S T S A R C E S

Standards for Safety Related Complex Electronic Systems

A n n e x 5

Tools for Software fault avoidance

Task 3: Common mode faults in safety systems

F i n a l R e p o r t o f W P 1 . 2

Philippe Charpentier

INRS

E u r o p e a n P r o j e c t S T S A R C E S

Contract SMT 4CT97-2191

Contents

SUMMARY

This document deals with the question of common mode failures, inherent to the concept of redundancy. Problems posed by these failures are treated at a global system level, without focusing on aspects linked solely to software diversity.

Definitions are provided to adopt a common and unambiguous language. Some bibliographic data on the nature of phenomena at the root of common mode failures are then presented.

Actions to be taken to fight against these phenomena are also proposed, that concern fault tolerance, fault avoidance and fault forecasting. General measures are given for each of these headings, then measures specific to common mode failures. Although closely linked, software and hardware are treated separately.

A check-list is given, that organises the points to be checked at the different phases of the life cycle of a product, and hence to ensure the arrangements made regarding the tolerance, avoidance, and forