requirement) : the application of this requirement is left to the user's discretion.

3.3. Architecture

The architecture of a safety-related Complex Electronic System (CES) is one of the key elements in achieving the required Category (CAT) or Safety Integrity Level (SIL). This is the reason why the architectures play a prominent role in connecting categories and safety integrity levels.

Requirements on architecture are related to hardware and software. In the first paragraph of this chapter hardware architectures which are commonly used in machinery (so-called “designated architectures”) are described and an overview of the Markov simulations of these architectures is given. More detailed information on the results can be found in Annex 6. In the second chapter the influence of software architecture is analysed. A quantification of this influence can only partially be considered by using the common cause factor. Annex 6 gives more information on the influence of software fault avoidance through architecture.

3.3.1. Designated CES Architectures for Machinery

3.3.1.1. The Role of Architectures for Safety Related Systems

Studying EN 954-1 one can interpret the requirements for the different categories as architectural constraints for the design of safety-related parts of control systems. In the categories B and 1 a single channel architecture without monitoring facilities is described while category 2 specifies a single channel architecture with monitoring and redundant switch off path and categories 3 and 4 normally need redundant signal processing and reaction to fulfil the requirements. Transferring the requirements of EN 954-1 into complex electronic safety-related systems will lead to different system architectures that are in use in the machinery sector.

With these common architectures brought into different Markov models the probability of a dangerous failure per hour (PDF) or the probability of failure on demand (PFD) can be calculated for different diagnostic coverages, mean times to failure and common cause factors. The results of these calculations can be used to establish relationships between commonly used architectures in the machinery sector and different safety integrity levels as defined in IEC 61508-1.

The following paragraph will describe commonly used architectures for CES in the machinery sector. Several assumptions are made which are typical for these architectures. A proof test is not standard in the machinery sector for CES. Therefore the proof test interval was set to 10 years which is the average time a safety-related CES is in use today (called the “mission time”). Chapter 3 of Annex 6 will describe how the PFD and the PDF have been calculated taking into account the demand on the CES and determining the average over the mission time.

3.3.2. Common Architectures for Machinery

3.3.2.1. Single channel system without fault detection in accordance with category B or 1 of EN 954-1

The categories B or 1 according to EN 954-1 imply that the system does not provide any capability of detecting internal faults. For category 1 not only basic but well-tried safety principles and components must be used, which means that a higher reliability is achieved and therefore the probability of a system failure is lower than in category B (see EN 954-1, 6.22).

If we assume the system comprising a sensor (S), a programmable electronic device (PED) with integrated power supply for signal evaluation and a drive (D) that is controlled by the PED it can be represented in a block diagram by a simple series system. This is shown in Figure 5.

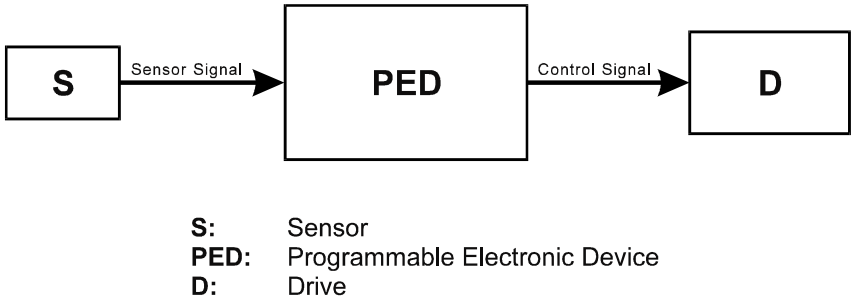


Figure 5 : Block diagram of a single channel system without fault detection

Normally a single electronic device is not regarded to be a well-tried component. Thus, it is not possible to realise a category 1 single channel safety system using a PED.

Three assumptions have been made in order to determine the SIL :

[1] Switching off the drive is the appropriate action to generate a safe state of the machine (in IEC 61508 called “equipment under control” (EUC)) the drive is belonging to.

[2] The safety system is not able to induce a hazardous situation by itself. The worst case which can occur is a dangerous failure, i.e. the system cannot perform it's intended safety function.

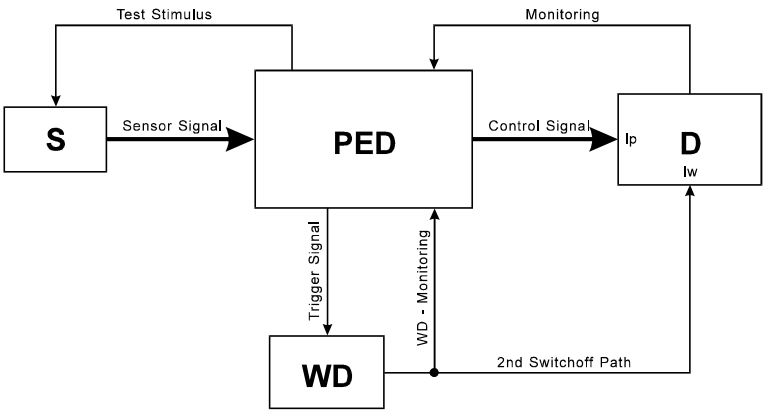
[3] Failures are only revealed by a demand on the safety function. This leads to a hazardous situation which will be followed by a repair.

3.3.2.2. Single channel system with implemented tests in accordance with category 2 of EN 954-1

Category 2 of EN 954-1 requires self checks to be executed by the safety related system "at suitable intervals". The tests may be initiated either manually or automatically. If a fault is detected an output signal shall be generated in order to initiate an "appropriate control action". Whenever possible a safe state shall be induced.

These requirements imply "that the occurrence of a fault can lead to the loss of the safety function between the checking intervals". Additionally it must be remarked that many of the typical testing techniques do not provide a diagnostic coverage of 100%. Therefore there may exist faults within the safety device which cannot be detected by the checks.

A representative system architecture for category 2 is presented by the block diagram of Figure 6.



S: Sensor
PED: Programmable Electronic Device
D: Drive
WD: Watchdog

Figure 6 : Block diagram of a single channel system with implemented tests

Compared with the simple system of Figure 5 a watchdog (WD) has been added in order to monitor the operation of the programmable electronic device (PED) which is thought to be represented by a microcontroller system. In the PED a power supply is integrated. The drive (D) has two separate inputs, the first (Ip) - as usual - for the PED and a second one (Iw) for the watchdog, each providing full switch-off capability. The system is also performing periodic tests of the sensor, the switch-off path(s) of the drive and the watchdog.

Several assumptions have been made in order to ease the creation of a suitable Markov model :

[1] Switching off the drive is the appropriate action to generate a safe state of the equipment under control (EUC) the drive is belonging to.

[2] The safety system is not able to induce a hazardous situation by itself. The worst case which can occur is a dangerous failure, i.e. the system cannot perform it's intended safety function.

[3] The programmable electronic device (PED) is periodically performing a self test. The detection of a dangerous fault in the PED is acknowledged by the absence of trigger pulses within the watchdog (WD) time out period. This online test is characterised by the test rate and the diagnostic coverage C_{PE} which is assigned a value between zero and one. C_{PE} is the conditional probability that a dangerous failure of PED will be detected, given that it has occurred. In this case the PED is no longer able to cut off the drive via input Ip although this might be necessary. If the fault is detectable the drive will be cut off by the watchdog via input Iw (presumed that WD and Iw both are operational).

[4] The sensor and the drive-internal switch-off path beginning with input Ip of the drive are tested periodically by the PED. The diagnostic coverages are assumed to be equal to one as long as the tests are carried out.

[5] The watchdog is also tested by the PED. The diagnostic coverage is supposed to be equal to one. There are two ways to monitor the operation of the watchdog. It's output signal can either be directly reread by the PED or the drive-internal switch-off path beginning with input Iw of the drive can be included in the test loop. In the latter case the switch-off path is included in the test loop. This can be expressed by the diagnostic coverage which is set either to zero or to one.

[6] Any failure which has been detected successfully will drive the system to a non-volatile safe state with the drive cut off. The system is assumed to be disconnected from the power manually until it has been repaired or replaced by a new one.

[7] If the PED has failed it will no longer perform any tests of PED-external components, i.e. S, Ip, WD and Iw are not tested in case of a failure of PED.

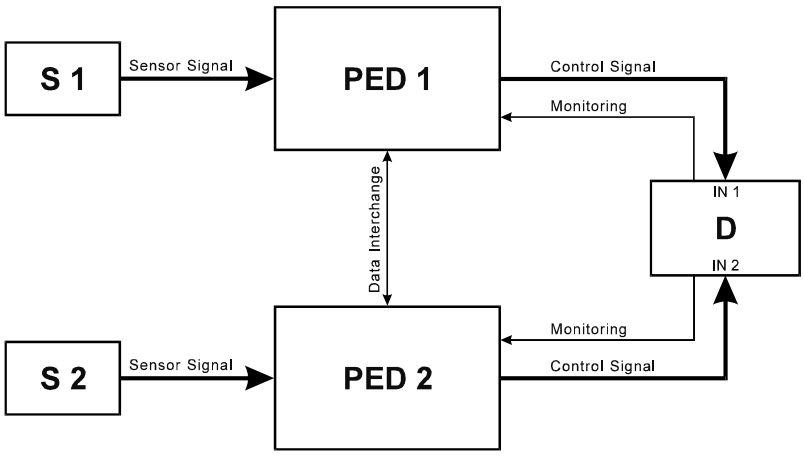
[8] In order to describe the drive by a single dangerous failure rate this rate is equally distributed to both inputs Ip and Iw respectively.

3.3.2.3. Dual channel system with comparison in accordance with category 3 or 4 of EN 954-1

EN 954-1 requires a category 3 device to remain operational if a single fault is present in any part of the system. Besides, "whenever reasonably practicable the single fault shall be detected at or before the next demand upon the safety function." This includes that not all faults should be detected and that "the accumulation of undetected faults may lead to an unintended output and a hazardous situation at the machine." Common mode failures shall be taken into account.

In addition to above-mentioned demands there are more rigid requirements to be fulfilled by a system that claims for category 4. The single fault shall be detected "whenever possible" and, "if this detection is not possible, then an accumulation of faults shall not lead to a loss of safety functions."
The problem of providing the safety functions after the occurrence of a fault is often solved by the implementation of redundancy. A typical example for homogeneous redundancy is given by the dual channel system depicted by Figure 7. Whether category 3 or 4 can be met depends on the extent to which faults can be detected or tolerated.

The system comprises two sensors (S1, S2) of same type and two programmable electronic devices (PED1, PED2) of identical type with integrated power supply in each PED combined with a single drive (D). Either of the PEDs is connected to an individual input (IN1, IN2) of the drive. In reality the PEDs will usually be given by microcontrollers. The cross link between them is intended for data interchange.



S 1, S 2: Sensor
PED 1, PED 2: Programmable Electronic Device
D: Drive

Figure 7 : Block diagram of a dual channel system with comparison

Again, there is a number of reasonable assumptions which have been made in order to derive a suitable Markov model :

- Assumption 1 : Switching off the drive is the appropriate action to generate a safe state of the equipment under control (EUC) the drive is belonging to.
- Assumption 2 : The safety system is not able to induce a hazardous situation by itself. The worst case which can occur is a dangerous failure, i.e. the system cannot perform it's intended safety function.
- Assumption 3 : Periodic online tests are carried out by the two programmable electronic devices (PEDs). The complete set of tests includes :
    - a self-test of PED1 controlled and monitored by PED2,
    - a self-test of PED2 controlled and monitored by PED1,
    - a test of the drive-internal switch-off path beginning with input IN1 of the drive, performed by PED1,
    - a test of the drive-internal switch-off path beginning with input IN2 of the drive, performed by PED2,
    - a comparison of the output signals of the two sensors (S1, S2), performed by PED1 and PED2 together.

    Each of the tests is checking subfunctions which are performed by the different components. Performing all subfunctions properly is a pre-condition for the safety system to provide it's intended safety function(s).
- Assumption 4 : The mutually contolled and monitored self-tests of the PEDs are characterised by a diagnostic coverage, which can be assigned a value between zero and one.
- Assumption 5 : The diagnostic coverage related to the sensors is equal to one. In some cases the feature will be implemented, in others it won't. This can be expressed by the diagnostic coverage which is set either to zero or to one.
- Assumption 6 : The diagnostic coverage related to the drive-internal switch-off paths beginning with inputs IN1 and IN2 of the drive is equal to one. In some cases the feature will be implemented, in others it won't. This can be expressed by the diagnostic coverage which is set either to zero or to one.[3]

[3] For machinery normally only a few digital sensors like switches are used. Monitoring of the drive is also done by digital signals. Thus a 100% diagnostic coverage is possible.

- Assumption 7 : Any failure which has been detected successfully will lead the system to a non-volatile safe state with the drive cut off. The system is assumed to be disconnected from the power manually until it has been repaired or replaced by a new one.

- Assumption 8 : If one PED has failed dangerous it will no longer perform the test of it's related drive input. The comparison of the output signals of the sensors is also inhibited.

- Assumption 9 : The same dangerous failure of both sensors at the same time is not detectable because they deliver identical (wrong) output signals. This can not be revealed by a comparison.

- Assumption 10 : The failure rate of each input channel of the drive is given by : $\lambda_I = 0.5 \cdot \lambda_D$

- Assumption 11 : Common cause effects do not hit complete channels but the two sensors, the two PEDs and the two switch-off inputs of the drive separately.

3.3.2.4. Dual channel system in mixed technology in accordance with category 3 of EN 954-1

In many applications a mixed technology is used in order to implement a safety function. A first channel is given by a standard programmable logic controller (PLC) with integrated power supply and no specific online tests, while the second channel is formed by electromechanical means. Online tests are carried out by the PLC to check the elements of the electromechanical signal path.

As an example the simplified schematic of Figure 8 depicts the implementation of an emergency stop function employing a PLC and a relay circuit.

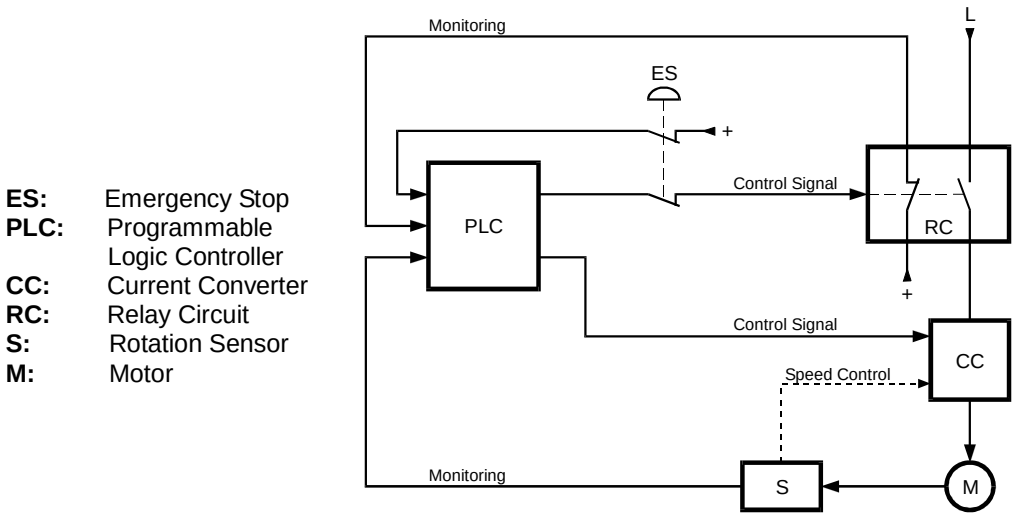


Figure 8 : Implementation of an emergency stop function using mixed technology

We assume a machine where a current converter (CC) is controlled by a standard PLC. The rotation sensor (S) is part of the speed or position control of the current converter and can be used by the PLC to monitor the motor movements.

The safety function to be implemented is the emergency stop of the dangerous movement as soon as the emergency stop device (ES) is actuated. The actuator contains two mechanically forced contacts, either of them providing a separate output signal. One of which is processed by the PLC while the other is led to a relay circuit (RC) consisting of 2 relays (or contactors respectively) with forced contacts. The emergency stop function is executed by both the PLC via the current converter and the relay circuit. A failure of the opening of the contacts of the emergency stop actuator device is excluded. Independent random failures are supposed to happen to the PLC, the current converter, the relay circuit and the sensor while the emergency stop actuator ES is imputed not to fail to open it's contacts if the button is pressed.

The PLC software is designed so that the opening of the contact of ES immediately leads to a stop signal for the current converter. Four online tests can be modelled by our Markov model. If one of the tests is not implemented in reality the pertinent test rate may be set to zero.

For the online tests the following assumptions are made :

- PLC diagnostic test : as said before a standard PLC is used. Therefore we assume only simple online tests like a watchdog and parity bit test of the memory which are common today also for standard electronics. This will result in a low diagnostic coverage of 30%. We assume that the PLC after failure detection permanently switches off the outputs connected with CC and RC.

- CC diagnostic test : in suitable time intervals e.g. once per day or during maintenance the PLC switches off the motor movement using the current converter (CC). In parallel the PLC monitors the output signal of the rotation sensor (S) so that it can detect the reaction of CC. If the movement is not stopped by CC the PLC permanently stops the motor via the relay circuit (RC).

- Rotation sensor diagnostic test : the diagnostic test of CC can only be effective if the rotation sensor (S) is able to detect the motion of the motor. To check this the PLC is reading the sensor signal after switching on the motor. If the motion is not detected the PLC permanently stops the motor using the relays circuit (RC).

- Relay circuit diagnostic test : after a normal stop of the motor using CC and after executing the CC diagnostic test the PLC switches off the control signal for RC. Simultaneously the PLC monitors the corresponding contact(s) of the relay circuit (RC). If RC does not react properly the PLC permanently stops the motor via the current converter (CC). Because of the test's simplicity the diagnostic coverage can reach 100%.

3.3.2.5. Triple channel system with comparison in accordance with category 4 of EN 954-1

In few (rare) cases the problem of providing the safety functions after the occurrence of a fault is solved by the implementation of triple redundancy. A typical example for homogeneous redundancy is given by the triple channel system depicted by Figure 9. Whether category 3 or 4 can be met depends on the extent to which faults can be detected or tolerated.

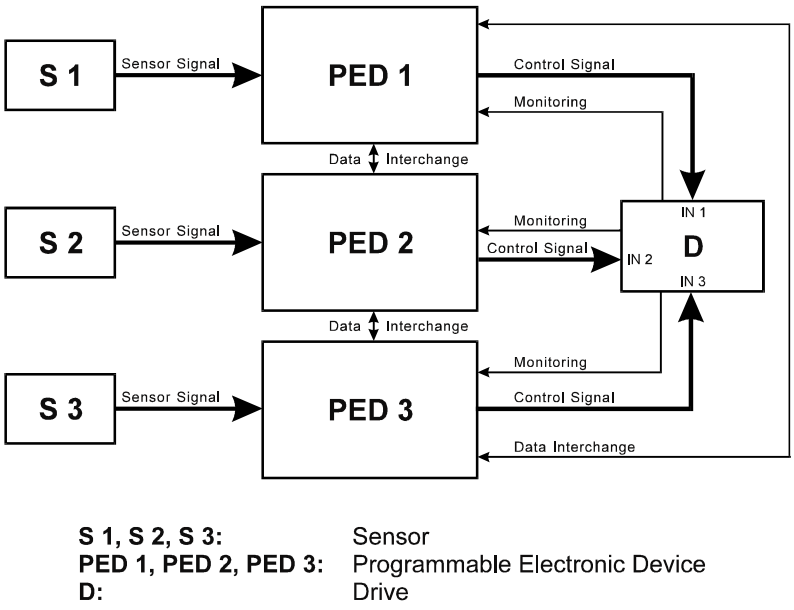


Figure 9 : Block diagram of a triple channel system with comparison

The system comprises three sensors (S1, S2 and S3) of same type and three programmable electronic devices (PED1, PED2 and PED3) of identical type (with integrated power supply) in connection with a single drive (D). Each PED is connected to an individual input (IN1, IN2 and IN3) of the drive. In reality the PEDs will usually be given by microcontrollers. The three cross links between them are intended for data interchange. More or less the same or equivalent assumptions are made as for the dual channel architecture in chapter 3.3.2.3 in order to derive a Markov model which can deliver comparable results.

3.3.3. Designated architectures of CES for the machinery sector

It could be shown in Annex 6 that typical architectures used in machinery which fulfil the requirements of EN 954-1 can be linked to the SILs of IEC 61508. Figure 10 compiles some results obtained by the Markov models presented in Annex 6.

In order to make different architectures comparable the input parameters for identical or similar functional units have been set to the same values. In other cases reasonable values have been chosen.

Unless otherwise noted, the following input data have been assumed :

- MTTF of sensors, PEDs and PLCs : 15 years
- MTTF of switch-off paths of the drive : 30 years
- MTTF of a watchdog : 100 years
- MTTF of a relay circuit (two contactors) : 50 years
- Mission time (life time) : 10 years
- Repair rate (after failure detection or hazardous event) : 1/(8 hours)
- All test rates of single channel systems : 1/(15 min)
- All test rates of dual or triple channel systems : 1/(10 s)
- All demand rates of single channel systems : 1/(24 hours)
- All demand rates of dual or triple channel systems : 10/hour

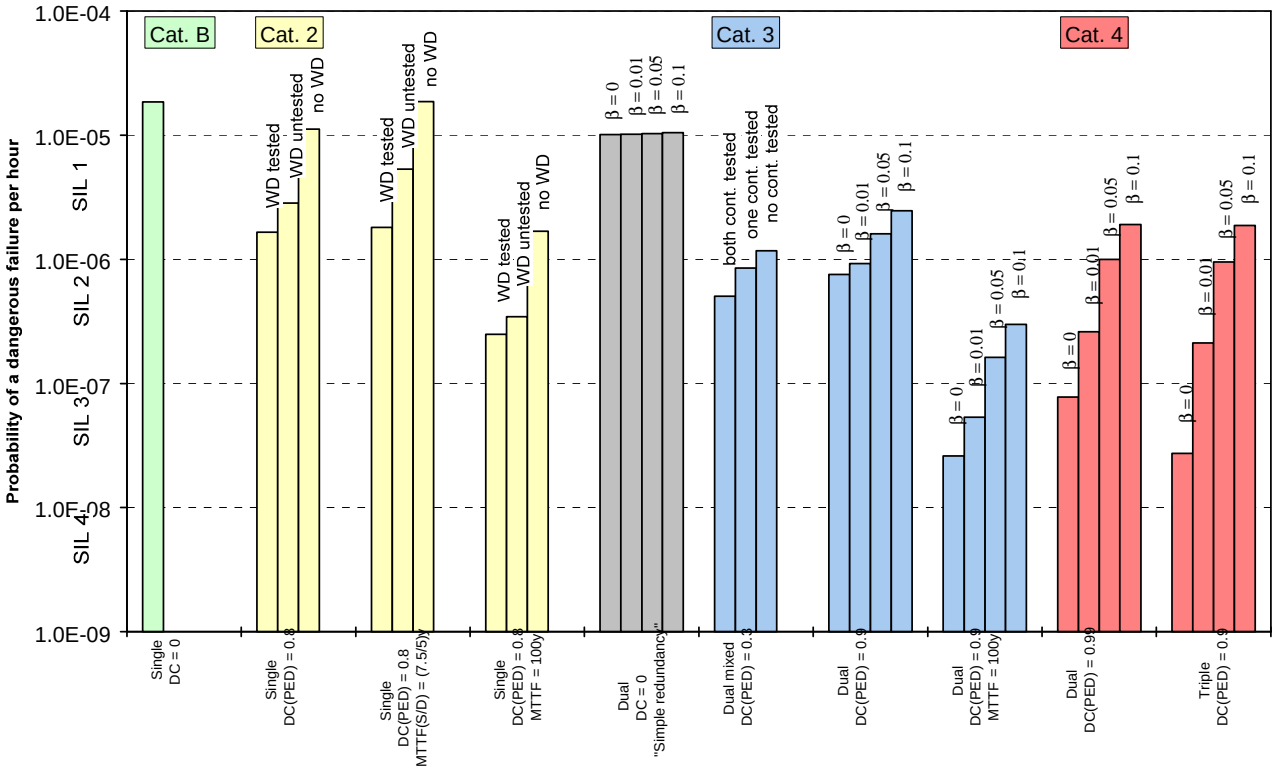


Figure 10 : Comparison of different architectures used in machinery

All evaluations have been executed applying the high demand procedure. As shown in Figure 10, SILs 1 to 3 can be achieved by system architectures belonging to different categories. For category B no link to a SIL is possible. With category 2 and suitable tests running in a time interval which is about 100 times smaller than the mean time between demands SIL 1 is achievable (for more information see Annex 6). Redundancy without any diagnostic tests running is comparable to category B systems and cannot be used even for SIL 1. Redundancy in mixed technology may achieve SIL 2 if online testing of the periphery is implemented. To achieve SIL 3 a redundant system needs to have 99% diagnostic coverage or a much better MTTF of the subsystems than we presumed for our reference systems. Given appropriate conditions SIL 3 is possible with a triple redundant system.

Figure 10 demonstrates that simple doubling of signal processing paths and implementing no online tests ("simple redundancy") does not provide a significant gain if the mission time has a similar order of magnitude as the MTTF of a single channel. Other investigations have shown that "simple redundancy" can only have a positive effect if the mission time is one order of magnitude smaller than the MTTF. For simple systems (e.g. contactors or valves) which can be proof tested once a year (i.e. 100% diagnostic coverage for all subsystems) simple doubling of the hardware may be useful. For complex subsystems like ASICs or PEDs simple doubling is only useful if the MTTF is one order of magnitude bigger (possible e.g. for some ASICs) than the mission time (life time) of the safety system. In all other cases online diagnostics are essential also in redundant safety-related systems.

These results compiled in Figure 10 could be helpful for standardisation. A link may be drawn between SILs and the categories for so called designated architectures. The architectures introduced in this chapter are proposed to be considered as designated architectures for the machinery sector. A manufacturer who can prove that his architecture is equivalent to one of the designated architectures, has to determine the $MTTF_{dangerous}$ of his subsystems, the diagnostic coverage of the online tests and, in case of redundant systems, estimate the common cause factor. Then he may derive the SIL out of a table. As an example, a table of this kind is presented in the following. This table is the compilation of results achieved by choosing particular input data. New Markov modelling will be necessary only if system architectures and/or parameters for the subsystems are used, which are not listed in the table.

There are several data banks which could be employed to determine the MTTF of hardware components, for example FARADIP.THREE[9], MIL HDBK 217 or the Siemens Standard SN 29500[10]. The results obtained are sensitive to the data bank used and therefore to attain comparable results one should preferably use only one of them. The diagnostic coverage can be determined using the failure model in annex A of part 2 of IEC 61508. Part 6 of IEC 61508 may be helpful to estimate the common cause factor β. Standardisation could specify one methodology for estimating the CCF. With this proposal a link between the two standards IEC 61508 and EN 954-1 is possible, but it is not a fixed link between categories and SILs.

3.3.4. Conclusions for designated CES Architectures for machinery

Markov models turned out to be a very appropriate tools for determining the Safety Integrity Level (SIL) of safety related systems. Enriched by a specific feature allowing for the online test rate and the demand rate on the safety function the probability of a failure on demand (PFD) and the probability of a dangerous failure per hour (PDF) can be calculated. Thus the method is covering the "low demand mode of operation" as well as the "high demand or continuous mode of operation" as defined in IEC 61508.

Applying the developed technique to a number of typical system architectures commonly used in machinery systematic investigations could be carried out. Hence a lot of information could be obtained about the influence of fundamental system parameters such as the component's mean time to failure (MTTF), the diagnostic coverage (DC) and the repetition rate of automatic online tests, the demand rate on the safety function and the common cause factor (β) in multiple channel architectures. This information is useful not only for the validator but may also help the system designer to chose an appropriate system structure and to fix basic design parameters for a given application.

Some of the results obtained by Markov modelling are listed in the following :

- Assuming reasonable failure rates of the components it will be hard to achieve even SIL 1 with an untested single channel system. Theoretically the situation may be improved by introducing regular proof tests but this is neither realistic for complex electronics nor are proof tests standard for electronic devices in machinery.

- Online testing is essential not only for single channel structures but also for multiple channel architectures. Using components with a sufficient (not unrealistic) MTTF SIL 3 can be achieved with a diagnostic coverage of 90% or more. The simple watchdog of a typical single channel system will usually provide a lower diagnostic coverage so that only SIL 2 may be attainable.

- The influence of the online test interval depends on the system architecture. Considering a single channel system for maximum test efficiency the test interval should be chosen smaller by a factor of at least 100 than the mean time between demands on the safety function. In a dual channel system the test interval may be much smaller since it is related to the MTTF of one of the channels.

- Common cause failures can substantially reduce the SIL of especially homogenous redundant structures. The benefit of a dual or triple channel system may be completely lost due to a common cause factor of a few percent only.

The system architectures having been made subject to the investigations of this report are proposed to be considered as "designated architectures" for the machinery sector. Exemplary evaluation results listed in a table provide a helpful overview on the performance of different technical solutions. Involving additional input parameters like MTTF of system components and diagnostic coverage, the table is able to draw a link between the categories of EN 954-1 and the SIL according to IEC 61508. Furthermore, in some cases even validation concerning the Safety Integrity Level to be achieved by a particular safety related system may be simplified by application of this table.

SIL	System Architecture	Mean Time to dangerous Failure $MTTF_d$ (years)	CCF β (%)	Diagnostic Coverage (each Channel) (%)	Cat.
		In/Processing/Out		In/Processing/Out	
-	Single PE, Single I/O	15/15/30	-	0/0/0	**B**
	Single PE, Single I, Ext. WD(u/t)	15/15/30	-	0/60/0	**B**
	Dual PE, Dual I/O, 1oo2	15/15/30	5	0/0/0	**B**
1	Single PE, Single I, Ext. WD(u/t)	15/15/30	-	100/60/100	**2**
	Single PE, Single I, Ext. WD(u/t)	7.5/15/10	-	100/60/100	**2**
	Dual PE, IPC, Dual I/O, 1oo2	15/15/30	5	100/60/100	**3**
	Dual PE, IPC, Dual I/O, 1oo2	15/15/30	10	100/90/100	**3**
	Dual PE, IPC, Dual I/O, 1oo2	45/15/60	10	100/90/100	**3**
	Triple PE, IPC, Triple I/O, 1oo3	15/15/30	5	100/60/100	**3**
	Triple PE, IPC, Triple I/O, 1oo3	15/15/30	10	100/90/100	**4**
2	Single PE, Single I, Ext. WD(t)	15/15/30	-	100/90*/100	**2**
	Dual PE, IPC, Dual I/O, 1oo2	15/15/30	1	100/90/100	**3**
	Dual PE, IPC, Dual I/O, 1oo2	30/30/60	5	100/90/100	**3**
	Dual PE, IPC, Dual I/O, 1oo2	7.5/15/10	1	100/99/100	**4**
	Mixed Dual Processing, Dual O, 1oo2	∞/(15/100)/(15/100)	-	0/(30/100)/(100/100)	**3**
	Triple PE, IPC, Triple I/O, 1oo3	15/15/30	1	100/60/100	**3**
	Triple PE, IPC, Triple I/O, 1oo3	100/100/200	10	100/90/100	**4**
3	Single PE, Single I, Ext. WD(t)	30/30/60	-	100/99*/100	**2**
	Dual PE, IPC, Dual I/O, 1oo2	45/45/90	1	100/99/100	**4**
	Triple PE, IPC, Triple I/O, 1oo3	100/100/200	1	100/90/100	**4**

Conditions for single channel systems :
All test rates : 1/(15 min)
Demand rate : 1/(24 h)
Repair rate : 1/(8h)
Mission time (life time) : 10 years
$MTTF_d$ of watchdog: 100 years
$MTTF_d$ of switch-off path for watchdog:
WD(u/t): Watchdog and pertinent switch-off path untested or tested
WD(t): Watchdog and pertinent switch-off path tested
(* not achievable by simple watchdog)

Conditions for dual or triple channel systems :
All test rates: 1/(24h)
Demand rate: 10/h
Repair rate: 1/(8h)
Mission time (life time): 10 years
$MTTF_d$ of output sensor of mixed system: 15 years
equal to normal switch-off path (output sensor not tested)

IPC : Inter-processor communication

Table 4 : Possible designated architectures for machinery

3.3.5. Influence of Software Architecture

3.3.5.1.Software Architecture and Common Cause

Partial or full redundancy is a powerful tool to achieve higher CATs and if used in connection with on-line tests also higher SILs. Hardware redundancy in CES is often connected with software redundancy. In parallel to hardware architecture the influence of software architecture has to be analysed as a part of design.

The question of common mode faults, inherent to the concept of redundancy, then arises. The example of compilation is very revealing. Two identical source software products can produce erroneous executable codes if these codes are generated by the same erroneous compiler. The common source of faults is therefore the compiler which systematically introduces errors into the programmes. These errors, if no precaution is taken, produce common mode failures that, in certain cases, can diminish the safety of the application.

Standardisation bodies have therefore taken care to draw the attention of designers and of those responsible for the evaluation to the problems linked to these types of failures. Hence, a note associated with category 4 of EN 954-1 lays down that.

If further faults occur as a result of the first single fault, the fault and all consequent faults shall be considered as a single fault. Common mode failures shall be taken into account, for example by using diversity or special procedures to identify such faults.

Common mode failures (the result of a single initial fault) are equivalent to single faults, and must therefore not affect the safety of the application. This is a very strong obligation as, strictly speaking, only recourse to a diversified and validated structure satisfies the requirements of category 4 for such failures.

In the preceding example, the designer would be led to use two distinct and validated compilers if it turns out that errors can be introduced on compilation.

The standard proposes two ways of taking these failures into account : diversity or the use of procedures to identify common mode failures. The latter is not open to question and must be followed as soon as a redundant structure has been employed, otherwise the benefits stemming from it may be lost. In contrast, recourse to diversity must be studied carefully so as not to lead designers to these complex means as, in certain cases, they can consist of two distinct developments and products. In addition, the efficiency with respect to common mode faults can sometimes be questionable if no precautions have been taken.

Going back to the example of software, it can be tempting to design two different software to carry out the same function. The limitations of this technique, a priori attractive, nevertheless quickly become apparent. It is indeed very difficult to give a final guarantee of the absence of common points between two such programmes. The same specification is often used which, if erroneous, will lead to two programmes with similar failures for the same input data ; both programmes may have been designed and coded by two people or two teams with a similar culture, generating software errors with identical consequences, etc.

This example quickly demonstrates that the diversity laid down in EN 954 cannot be imposed without precautions to deal with common mode phenomena.

3.3.5.2. Common Mode Failures (CMF) and their appearance mechanism

Defining common mode failures avoids any confusion or misunderstanding relative to the problems posed and their possible solutions. Indeed, many use the terms common mode faults, common mode failures, and common cause failures indifferently when referring to common phenomena affecting several distinct entities. This paragraph is intended to clarify the concept of Common Mode Failure by explaining the sequence of phenomena involved in the lead up to these failures.

Fault / Error / Failure

A reminder should first be given of what a failure is. The failure are the transition from correct service to incorrect service[11 & 12]. The failure occurs when the service provided no longer conforms to the specification. The deemed or supposed cause of a failure is a fault, a definition that constitutes a shortcut who introduces the intermediate concept of error.

This terminology is in everyday usage within the scientific community. It is more accurate than that found in the EN 954 standard which does not distinguish between failures and faults. A fault is thus defined as the state of an entity unable to accomplish a required task but not including incapacity due to preventive maintenance or other pre-programmed actions.

Failure mode / Failure mechanism

With failure defined, the mode of failure can then be defined, something that should not be confused with failure mechanism, which is the physical process (e.g. corrosion) that has led to the failure[13]. To all intents and purposes, the failure mode as being the observable manifestation of the failure (e.g. short circuit, transistor output stuck, cut in a circuit). This definition is of course relative to the level of observation of phenomena; the failure of a transistor can be considered as global system fault.

Common Mode Failure

With this clarification made, the definition of Common Mode Failures[14] clearly synthesises the different wordings found in the bibliography[15]. Common mode failures are failures affecting multiple entities, simultaneous and multiple, dependent on a single initial cause. This definition has the advantage of applying to all types of architecture, without involving the nature of the redundancy.

Simultaneity is an important point that should be noted when talking about common mode failures. states that failures can take place at identical or distinct times but, that at a given moment, the failure states coincide. The length of the time interval is crucial as it indeed allows discrimination between correlated failures and multiple independent failures.

Common mode failure appearance mechanism

The definition given by J.C. LAPRIE clearly shows the sequence of phenomena leading to common mode failure. At the outset is a common cause capable of generating correlated faults. These faults should be distinguished from independent faults, which are attributed to different causes. Correlated faults are at the root of similar errors that, when activated or revealed, provoke a common mode failure. The following diagram (Figure 11) is therefore obtained.

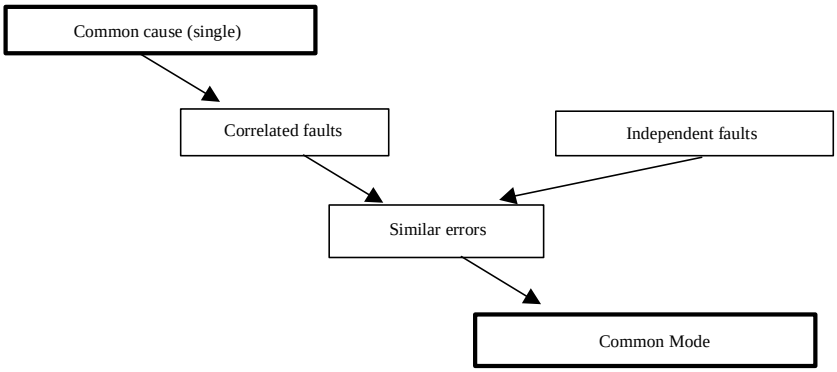


Figure 11 : Phenomena leading to a common mode failure

Common mode failures usually originate from a common cause. In certain cases, it may be that independent faults lead to similar errors. On account of their low probability of appearance, these faults will not be considered in the remainder of this document.

3.3.5.3.Software Diversity

Diversity is a technique which consists in creating n-versions of an entity (hardware or software) whilst introducing one or several differences into each entity or its development process in order to avoid common mode failures.

Functional diversity is often quoted to overcome common mode failures[16]. It consists in acquiring different parameters, using different technologies, different logic or algorithms, or different means of activating outputs to provide several ways of detection[17].

It is, however, difficult to justify diversity throughout the processing chain, except for very high safety applications. The most important advantages of diversity are at CPU, interface memory, programme, and data format level.

Diversity is seen as a solution to common mode failures that cannot be predicted. It is complementary to the concepts of independence and separation.

Software diversity is in fact a subset of design diversity, which is isolated on account of its importance. It's the development and the execution of different but functionally equivalent versions (or variants) in order to detect eventual errors by comparing in real time the results attained.

Following the identification of a state of error recovery is undertaken by :

- Backwards recovery in which the system returns to the state previously saved.
- Forwards recovery in which the system branches in a new state (usually in degraded mode) in which the system is able to operate.
- Error compensation, based on an algorithm using the redundancy built into the system to furnish the right reply.

Error detection by verification of the results arising from the different variants can be undertaken by :

- Acceptance (internal) test to check the results of the execution of a programme. The calculation of the checksum represents a typical example of an acceptation test.
- External coherence, in which results are checked by means of external intervention.
- Automatic checking of numerical results. This is the verification of the numerical accuracy of algorithmic results, for example for floating calculations for which a minor error in a result could propagate to take on ever increasing importance.

More often than not, versions are designed by different teams, to achieve the same safety objective. As for design diversity, it is assumed that different designers will not make the same mistakes. Neither the bibliography nor the field experience permit to know the conditions of optimal software diversity (it may be enough to produce two different designs from the same specification)[18]. In fact, the software must have sufficiently diversified dynamic and logic parameters to be considered as diversified.

Nevertheless, UCRL proposes classifying the factors increasing software diversity in the following decreasing order of importance :

- different algorithms, logic, and programme architecture,
- different sequences and order of execution,
- different programming languages,
- different programming environment.

Two basic techniques are used for fault tolerance LYU.

3.3.5.4. Recovery Blocks

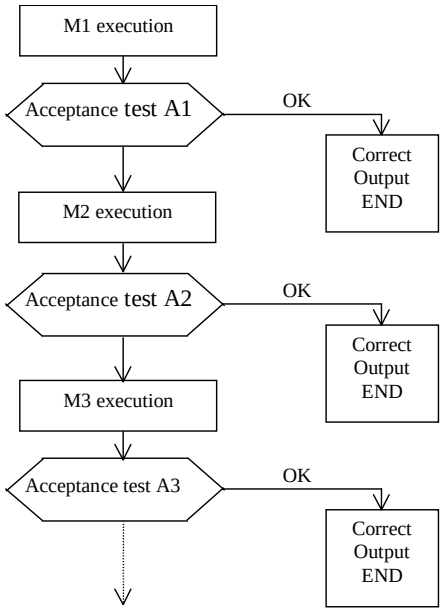


Figure 12 : Example for Recovery Blocks

Several blocks functionally equivalent (M1, M2, M3, etc.) are created and executed sequentially as long as an error is detected by the modules undertaking the acceptance tests (A1, A2, A3, etc.) assigned to each block Mi (Figure 12).

Proper application of this principle means that the acceptance tests (A1, A2, A3, etc.) should be distinct but in practice a single test common to all the blocks is often developed. An extreme case consists in adopting an acceptance test that is similar to the blocks and then comparing the output from the monitored blocks with the results of the acceptance test. One of the problems posed by this method in a monoprocessor environment is found in the sequential nature of the execution of the versions.

3.3.5.5. N-version programming

N-version programming has been the subject of academic experiments intended primarily to ascertain the efficiency limits with respect to common mode failures. This technique consists in running multiple versions (N) of a software product in parallel, and taking a majority vote to determine the final result. The number of versions depends on the number of faults that are to be tolerated (3 versions will be able to tolerate 1 fault). The assumption on which its efficiency is established is based on the following diagram (Figure 13).

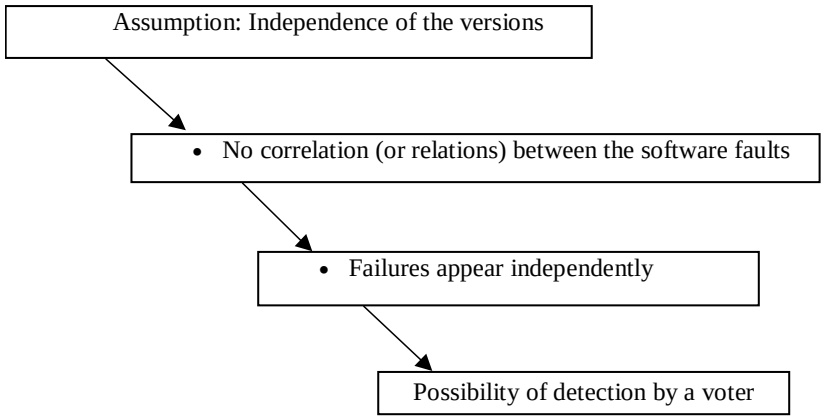


Figure 13 : Principle of N-version programming

In order to be fully efficient, this technique must be carried out in line with the following rules :

- Requirements must be specified and analysed with formal methods.
- The specification documents must be debugged and stabilised before the development of any components (for example by developing final code prototypes).
- A protocol must exist in order to know and solve the problems. This protocol should contain measures ensuring independence in development and should not introduce correlated faults such as, for example, communication errors, common lack of knowledge , or exchanges of erroneous information between the development teams.
- Verification, validation and the test must all be formalised and must show absence of correlated faults.
- The specifications, design and the code must all be tested.

The advantages of N-versions programming are :

- Simplification of the test as it is enough to run N programmes with the same inputs and compare the outputs obtained.

- The reliability of each version can be lower, the contribution at the global level provided by the comparison. It must, however, be sufficient not to degrade the reliability level of all N versions.

- The higher development costs can be compensated by a reduction in validation costs, these advantages being linked to the assumption of non correlation of the N versions.

These advantages are minimised by the conclusions of various experiments :

- The increases in reliability provided by N-versions programming depend on the non correlation of the failures of the different versions[19 & 20]. Experience and probabilistic calculations have shown that there is no true independence between the different versions developed[21]. The rate of appearance of correlated failures obtained following the experimentation is much higher than that calculated by making the assumption of fault independence. Strictly independent developments are therefore not enough to guarantee significant benefits in terms of reliability.

- Even if, on average, substantial benefits are possible using N-versions programming, these benefits are so variable that it is still possible to combine several versions to obtain a poor result.

- The results are relative to the experiments carried out. Extension to any type of application is therefore difficult.

- The use of different languages to create different versions of a software product does not have a major impact on reducing the causes of correlated failures.

In addition to these conclusions, various lessons can be drawn from the experiments carried out :

- The different experiments were conducted on relatively simple modules of reduced size. When large programmes composed of numerous interconnected modules are involved, identifying and separating the critical parts and only applying N-versions programming to these parts.

- A general rule applies to the development of diversified programmes ; the earlier the development teams come into contact, the greater the chance of introducing common mode faults[22].

- At the outset of the software life cycle there is necessarily a common specification. However, there must be a minimum of different design processes, to avoid errors at this level being propagated throughout the software life cycle.

- Diversity creates a dilemma that is difficult to solve ; not stating the algorithm in a specification does promote diversity but may generate faults due, for instance, to level of understanding of the programmers.

3.3.6. Key Questions for Software Fault Avoidance through Architecture

There are checklists developed in WP1.2d which rise the following key questions during the development of the software architecture in order to deal with software common mode failures. The key questions are structured along the safety life cycle for software development.

3.3.6.1. Software specifications

Have the software specifications been developed by different people ? If no, what are the links between these people ?

The specifications can be written in different forms.

It is very difficult, if not impossible, to introduce real diversity, at software level, links often being necessary between two teams responsible for drawing up the specifications. Have the common mode faults likely to affect the software been determined ? What are the potential sources of CMF at the system level ?

CMF determination is vital to take these phenomena into account correctly.

In particular, the possibility of introducing systematic faults over the course of every stage of the software life cycle is to be analysed.

Does the software redundancy chosen take account of common modes ?

Taking CMF into account can / must influence the choice of structure : temporal, structural, functional diversity, etc.

What type of diversity has been specified for the software :

- Human diversity ?
- Functional diversity ?
- Signal diversity ? If so, Input signal diversity ? Output signal diversity ? Signal processing diversity ?
- Equipment diversity ?

The types of diversity chosen are determined by the CMFs likely to affect the software. Of course, software and hardware diversity are linked.

Homogeneous diversity does not guard against design faults.

Has the software been developed in accordance with a quality plan or quality requirements ?

Following a quality plan for the development of a software product contributes to fault avoidance (whether common mode or not). It is vital, in particular when the software architecture is homogeneous.

A minimum software quality must be ensured, even in the case of a diversified structure. Quality avoids the introduction of software faults, in particular at the maintenance stage.

3.3.6.2. Software Design

In the case of diversity, have the different software products been designed by different people ?

Be attentive to possible links between teams responsible for designing different software versions.

Skills of the designers ?

A lack of designer skill can lead to the introduction of similar faults in the different versions of a software product.

Are the algorithms identical from one channel to another ?

Are the development tools employed identical from one channel to another ?

What logic separation is there between redundant channels ?

Separation is a constructive technique that does not propagate the failures of one function to another, thereby limiting the analyses to sensitive points.

Are there shared memories that could propagate a logic fault from one channel to another ?

Is software execution desynchronised ?

Desynchronisation of the execution of two programmes creates an offset favourable to reducing faults stemming from common external perturbation, for example electromagnetic perturbation.

3.3.6.3. Software coding

In the case of diversity, have the different software products been coded by different people ?

Be attentive to possible links between teams responsible for coding the different versions of a software product.

Have different programming languages been used for each channel ?

Have different compilers or assemblers been employed ?

3.4. Design and development

There is no simple answer to know how software and hardware of safety-related parts of machine control systems should be designed. Several aspects will be important to reach the objective of a control system safe enough for its intended application. Safety considerations all through the life cycle will be important.

This Chapter describes some aspects of software, microcomputer hardware and complex integrated circuits which have to be addressed during the development phase of the life cycle :

- How should safety-related software be developed to avoid faults ?
- What measures can be implemented to detect faults in microprocessor circuits before they manifest themselves as failures of the machine ?
- How should complex integrated circuits be designed ?

The requirements are intended to guide the designer during the development work.

3.4.1. Software

The software of the control system is the specification for most of the functionality of the machine. Both safety-related and non safety-related functions can be software controlled. Even if the specification used as input to the software development is fault-free, mistakes in the design process may introduce software faults. Software failures and other unexpected behaviour might lead to the creation of dangerous faults in the machine control system. A set of requirements for all technical activities associated with software development should be followed in order to avoid faults in software.

To obtain a software package of satisfactorily high quality, a number of activities, a certain organisation and a number of principles must be established. Software product requirements for design and development should cover :

- Interface with system architecture.
- Software that can be parametrized by the user.
- Pre-existent software.
- Software design.
- Development languages.
- Software coding.

3.4.1.1.Interface with system architecture

Interface with system architecture	Level	
	1	2
Software safety requirements as well as the determination of expected events should arise from safety analyses at system, functional and hardware level, etc.	/	O
The list of constraints imposed by hardware architecture on software should be defined and documented. Consequences of any hardware/software interaction on the safety of the machine or system being monitored should be identified and evaluated by the designer, and taken into account in the software design. Constraints such as : • protocols and formats, • input/output frequencies, • by rising and falling edge or by level, • input data using reverse logic etc. Listing these constraints allows them to be taken into account at the start of the development activity, and reduces the risk of incompatibilities between software and hardware when the former is installed in the target hardware. The consequences of any software errors should be studied at system level in particular.	O	O

Table 5 : Interface with system architecture

3.4.1.2.Software that can be parametrized by the user

The following requirements concern the development of software products that are designed to allow end-users to set their own parameters. The end-user may be either the person responsible for integrating the product into the system or else the user.

Such software can have different degrees of complexity : a table of messages or a system option. In keeping with the spirit of the present document, the definition of software parameters is limited and defined precisely in the specification documents. This excludes any modifications that might cause doubt about the exact version of the software, since this type of modification should always be undertaken by the software designer.

Certain systems can also include optional functions that are selected through the use of parameter-setting options in the software.

Software that can be parametrized by the user	Level	
	1	2
The parameters should be formally specified (type, relations, …) in the form of memory arrays. Moreover, the software and the parameters should be possible to modify without affecting each other.	R	R
Software specifications should define mechanisms that can be used to prevent the possibility of any parameters set by the user can affect the system safety. In so far as modifiable parameters are concerned, these mechanisms should provide protection against : - undefined or invalid initial values, - values falling outside functional limits, - data alteration. The definition of software parameters by users should be kept within the limits established by the system specifications approved by the analyst. In particular, test procedures should include different parameter values and different types of software behaviour. The designer should ensure that only those parameters which can be modified are accessible to the user.	R	O

Table 6 : Software that can be parametrized by the user

3.4.1.3.Pre-existent software

The term "pre-existent" software refers to the source modules that have not been developed specifically for the system at hand, and are integrated into the rest of the software. These include software products developed by the designer for previous projects, or commercially available software (e.g. modules for scientific calculations, sequencers, etc.). Such software components are called COTS (Commercial Off-The-Shelf) Software.

When dealing with this type of software, and especially in the case of commercial software products, the designer does not always have access to all the elements needed to satisfy the previous requirements (e.g. what tests have been carried out, is the design documentation available). Specific co-ordination with the analyst is therefore necessary at the earliest possible moment.

Pre-existent Software	Level	
	1	2
The designer should indicate the use of pre-existent software to the analyst, and it is the designer's responsibility to demonstrate that pre-existent software has the same level as the present requirements. Such a demonstration should be done : - either by using the same verification activities on the pre-existent software as on the rest of the software, - or through practical experience where the pre-existent software has functioned on a similar system in a comparable executable environment (e.g. it is necessary to evaluate the consequences of a change of the compiler or of a different software architecture format). The goal of indicating the use of pre-existent software is to open up consultations with the analyst as early as possible about any eventual difficulties that this type of software might cause. The integration of pre-existent source modules can be the cause of certain anomalies or unsafe behaviour if they were not developed with the same rigour as the rest of the software.	O	O
Pre-existent software should be identified using the same configuration management principles that were applied to the rest of the software. Perfect configuration control should be exercised over all the software components, regardless of their origin.	O	O

Table 7 : Pre-existent Software

3.4.1.4.Software design

Software Design	Level	
	1	2
Description of the software design should include at the very least : - a description of the software architecture that defines the structure decided on to satisfy specifications, - a description of inputs and outputs (e.g. in the form of an internal and external data dictionary), for all the modules making up the software architecture, - sequencers and interruptions, - global data, - a description of each software module (inputs/outputs, algorithm, design particularities, etc.), - libraries used**,** - pre-existent software used.	O	O
Software should be modular in order to facilitate its maintenance : • each module or group of modules should correspond, if possible, to a function in the specifications • interfaces between modules should be as simple as possible The general characteristic of correct software architecture can be summed up in the following way : a module should possess a high level of functional cohesion and a simple interface with its environment	O	O
Software should be designed to limit those parts associated with safety: - data/functional architecture: strict limitation of global variables, implementation of operators on state variables (visibility), - control of the layout of arrays in memory (risk of array overflows).	O	O

Table 8 : Software Design

3.4.1.5.Development languages

The goal of these requirements is to ensure that the designer uses a programming language and development tools that are well adapted to the software being developed, and that these tools do not lead to the introduction of errors in the executable code on the target machine. The following requirements can be applied to any language used (if more than one language is used simultaneously on the same system). Generally, the most widely used languages include :

- an assembly language, specific to the microprocessor or microcontroller employed,

- an advanced programming language such as C.

Development Languages	Level	
	1	2
The selected programming language should correspond to the characteristics of the application, and should be fully and clearly defined or at least limited by clearly defined characteristics. The characteristics of the application refer to such factors as size, type (industrial or scientific software, management, etc.), and any performance constraints. For example, COBOL does not satisfy the development requirements of a monitoring/control application on an industrial machine. Any deficiencies in the language can be avoided using appropriate coding rules.	R	O

Table 9 : *Development Languages*

3.4.1.6.Software coding

Coding	**Level**	
	1	**2**
The source code should: a) be readable, understandable, and subject to tests, b) satisfy design specifications of the software module, c) obey the coding manual instructions.	O	O
The coding rules applicable to a given software product should be outlined in detail in a coding manual and used to develop the software. The coding manual should : • indicate what programming principles should be applied and prohibit any uncertain language aspects • describe rules for source code presentation and documentation • include conventions used in naming components, subroutines, variables and constants. A set of rules are provided in Appendix A of this document. For example, indented presentation of different blocks of instructions, use of blank lines, contents of the source file header (author's name, input and output data, modified data, etc.). These conventions help to improve software legibility and maintenance.	R	O

Table 10 : *Coding*

3.4.2. Fault detection in microcomputer hardware

Programmable electronic systems (PES) have the ability to detect faults within themselves before a fault is manifested as a failure of the system. The techniques and measures used focus on different parts of the electronic hardware and may require different amount of system effort. It is regarded as state-of-the-art to implement techniques for fault detection in PES used in safety-critical applications.

3.4.2.1.Diagnostic coverage

It is easy to imagine some possible faults which can cause unexpected behaviour of the machine which is controlled. A bit in a memory cell may be stuck at ”0” or ”1”. The output circuits may be stuck at ”ON”. A software fault may cause a task to enter an ”eternal loop”. Perhaps interruptions in the supply power, or variations in the voltage level, may influence the execution of the software. Data transferred on serial communication lines may be distorted by interference. An internal CPU fault might cause incorrect execution. There are techniques and measures to detect such faults before the machine gets out of control.

When numerical values are needed in calculations, 60% is used for “low” coverage, 90% is used for “medium” coverage and 99% is used for “high” coverage.

The quantification of the diagnostic coverage for different methods of fault detection in memory and I/O units is based on values extracted from the IEC 61508 standard. Those values are the results of theoretical studies based on a simplified probabilistic approach. The lack of data concerning the various types of memory chips, and the assumption that the potential faults are equally distributed introduce a number of uncertainties.

Simple tests will not detect all hardware faults. Elaborate tests will detect many hardware faults at the cost of much processing effort spent. The diagnostic coverage, DC, is defined as the fractional decrease in the probability of dangerous hardware failure resulting from the operation of the automatic diagnostic tests. [IEC 61508-4, clause 3.8.6] See formula 1. If the test detects all faults, the coverage is 100%. If no faults are detectable, the coverage is 0%.

$$\text{diagnostic coverage DC} = \frac{\textit{the probability of detected dangerous failures}}{\textit{the probabilty of total dangerous failures}} \qquad [1]$$

Another definition of diagnostic coverage, $DC_{N,}$ (see formula 2) relates to the fraction of total number of different failures that is detected during a particular test. There can be large differences between the two ways to define the diagnostic coverage. The probability based approach distinguishes between faults which occur with different probability, while the number based approach does not.

A testing technique which detects faults occurring with high probability is very well likely to have a high DC, but may have a low DC_N if there is a large number of low probability faults which are not detected.

diagnostic coverage $DC_N = \frac{\textit{the number of dangerous failures detected}}{\textit{the total number of dangerous failures}}$ [2]

It may be hard to find numerical values for the probabilities of different faults. Sometimes the assumption is made, that all faults have the same probability. This is always an approximation of reality.

It is possible to make a numerical calculation of the coverage of some methods. While the coverage of some other methods may have to be expressed in qualitative ways such as ”high/medium/low”. An estimation of the diagnostic coverage will be needed to be able to compare two diagnostic test methods.

A translation from the qualitative definition “low/medium/high”, to a quantitative measure expressed as a percentage will be needed. This report has chosen to follow the definitions suggested by the IEC 61508 standard. (see Figure 14). High coverage is used for techniques and measures with a probability higher than 99% to detect a fault. Medium coverage means a probability less than 99%, but higher than 90%.

Low coverage will correspond to a diagnostic coverage greater than 60%, but lower than 90%. Techniques and measures offering less than 60% probability to detect faults are to be avoided in safety-related parts of control systems.

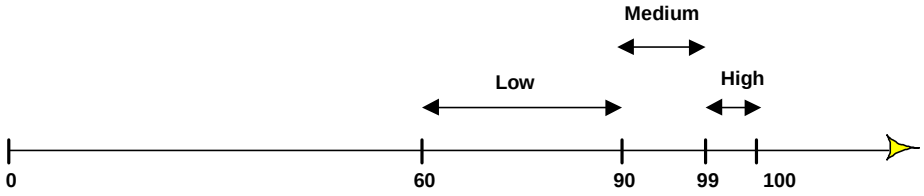


Figure 14 : Diagnostic coverage defined as low, medium and high

Similar uncertainties are introduced for other parts of the PES. The probability of different faults in the processing unit will depend on the type of processor, the manufacturer, the production process, the design etc. It is hardly possible to state a probability that will be valid in all cases. Assuming equal probabilities of all possible faults, the previous expressions [1] and [2] for the diagnostic coverage become equivalent.

Faults in the programme sequence will have different probabilities depending on the programming language, the experience of the programmer, the testing effort etc. It will not be easy to state a probability-based diagnostic coverage for the programme sequence monitor.

The most valid estimation of diagnostic coverage for a fault detecting method should at this stage be limited to one of the three levels referred earlier in this section: low, medium or high. The level chosen may be different if probability, or numbers of errors, is used for the definition of diagnostic coverage. However, the level will be the same if all faults are equally probable.

Diagnostic test interval

Methods for fault detection can be used at power-up to establish if the electronic system is fit to start operating. Faults should be detected at start, and operation must not be allowed to start when faults are detected. Such a power-up test will not detect faults, which occur during operation. Thus it will not be suitable for systems of high safety integrity with strict requirements for behaviour at fault.

Power-up testing is sometimes used in systems with moderated moderating safety requirements, and short operating time between power-ups.

Fault detection may also be performed at run-time to find faults occurring during operation. The will be of great importance to decide for which applications the system can be used. The test interval will be the maximum time during which an undetected fault may exist.

The is defined as the interval between on-line tests to detect faults in a safety-related system that have a specified diagnostic coverage [IEC61508-4, clause 3.8.7].

The application will decide the requirement for, which depends on the application of the safety system and its architecture. Normally within a given time interval, the occurrence of a component failure is much less probable than the execution of an online test. For a single channel system, the depends on the mean time between operation demands (see chapters 5.3 & 6.3 of Annex 6).

3.4.2.2.Methods to detect faults

Processing units

The processing unit of the PES may suffer from faults, which may cause malfunction. Examples of such faults could be bit errors in one of its internal registers, or malfunctioning of the instruction decoder. Self-testing is employed by designing software routines, which test the functionality of the processing unit. Certain operations are performed, and there will be only one correct result of such a test. The principle of letting a processor, which might be to check defect itself, relies on the assumption that a fault will corrupt the result of the self-check. The self-test has to be designed in a such way that the risk for a fault corrupting the test itself is negligible.

Invariable memory ranges

Semiconductor memories may fail to work as intended. A fault in the invariable memory will corrupt the source code and constants stored there. The instructions to the processing unit may be distorted, and important constants and parameters will be incorrect. This will result in an unpredictable behaviour of the machine control system. It is important to try to detect such faults before disturbing of the execution.

The methods for checking of invariable memory all rely on reading the memory cells and compare the read values to what originally stored there. This may be done by direct comparison with a duplicate area, or by calculating a checksum (or signature). The checksum will be compared then to the sum, which was originally calculated and stored in memory.

Memory tests can be quite time consuming. It may not be possible to cover the complete address range at one time. Run time checking is often performed by checking only a limited address range each time the memory test task is started. After a large number of calls to the memory test task, the complete memory will have been tested.

Variable memory ranges

Semiconductor memories may fail to work as intended. A fault in the variable memory will corrupt all variables stored in memory. This X will result in an unpredictable behaviour of the machine control system.

The methods for checking variable memory rely on different ways to stimulate the memory cells, and check that the function (works ?) correctly. Test patterns of different complexity can be written and then read back. A complex test pattern will have a higher diagnostic coverage than a simple test pattern.

Also testing of variable memory can be time consuming, as described above for invariable memory.

I/O Units and Interface
Interfaces to external units are present on many safety-related machine control systems. They can be analogue, digital, serial or parallel. The external communication through these interfaces can be of great importance for the safety. The status of the I/O units and interfaces may be critical, and should therefore be monitored.

Data paths
Even a physically small PES may consist of several internal units, which communicate. Examples of such data paths are electrical parallel buses, serial buses and optical fibres. The data paths between the internal units may fail, and should be checked.

Important signals, such as alarm signals, originating from one unit should be detected and processed by the appropriate receiving unit.

Power supply
All safety-related programmable electronic systems should have some kinds of circuitry to ensure that operation will not be started before. The behaviour of the processor and other electronic circuits is only specified for a specific voltage range.

Monitoring of the supply voltage level will also be important during run-time. Interruptions of the power supply will cause the electronic hardware to enter an undefined range where the exact behaviour cannot be foreseen. The monitoring circuit must give an alarm in time before than the voltage reaches a threshold value. The PES should have a ”graceful death” bringing the controlled machinery to a safe state.

The programmable electronic system should have then time to take proper action to enter a safe state. It may also be necessary to save machine status and calculated values in a non-volatile memory.

The output signal from the hardware circuit monitoring the power supply can be either static or dynamic. The output from a static monitor will indicate proper power supply with a constant high or low output signal. A change of the output will give an alarm to the processor. A monitor built on a dynamic principle will have a dynamic output signal. Power failure will be indicated by a constant signal. The most common way to implement a supply power monitoring is to use a standard commercial supply power monitor circuit. Such ICs are available from several semiconductor manufactures. A disadvantage with this type of circuit is that it is normally not testable, i.e. a fault in the circuit will not be noticed before the PES is shut down in an uncontrolled way.

Also over-voltage may create unwanted behaviour of the control system. Checking facilities can also be implemented to react on over-voltage before the specified operating voltage is exceeded, and the behaviour cannot be guaranteed. One example when over-voltage detection is needed may be a dual-channel system using a single power supply.

Programme sequence

The execution sequence of the software may be distorted by either software faults, hardware faults or environmental disturbances. This will with great probability lead to incorrect behaviour of the programmable electronic system. The consequences of such a fault are not possible to foresee since ”anything may happen”. However, there are well-established techniques to monitor the programme sequence.

The monitoring of programme sequence may be realised both in hardware and in software. A combination of a hardware unit (”watchdog”) with a logical monitoring realised in software will be the most powerful. But also less sophisticated techniques built on simple hardware or software solutions will certainly be able to detect some errors in the programme sequence.

There are several hardware solutions for watchdog functionality. Many microcontrollers offer a watchdog circuit on-chip. The watchdog is then programmed, activated and controlled through internal registers of the controller. There are also special circuits available to supervise microcontrollers. Such circuits often offer the facility of a watchdog. Another solution is to design a separate hardware circuitry based on a monostable flip-flop, which has to be retriggered at a specified interval.

The watchdog circuit is preferred to be hardware independent from the processor itself. The same error, which causes the fault in programme sequence, should not cause also the watchdog to stop functioning. There is an increased risk for this if the watchdog is integrated on the same chip as the processor. Special caution must be paid if the time base used by the processor and the watchdog is the same. A clock fault might then affect both the microcontroller and the watchdog.

A good watchdog should be tested or fail-safe. Systems with high requirements for functional safety may let the processor fake a watchdog alarm at every power-up, or at a periodic interval. The processor will then detect if the watchdog is non-operational. Otherwise there is a risk for a fault in the watchdog to pass unnoticed, until it really will be needed. Another possibility is to design a watchdog circuitry where single hardware faults will cause an alarm.

A detected fault in the program sequence will require different kinds of actions depending of the type of system. It is important that the correct action is taken. It will not be enough to design a watchdog into the system, and not specify properly which action shall be taken at fault. Most machines have a safe state, which shall be entered when a fault is detected. A watchdog circuit may force the processor and the outputs to the safe state where signals are inactive. Another possibility is that a watchdog alarm will cut the power to the outputs and leave the machine in safe state.) This safe state must not be possible to exit without a dedicated action of reset.

For applications with requirements for high availability, the watchdog alarm may be used to isolate one of the processors and let a redundant unit take over without affecting the normal operation of the system.

3.4.2.3.Requirements

The requirements listed in this report are combined to form adequate safety principles for a certain category (B, 1, 2, 3 or 4). Different safety principles are listed for the different architectures in category 3 and 4. A dual-channel system employing safety principles of medium diagnostic coverage may provide the same probability for failure of the safety function, as a single channel system using safety principles with high diagnostic coverage.

Some of the checking principles may be applied either at power-up, or continuously during run-time. This will be much depending on the application and no specific requirements are given for when (or how often) the checking must take place.

Processing unit	**Category**					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.1.a The CPU shall be checked for stuck-at failures of registers and internal RAM.			x	x	x	x
4.1.b The decoding and execution of instructions shall be checked.			x		x	
4.1.c All registers must be checked.			x		x	
4.1.d Faults in the processing unit shall be indicated by the PES.			x	x	x	x
Minimum diagnostic coverage	-	-	High	Medium	High	Medium

Table 11: Safety principles for monitoring of the processing unit

Invariable memory ranges	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.2.a The PES shall be able to detect faults in the invariable memory.		x	x	x	x	x
4.2.b The complete address range must be checked.			x		x	
4.2.c Memory failures shall be indicated by the PES.		x	x	x	x	x
Minimum diagnostic coverage	-	Low	High	Medium	High	Medium

Table 12: Safety principles for monitoring of invariable memory

Variable memory ranges	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.3.a The PES shall be able to detect faults in the variable memory.		x	x	x	x	x
4.3.b The complete address range must be covered.			x		x	x
4.3.c Memory failures shall be indicated by the PES.		x	x	x	x	x
Minimum diagnostic coverage	-	Low	High	Medium	High	Medium

Table 13: Safety principles for monitoring of variable memory

I/O Units and Interface	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.4.a The PES shall automatically check the input and output units (digital, analogue, serial or parallel).			x	x	x	x
4.4.b Faults detected in the internal communication shall be indicated.			x	x	x	x
Minimum diagnostic coverage	-	-	High	Medium	High	Medium

Table 14 : Safety principles for monitoring of I/O units and interface

Data paths	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.5.a The PES shall automatically check the internal communication.			x	x	x	x
4.5.b Faults detected in the internal communication shall be indicated.			x	x	x	x
Minimum diagnostic coverage	-	-	High	Medium	High	Medium

Table 15: Safety principles for monitoring of data paths

Power supply	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.6.a The PES shall be able to detect decreases in the supply voltage, and the execution of the processor must be halted in a controlled way.	x	x	x	x	x	x
4.6.b The supply voltage monitoring circuit must be dynamic, i.e. correct supply voltage is indicated by a dynamic signal.			x or 46c		x or 46c	
4.6.c The supply voltage monitor circuit must be fail-safe, i.e. a fault in the circuitry shall lead to a power fail alarm.			x or46b		x or46b	
4.6.d Power supply failures shall be indicated by the PES.			x	x	x	x
Minimum diagnostic coverage	Low	Low	High	Medium	High	Medium

Table 16: Safety principles for monitoring of supply power

Programme sequence	Category					
	B	**2**	**3 Single**	**3 Dual**	**4 Dual**	**4 Triple**
4.7.a The PES shall have a watchdog implemented in hardware to monitor the program sequence	x	x	x	x	x	x
4.7.b The hardware (especially the time base) used for the watchdog shall be independent of the processor it is supposed to supervise.			x		x	
4.7.c There shall be software means of monitoring the program sequence.			x	x	x	x
4.7.d The watchdog must be automatically tested by the software at power-up or a periodic intervals.			x or 47e		x or 47e	
4.7.e The watchdog must be fail-safe, i.e. a fault in the watchdog circuitry will lead to a watchdog alarm.			x or 47d		x or 47d	
4.7.f A watchdog alarm shall be indicated by the PES			x	x	x	x
Minimum diagnostic coverage	Low	Low	High	Medium	High	Medium

Table 17: Safety principles for monitoring of program execution

3.4.2.3.1.Complex Integrated Circuits

Application-Specific Integrated Circuits (ASICs) are used in electronic systems dedicated to the management of safety functions. These complex integrated circuits may incorporate several million transistors, and so cause problems for the evaluation of the functional safety of the systems that include them.

For discrete components (relays, transistors, resistors, capacitors, etc.) the analyst can evaluate the safety level by simulating virtually all the device's fault situations, using a practically exhaustive list of possible failures. For complex electronic circuits such as ASICs, this exhaustive approach is impossible. To evaluate the operational safety characteristics it is necessary to know the failure modes of the components used, and this is not possible for these circuits. The traditional methods of testing performance in the presence of faults are inadequate. It is therefore necessary to tackle the evaluation not only by updating subsequent tests on the finished products but also by extending the field of investigation from the origin of the faults considered: errors of specification, design or production, internal faults, or external effects.

The validation test scheme results in the sum of independent verification steps during the implementations process. The complete, uninterrupted sequence of verification steps provides the objective evidence („validation”) that the final result (e. g. the programmed FPGA) fulfils the initial requirements for the intended use and the required safety category.

3.4.2.3.2.Technology

The term ”complex component” may be applied to a wide variety of devices. The range spans different process technologies, different design and implementation methodologies as well as different levels of complexity. A number of typical products and design methodologies for integrated circuits are described below. The number of parties involved in the design and validation process varies as well as the responsibility for the work packages within the design flow. CICs may be divided into three main families: programmable circuits, prediffused arrays, and precharacterised arrays.

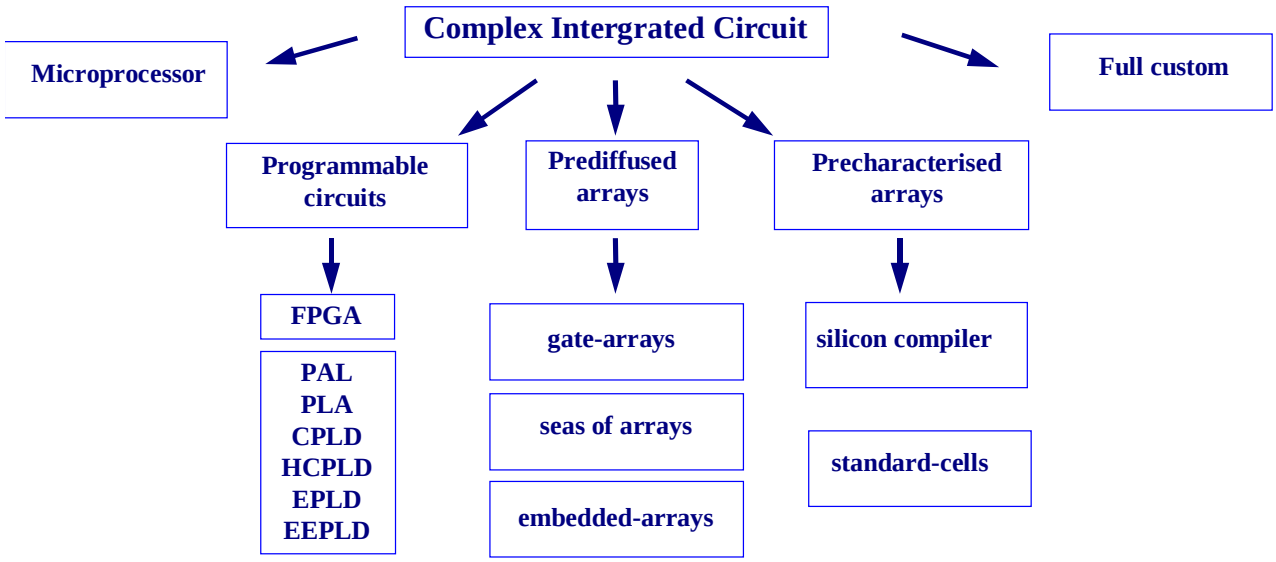


Figure 15: The families of CICs

Programmable circuits are components made up of matrices of gates, connecting tracks and complex cells such as registers, bistable devices, and so on. The user makes the interconnections between the cells according to his application purposes using a programming tool. The different arrangements of cells, the complexity available and the interconnection technologies used determine the different sub-families of programmable logic devices (PLDs) :

- Programmable Array Logic (PAL) circuits consisted solely of one programmable AND matrix and another fixed OR matrix.
- Programmable Logic Array (PLA) circuits consisted of programmable AND and OR matrices.

These circuits can very easily incorporate more complex cells, but are now obsolete.

- Complex Programmable Logic Device (CPLD) and High Capacity PLD (HCPLD) circuits are a development of PLDs containing a large number of very complex basic cells.

In all these PLD circuits, the cells are interconnected in arrays and the user removes the unwanted connection points by breaking the track. This programming is not reversible.

- The Erasable Programmable Logic Device (EPLD) is a PLD that can be programmed electrically and erased using UV light using the EPROM memory technique.

- The Electrically Erasable Programmable Logic Device (EEPLD) is a PLD that can be programmed and erased electrically using the EEPROM memory technique.

These two sub-families encompass erasable and reprogrammable PLDs, techniques that are very useful in prototyping.

- Field Programmable Gates Array (FPGA) circuits employ two interconnection techniques : the non-melt technique for which the user sets up connection points by breaking down a dielectric (an irreversible configuration) and SRAM for which the configuration of the connections, stored in a ROM memory, is automatically loaded into a solid-state RAM each time the circuit is switched on. The interconnections are made by MOS transistors turned on by commands from the RAM (reconfigurable). They include complex cells such as registers, multiplexers, etc., and represent strong competition for the pre-diffused family.

A prediffused circuit is an incomplete circuit. The deep layers of the component are made beforehand by the constructor. The user designs the interconnections of his circuit in tracks provided for this purpose using a CAD method. The circuit will then be finished by the constructor who creates these connections on a final layer of aluminium. This family is subdivided into three sub-groups :

- Gate Arrays are organised into rows of basic cells and interconnection tracks that are fixed in location and size.

- Seas of Gates or ”silicon seas” are circuits with a high density of transistors but no tracks. The interconnections are made on top of these transistors by a special metal layer, giving the user considerable flexibility for defining functions and connections.

- Embedded Arrays offer composite solutions that employ the best features of the various families : the complexity and optimisation of precharacterised circuits, the short development time of prediffused circuits, the high density of seas of gates, and so on.

With precharacterised arrays, the user has a software library of standard cells that are defined and characterised by the constructor. He chooses the cells necessary for producing the functions required and can design all the interconnection masks. This circuit is more optimised than a prediffused circuit.

The most evolved form of precharacterised circuit is the silicon compilation. This circuit is optimised as regards the parametrisable cells, RAM, ROM, multiplexers, connection of logic functions, and so on, using a description of the component in a high level language.

	PLD CPLD	**FPGA**	**gate array**	**cell based ASIC**	**core based ASIC**	**full custom ASIC**	**standard IC**
functional specification	C	C	C	C	C	C	V
implementation	C	C	D	D	D, M	D	V
place & route, layout	V	C	V	D	D, M	D	V
wafer production	V	V	V	V	V	V	V
packaging	V	V	V	V	V	V	V
test	V, C	V, C	V, D	V, D	V, D	V	V
responsibilities	V	IC / ASIC vendor (manufacturer)					
	C	end customer, system and application development					
	D	ASIC design centre					
	M	macro core (pre-designed functional blocks) vendor					

Table 18: Overview of Integrated Circuits

Standard IC are manufactured in large quantities and applied for different applications. Functionality, validation, production and production tests are solely in the hand of the semiconductor vendor. Manual manipulations and optimisations at layout level are frequently used to reduce required area. Not designed for safety-related systems, fault avoidance during the design process is only adequate for standard products. Frequent changes in production process, process technology and layout are likely for cost and yield optimisation. Number of components manufactured using a certain process or mask revision is not publicly known. Design and production of full custom ASIC is similar to standard IC, with functionality defined by end customer.

Core Based ASIC is based on pre-layouted or generated macro cores, connected by additional logic. Examples for pre-layouted macros are standard microprocessor cores, peripheral components, communication interfaces, analogue blocks, special function I/O cells. Examples for generated macros include embedded RAM, ROM, EEPROM or FLASH. Generated blocks are assumed to be „correct by construction”, based on design rules. Pre-layouted or generated macros are process specific but may be ported to different technologies. In most cases, the macro cores are not identical to the original discrete off-the-shelf components (different process, provided by a third party).

Cell Based ASIC are based on logic primitives (like AND, OR, Flip-Flop, Latch) taken from a cell library. The gate-level netlist containing the logic primitives and the interconnections is usually created from a high level hardware description language (VHDL, Verilog) using synthesis tools. The functional and timing characteristics of the logic primitives is characterised in the cell library;

these parameters are used to drive the synthesis tool and are also used for simulation. In addition, layout tools are used to place the cells and to route the interconnects.

Multi Chip Module (MCM) are multiple chips (dies) and passive components mounted on a common substrate and assembled into a single package. In most cases, package and outline is similar a standard IC. The chips (dies) use for MCM production are usually pre-validated, but not finally characterised. Thus, testing under environmental conditions needs to be done at MCM level. MCM is primarily a different packaging technology. Design methodology for the individual parts of the MCM is mostly identical to the design methodology for system build on conventional printed circuit boards. Therefore, MCM are not further discussed in this report.

Chip On Board. (COB) has a die which is bonded directly on the printed circuit board and hermetically sealed afterwards, instead of using chips (dies) in conventional packages. COB is primarily a different packaging technology, thus no further discussed in this report.

3.4.2.3.3.Faults in ASICs

For safety-related integrated circuits, the different product types require different validation concepts. For example, the layout and placement of the cells of a gate array or a FPGA is fixed; components based on these predefined structures are manufactured in larger numbers, thus the structure itself might be considered as „proven in use“ after some time. For the other product types listed in the previous paragraphs, the structure is define during the layout process. Thus, especially for deep sub-micron processes, interference between neighbouring cells or interconnections are possible, with actual influence on the chips functionality. It is obvious that this situation has to be considered during validation testing and fault injection.

Generally speaking an application developed using an ASIC will be more reliable than one with standard circuits. First of all, reliability is inversely proportional to the number of connections between units and, secondly, low power also means better reliability.

However, for these complex components :

- Knowledge of faults, and any cause/fault correlation, are no more than partial.
- Reducing the physical size of the basic components creates new faults.
- The growing complexity increases the probability that design faults will appear.
- The more limited spread of ASICs compared with standard circuits means that interpreting the feedback of experience is more difficult.

3.4.2.3.4.Phase model

It is useful to identify the major steps that lead to a production-ready component. This ”phase model” is intended to be more general than the two design flows given in chapter 3.4.3.3. Based on the phase model from the IEC 61508, the following phases are identified :

[1] Specification: Textual or formal description of the device’s functionality.

[1] Design Description: Formal description (e. g. Boolean Equations, Schematic, (V)HDL) that may be automatically translated into a fusemap / bitstream (PLD, FPGA) or gate level netlist (Gate Array, ASIC).

[2] Implementation: Transformation of the design description into a netlist / fusemap / bitstream that may be used to produce or program the component. This phase is subdivided into two phases: ”Implementation I” maps the design description into the primitives of the target device (logic blocks, gates), ”Implementation II” produces the final information required for the component production or programming (fusemap or bitstream file, layout database).

[3] Production: Production (programming) of the component, based on the output of the implementation phase.

[4] Post Production : The component is available for standard system integration and validation tests.

3.4.2.3.5.Design Flow

The description of the development work can also be made as a flow chart of activities. Simplified design flows for Application Specific Integrated Circuits (ASICs) and Programmable Logic Devices/Field Programmable Gate Arrays (PLD/FPGA) show the different methodologies, design steps and tools typically used for the development of complex components (for further details on safety issues see Annex WP 3.3).

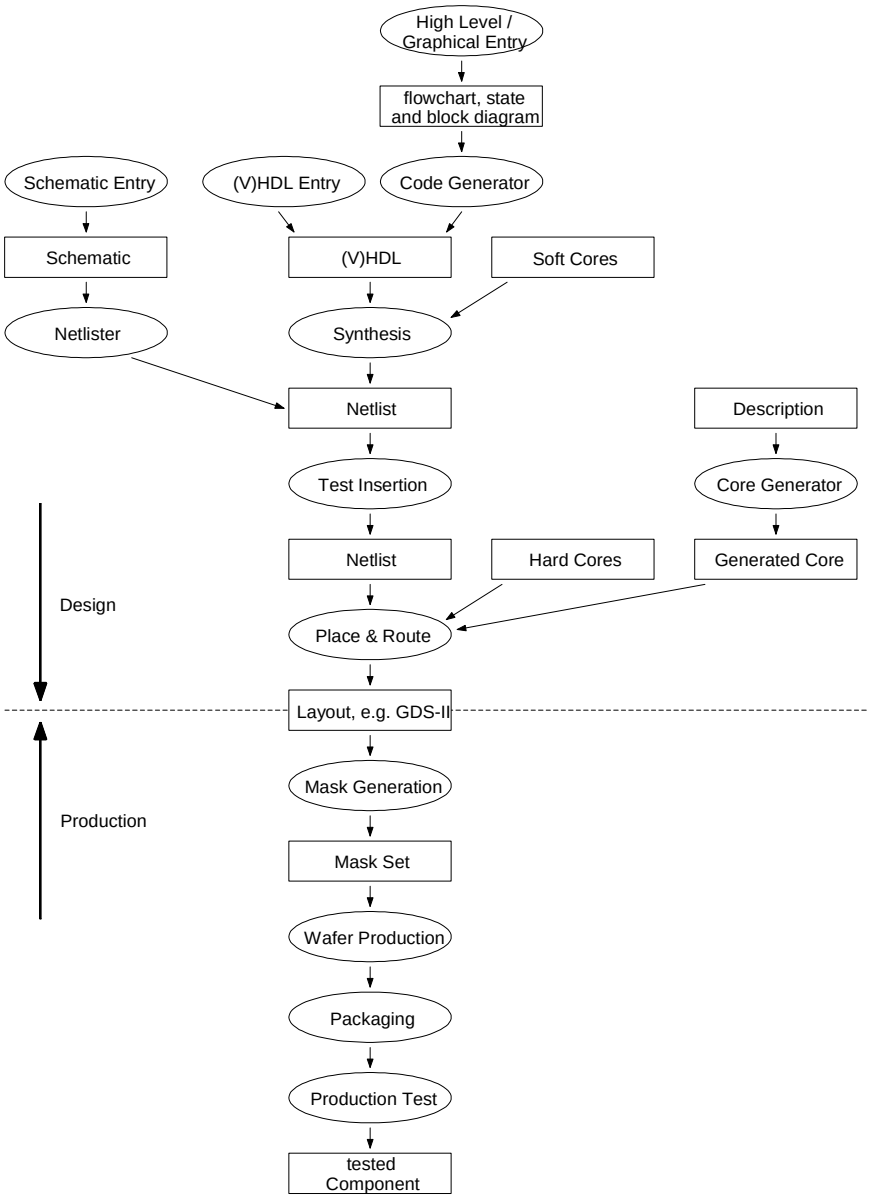


Figure 16 : ASIC Simplified Design Flow

It is possible to identify several potential safety hazards in all the activities during the design. The design of complex components for safety-related machine control systems will require attention to hazards such as :

- Vendor-dependent quality of the soft core or macro libraries. Correctness is not guaranteed.
- Faults during the synthesis process (caused by the synthesis tool).
- Only faults covered by the test pattern set of the production are revealed. Thus, high fault coverage is mandatory.

- For PLD type devices, timing is assumed to be ”correct by construction”, so the actual timing is not verified.
  - Only the successful programming may be checked by reading out the programmed pattern in the production. This does not guarantee correct behaviour of the device (same reasoning as for volatile devices).

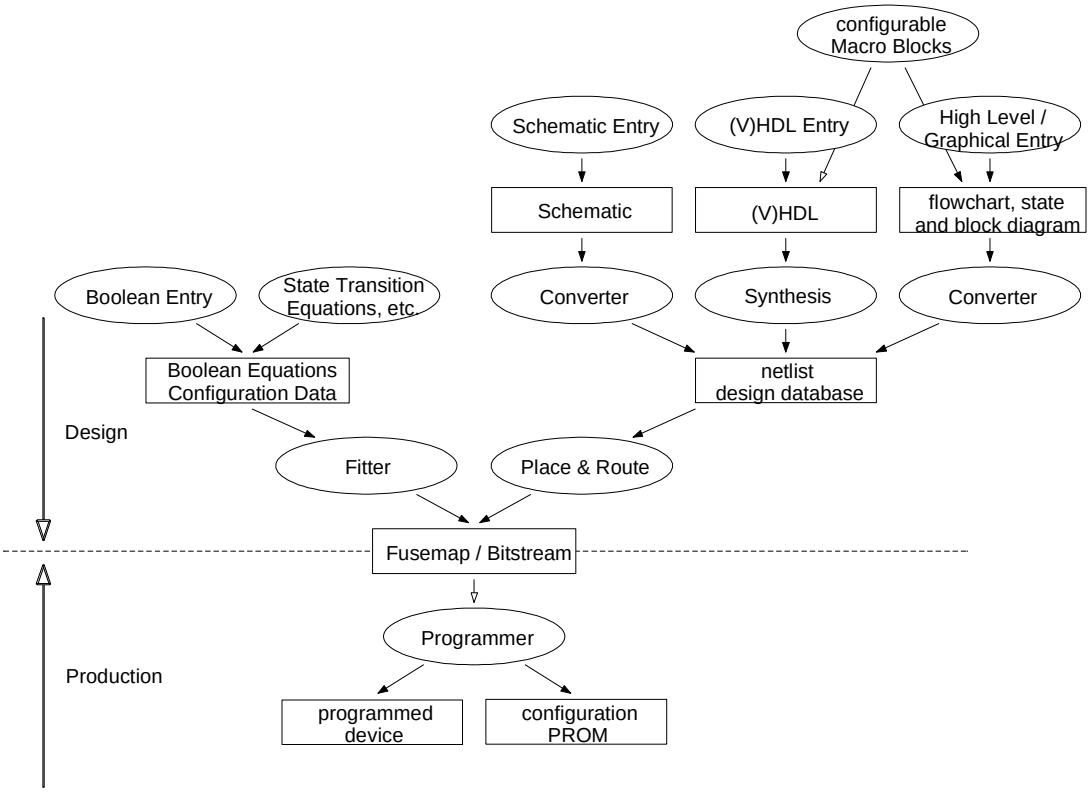


Figure 17 : PLD/FPGA Simplified Design Flow

3.4.2.3.6.Field experience

The definitions of IEC 61508 (part 2) for class A and B components implies that „field experience should be based on at least 100.000 hours operating time over a period of two years with 10 systems in different applications.“ Especially for complex standard components, it is not known to the end user whether the devices that are actually used on the circuit board are manufactured for the required period of time with the current mask revision and on the current process line. Even if the standard component is available for many years, modifications during that period of time are most likely, contradicting the requirements laid down in IEC 61508.

For complex application specific integrated circuits (ASICs), the terms „experience“ or „proven in use“ should be clarified and related to the different inputs for the design process :

- Process technology.
- Design rules for cell placement, interconnect and layout.
- Pre-layouted or generated macro cores.
- Cell libraries, including layout information and simulation models.
- Soft macros.
- Design tools: layout, synthesis, simulation.

In addition, rules for fault-avoidance during the design process and for the validation tests after migration to another process technology have to be worked out. Equal strategies must be used for both standard integrated circuits and application specific circuits, because, for example, it is not acceptable that changes in standard IC are silently accepted while the same modification of an application specific device requires additional validation tests.

3.5. Validation

3.5.1. Introduction

In general, validation process is made to confirm by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled [IEC 61508-4, clause 3.8.2]. When validation is related to safety related parts of a control system the purpose is to determine the level of conformity to their specification within the overall safety requirements specification of the machinery [pr EN 954-2 1999].

Carrying out a validation process can be a laborious task especially for complicated systems, which have got high safety demands. However, although the process can be laborious it is also necessary. Validation is often needed for the following purposes :

- to prove customers that the product is applicable for the intended purpose,
- to prove authorities that the product is safe and reliable enough for the intended purpose,
- to prove the manufacturer that the product is ready for the market,
- to prove the reasons for specific solutions,
- to have documentation to help future alterations of the product,
- to prove the quality of the product.

The validation process has been growing to meet the common needs as the technology has developed. Simple systems can be analysed (FMEA) and tested (fault injection) quite thoroughly. Systems with moderate complexity can be analysed quite thoroughly, but the tests cannot cover the whole system. Very complex systems cannot be analysed totally in details and also thorough tests are not possible. A number of different methods are needed in the process. Analyses are needed at least in the system level and the detailed component level, but also requirements related to different lifecycle phases have to be fulfilled. This means that attributes such as quality control, correct design methods and management become more important since most of the failures or errors are related to these kind of issues.

Confidence is a very important factor related to the validation process. The user of the validation documents has to trust the validation quality, otherwise the validation has no meaning. The validation activities are actually carried out to convince someone that the product is properly designed and manufactured. One way to increase the confidence is to perform the validation process according to existing requirements and guides, and to have objective experts involved in the validation process.

3.5.2. Validation process

Safety validation process consists of planning and actual validation. The same process can be applied also for subsystems. A checklist or other guide is needed in the process to include all necessary actions to the safety validation plan.

The phases of the validation process are presented in Figure 18. First, validation plan is made according to known validation principles. Then the system is analysed according to the validation plan and the known criteria and design considerations. Testing is carried out according to the validation plan and the results of the analysis. All the phases have to be recorded in order to have a reliable proof of the validation process and the documents to help future modifications.

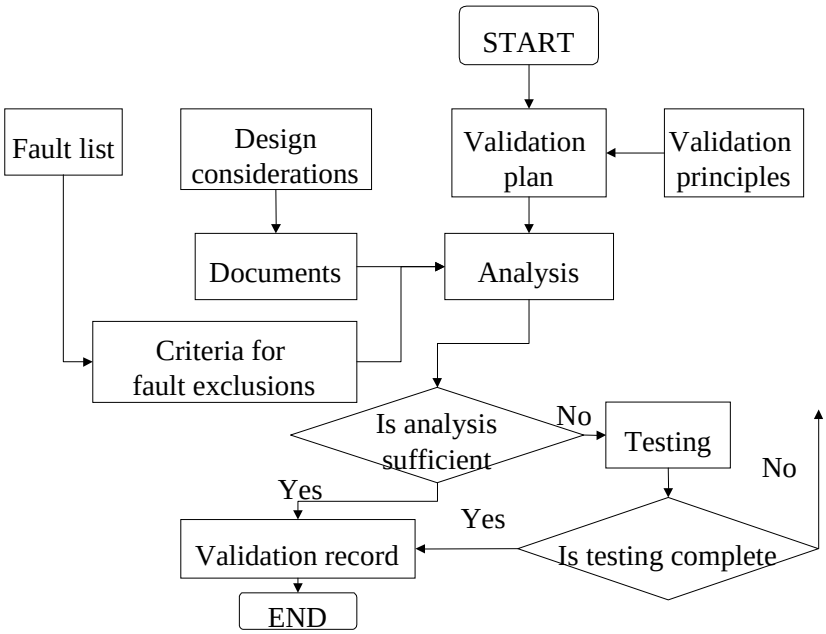


Figure 18 : Overview of the validation process [prEN 954-2 1999]

Validation planning

The purpose of safety validation planning is to plan out that the safety requirements (e.g. standards EN 954 or IEC 61508) are tested or analysed. Safety validation planning is performed to facilitate and to enhance the quality of safety validation. The planning states the organisation and, in chronological order, the tests and verification activities needed in the validation process. A checklist is needed in the planning process in order to include all essential analyses and the tests into the safety validation plan. Such checklist can be gathered from IEC 61508-1, prEN 954-2 or Nordtest Method. Large control systems may include separate subsystems, which are convenient to validate separately.

The main input for safety validation planning are the safety requirements. Each requirement shall be tested in the validation process and the passing criteria shall be declared in the plan. It is also important to declare the person(s) that makes the decisions if something unexpected happens, or who has the competence to do the validation. As a result, safety validation planning gives a guideline how to perform safety validation.

Validation

The purpose of safety validation is to inspect that all safety related parts of the system meet the specification for safety requirements. Safety validation is carried out according to the safety validation plan. As a result of the safety validation, it is possible to see that the safety related system meets the safety requirements since all the safety requirements are validated. When discrepancies occur between expected and actual results it has to be decided whether to issue a request to change the system or the specifications and possible applications. Also, it has to be decided whether to continue and make the needed changes later or to make changes immediately and start the validation process in an earlier phase.

3.5.2.1.Validation by analysis

Different analysing techniques are needed in different phases of the design. At first, hazard identification and risk analysis techniques are useful. Such techniques are e.g. “hazard and operability study (HAZOP)”, “preliminary hazard analysis (PHA)”, and techniques, which use hazard lists. There are many techniques for software verification and for probabilistic approach to determine safety integrity. In software verification the software errors are searched systematically by using e.g. data flow analysis, control flow analysis, software FMEA, or sneak circuit analysis (see IEC 61508-7). In probabilistic approach, it is expected that the verification process is already carried out and statistical values are used to calculate a probabilistic value for executing the program correctly. There are also methods for verifying component, such as ASIC, designs. This chapter, however, is concentrating on analysis techniques, which are used in analysing control systems.

There are two basic type of techniques for analysing systems :

- Top-down methods (deductive), which begin with defined system level top event(s) and the initiating factors are concluded.
- Bottom-up methods, which begin with single failures and the system level consequences are concluded.

Both analysing techniques have their advantages and disadvantages, but after all the value of the results depend on the analyst. The techniques can, however, make the analyst more observant to detect certain type of failures or events. Bottom-up methods tend to help the analyst to detect all single failures and events, since all basic events are considered. Top-down methods tend to help the analyst to detect how combined effects or failures can cause a certain top event. Top-down methods are good if only the critical events have to be analysed. Bottom-up methods are good if the whole system has to be analysed systematically. The basic demand is that the analysing technique has to be chosen so that all critical events are to be detected with the minimum duty. Top-down methods give an overview of the system, show the critical parts, systematic failures and human factors. Bottom-up methods consider the system systematically and a lot of failures are found.

Combined bottom-up and top-down approach is often likely to be an efficient technique. The top-down analysis provides the global picture and can focus the analysis to areas that are most significant from the overall performance point of view. Bottom-up methods can then be focused on the most critical parts. Bottom-up analysis aims at finding ”the devil that hides in the details”.

The most important point after choosing the analysing method is to concentrate on the weak points of the method. This can be done by using strict discipline.

FMEA

When the safety and performance of a control system is assessed, the failure mode and effect analysis (FMEA) is the most common tool used. There is an international standard (IEC 812. 1985) which defines the method. FMEA is a bottom-up (inductive) method which begins with single failures, and then the causes and the consequences of the failure are considered. In the FMEA, all components, elements or subsystems of the system under control are listed. FMEA can be done on different levels and in different phases of the design which affects the depth of the analysis. In an early phase of the design, a detailed analysis cannot be done. Also some parts of the system can be considered so clear and harmless that deep analysis is not seen necessary. However, in the critical parts, the analysis needs to be deep and it should be made on component level. If safety of the system depends very much on a certain complex component the analysis may include even some inner parts of the component. This can mean e.g. software analysis or consideration of typical failures related to a certain logical function.

In prEN 954-2 there are useful lists for FMEA on failures of common components in different types of control systems. The standard gives probable component failures and the analyst decides if the failures are valid in the system considered or if there are other possible failures. If functional blocks, hybrid circuits or integrated circuits are analysed the list in prEN 954-2 is not enough. Also systematic failures and failures typical to the technology (microprocessors, memories, communication circuits, etc.) have to be considered since those failures are more common than basic random hardware failures.

FMEA is intended mainly for single random failures and so it has some weak points :

- Does not support detection of common cause failures and design failures (systematic failures).
- Human errors are usually left out; the method concentrates on components and not the process events. Series of actions causing a certain hazard are difficult to detect.
- Sequential failures causing a hazard can also be difficult to detect, since the basic idea of the method is to consider one failure at a time. If the analysis is made with strict discipline it is possible to detect also sequential failures. If a failure is not detected by the control system other failures (or events) are studied assuming the undetected failure has happened.
- Systems with a lot of redundancy can be difficult to consider since sequential failures can be important.
- The method treats failures equally, and so even failures with very low probability are considered carefully. This may increase the work and cause a lot of paper.
- In a large analysis documentation it can be difficult to identify the critical failures. It can be difficult to see which failures have to be considered first and which means are the best to take care of the critical failures.

However, FMEA is probably the best method to detect random hardware failures, since it considers all components (individually or as blocks) systematically. Some critical parts can be analysed on detailed level and some on system level. If the method seems to become too laborious, the analysis can be done on higher level, which may increase the risk that some failure effects are not detected.

The FMEA table includes always the failure modes of each component and the effects of each failure mode. Since the analysis is carried out to improve the system or to show that the system is safe or reliable enough, some remarks and future actions are also always needed in the table. Severity ranking is needed to ease the comparison between failure modes and therefore it helps to rank the improvement actions. When the analysis includes criticality ranking it is called failure mode, effects, and criticality analysis (FMECA). The criticality and probability factor can be a general category, like impossible, improbable, occasional, probable, frequent, or e.g. exact failure probability values can be used. In many cases exact values are not available, they are difficult to get, or they are difficult to estimate. The circumstances affect very much the probability of a failure.

FTA

Fault tree analysis (FTA) is a deductive technique that focuses on one particular accident or top event at a time and provides a method for determining causes of that accident. The purpose of the method is to identify combinations of component failures and human errors that can result in the top event. The fault tree is expressed as a graphic model that displays combinations of component failures and other events that can result in the top event. FTA can begin once the top events of interest have been determined. This may mean preceding use of preliminary hazards analysis (PHA) or some other analysis method.

The advantages of FTA are typically :

- It can reveal single point failures, common cause failures, and multiple failure sequences leading to a common consequence.
- It can reveal when safe state becomes unsafe.
- The method is well-known and standardised.
- The method is suitable for analysing the entire system including hardware, software, any other technologies, events and human actions.
- The method provides a clear linkage between qualitative analysis and the probabilistic assessment.
- It shows clearly the reasons for a hazardous event.

The disadvantages of FTA are typically :

- It may be difficult to identify all hazards, failures and events of a large system.
- The method is not practical on systems with large number of safety critical failures.
- It is difficult to introduce timing information into fault trees.
- The method can become large and complex.
- The method gives a static view to system behaviour.
- The method typically assumes independence of events, although dependencies can be present; this affects the probability calculations. The dependencies also increase the work.
- It is difficult to invent new hazards that the participants of the analysis do not already know.
- Different analysts typically end up with different representations, which can be difficult to compare.

Quite often probability calculations are included in the FTA. FTA can be performed with special computer programs, which give easily proper documentation. There are also programs, which can switch the method. You only feed the analysis once, and the program shows information in the form of FTA or FMEA.

Analysing strategy

The traditional way to analyse an electronic control system is to apply a bottom-up approach by using failure mode and effect analysis. The method is effective and it reveals random failures well. The method is good for systems, which can be analysed thoroughly. Systems are, however, getting more complex and so the top-down approach is getting more and more applicable. A top-down approach like fault tree analysis helps to understand the system better and systematic failures can also be better revealed. The top-down approach reveals well also other failures than random failures, which are better revealed by the bottom-up approach.

Another development due to increasing system complexity has been analysis on module by module basis rather than on component by component basis. Non-programmable electronic systems with moderate complexity can and should be analysed on component by component basis and, in some cases (large systems), also on module by module basis to cover complicated module/system level errors. To analyse complex programmable systems at the component by component basis by using bottom-up analysis (FMEA) would require a lot of resources and yet the method is not the best way to find certain failures. The system functions can be better understood at module or system level than at component level and so the quality of the analysis can be improved in that part.

The system analysis can be started from the bottom so that first each small subsystems are analysed and finally the system as a whole. In the so called V-model, the system is designed from the top to the bottom (finest details) and then validated from the bottom to the top. The analysis should, however, be made during the design process as soon as possible to minimise possible corrections. Thereby the system should be analysed by starting from the top at system/module level. Then detailed component level analysis can be made in modules which were found critical at module level analysis. This method reduces the resources needed in the analysis.

More and more often the bottom-up analysis tends to become too massive and laborious. Some tactics are needed to minimise the work and amount of documentation. One strategy can be to document only critical failures. Another tactics can be to start the analysis on the most questionable (likely to be critical) structure and then first document the items and effects. The failure modes and other information are added only to critical failures. The FMEA table may look then rather empty, but it saves work.

Complex modules and systems

Many complex components are at the present time too complex to be validated thoroughly (with reasonable resources) and programmable components are getting even more complex and specially tailored (e.g. ASIC, FPGA). This means that, for safety purposes, the systems including complex components have to cope with faults by being fault tolerant or by activating automatically safety functions. This can be achieved by concentrating on the architecture. Architecture can be best understood on system/module level and, therefore, also the analysis carried out on system or module level reveals best the architectural weaknesses. On complex systems there are nearly always also some design errors (hardware or software), which can be hard to find at component level. At module level the analysis can be made thoroughly. One factor supporting module level analysis is the quality of the analysis. An increasing number of components in a unit to be validated corresponds to a reduction in the efficiency of the analysis. Although module level analysis is becoming more and more important one cannot neglect the analysis at component level because certain failures can be better seen at the component level. A resource saving strategy is to concentrate on critical failures at all analysing levels. The category (according to EN 954) or the SIL (according to IEC 61508) affects how detailed the analysis should be performed.

Usually both system/module level and component level analyses are needed in validating complex systems. Analyses on system/module level are performed in order to determine the critical parts of the system, and component level analyses are carried out for those parts of the system.

For module level analysis there are some references, which give hints for failure modes of modules. For some standardised systems some advise for analysis can be found. For system/module level analysis failure modes resemble failures at component level, but the analyser has to consider the relevant failure modes.

Complex components

Complex components hold more than 1000 gates and/or more than 24 pins [EN 954-2 1999]. The definition gives just a rough estimate which component could be complex. The amount of possible different random failures in such a component is large. Only the combinations two out of 24 are 276. This is just the amount of simple short circuits in a small (according to the definition) complex component. Complex components have got several failure modes. If one analyses blindly all combinations the result would be a lot of irrelevant failures. Failure exclusions are needed in order to focus the resources on the critical failures.

the basic failures to be considered in the analysis can be simple compared to the actual failures that can happen inside the component. Such component specific failures can be e.g. a failure in the microprocessor register or a failure in a certain memory location.

In the draft IEC 61508-2 failures typical to certain component technology (e.g. CPU, memory, bus) are considered instead of the pins (input, output etc.) of the component. A single component can include several technologies.

3.5.2.2.Testing methods

Black box (BB) and white box (WB) strategies

BB and WB are two basic approaches used for modelling systems, which can be applied in any phase of the design process. They are characterised depending on the aspect the analyst, designer or tester focus his attention, that is, in the operation of the external functioning of the system, or in the details of internal operation.

In general, it is said that a **functional approach** (or BB) is used referring a system, when the system is considered as a whole, emphasizing its external perceived behaviour. That is, when the system is viewed as a **black-box** that interacts with other entities or systems through its inputs and outputs. It is the definition of a system from the user point of view.

On the other hand, under **structural approach** (or WB) a system is defined as a set of components bound together in order to interact; every component is in turn another system. The recursion stops when a system is considered as being atomic: any further internal structure can not be discerned, or is not of interest and can be ignored. In this case, the system is transparent to the person handling and this is why structural viewpoint is called **white box**, though it would be more appropriate to use the term glass box.

This way, for example, a Galpat test applied to a RAM memory may be seen like a structural test among others within an analysis of system architecture, or like a functional test from the RAM memory point of view. Although at first glance they seem two separate approaches, in practice the boundary between function and structure is fuzzy.

BB testing versus WB testing

In the testing field, functional and structural approaches are used to create models which describe specific aspects of the system. The aim of the test is to check whether those aspects fulfil the requirements or not, this means, to cover as much as possible the model. To do that the tester use different coverage criteria which represent actually selection criteria for test input data.

This way we find, for SW testing, structural models like the control flow graph (in addition with the source code listing) and coverage criterions as statement testing or decision and branch testing, for unit/component level testing mainly. And in the case of HW, structural models like a topological gate diagram and coverage criterions like single-paths sensitization, for component level testing.

In **functional testing** test data are derived solely from the specifications (i.e. without taking advantage of knowledge of the internal structure of the system).

If one wishes using this approach to find all the faults in a system, the criterion is exhaustive input testing (i.e. the use of every possible input condition as a test case).

Hence, to test a system, one would have to produce a huge number of test cases, and this without taking into account the additional test cases due to the special requirements of safety systems on the behaviour in case of fault (category).

This reflection shows that in general, exhaustive input testing is impossible in big systems. Two implications of this are that :

- The test of a complex system can not guarantee that it is fault-free ; and
- A fundamental consideration in testing is one of economics.

In **Structural testing** the tester derives test data from an examination of the system internal logic (often at the neglect of the specification).

Structural tests are inherently finites. Unlike the functional testing, it is assumed that structural testing can not make a complete test of the system. Moreover, a structural test in no way guarantees that a system matches its specification (due to misconceptions, missing parts, etc).

Then, since exhausted testing is out of the question, the objective should be to maximise the yield on the testing investment (i.e. maximise the number of faults found by a finite number of test cases). Doing so will involve, among other things, being able to peer inside the system and making certain reasonable assumptions about the system. This will form part of the test case design strategy.

For a specific selection criteria, the test input data may be generated according with two different procedures : **deterministic choice or probabilistic choice**. In the first case which defines the deterministic test, the test input data are determined by a selective choice whenever they satisfy the criteria considered. In the second case which defines the statistical test (or random test), the test input data are generated randomly according with a probabilistic distribution of the input domain in relation to the criteria considered.

Both data generation procedures may be combined with the two type of system models producing the following test approaches: functional deterministic or functional statistical tests and structural deterministic or structural statistical tests.

In general, statistical tests use a large number of input data and require that the probabilistic distribution to be representative of the input domain or use environment. On the other hand, the results have direct applicability and statistical confidence intervals can be developed for reliability parameters. Deterministic tests improve the efficiency concentrating in faults, and the main disadvantages are the effort required to establish the most probable faults and the conditions to expose them.

For safety related electronic control systems, to be applied in the machinery field, seems that a deterministic test provide enough testing, without being necessary to apply statistical testing.

A **test strategy** has to consider other essential aspects like, the life cycle phase, the level of abstraction and the test objectives (verification and evaluation), in addition to the type of system model (functional or structural), test data selection criteria and the procedure for generating the inputs.

The abstraction level refers to the different domains to represent a system (analytical, simulation and physical) and also to the depth within a domain. For instance, in validation phase the tests are usually applied at different levels of the prototype (system level, board level and component). In this phase, components usually are treated as black boxes.

Functional approach will be predominant during validation stage, where it is necessary to handle large objects and abstract from the details in order to verify high level functions (system requirements). Nevertheless, many times the limited number of functional tests carried out need to be complemented by a structural strategy concentrated above all on checking critical components (for instance, a comparison and synchronisation routines in a program of a redundant system, error detection mechanisms, …).

In general, the highest structural coverage values are assigned to the early design phases (e.g., unit or component testing) and requirements are reduced as we go up in the design process (e.g., in SW testing, 100% statement and decision coverage is mandatory for unit testing but this goal result unreachable when testing larger size items).

Some researches in the field of SW testing have given theoretical results on the relative structural techniques effectiveness considering only inclusion relations. However, in practice, **the relative effectiveness of a testing technique** will depend on the number of test cases necessary to satisfy its corresponding criterion (testing cost) and the probability that there exist a type of fault detected by this test criteria and not by others.

The inclusion relation is not valid to order all structural coverage criteria. And it does not serve to establish a relationship between structural and functional criteria.

Analysing these approaches from the involved persons point of view, in particular the designer and tester, it is find that designer when act as a tester are by nature biased toward structural considerations while independent tester due to his ignorance of structure (no preconceptions) are bias-free and can better deal with functional tests.

In conclusion, neither functional nor structural tests prove to be absolutely useful strategies: both have limitations and both target different faults. What is proposed is to combine elements of both BB and WB testing to derive a reasonable, but not air-tight, testing strategy. Reasonable in the sense of an acceptable relation between the number of test cases and sought guaranties (risk reduction requirements).

Fault injection testing

According to EN 954 standard, one of the test methods for validating the safety of control systems is fault injection.

In general, fault injection is a privileged technique for validating a relevant feature of safety systems: the behaviour in case of fault. Moreover, among the different fault injection techniques, there are a group of techniques that allow the injection of faults on a prototype (see fig. 5 in the Annex 9). These particular methods (grouped into two branches: physical fault injection and SW implemented fault injection methods) provide accurate data on the performances achieved by a real system close to the final (the whole system, HW + SW), still within the development process, i.e. before launching a system. Accurate, in the sense of dynamical behaviour of a system near the final, and without the precision problems of simulation methods.

On the other hand, the value of the test results will depend on how the methods are apply, that is what are the selected FARM attributes for the test.

Fault injection is an indispensable complement for other validation methods currently available (analytical techniques, functional and structure-based testing, symbolic tests). This complementarity, for instance can come to the point of establishing a close relationship with analytical models (e.g., Markov models), providing: assistance in the characterisation of the initial analytical model (a better understanding of failure processes, usually described by macroscopic models, assigning values to coverage parameters of failure control mechanisms, …) and validating or even refining the analytical model.

Validation can be understood as a process dealing with removing residual design faults and evaluating the performances achieved even beyond the specified hypothesis.

When fault injection is applied to remove the residual faults (systematic faults) in the system, tester uses principally a structural approach. Tester is concerned with checking whether every particular measure fulfil or not the detailed design specifications. To do that, he concentrates on designing a small number of test input data that explore the main features. This kind of testing will demand a rather deep knowledge of the system. It is convenient to use also stress workloads.

In contrast, when fault injection deals with estimating the influence of the occurrence, presence and consequences of faults in the system during its operational phase, the strategy turns more functional or black box. That is, tester is interested in applying a large number of faults so that they represent a significant part of all potential fault of the system, and achieving in that way a sufficient guarantee to state that the system fulfils its behaviour in case of fault requirements. In this case the system will be charged with real workload (a profile of the activity of the system-environment in operational phase).

For either systems validation or the validation of the application of components, the study carried out on the different injection techniques shows that currently the most popular ones for applying at the validation stage are the physical fault injection at pin level and the software implemented fault injection.

A fault injection test may be characterised by FARM attributes. The definition of these attributes for an injection campaign will depend mainly on the system, the safety requirements and the injection method used.

Set of faults (F). It characterises the faults by the type, location and duration.

Fault type : EN 954-2 and IEC 61508 standards both of them include fault lists for electronic components which indicate the type of faults to be taken into account for every kind of component. Additionally, the standard IEC 61508 relates the type of faults for each component to be detected by measures to control HW failures and the diagnostic coverage levels required from these measures. The above mentioned two methods allow inject physically or emulate the most of faults corresponding to low and medium diagnostic coverage (e.g., stuck-at and DC model) in the fault list of IEC 61508.

Location : It depends on the injection method. Physical fault injection at pin level allows inject faults in the pins of components, board connectors or even at external input/output level. While SW implemented fault injection corrupts memory cells and processor registers to emulate faults.

Duration : The timing feature of a fault may be classified in: permanent, transient (external faults) and intermittent (internal faults). Transient faults are considered as the more aggressive in perturbing the system operation, and also the more difficult to inject and control; specially in terms of reproducibility. The fault duration influence in the safety performances and so transient faults should be taken into account at least in the higher requirement levels (categories 3 and 4).

Set of activation input data (A). It characterises the input domain through two parameters: activation inputs (or activation trigger) and workload.

Activation trigger : There exist two possibilities depending on the injection time: a) pre-runtime, fault injection before the system start up, (i.e., the system will not be interfered any more when it is running); b) runtime, faults are injected during system operation.
When injecting during runtime, the trigger signal for fault activation may be controlled by time or spatial (state) parameters. Selection of these parameters will depend on the system features.

Workload : It refers to the application program characteristics. When validating a general purpose system, tester uses special workloads (application programs) and input data to exercise as much as possible the different parts or functions of the system in order to sensitise the injected faults. Obviously, in the case of an embedded system is not needed a workload, and tester only must select an appropriate set of input data from the input domain (use environment should be considered).

In the case of testing auto-test routines (e.g., for RAM memories), the set of input data is implicit in the routine.

Set of read output data (R) : It is derived from the tests and will include specific system outputs and intermediate states, selected to show the success or fail of a control measure in detecting the injected fault. In order to calibrate the system diagnostic parameters like coverage and latency times in error detection and/or recovering, it is required to obtain the number of errors detected, errors isolated and errors recovered, as well as the system intrinsic tolerance and the effectiveness of the injection campaign.

Set of measures (M) : It is obtained from the readings and it will be oriented to the validation objective, that is to fault removal or to fault forecasting. The typical parameters measured in a test campaign are the coverage factor and the latency time.

Coverage factor is referred to the coverage of the system to be validated in presence of faults, which depends strictly on the orientation given to the injection campaign.
The effect of a fault in a system depends on the injected fault as well as the system activity at the time the injection is effective. So the parameters F and A have to be considered. Coverage factor of fault detection mechanisms is formally defined as the space represented by the product of both sets of parameters.

Latency times : These times can refer to the detection as well as to the identification and isolation of the error and system recovery. The negative impact of a high latency in some of these actions makes necessary their estimation when working with critical times.
Another factor to take into account when evaluating the safety performances is the length of testing. In general it is not possible to inject all potential events from the domain formed by the sets F and A, and the coverage factor will be estimated using a subset of the domain. Thus, we will use the *Confidence Interval* to estimate if the sample is representative.

For complex systems, to make an equilibrated sample for the system can be unfeasible. In these cases it is used the Stratification, an statistical option that reduces the complexity of the sample and offers the possibility of being oriented to concrete goals in the validation.

3.5.3. Verification and validation of software

3.5.3.1.Presentation

3.5.3.1.1.Specific characteristics of safety software

Software is an intellectual creating including programs, procedures, rules and all associated documents, related to implementation of the programmed system. Software is materialised by specifications, a code (program) and documentation.

Software development is often difficult to control. Moreover, software is rarely a finished product; it evolves from one version to another, within very short periods of time. It is a paradoxical product, which may become obsolete, but is not subject to wear. On the contrary, it is best when used frequently. Finally, software development is essentially devoted to product design and testing and little emphasis is placed on series production.

One of the most important characteristics of software is that it is a product with countless inputs and which processes combinations far greater than the brain capacity. As a consequence, software behaviour cannot be fully apprehended by man. It is therefore separated into different modules. Nevertheless, it remains difficult to fully control the complexity of the product.

3.5.3.1.2.Evaluating safety software

The problem raised by software evaluation is to obtain justified confidence in the software behaviour. The software is often analysed according to the method used for its development. The evaluation is then based on a wide variety of criteria such as its structure, its development process, or the manner in which it was written, even though, in fact, only its behaviour should be evaluated. This is why it is rather difficult to distinguish between development methods and evaluation methods. These two types of methods increasingly overlap one another.

Finally, it is interesting to note that there are no specific methods for critical software. The methods used for critical software and those used for traditional software differ by the requirements of the standards. The major difference, in fact, resides in the budget and the time devoted.

Evaluating software may have highly varied significations. In general, two levels of evaluation are frequently distinguished : validation and verification.

3.5.3.1.3.Software verification and validation requirements

The requirements for software verification and validation include that the analyst should be able to carry out the evaluation of software conformity to the present requirements by conducting any audits or expertises deemed useful during the different software development phases. All technical aspects of software lifecycle processes are subject to evaluation by the analyst. The analyst must be allowed to consult all verification reports (tests, analyses, etc.) and all technical documents used during software development. Evaluation of software conformity to the present requirements is performed for a specific, referenced software version. Any modification of previously evaluated software which has received a final opinion from the analyst should be pointed out to the latter in order that any additional evaluation activities can be carried out to update this opinion.

3.5.3.2.Verification of software

3.5.3.2.1.Presentation

The purpose of verification activities is to demonstrate that software products stemming from a given phase of the development cycle conform to the specifications established during the previous phases and to any applicable standards or rules. They also serve as a means of detecting and accounting for any errors that might have been introduced during software development.

Software verification activities contain series of tests and analyses. In addition to tests, verification activities can rely on techniques such as reviews, inspections, checkings, cross-readings and code verification. In some cases reviews and analyses can replace some tests (e.g. in the event that a test cannot be carried out because it would cause the deterioration of a hardware component). Internal reviews permit the designer to ensure at key points in the development progress, that the product will reach its objectives.

It is important to note the rapidly increasing cost of correcting an error in relation to the phase at which it is discovered. The optimal cost of correcting an error corresponds to the earliest possible moment in the lifecycle. For example, an error discovered in the specification stage by means of verification of the coherence between the software specification and the system specification costs significantly less than if this same error is discovered at the end of the development cycle (through software or system validation). A discovery made late in the process requires that all development phases influenced by this error, and already completed, must be undertaken anew.

Software tests can be carried out at different phases of the lifecycle :

module tests at the level of each individual. These tests focus on software modules and their conformity with the detailed design. This activity is indispensable for large and complex software products, but is only recommended for the relatively small software products dealt with here. This conformity can also be demonstrated using static techniques (e.g. re-reading the code). This phase of the verification procedure allows detection of the following type of errors :

- inability of an algorithm to satisfy software specifications,
- incorrect loop operations,
- incorrect logical decisions,
- inability to compute valid combinations of input data correctly,
- incorrect responses to missed or altered input data,
- violation of array boundaries,
- incorrect calculation sequences,
- inadequate precision, accuracy or performance of an algorithm.

software integration tests. These tests (equivalent to the software design verification**)** focuse on the correct assembly of software modules and on the mutual relationships between software components. Software integration tests form the principal component of this verification. It can be used to reveal errors of the following kind :

- incorrect initialisation of variables and constants,
- errors in the transfer of parameters,
- any data alteration, especially global data,
- inadequate end to end numerical resolution,
- incorrect sequencing of events and operations.

validation tests. The purpose of these tests (equivalent to the software specification verification) is to detect errors associated with the software in the target system environment. Validation tests are the principal component of software specification verification. Errors detected by this type of verification include :

- any incorrect mechanism to treat interruptions,

- insufficient respect of running time requirements,
- incorrect response from the software operating in transient mode (start-up, input flow, switching in a degraded mode, etc.),
- conflicts of access to different resources or organisational problems in the memory,
- inability of integrated tests to detect faults,
- software/hardware interface errors,
- stack overflows.

3.5.3.2.2.Software verification requirements

The general verification requirements include the following items :

[1] The software verification strategy used at the different software development steps and the techniques and tools used for this verification should be described in a Test Plan before being used. This description should, as a minimum, include :

- identification of the software and its safety-related components that will be submitted to validation procedure before use,
- organisation of the verification activities (integration, validation, etc.) and any interfaces with other development activities,
- independence of the verification (if applicable): the verification strategy should be developed and implemented, and the test results should be evaluated independently (by an individual, department, or organisation) in relation to the size of the development team,
- verification methods and tools used (*types of tests, etc.),*
- environment of the verification (*test equipment, emulators, etc.),*
- manner in which test results were verified,
- a traceability matrix demonstrating the correspondance between the tests to be undertaken and the objectives of the tests defined.

[2] Verification of a new software version should include non-regression tests.

[3] Directives for drawing up test procedures should include a description of the input data to be used (value), a description of the expected output (value) and criteria on which test results will be judged acceptable (tolerance).

[4] The tests formalised in reports should be able to be carried out again (e.g., in the presence of the analyst).

A verification report should be produced for each verification activity, and should identify and document all distortions (non-conformities) with respect to the corresponding specifications,rules or standards (design, coding) and any quality assurance procedures that may exist. The **verification requirements** also include requirements for reviews and code verification. Internal reviews at key points in the development process allow the designer to ensure that the product will achieve the objectives set. An external specification review (with the analyst) should be held at the end of the software specification phase, and, respectively, an external validation at the end of the validation phase. The result of each review should be documented and archived. It should include a list of all actions decided on in the review process, and the review conclusion (decision on whether or not to move on to the next activity). The activities defined in the review should be monitored and treated.

Activities involving analysis and software specification verification should verify the exhaustiveness and adequacy of the software specifications with respect to the system specifications as well as the traceability with respect to the system specifications. Analysis activities and software design verification should verify the conformity to specifications.

Code verification corresponds to the first step in the verification of the actual code once it has been written. This is a "static" verification in so far as it is based on cross-readings, inspections, etc. It is only after this point that dynamic verification procedures (module tests, integration, validation) will make up the principal verification methods. The code verifications include source code and data verifications. Code verification (static analysis) should ensure that the code conforms to the software design documents and coding rules.

3.5.3.2.3.Software test requirements

The **software test requirements** include general verification requirements for software specification (validation tests), software design (software integration tests) and detailed design (module tests). The test objectives must be adapted to the safety integrity level of the software, to the type of software, and to the specific factors at work in adopting a given software product. These criteria determine the types of test to be undertaken (functional tests, limit tests, out of limit tests, performance tests, load tests, external equipment failure tests, configuration tests) as well as the range of objects to be covered by the tests (functional mode tests, safety function tests, tests of each element in the specification, etc.).

The **validation tests** should be carried out in conditions representative of the operational conditions of the system The test coverage of these tests should be made explicit in a traceability matrix and should demonstrate that each element of the specification, including safety mechanisms, is covered by a validation test, and that it is possible to verify the real-time behaviour of the software in any operational mode.

Validation results should be recorded in a validation report - available for each delivered software version - that should cover at least: the versions of software and system that were validated, a description of the validation tests performed (inputs, outputs, testing procedures), the tools and equipments used to validate or evaluate the results, the results showing whether each validation test was a success or failure, and a validation assessment : identified non-conformities, impact on safety, decision as to whether or not to accept the validation.

Integration tests should be able to verify the correct sequencing of the software execution, the exchange of data between modules, the respect of the performance criteria, and the non-alteration of global data. The test coverage should be given explicitly in a traceability matrix demonstrating the correspondence between the tests to be undertaken and the objectives of the tests defined. The integration test results should be recorded in a software integration test report, which should, as a minimum, contain the version of the integrated software, a description of the tests performed (inputs, outputs, procedures) and the integration tests results and their evaluation.

Module tests should verify, using input data, that the modules fulfil the functions specified at the detailed design stage. The test coverage should be given explicitly in a traceability matrix that demonstrates the correspondence between the tests to be undertaken and the objectives of the tests defined. Module test results should be recorded in a report that contains at least the version of the module tested, the input data used, the expected and observed results, and an evaluation of the results (positive or otherwise).

3.5.3.3.Validation of critical software

3.5.3.3.1.Methods of validation

Three types of specification language may be distinguished: specification in ordinary language, semi-formal specification and formal specification. Informal specifications written in ordinary language are generally incomplete, incoherent, ambiguous, contradictory and erroneous. As a consequence, it seems reasonable that they should not be used for safety software. Contrary to ordinary language, specifications implemented by **formal methods** are precise and the semantics of notations is clearly defined. If one is familiar with the representation used, formal methods are a good means of communication and documentation. In fact, formal methods are more than a tool for representation; they are also a technique for drafting specifications which restrains the designer to make abstractions and finally results in a better comprehension and modelisation of the specifications. It is sometimes even possible to make simulations. Use of a formal method requires considerable investments in time and training. However, formal methods are a significant step forward for the development and evaluation of critical software.

There are six level method for the evaluation process of critical software: The first step is to make certain that software specifications are in compliance with user needs. This verification is relatively difficult to make since the user often expresses his needs in an informal, incomplete, imprecise or yet incoherent manner. This activity therefore mainly rests on the experience and the know-how of experts in the field.

The second level of evaluation corresponds to moving from specifications to the final code ; the final code must be in compliance with the software specifications. This evaluation is in fact devoted to the software development process. Its success depends on the methods and tools issued from the software engineering.

The third level of evaluation, corresponding to moving from the final code to the software behaviour, consists in executing the final code to check the software behaviour. This level of evaluation is based on dynamic methods and is automatically controlled for the most part.

The fourth level of evaluation consists in making certain that the final code is in compliance with the user needs. For the same reasons as those expressed for the first level of evaluation, which are inherent to the nature of the user needs, this compliance is very difficult to demonstrate.

The fifth level of evaluation, corresponding to moving from specifications to software behaviour, consists in checking that the software behaviour is in compliance with what is described in the specifications. This activity was previously referred to as VERIFICATION. It is also mentioned in documentation as the answer to the question, “Have we built the software correctly”. To date, this evaluation may be done by tests for which the initial sets have been elaborated based on specifications.

Over a longer period of time and by the intermediary of more elaborate methods, this evaluation may be done by program synthesis techniques or specification simulation techniques.

Finally, the sixth level represents the total evaluation activity (see V-cycle in chapter 3.1.3.1).

3.5.3.3.2.Specification and validation procedure

The role played by specification for operating safety is to explicit “what to do ?”, resulting from refining the specifications after functional analysis and preliminary risk analysis. It forms the interface between the analyst and the software designer, specifying the safety restrictions, such as execution time, inputs and outputs, he behaviour desired in case of failure, etc.

Seven French companies were contacted in order to report on the methods and tools used in the software development process, notably for critical software. Very few companies use specific methods. Among the companies which systematically implement software engineering methods and tools are companies involved in the automobile and nuclear industries.

Therefore it was very difficult to establish a procedure based on industrial practices concerning critical software specification. The interviews obtained with specialists from different horizons, enabled us to synthesise the following procedure for specification.

The specification phase is often based on the experience of the person in charge of drafting the specification. It is necessary to :

- Provide a precise definition of the composition and role of the analysis and specification team.
  - have final users intervene early ?
  - plan project reviews and their contents ?
  - have the client express his needs as extensively as possible ?
- Facilitate “client”-developer” communication.
- For specification, in so far as possible, use an adequate method and possibly an adequate tool.
  - (CASE tools ensure the unity of the dictionary and make it easier to avoid systematic transcription faults),
  - a good drawing is worth 1000 words.
- Imagine being in the client’s position and adopt his logic and his manner of expressing himself.
- Use neutral vocabulary for both parties, client / specification person or team.
- Incite the client to enter into the developer logic, in order that he may formalise his need better.
  - the client will thus express the needs he considers “evident” and therefore not necessary to be stated.
- Begin by making a functional analysis and a specification of the overall system.
- Make as complete a description as possible of the environment-operator-application interactions.
- Define the role of the operator.
- Take into account the application ergonomics : screens, alarm control, diagnostic, interventions for maintenance, etc.
- Divide to rule better.

- wait for the right moment in the system description before dividing specification tasks among the members of a team,
- decrease the complexity by carefully chosen divisions,
- minimise the information exchange flows,
- do not decompose the system into more than three level, since complexity increases quickly and overall control may be lost,
- decompose critical functions into primitive functions. Impasses may thus be highlighted.

- In parallel, during specification, specify the manner in which to check objective expectations (acceptance files) and the means necessary to complete the verification.
- Take the referential into full consideration : standards, guides, technical documents, etc., before referring to them in the specifications.
  - select elements which may be realised and measured in relation with the size of the application, the structure and culture of the company which develops the software.
- Validate the specifications by an internal audit type action done by a person / team other than that concerned by specification and development.
- Include the final user of the application.

3.5.4. Validation of hardware

3.5.4.1. Validation of fault detection principles

Programmable electronic systems (PES) have the ability to detect faults within themselves before a fault is manifested as a failure of the system. The techniques and measures used focus on different parts of the electronic hardware and may require different amount of system effort. It is regarded as state-of-the-art to implement techniques for fault detection in PES used in safety-critical applications.

All safety critical systems should undertake basic measures to detect the faults, and possibly control the failures which might occur. The standard EN 954-1 specifies 5 categories for system behaviour at fault. Basic safety principles should be implemented for categories B, 1, 2, 3 and 4. For categories 1, 2, 3 and 4 also well-tried safety principles have to be implemented. The presence and performance of the safety principles must be validated.

Methods for fault detection

There are several aspects in a programmable electronic system which can be automatically self-checked. The following table gives examples of different techniques.

Component	Technique
Prosessing unit	self test of the execution of the instruction set. self test of registers by patterns or walking-bit [IEC61508-7, clause A.3.1, A.3.2] reciprocal comparison by software between two processing units [IEC61508-7, clause A.3.5]
Invariable memory ranges	checksum [IEC61508-7, clause A.4.2] 8-bit signature [IEC61508-7, clause A.4.3] 16-bit signature [IEC61508-7, clause A.4.4] replication [IEC61508-7, clause A.4.5]
Variable memory ranges	• RAM test ”checkerboard” or ”march” [IEC61508-7, clause A.5.1] • RAM test ”walkpath” [IEC61508-7, clause A.5.2] • RAM test ”galpat” [IEC61508-7, clause A.5.3]
I/O Units and Interface	multi-channel parallel output [IEC61508-7, clause A.6.3] monitored outputs [IEC61508-7, clause A.6.4] input comparison/voting [IEC61508-7, clause A.6.5]
Data paths	• inspection using test patterns [IEC61508-7, clause A.7.4] • transmission redundancy [IEC61508-7, clause A.7.5] • information redundancy [IEC61508-7, clause A.7.6]
Power supply	over-voltage protection with safety shut-off [IEC61508-7, clause A.8.1] monitoring of secondary voltages [IEC61508-7, clause A.8.2]. power-down with safety shut-off [IEC61508-7, clause A.8.3].
Program sequence	an on-chip watchdog with separate time base without time-window, e.g. Motorola microcontroller 68HC11 [IEC61508-7, clause A.9.1] a watchdog with separate time base without time-window, e.g. a Maxim microprocessor supervisor IC [IEC61508-7, clause A.9.1] logical monitoring of programme sequence implemented in software[IEC61508-7, clause A.9.3] combination of temporal and logical monitoring of programme sequence [IEC61508-7, clause A.9.4]

Table 19 : Methods for fault detection

3.5.4.2.HW validation tests

Nowadays, complex components, like microprocessors, memories (RAM, EPROM, Flash), programmable logic (PLD, FPGA), ASICs and other high integrated circuits may be used as building blocks for safety related electronics. Due to large scale integration, it is possible today to integrate a whole system – that required a board or a assembly of boards some years ago – onto a single chip.

Well known state of the art validation test may be applied on system (component) level. These „Safety validation tests for electronic systems“ are assigned to the safety categories (CAT 1-4) introduced in EN 954-1.

Technique/measure	Cat 2	Cat 3	Cat 4
Functional testing	HR high	HR high	HR high
Functional testing under environmental conditions	HR high	HR high	HR high
Interference immunity testing	HR high	HR high	HR high
Fault injection testing	HR high	HR high	HR high
Expanded functional testing	– low	HR low	HR high
Surge immunity testing	– low	– low	– medium
Black box testing	R low	R low	R medium
Statistical testing	– low	– low	R medium
“Worst case” testing	– low	– low	R medium

Table 20 : Safety validation tests for electronic systems

Notation :

qualitative rating for this method (first line)	HR	method is highly recommended for this safety category
	R	method is recommended for this safety category
	–	method is not required, but may be used
required test coverage of this method (second line)	high[4]	a high degree test coverage is required
	medium	a medium degree of test coverage is required
	low	a acceptable degree of test coverage is required

The detailed analysis of these existing methods reveals a number of potential limitations when confronted with the validation of a complex component :

- Complexity : the component might be far to complex for an adequate validation; it is not possible to reach the coverage figures for the given category.
- Controllability : interconnections and logic inside the component is not directly controllable.
- Observability : the reaction to input stimuli might not be observable; attaching probes is either not possible (internal signals) or affects the test results.

[4] “high” replaces the misleading “mandatory” used in tables in existing standards, e. g. in the standard IEC 61508.

Moreover, an additional drawback of the listed validation tests is the fact that they are applicable only very late in the development process, because a “real” hardware is required to run most of the tests. The system that is used during the validation test has to be as close as possible to the one that will be used in the field, otherwise the result of the validation test is not expressive at all.

For complex components, validation testing has to go “beyond the surface” of the component and is advised much earlier in the development process. For example, functional testing has to start at module level – using modules with very limited complexity – and has to accompany the hierarchical (bottom up) integration of the modules to more complex building blocks, step by step, until the complete functionality of a “complex component” is reached and all application and safety requirements are met.

Table 21 gives an example of the phases that may be identified as major steps in the design process of a complex component (PLD, FPGA, Gate Array or ASIC).

Phase	**Output (PLD/FPGA)**	**Output (Gate Array, ASIC)**	**level of detail**	**usability for verification (formal or simulation)**
Specification	Specification Documents (pure textual or semi-formal, e. g. using block and state diagrams, pseudo-code)		“high level” description with low level of detail	partial (only for those parts described semi-formal)
Design Description	Formal description of the functionality of the device, usable for automatic translation.		(virtual) components, blocks, processes RTL (“register transfer level”)	functional aspects (RTL level) no explicit information about timing behaviour
Implementation I	primitives netlist, (propriety) database	gate level netlist	FPGA primitives, ASIC gates; interconnections Gate Level	all functional aspects (Gate Level) estimated timing
Implementation II	Fusemap / bitstream	layout database (e. g. GDS-II)	physical placement and interconnection	all functional aspects (Gate Level) actual timing
Production	programmed device (or configuration PROM	packaged and tested device	Component	device characteristics (overall functionality, timing)
Post Production	Board / System		“black box”	black box testing only

Table 21 : Phase Model

The linkage between the phase model (Table 21) and safety validation testing (Table 20) is described in detail in the technical annex for WP 3.3. This includes a general overview and details about individual test sets for different technologies.

To help to decide what level of validation testing is required during the design and implementation process, the following classification that is based on “testability” is proposed :

- A component is of **low** test complexity if it is adequate to run the standard validation tests on the final component and to reach the required test coverage. For those “simple” components, no modification of the standard validation approach is required; nevertheless it might be advised to run some validation tests during the design process.

- A component is of **medium** test complexity if running the standard validation tests on the final component achieves a test coverage for at least one test that is one level less than required (e. g. “medium” coverage of functional testing instead of the required “high” coverage). For these components, running part of the validation tests during the design and implementation process is required, to improve test coverage. This is shown in Table 22.

- A component is of **high** test complexity if running the standard validation tests on the final component achieves a test coverage for at least one test that is two or more level less than required (e. g. “low” coverage of functional testing instead of the required “high” coverage). For this level of complexity, a detailed knowledge about the component and its intended use is required to find an adequate test strategy. Thus, no general recommendations are given.

Technique / measure	Cat 1,2		Cat 3		Cat 4	
	During Design Flow	Post Production	During Design Flow	Post Production	During Design Flow	Post Production
Functional testing	HR high / high	medium	HR high / high	medium	HR high / high	medium
Functional testing under environmental conditions	HR high / high	medium	HR high / high	medium	HR high / high	medium
Interference immunity testing	HR medium / –	medium	HR high / –	high	HR high / –	high
Fault injection testing	HR high / high	medium	HR high / high	medium	HR high / high	medium
Expanded functional testing	– low / low	low	HR low / low	low	HR high / high	medium
Surge immunity testing	– low / –	low	– low / –	low	– medium / medium	low
Black box testing	R low / –	low	R low / –	low	R medium / medium	low
Statistical testing	– low / low	low	– low / low	low	R medium / medium	low
“Worst case” testing	– low / low	low	– low / low	low	R medium / medium	low

Table 22 : Validation Tests for Components with Medium Test Complexity

Note : For the interpretation of this table see definitions below.

When the validation testing is moved to an earlier point in the design flow, the subsequent steps need to be more thorough verified, to ensure that the results of the validation are still valid for the final component. The technical annex (final report for WP 3.3) lists the verification steps that need to be carried out, starting at the validation test in the design process and ending at the final component.

The coverage for each step needs to be at least as high as the coverage for the validation test itself (Table 22).

4. APPLICABILITY OF EN 954 AND IEC 61508 TO THE MACHINERY SECTOR

4.1. Introduction

Work-package 4 (WP4) had as its objective a comparison of the methodologies and requirements of two standards, namely, IEC 61508 ‘Functional safety of electrical/electronic/programmable electronic safety-related systems’ and EN 954 ‘Safety of machinery - Safety related parts of control systems’. This study was performed in order to establish whether these two standards are likely to set the same or differing requirements when applied to machinery control systems.

Both standards propose a structured approach towards the design of safety-related control systems but differ in that EN 954 is designed to address all types of control system technologies whilst IEC 61508 has been primarily (but not exclusively) designed to apply to electrical/electronic/programmable electronic (E/E/PE) based control systems. The standards require that the safety-related functions of the control system are classified ; IEC 61508 requires that the control system be allocated a safety integrity level (SIL) whilst EN 954 uses a concept of safety performance and places the system into one of five categories. There is a significant difference in the way that SILs and categories are derived and defined. It is the problems that this difference causes that were the basis for the tasks performed in WP4, especially when the two classifications are compared with a view to developing a strategy to link them.

IEC 61508 uses a safety lifecycle approach to ensure that the design of an CES safety-related control system is systematically carried out. This lifecycle, as a technical framework, was examined to assess its suitability for the design of machinery control systems.

WP4 comprised three principal tasks that were focussed upon an examination of the IEC 61508 and EN 954 standards from the perspective of their practical implementation at a machines safety-related control system. This work included identifying the common requirements and differences, mapping schemes to link SILs and categories, and performing a machine validation exercise to consider the application (albeit retrospectively) of these standards to an existing machine in relation to specific hazardous events.

Additionally, Annex 3, Annex 4, Annex 5, Annex 6 and Annex 7 provided information on this subject.

4.2. Common requirements & differences between EN 954-1 and IEC 61508

The following are considered to be factors in the comparison of EN 954-1 and IEC 61508 using the safety lifecycle model as a technical framework.

General

- EN 954-1 does not take the hierarchical system oriented view that is a strong feature of IEC 61508.
- IEC 61508 refers to safety-related systems, which are seen as being wrapped around the "equipment under control" (EUC) to provide a "level of safety". EN 954-1 refers to "safety related parts of control systems".
- IEC 61508 requires the production of documentation at each phase of the Safety Lifecycle. The only specific documents required by EN 954-1 are the validation plan and validation report.
- IEC 61508 has a strong formal structure with clearly defined objectives and requirements specified for each phase of the safety lifecycle. EN 954-1 is much less structured and careful examination is necessary to extract the key requirements.

Scope

- EN 954-1 applies to safety related parts of control systems, regardless of the type of technology used. IEC 61508 is primarily concerned with CES systems.
- IEC 61508 addresses the entire lifecycle from the concept phase through to decommissioning. EN 954 is restricted to the design phase.
- IEC 61508 takes account of the entire system comprising EUC, safety-related system(s) and external risk reduction facilities. EN 954-1 is only concerned with the "safety related parts of control systems".

Competence of persons

- Addressed by IEC 61508, not by EN 954-1.

Safety management

- Addressed by IEC 61508, not by EN 954-1.

Concept

- Addressed by IEC 61508, not by EN 954-1.

Hazard & risk analysis

Both standards require :

- carry out a hazard and risk analysis ;
- consider elimination of hazards ;
- include fault conditions, reasonably foreseeable misuse and human factors ;
- identify events leading to hazards ;
- assess frequencies (or probabilities) of hazards events ;
- identify potential consequences ;
- assess risk associated with each hazardous event ; and
- identify the necessary risk reduction, for each hazard.

Differences

- IEC 61508 refers to “hazardous events of the EUC”. EN 954-1 refers to time/frequency of exposure to hazard.
- IEC 61508 allows quantitative or qualitative techniques. EN 954-1 emphasis is on qualitative/empirical techniques.
- IEC 61508 requires a “level of safety” (based on the tolerable risk) to be identified for each hazard. EN 954-1 simply refers to the “appropriate risk reduction”.
- IEC 61508 requires the information and results from the hazard and risk analysis to be documented. EN 954-1 has no documentation requirement.

Specification of safety functions

- IEC 61508 requires specification of all safety functions included in the “total combination of safety-related systems and external risk reduction facilities”. EN 954-1 only requires specification of the safety functions “to be provided in the control system”.
- IEC 61508 requires both a functional description and specification of SIL. EN 954-1 only requires a functional description.
- EN 954-1 lists common safety functions and associated characteristics applicable to machinery.

- IEC 61508 allows for safety functions to be allocated between the safety-related systems and external risk reduction facilities. EN 954-1 only addresses those safety functions implemented by the “safety-related parts”.

Derivation and specification of performance requirements for control systems

- IEC 61508 specifies a formal process whereby, for each hazard, the necessary risk reduction is derived from the EUC risk and the level of safety. It is then necessary to specify how the level of safety (and associated risk reduction) will be achieved. This is done by describing what the safety-related systems will do (i.e. the safety functions) and with what probability they will do it as required (i.e. the safety integrity). At this stage the safety-related systems can take the form of external facilities or control systems (of any technology). Then the individual safety-related systems should be specified, both in terms of functionality and effectiveness (as relating to a specific technology) so that all the safety functions are implemented with the required level of safety integrity (taking into account the total effect of all the designated safety-related systems). It should be noted that the level of effectiveness of the individual safety-related systems is also measured by the parameter “safety integrity”. IEC 61508 requires that the information and results of the safety requirements allocation process shall be documented.

- EN 954-1 requires that the measures for risk reduction by control means should be “decided” and specified in terms of functionality and category. The methodology to translate risk reduction (associated with particularly hazards) to performance requirements of safety related parts of control systems is not specified.

- IEC 61508 requires that the “effectiveness” of the safety-related control systems be classified according to “safety integrity”. Safety integrity is a quantified measure of the effectiveness of a safety-related control system and encompasses hardware reliability as well as control/avoidance of failures due to systematic faults.

- EN 954-1 requires that safety related parts of control systems be categorised according to resistance to faults. The performance measures associated with the categories are a description of measures taken to avoid or control failures and are not quantified.

- IEC 61508 requires that overall safety functions and safety integrity requirements are documented in an “Overall Safety Requirements Specification”. The corresponding requirements for individual CES safety-related systems are documented in the “CES Safety Requirements Specifications”.

Design

- Both standards require that the design meets the specified safety requirements, but IEC 61508 requires that the design documentation should identify and justify the techniques and measures chosen to achieve the SIL. With IEC 61508, extensive tables of recommended techniques and measures (for both hardware and software) are provided. EN 954-1 simply requires a "list of the design features which provide the design rationale for the category achieved".

- IEC 61508 recommends architectural constraints, EN 954-1 does not address architecture (other than as may be necessary to achieve the fault behaviour according to category).

Behaviour under fault conditions

- Both standards require consideration of behaviour under fault conditions. In IEC 61508, fault requirements depend on SIL, the extent of diagnostic coverage, knowledge of component failure modes, testability of components and knowledge of component reliability. In EN 954-1, fault requirements are dictated solely by choice of category.

Diagnostic coverage

- IEC 61508 makes recommendations regarding the level of diagnostic coverage provided by the techniques and measures used to control failures. EN 954-1 similarly accepts that not all faults may be detected. In category 3, the required measures for fault detection are required to be graded according to consequence and probability of failure and technology used. In category 4, the inability to detect certain faults leads to the requirement to show that an accumulation of faults does not lead to loss of the safety function.

Proof checking

- IEC 61508 requires that proof checks be undertaken so that the probability of failure on demand remains within the specified safety integrity level. Proof checking is not addressed by EN 954-1.

Integration

- Integration (software, hardware, modules, sensors, actuators) of CES systems is addressed by IEC 61508, not by EN 954-1.

Operation & maintenance

- Both standards require information for operation and maintenance.

Validation

- Both standards require validation to demonstrate that the safety functions have been implemented according to specification.

Modification

- Addressed by IEC 61508, not by EN954-1.

Verification

- Required by IEC 61508, not by EN 954-1.

Functional safety assessment

- Required by IEC 61508, not by EN 954-1.

Decommissioning

- Addressed by IEC 61508, not by EN 954-1.

4.3. Practical difficulties encountered during machine validation using the EN 954-1 & IEC 61508 standards

Task 2 of WP 4 was to examine the retrospective application of the EN 954-1 and IEC 61508 standards to existing machinery as part of a machine control system validation exercise.

The fundamental aim of this exercise was neither to assess, nor test, the machine, but to identify the differences between the approaches taken by the two standards. Therefore, the exercise was not carried out in unnecessary detail where this would not have been beneficial towards the aims of the STSARCES Project. For example, where EN 954-1 and IEC 61508 make normative reference to other standards, the requirements of each reference were only considered in the context of the validation exercise. Consequently, the validation methodology described in the WP4 Task 2 report should not be used as the basis for other assessments.

Useful hints arising are given in the following sub clauses.

4.3.1. Selection of the machine and safety-related control system to be validated

The requirements for the safety-related control system were :

- it has sufficient technical complexity in the configuration of its control system(s) to allow sufficient application of either standard ;
- it should include a programmable electronic system ;
- it is a practical application within an existing machine ;
- the manufacturer, or its designer, should be readily contactable, if necessary, to elucidate design criteria or details of its operation ; and
- the manufacturer should be willing to co-operate with the project and to provide the necessary technical material to allow validation to be effected.

It was decided that a suitable machine for this exercise would be a hydraulic press manufactured in the UK. Technical details of the machine under examination were as follows :

- Multi-axis direct numerical control (DNC) controller ;
- Hydraulic operation with individual servo control of the position of each end of the beam together with hydraulic pressure control ;
- Sizes from 30 to 3000 tonnes, specifically 100 tonnes on the machine examined ; and
- Photoelectric curtain allowing normal photoelectric guarding[5] or guarding in association with single- or double-break stroke initiation.

4.3.2. Hazardous events considered

A full examination of the control system of the machine would neither have been cost-effective or capable of yielding results additional to those obtained by a limited analysis. Therefore :

- to avoid repetition in the analysis, the operation of the machine was considered only in manual mode (i.e., neither single- nor double-break modes of initiation were considered.) ; and
- the most important hazards associated with the machine were determined in order to define the scope of the assessment.

[5] A photoelectric system is colloquially referred to as a photoelectric guard, despite the fact that it does not prevent access to the danger area, and sometimes as an intangible guard. A more accurate term is an Active Opto-electronic Protective Device (AOPD). However, as the term photoelectric guard is more commonly used and understood this term will be used throughout this document.

The hazardous events identified as being within the scope of the examination were :

[1] Aberrant stroke : An uninitiated stroke occurs, which cannot be prevented by obscuring the photoelectric guard (referred to as an unguarded stroke).

[2] Incorrect mute : The muting position aberrantly changes so that muting of the photoelectric guard occurs with the tool more than 6mm above the workpiece or the guard fails in a dangerous mode.

[3] Failure of the rear-gate interlock : If this interlock were to fail, access could be obtained to the rear of the working parts of the machine.

The validation exercise was carried out separately for each standard with the intention of minimizing the "cross-talk" between the respective examinations.

In order to make the exercise as realistic as possible, it was decided to adopt an approach which would, as nearly as could be envisaged, follow that expected to be taken by a machinery designer faced with the use of the standards in a working environment (i.e. not necessarily as the designers of the standards would have intended).

4.3.3. Matters arising from the application of EN 954-1

[1] The standard is intended to be applied during the design of a control system and not during a validation exercise. As a result, some of the steps in the methodology were inappropriate. To achieve adequate safety using EN954 advice on validation should be given.

[2] Where the standard does not give guidance for the validation other approaches have to be taken into account to follow the validation process from start to finish. There are a large number of minor requirements and 'give aways'. For example, the fundamental requirements of the various categories are simple to follow and relate to fault tolerance.

However, having established the requirements for Category 3, one finds that it is not necessary to detect ALL single faults but only SOME (see Table 2 ‘Summary of requirements for categories’ EN 954-1:1996). A subjective decision must be taken as to which faults need, or do not need, to be detected.

In addition to the standard the results of Clause 3.5 give useful advice.

[3] EN 954-1 has been designed as a standard with a practical means of assessment and implementation. Unfortunately, what appears at first sight to be a very practicable method (i.e., based on a simple analysis of fault tolerance) becomes very subjective when applied practically.

Annex B of EN 954-1 is the only way of determining the required Category for a system, other then by examining an existing system (which itself may not have been categorized correctly). Because of the subjective nature of Annex B, different assessors may come to different conclusions when determining the category as there is no absolute means of objectively determining the category required for any particular system.

More detailed advice could have been given to users of this annex. For example, research could have established the probability of the operator avoiding hazards in a variety of industrial applications and under varying conditions (e.g., approach speed) and the data tabulated in the standard.

To achieve a given risk reduction a closer consideration should be made of systematic failures/faults,of the $MTTF_d$, Diagnostic coverage and of Common Cause Failures. See clause 3.3.3 for further advice.

[4] The principles of EN 954-1 are based on single/multiple component failures leading to a hazard being realised. This, at first sight, seems to be a very simple way of defining the integrity of the safety functions. However, the examination of the control system indicated that there are many component failures which, in combination, could lead to the hazard. However, many of these failures are considered to be unlikely, highly unlikely or even incredible and could be very different when using different technologies. Because the decision to exclude such failures from the analysis can be a subjective task, it is recommended, where known, to consider failure rates from databases or field experience. This will help minimise subjectivity in validation.

[5] EN 954-1 gives no means of assessing or ensuring the integrity of software. Clauses 3.3.4, 3.3.5 and 3.5.3 give advice on the integrity requirements for software.

[6] To justify that the press has been designed using the principles of EN 954-1 and validated to its safety specification; a validation report (as described at Clause 8.5 of EN 954-1) and the technical construction file should be available to the assessor.

[7] EN 954-1 mentions maintenance but does so very weakly. In any safety-related protection system (which may be called to operate only infrequently), regular manual proof testing (in the absence of automatic diagnostics) is an important factor in maintaining the integrity, which will vary approximately linearly with the

frequency of the manual proof checks. In the machinery sector at the present such an approach is not often followed.

[8] EN 954-1 is a design standard so does not give advice on the manufacture of the system being designed. A well-designed system that is poorly manufactured could have a reduced integrity. For example, a multi-channel system, whose wiring has been segregated in order to avoid common-cause failures, could have the wiring strapped together as a single loom leading to a potential for common-cause failures. It was noted that the validation stage, i.e. type testing, couldn't account for variations between manufactured items resulting from, for example, a poorly specified manufacturing stage. It is essential that the quality system of the manufacture assures no deviations from the approved test sample.

[9] By assuming that subsystems are single components and applying the fault exclusion principle, it is possible to determine a Category without the need for complex calculation. However, the failure rate of a complex subsystem may be considerably higher than that of a single component. Therefore, the Category of a dual-channel subsystem cannot be considered equivalent to a dual-channel system at the component level, e.g., an interlock based on 2 relays cannot be compared with one based on two complex programmable logic controllers (PLCs), even if both interlocks achieve Category 3. Hence, two systems, each having the same Category, may be considered to be equivalent **only** if they use the same technology and a comparable number of components.

[10] A number of factors will considerably distort the hierarchy of Categories[6]. For example :

- the standard is based on system behaviour in the presence of faults. Modern technology allows the incorporation of sophisticated automatic diagnostics with a coverage approaching 100%. A single-channel system with sophisticated diagnostics may have a higher integrity than a crude multi-channel system. Although the standard allows faults to be excluded, it does not give advice on how this problem should be addressed.

- a highly reliable system, based on simple technology (e.g., a mechanical blocking bar) which because of its single-channel status would be assigned a Category of 1, may in practice have an integrity comparable, or even higher than, that of a Category 4 system employing a complex and, therefore, difficult to validate technology.

  The possibility of making a misleading assessment can be minimised by considering probabilistic aspects to estimate the real amount of risk reduction (see clause 3.3.1 – 3.3.3).

[6] Although the standard clearly states otherwise, it appears inconceivable that the hierarchy was not developed on the basis that a monotonic relationship exists between the integrity of the safety related parts and the Category.

4.3.4. Matters arising from the application of IEC 61508

[1] The first, and probably the most important, obstacle in using IEC 61508 involved the determination of what is an acceptable level of risk. This may require an iterative process in order to obtain an acceptable value, which will depend on a number of factors, such as :

- what may have been established as custom and accepted engineering practice in the industry concerned ;
- the cost effectiveness of improving safety beyond any particular level (e.g., the "law of diminishing returns") ; and
- what competitors and other organizations using similar types of equipment have deemed to be practicable.

Existing accident rates involving presses, (obtained from internal HSE sources), although not comprehensive, were used in this validation exercise to establish an acceptable level of risk. Such information will not be easy to obtain by designers and validators in the field of machinery and other methods may be more appropriate.

Additionally, it may be unwise in some circumstances to quote acceptable or tolerable rates of a particular level of injury. Therefore, it may prove necessary for target SILs to be determined by opaque means, possibly qualitative, for the various sectors of industry. The determination of target SILs is a critical and not necessarily easy task that would be helped considerably by the availability of a suitable, possibly industry-specific, methodology for dealing with it. It may be necessary, in the first allocation of an SIL, to consider the full bandwidth of possible SILs.

[2] IEC 61508 appears to have been conceived with the process industries in mind. As a result, the determination of SILs depends on the risk reduction provided by safety-related protection systems, which operate in parallel with the control system of the EUC and put the EUC into a safe state if a failure of the control system occurs. A safe state may therefore be the continuation of the process, while in the machinery sector the safe state commonly is the shut down of the hazardous movements.

Many machinery control systems have traditionally been based on relay technology, and since machines are mostly cyclic in their operation, it is possible to test most, if not all, of the individual components in the control system at every cycle of the machine and employ redundancy. This leads to a fault tolerance of 1, or more, with a short interval between tests and consequently control systems have a high integrity.

Therefore, in the case of many machinery safety functions, the concept of risk reduction, as used in IEC 61508, is inappropriate and a SIL must be calculated from the failure rate of the control system.

[3] Because IEC 61508 is new (not published in its final form at the time of this assessment), few, if any, manufacturers have used it. Thus, it was difficult for the manufacturer of the pressbrake under examination to deliver the documentation prepared to show compliance with IEC 61508. This was especially true with respect to the quality procedures used in the design of the machine. This documentation is necessary; otherwise it is not possible to determine whether the quality requirements have been satisfied in the design. It is important to carry out all assessments during all stages of the lifecycle (see clause 3.1).

It is recommended that the formal and detailed documentation for installation, commissioning, operation and maintenance is delivered by the manufacturer. Documents relating to design procedures, e.g., quality assurance, should be available to the assessor for the validation. This is necessary for all systems regardless of their origin (inside or outside of the EU).

[6] For a quantitative assessment, good failure-rate data is required. Data is available on the most frequently encountered modes of failure of most of the components, e.g. a relay failing to energise. However, in safety-related systems, many components are automatically tested to ensure that the frequently encountered modes of failure are revealed. Therefore, the remaining modes of failure, on which there is likely to be insufficient data (e.g., the failure of a single relay contact or a relay spontaneously changing from the de-energised to energised states as a result of, for example, a spring breaking), are the ones which cause difficulty. It may help to use the experience of applications in other sectors to achieve more representative estimates of these data.

[7] In order to determine the probability of injury if the relevant safety function were to fail a number of assumptions had to be made. For example, in the case of Hazardous Event 1, it was assumed that the operator places his/her hands in the press once per minute. A different assumption would change the target SIL and the validation. An ideal design would be that the operator's hands are NEVER placed in the press, but this is often not possible.

[8] Because the outcome of the quantitative analysis using IEC 61508 is likely to depend on a number of highly subjective assumptions, it will be possible to tailor the outcome of the analysis to suit one's particular needs. Some of these assumptions will be difficult to challenge. The use of the experience of applications in other sectors may improve these assumptions.

[9] Clause 7.4.4.3 of Part 2 of IEC 61508 requires that "Any failure-rate data used shall have a statistical confidence level of at least 70%". This level of confidence is unlikely to be realized in practice, for the reasons described in the previous paragraph. In practice, the use of the best available data is better than not carrying out a quantitative reliability assessment; if necessary, worst-case assumptions could be made.

[10] The proof-test intervals used in the validation exercise were based on the manufacturer's recommendations. This important information should always be available, perhaps by being directly labelled on the machinery.

[11] At first sight, the documentation requirement of IEC 61508 does appear to be burdensome. However, this need not be the case. What the standard is, in fact, requiring is that the development, etc., is broken down into discrete stages (i.e., the lifecycle), careful thought is given to each of these stages, and the results of this are put onto paper for use in later stages and for demonstrating the adequacy of the system. Looked at in this way, the IEC 61508 lifecycle is no different from any other well-organized process.

Clearly the documentation requirements will increase with the complexity of the system.

[12] The application of quantified risk analysis to machinery is more complex compared to its application to process control systems due to the synchronous interactions between the persons at risk, the control system and the cyclic nature of operation of the machine. In such situations, a calculation (e.g., of probability of failure on demand) involving steady-state conditions, as would be applicable to the control system of a process plant, is unlikely to be realistic. Instead, the timing of the automatic tests and periods of high risk in relation to the machine cycle must be considered in detail in the calculations.

[13] A complete understanding of the operation of the system is required for validation to be meaningful. This is true of an assessment being carried out using either EN 954-1 or IEC 61508 ; however, in the case of the latter, where a quantitative analysis is carried out, large variations in the calculated failure rate could result from minor mistakes in determining functionality.

[14] At first sight, the use of the architectural constraints on the hardware safety integrity appeared to have a number of failings, for example :

- the diagnostic coverage (fail-safe fraction) is used as a parameter to determine the SIL ceiling ; however, in the case of automatic diagnostics, the rate at which the diagnostics are carried out is ignored ;

- the diagnostic coverage may be irrelevant in calculating the architectural constraint. In reality, what may be most important, for example, is whether the PES output used by the function is monitored ;
- no account is taken of the fact that some single-channel systems may inherently be reliable and so perform as well as a multi-channel system ;
- the fail-safe fraction for a single component (e.g., such as a mechanical scotch) may be even more difficult to determine than the diagnostic coverage of a computer-based system ;
- all that the diagnostic coverage could lead to (assuming an appropriate repetition frequency) is an effective reduction in failure rate. Therefore, a system with a failure rate of l and no diagnostics is effectively no different to a system with a failure rate of 1ool and a diagnostic coverage of 99%; however, the former would be severely penalized by the architectural constraint ; and
- no account is taken of manual proof checking.

However, the architectural constraints should be viewed as a means of ensuring that the quantified analysis is not abused or used in error. For example, in the case of the calculations for this press :

- a number of assumptions have been made ;
- the calculations are inexorably linked to the architecture, self monitoring and cyclic operation of the press ; and
- the manual for the press indicates that a daily check should be carried out on the rear-gate interlock. The frequency of this check will have a considerable impact on the integrity of the interlock. If no checks were carried out in practice, the actual (as opposed to the calculated) integrity of the interlock would be considerably reduced.

The architectural constraints are intended to put a ceiling on the SIL that can be assigned to any particular system in order to prevent either inadvertent (or deliberate) misuse of the quantitative analysis. As a result, the architectural constraints will ensure that the integrity level cannot be inflated significantly beyond the actual level achievable for any particular system. This will prevent inflated SILs being claimed and, as a result, ensure that an appropriate level of safety is maintained.

[17] The use of Tables 2 and 3 of Part 2 of IEC 61508 used to combine the architectural constraints of several subsystems, require clarification as to their use.

4.4. Conclusions from machine safety-related control system validation exercise

[1] Today machinery safety systems are not developed from scratch using a life-cycle approach. Instead, as a new machine is developed, the experience gained from previous machines is modified slightly in order to make improvements to the overall design. Hence, safety requirements are unlikely to be developed for any particular machine. Instead, the safety systems of new machines will be designed to be no worse than those of existing machines. The use of IEC 61508 will require a radical change to the machinery design/development process in that safety must be addressed using an absolute, rather than relative, approach.

[2] IEC 61508 uses quantitative calculation of the overall failure rate as well as qualitative techniques, where insufficient information is available for a quantitative determination (for systematic failures), for determining safety integrity. EN 954-1 attempts to avoid the need for a quantitative calculation by using a simple methodology - the risk graph. Unfortunately, the application of the methodology is not straightforward in other than the simplest of systems, and requires a subjective application of engineering knowledge.

[3] IEC 61508 covers all stages of the lifecycle of a system. EN 954-1 considers only the design (and validation of the design).

[4] The greatest problem in using a quantitative approach to risk assessment, as described in IEC 61508, is the availability of suitable data. Two types of data are required :

- Failure rate data for the components and subsystems: It may be necessary to use data from generic components, or for outdated components ; however, data can be obtained (or estimated) for most components, although it is likely that some assumptions may be necessary.

- Levels of acceptable risk: The level of acceptable risk is a societal parameter and is difficult to determine, being dependent on perceived, rather than actual, risk. The guidance in IEC 61508 uses the ALARP value but gives no help in determining what that value should be. The author made an assumption that existing hazard rates were acceptable but this assumption need not be valid in all cases. The author considers that this problem may present the most difficulty in using IEC 61508 until industry-specific guidance documents, based on IEC 61508, provide guidance in this area. However, the publication of such guidance could give alarm to those at risk.

[5] A number of assumptions had to be made in order to carry out the quantitative analysis described in IEC 61508. These were subjective had a significant effect on the SILs. There may be a high dependence on basic (and possibly subjective) assumptions in the quantitative analyses of many other systems. Some of these assumptions will be difficult to challenge and could lead to failure-rate predictions being distorted to meet the needs of other agendas.

[6] If a methodology, that will enable target SILs to be determined without significant subjectivity is not available, the uncertainty in the outcome of the quantitative analysis used in IEC 61508 may be large. In the author's opinion, the production of such a methodology should be given a very high priority otherwise it will not be possible to fully exploit the guidance in provided by IEC 61508.

[7] Generally, existing safety-related electrical control systems at machinery have not been designed using the guidance contained in IEC 61508 (of which all parts were not published at the time of writing of this report) and, as a consequence, suitable documentation, required in order to verify the various safety lifecycle stages, is not likely to be available. Documentation, in a form suitable for assessment purposes, will become available only when IEC 61508 gains credibility in machinery manufacture. Until this time, it will be difficult to carry out assessments of safety-related electrical control systems at machinery, especially in relation to the quantitative analysis.

[8] IEC 61508 enforces the manufacturer of a SRCPES to plan the overall lifecycle in a structured way by requiring an adequate documentation. At first sight, the documentation requirements for a simple machinery-control system appear to be excessive.

[9] Because shortage/incompatibility of documentation may prevent an adequate determination of the qualitative measures when a retrospective examination is carried out on a machine designed prior to the publication of IEC 61508, it will not be possible to determine whether (or not) suitable measures have been put in place to deal with systematic failures. Therefore, a retrospective quantitative assessment using IEC 61508, may prove to be inaccurate as the actual failure rate may be dominated by systematic failures, which are unlikely to be predictable quantitatively. Unfortunately, this will lead to an underestimate of the failure rate, i.e., the estimate will indicate that a system will be safer than it actually is.

[10] IEC 61508 takes a scientific approach by matching system integrity to risk. Also, wherever possible, it uses quantification, but acknowledges that qualitative measures may be followed where quantitative measures cannot be used. However, the qualitative measures have been determined (using engineering judgement) to be appropriate to the SIL. This should be compared with the approach taken by EN 954-1, which is based on fault tolerance.

[11] The principles of IEC 61508 require that a methodology is followed which encompasses all of the phases in the lifecycle of a system, e.g., concept, design, implementation, etc. If the methodology has not been used by the manufacturer, subsequent assessment using IEC 61508 will inevitably be difficult because of missing information. However, if IEC 61508 had been followed from the outset, the relevant information would have been available, facilitating validation.

4.5. Techniques & measures for machine validation

The findings and conclusions outlined above from the research undertaken in Annex 3, Annex 6, Annex 7, Annex 11, Annex 12 and Annex 13 indicate how to solve some of the practical difficulties that may be encountered when using the EN 954 and IEC 61508 standards. These problems, derived from the divergences that exist between EN 954 and IEC 61508, require that in order to establish a sound basis for the validation of safety-related control systems at machinery consideration should be given to :

- A linear mapping of the safety integrity levels of IEC 61508 to the categories of EN954-1 could not be established. This was primarily due to the category definitions in EN954-1 not placing any quantifiable requirements regarding the rate of failure of the safety functions. If the work outlined in clause 3.3.3 is further developed and standardised it may be possible to create some non linear mapping.

  However, it can be stated that, in a given technology, category 1 is likely to have a higher safety integrity level than category B and category 4 will have the highest safety integrity level.

- The qualitative approach of EN 954-1 is a desirable one from the machinery sector point of view and could be usefully developed and linked to IEC 61508.

- The principles of IEC 61508 (safety lifecycle and safety integrity levels) can be applied to E/E/PE control systems in machinery. IEC 61508 could complement EN 954-1 for E/E/PE systems but a qualitative approach leading to a safety integrity level would have to be developed.

- The non-hierarchical structure of EN 954-1’s categories is often misinterpreted into a hierarchical one. This is because the category definitions have to be carefully analysed to understand their full meaning. An informative annex interpreting the categories for different technologies may be useful.

- Although the categories are difficult to relate to risk, EN 954-1, as a document, does provides much useful information into the design strategies for safety and the requirements for safety functions.

- IEC 61508 covers all phases of equipment's life from concept through to decommissioning. In the machinery sector, very rarely would one party have responsibility across the entire lifecycle. It is considered that there is a need to delineate responsibilities. This is particularly so in the case of manufacturers who are producing machines or safety components for use in a variety of applications where it may not be practical for the manufacturer to undertake a complete hazard and risk analysis and identify suitable safety functions for all applications at an early stage in the safety lifecycle. In such cases the emphasis must be on the manufacturers to supply sufficient and suitable information (including the SIL) so users can take proper account of the equipment's performance characteristics in the final application.

- The developer and validator should have a deeper consideration of the systematic aspect of the machinery control system (see clause 3.1, 3.2 and 3.4).

5. USER'S GUIDE

5.1. Validation methodology for SRCES

To deal with the validation of SRCES, functional steps should be processed in the recommanded order of operations as follows :

- Obtain the allocation of safety requirements. Update the safety planning as appropriate during SRCES development.
- Determine the requirements for SRCES, including the safety integrity requirements, for each safety function. Allocate requirements to software.
- Start the phase of planning for SRCES validation.
- Access the architecture (configuration) for the SRCs logic system, sensors and final elements.
- Review with the software supplier/developer the hardware and software architecture and the safety implications of the trade-offs between the hardware and software. Iterate if required. The methodology for software validation is split into six main levels :
  - The first step is to make certain that software specifications are in compliance with user needs and safety requirements. Ensure that all necessary information for functions are complete, precise, explicite, coherent and correct.
  - The second level of evaluation corresponds to moving from specifications via the software design to the final code ; the final code must be in compliance with the software specifications.
  - The third level of evaluation, corresponding to moving from the final code to the software behaviour, consists in executing the final code to check the software behaviour.
  - The fourth level of evaluation consists in making certain that the final code is in compliance with the user needs. For the same reasons as those expressed for the first level of evaluation, which are inherent to the nature of the user needs, this compliance is very difficult to demonstrate.
  - The fifth level of evaluation, corresponding to moving from specifications to software behaviour, consists in checking that the software behaviour is in compliance with what is described in the specifications. This activity was previously referred to as verification.
  - Finally, the sixth level represents the total validation activity.

- Develop a model for the hardware architecture dedicated to safety-related systems. Develop this model by examining each safety function separately and determine the subsystem (component) that will be used to carry out this function. Deterministic and probabilistic analysis are required.

- For deterministic approach, two methods are generally employed to predict common mode faults :

  - Fault Tree Analysis (FTA), this deductive method starts out from a dangerous system failure, determined for instance by risk analysis, and looks for combinations of events that could lead to this failure. It reveals random, systematic and common mode faults. Ultimately, all the logic branches of a FTA must be developed through to the basic events. In practice, the tree is developed to be capable of analysing the effect of input, processing and output failures.

  - Failure Mode and Effect Analysis (FMEA), This is an inductive method that starts out from failures of the functions or components of the system to be analysed in order to determine the dangerous failures that could affect it. It highlights failures due to single failure modes that affect the software or the hardware.

- Use FMEA and Markov models for probabilistic approach. These techniques have been chosen because of their considerable capability of handling many of the technical features usually implemented in modern safety devices. Especially with Markov modelling, periodic events like online tests can be modelled quite comfortably.

- Establish the system parameters for each components used in the complex electronic safety-related systems. For each of the components, determine the following :

  - the mean time to restoration ;

  - the diagnostic coverage ; and

  - the probability of failure.

- Create a reliability model for each of the safety functions that the SRCES is required to carry out.

- Implement the design of the SRCES. Select measures and techniques to control systematic hardware failures, failures caused by environmental influences and operational failures.

- Integrate the verified software onto the target hardware and, in parallel, develop the procedures that users and maintenance staff will need to follow when operating the system.

- Together with the software developer, validate the SRCES. The purpose of safety validation is to check that all safety-related parts of the system meet the specification for safety requirements. Safety validation is carried out according to the safety validation plan. As a result of the safety validation, it is possible to conclude that the safety related system meets the safety requirements since all the safety requirements are validated. When discrepancies occur between expected and actual results it has to be decided whether to issue a request to change the system, or the specifications and the corresponding possible fiels of applications. Also, it has to be decided whether to continue and make the needed changes later, or to make changes immediately and restart the validation process in an earlier phase.

Finally, SRCES should also comply with generic safety requirements :

- Electrical safety.
- Electromagnetic compatibility : susceptibility and radiation is required by the European EMC directive.
- Environmental compatibility.
- Climatic and mechanical stress.
- Quality management in production, test field and revision handling. This is particularly important for software based systems.

5.2. What we cannot answer

From the research undertaken in this project, some limitations, sometimes inherent to the nature of the faced problems, need to be summarized :

- A fixed mapping of the safety integrity levels of IEC 61508 to the categories of EN954-1 could not be established. This is primarily due to the category definitions in EN954-1 not placing any quantifiable requirements regarding the rate of failure of the safety functions.

- The non-hierarchical structure of EN 954-1's categories is often misinterpreted into a hierarchical one. This is because the category definitions have to be carefully analysed to understand their full meaning. An informative annex interpreting the categories for different technologies may be useful.

- IEC 61508 covers all phases of equipment's life from concept through to decommissioning. In the machinery sector, very rarely would one party have responsibility across the entire lifecycle. It is considered that there is a need to delineate responsibilities. This is particularly so in the case of manufacturers who are producing machines or safety components for use in a variety of applications where it may not be practical for the manufacturer to undertake a complete hazard and risk analysis and identify suitable safety functions for all applications at an early stage in the safety lifecycle. In such cases the emphasis must be on the manufacturers to supply sufficient and suitable information (including the SIL) so users can take proper account of the equipment's performance characteristics in the final application.

- There is a large difference between software development practices and theoretical works which treat the subject. It is difficult to elaborate an operation safety methodology for small and average sized companies. Organisational restriction, and more particularly the lack of operation safety culture, greatly complicate the elaboration of an operation safety process. There are no specific software safety specification tools for small size applications.

- Complex components are indeed so complex that it is difficult to analyse them thoroughly, and it is very difficult to predict the failure modes of the components. Also, the inner programs related to programmable components may contain critical errors. All these reasons cause some uncertainty related to the analysis of the complex components. A single complex component alone cannot control a safety function safely enough, and some redundancy, diversity and/or monitoring is needed. This means that the architecture of the control system is of major importance and it can make the risks caused by complex components to become negligible.

- The increasing complexity of new systems (integrated function, processing speed, components and assembling technology miniaturisation, etc) is complicating largely the execution of tests late in design (e.g., in validation). Structural tests will not get to go in depth and will be applied more and more to external layers, what supposes a process of functionalization or change to more functional testing. These obstacles, are forcing to apply design techniques that make easier further testing of circuits and programs (so called, testability techniques), and direct testing from a physical to a simulation domain.
- Physical fault injection at pin level and Software implemented fault injection techniques has shown to be the most interesting techniques for fault injection into prototypes of programmable electronic systems. Each technique allows to introduce a subset of all potential faults in a system, so tester will have to choose the technique and plan the test properly depending the specific set of faults to emulate. Software implemented fault injection arises as a better option for emulating transient and precise internal faults, unlike Physical fault injection at pin level that allows to emulate, in an easier way, stuck at faults at data, address and control lines. However, it is difficult to say in what extent one technique cover the other because fault emulation capability of these techniques depend largely on the nature of the system and its load.

- Generally, existing safety-related electronic control systems for machinery have not been designed using the guidance contained in IEC 61508 and, as a consequence, suitable documentation, required in order to verify the various safety lifecycle stages, is not likely to be available. Documentation, in a form suitable for assessment purposes, will become available only when IEC 61508 derived standards like draft IEC 62061 gains credibility in machinery manufacturers. Until this time, it could be difficult to carry out assessments of safety-related electronic control systems at machinery, especially in relation to the quantitative analysis. This report intends to encourage manufacturers to fill this gap in the near future and to issue the step-by-step needed documentation in the course of the development of new safety products.

6. CONCLUSIONS

The objectives of the project plan have been more than totally answered. Not only preliminary results were already transfered to CEN/TC114 in 1999 in order to speed-up amendments of EN 954-1 and improvments to EN 954-2, which was the initial objective, but also essential tools based on the generic IEC 61508 were adapted to the specific machinery sector needs.

As a direct consequence, the more recent project of standard IEC 62061, not yet launched at the kick-off time of this project, has also already benefited from it.

An important contribution of the project deals with safety-related software development and validation techniques.

A major contribution has introduced modelling techniques and probabilistic assessment methods of dangerous failure rates and of suited architectures to achieve risk reduction. Markov models are one of the evaluation techniques dealt with during the STSARCES project. Concerning online tests performed automatically within a safety system, the immense influence of the diagnostic coverage could be demonstrated. The other aspect is the appropriate diagnostic test interval for a particular system architecture and application. EN 954-1 does not supply sufficient information on this topic. For a category 2 system its clause 6.2.3 is just requiring checks by the machine control system "at suitable intervals" without explaining what is "suitable". The test intervals of systems claiming for category 3 or 4 are not either specified in this standard.

Help can be provided by the Markov approach. By implementing a new feature in the Markov models we have been able to deliver some useful information concerning the adequate diagnostic test interval. It turned out that single channel systems and multiple channel systems are behaving quite different.

Interesting findings are described establishing a relationship between sufficient online test rates and the MTTFd of one of the redundant channels. They provide advice for the system designer as well as hints for the person carrying out the evaluation (for more detailed information on this topic see chapters 5.3. and 6.3 of Annex 6).

Comprehensive links could also be established between the category concept (EN 954 approach) and SILs (IEC 61508 approach) for given architectures and realistic reliability data.

6.1. Contribution of STSARCES to the EN954

The EN 954 standard is made of a part 1 (harmonised standard since 1996), of a user's guide FD CR 954-100 (harmonised in 1999), of a project of standard pr 954-2 (at the level of a CEN enquiry procedure) and of a project of revision of the standard EN 954-1.

During the two last years, in the course of the meetings of the Joint Working Group in charge of the EN 954, the STSARCES progress reports have been commonly used as an important input when one is drafting a requirement or a validation procedure concerning safety functions based on PES (Programmable Electronic Systems).

By lack of decisive knowledge, authors of the EN 954-2 were led to note and to write down the following :

- In control systems where the provision of the safety functions incorporates PES, it is inadequate to use only categories if :
  - the safety function of the control system relies solely on PES,
  - or the structure of the control system is complex,
  - or the contribution to the risk reduction at the machine is high.

In which case additional factors, eg systematic faults should also be taken into account faults (scope of EN 954-2).

- In a note, the CEN/TC 114-CLC/TC44X-JWG6 proposes to process this question through the amendment to EN 954-1 (1996).

It is at the level of the amendment of Part 1, where results of the STSARCES Project are wished impatiently because the software aspects (systematic faults) need to be introduce in the concept of the categories when PES are category 2, 3 or 4.

It is also indicated in the amendment that a machine application standard derived from the published IEC 61508 standard is under preparation by the IEC/TC 44/WG7 (here the basic concept is the Safety Integrity Level or 'SIL').

STSARCES results will allow defining credible and understandable links between categories (EN 954) and SILs in the draft IEC 62061. This connection is indispensable during the design and development phases of control circuits for the machinery which make use of mechanical components as well as hydraulic or pneumatic and electro-mechanical ones, based on the category concept, and also PES, better characterised by the concept of SIL.

A part of the WP4 results (a study of the links and divergences between IEC 61508 and EN 954, HSE, WP4 Task 1) has already been communicated in 1999 to the JWG6 and the presentation of the WP2.1 results is also wait for (Quantitative Analysis of Complex Electronic Systems using Fault Tree Analysis and Markov Modelling).

Experts involved in standardisation are convinced that STSARCES shall improve the validation methods of PES in their use for safety functions both in EN 954 and in IEC 62061.

6.2. Contribution of STSARCES to IEC 62061

Works to develop the standard IEC 62061 "Safety of Machinery-Functional Safety of Electrical, Electronic and Programmable control systems for Machinery" were initiated on March 1998 by the working group TC 44 WG7.

A first CD version is foreseen for the second half of year 2000, one year later than expected. This delay is largely due to difficulties in the interpretation of the IEC 61508 standard by persons not familiar with its concepts, and also in the necessity to take into account both standards at the same time, like IEC 61508 (probabilistic approach for CES devices) and EN 954 (deterministic approach for all types of technologies).

The purpose of the work is to develop a sector standard for machines, guided by the basic safety publication IEC 61508. This standard will define a hierarchy of safety performance levels by :

- Adapting the requirements of IEC 61508 to suit established principles of risk assessement and safety integration of machinery ; and
- Defining the methodology for the implementation of EN 954 within the hierarchy of performance levels.

This standard is intended for use by :

- The suppliers of machines, to enable the specification of the relevant safety-related performance levels of electrical, electronic and programmable control systems used on a machine ; and
- The designers and integrators of such systems, to enable them to meet the specified performance levels.

Until now the work is aiming at specifying a methodology for the integration of components (already certified previously) in order to develop safe control systems for machines. Requirements applied to components (e.g., safety light curtain) are those in the IEC 61508.

Results of STSARCES are again valuable here, when one considers problems raised by the integrated use of concepts derived both from IEC 61508 and from EN 954.

A part of the WP4 results (a study of the links and divergences between IEC 61508 and EN 954, HSE, WP4 Task 1) were transferred with the agreement of the STSARCES Steering Committee to the standardisation group as soon as on September 1999 to help solving the adaptation of the safety performance requirements of EN 954 within the overall functional safety philosophy of IEC 61508.

More recently on January 2000, a draft of WP report on Software aspects (Software quality and safety requirements, INRS, WP1.2, Aspect 1) was transferred to the working group with the intention to introduce the results as an annex of the standard, to the attention of designers of the embedded software used in the machinery.

6.3. Experience exchange between partners for validation of complex electronic systems for machinery

The management structure of the Project has deeply favoured communications between laboratories located in different countries. Some of them were having different levels of experience and also their industrial culture was not exactly the same. Furthermore it is well known that many manufacturers of safety devices are located also in these countries where are installed the major machines suppliers, as Germany for example. Participation of at least one German partner in almost any Task of the project was of a great benefit.

Apart from the main meetings in the course of the Project (the periodic six monthly meetings of the Steering Committee and the plenary yearly meetings) the organisation based on Work-Packages - WPs - with each of them under the responsibility of a WP manager, has induced thematic technical meetings with an in-depth investigation of specific problems and extensive exchange of experience during the visits of laboratories and installations :

- in BILBAO, MUNICH, TAMPERE for the WP 3 group headed by VTT although the distances had been a cause of important expenses. Because of some common points of interest, other partners had also to visit CNVM in Spain, like INRS (WP1 manager).
- in BONN in Germany and BORAS in Sweden for the WP 2 group headed by BIA.
- in NANCY and PARIS for the WP 1 group headed by INRS.
- in GRENOBLE at the JAY firm with INERIS within theWP 5 on innovative studies by the manufacturers and with INRS to validate the content of WP 1.2, aspects 1 and 2.

Due to the reduced funding possibilities for so many travels, good use was made of other opportunities allowing for short meetings, like the participation of several partners in Technical Committees for standardisation (CEN TC 114, IEC TC 44) or attendance to conferences.

Connections between Test-Houses and manufacturers have been difficult to maintain constant in time during the whole duration of the project.

In the first year, it appeared that a SME, well known on the market as a designer of innovative devices, could not allocate any availability of its expert persons to the Project because commercial problems had occur on a leading product which had to be re-designed. As it was an associated partner to a main contractor, the consortium had to find another solutions to validate as realistically as possible the analysis and testing procedures developed by the Test-House.

Another difficulty arose with a second manufacturer involved in the project, also due to commercial problems, but consequences on the programme could be reduced and the partner could participate to the final validation of the results.

6.4. Validation of the project by external manufacturers

A special seminar with manufacturers of safety related systems, not directly involved in the project, to inform them of the results and to improve the intelligibility of the final report presentation, was programmed near the end of the project.

In order to ensure the largest international attendance to such an event, this seminar was integrated to the most significant international conference organized on late 1999 on occupational safety, the MONTREAL International Conference on Safety of Industrial Automated Systems, 4-7 October 1999. This was made possible thanks to the Conference Scientific Committee, which included BIA, HSE and INRS, members of the STSARCES Steering Committee, and IRSST, the organizing Institution.

Five papers on the STSARCES results were presented by their authors in plenary sessions (an overview by the coordinator, and four technical reports on each work-package). Since the corresponding session chairmen were INERIS, BIA, HSE and INRS, the discussions could easily be oriented to sense the acceptance of STSARCES results by the attending manufacturers.

Finally, discussions could follow in a more informal manner after the sessions since a lot of them had their own stands in the exhibition installed at the same place. It was agreed that this lifecycle approach was well received, but a big concern was expressed on the need for a more in-depth collaboration between certification bodies and manufacturers in the near future, from the design stage until the final tests in vue of issuing a conformance certificate. There was a similar wish expressed by the Test-Houses for a deeper collaboration, extended through the whole "certification process" cycle.

7. PUBLICATIONS

- **-Brown, S.J. & Frost, S.** Health & Safety Executive, Electrical and Control Systems Unit, Technology Division. A study of the links & divergences between draft IEC 61508 and EN 954. STSARCES, WP4 Task1 Report Ref : STS-WP4-1001, Issue 02, September 1998.

- International Conference on Safety of Industrial Automated Systems, Montreal, 5-7 October 1999 **and 2nd European Project STSARCES Seminar :**

  - **-Villeneuve de Janti, P**. Institut National de l'Environnement Industriel et des Risques (INERIS), France. STSARCES : An European research initiative to harmonize validations methods on safety related complex electronic systems.

  - **-Reinert D. & Dorra M**. Berufsgenossenschaftliches Institut für Arbeitssicherheit (BIA), Germany. EU Project STSARCES-Hardware Safety : Quantitative analysis of complex electronic systems using Markov Modeling.

  - **-Charpentier P**. Institut National de Recherche et de Sécurité (INRS), France. Preventing software errors through quality control and testing.

  - **-Durka J.L. & Reinert D**. Institut National de l'Environnement Industriel et des Risques (INERIS), France; Berufsgenossenschaftliches Institut für Arbeitssicherheit (BIA), Germany. Design and validation of innovative technologies : ASICs in safety related systems for machines.

  - **-Frost, S & Ward G.R**. Health & Safety Executive (HSE), United Kingdom. Functional safety for machinery control : challenges for standardisation.

8. BIBLIOGRAPHY

1 DIN V VDE 0801/01.90 and A1/10.94 : Grundsätze für Rechner in Systemen mit Sicherheitsaufgaben und Änderung A1.

2 EN 954-1 (1996) : Safety of machinery - Safety-related Parts of control systems (Identical with ISO/IEC DIS 13849-1).

3 IEC 61508 : Functional Safety-Related-Systems: Part 1 : General Requirements; Part 2 : Requirements for electrical, electronic, programmable electronic systems; Part 3 : Software Requirements; Part 4 : Definitions and abbreviations of terms; Part 5 : Guidelines for the application of part 1; Part 6 : Guidelines for the application of part 2 and 3; Part 7 : Bibliography of techniques and measures.

4 DIN V 19250: Leittechnik. Grundlegende Sicherheitsbetrachtungen für MSR-Schutzein-richtungen. Beuth-Verlag, Berlin 1994.

5 Reinert, D.; T. Bömer : Modern Sensors as protective devices for the safety of machinery. Proceedings Volume 1 : 3rd Eurolab Symposium 5-7.6.1996 Berlin. Testing and Analysis for Industrial Competitiveness and sustainable Development. Wirtschaftsverlag NW. Bremerhaven 1996, pp. 215-224.

6 Reinert, D.; Schaefer, M.: Integrated safety in flexible manufacturing systems. In R.D. Schraft, G. Brandenburg, & W. Leidig, (Eds.), Tagungsband SPS/IPC/DRIVES98 (pp. 305-314). Heidelberg, Germany: Hüthig-Verlag 1998.

7 Reinert, D. et al : Validation of functional safety of programmable electronic systems according to IEC 1508. Preprints of the 5th International Working Conference on Dependable Computing for Critical Applications, Sept. 27-29, 1995.

8 EN 1050 Sécurité des machines. Principes pour l'appréciation du risque. (Machine safety. Risk appreciation principles). 1997-01.

9 FARADIP.THREE (Failure Rate and Failure Mode Data Bank and Failure Mode and Effect Analysis Package). Technis, Tonbridge, Kent UK 1997.

10 SN 29500 Failure Rates of Components, Part 1 – 7, Part 9 – 10. Siemens AG, ZT TN Corporate Functions Technical Regulation and Standardization, Munich and Erlangen 1982 – 1999.

11 GUIDE DE LA SURETE DE FONCTIONNEMENT. Laprie J. C. et al. CEPADUES EDITIONS – 1995.

12 MODE DE DEFAILLANCE DES CIRCUITS INTEGRES - Constat des problèmes posés. GROUPE DE TRAVAIL MDCI DE L'ISDF – 1994.

13 ARE COMPONENTS STILL THE MAJOR PROBLEM: A REVIEW OF ELECTRONIC SYSTEM AND DEVICE FIELD FAILURE RETURNS. Pecht M., Ramappan V. IEEE transactions on components, hybrids, ... - vol.15 - No. 6 - Dec. 1992 - pp. 1160, 1164.

14 A STUDY OF FAILURES BASED ON U.S. POWER REACTOR ABNORMAL OCCURRENCE REPORTS. Taylor J. R. Reliability of nuclear power plant - IAEA-SM-195/16 – 1975.

[15] DEFAILLANCES DEPENDANTES ET DE CAUSE COMMUNE. Villemeur A. Sûreté de fonctionnement des systèmes industriels. Ed. Eyrolles, 1988, pp. 371, 410.

[16] DEPENDABILITY OF CRITICAL COMPUTERS SYSTEMS EWICS/TC7 Ed. F.J. Redmill - 1988.

[17] Method for performing diversity and defense-in-depth analyses of reactor protection systems Preckshot G.G. Fission Energy and Systems Safety Program, Rapport UCRL-ID-119239, Dec 1994.

[18] HANDBOOK OF SOFTWARE RELIABILITY ENGINEERING Lyu M.R. Computing Mac Graw-Hill/IEEE Computer Society Press, 1995.

[19] Analysis of faults in a n-version software experiment: Brilliant S.S., Knight J.C., Leveson N.G. IEEE Transactions on software engineering, Vol. 16, N°2, Feb. 1990, pp 238, 247.

[20] An experimental evaluation of the assumption of independance in multiversion software: Knight J.C., Leveson N.G. IEEE Transactions on software engineering, Vol. 12, N°1, Jan. 1986, pp 96, 109.

[21] A theorical basis of multiversions software subject to coincident errors: Eckhardt D.E., Lee L.D. IEEE Transactions on software engineering, Vol. 11, N°12, Dec. 1985, pp 1511, 1517.

[22] An experimental evaluation of software redundancy as a strategy for improving reliability: Eckhardt D.E., Caglayan A.K., McAllister D.F., Vouk M.A., Kelly J.P.J. IEEE Transactions on software engineering, Vol. 17, N°7, July 1991, pp 692, 702.
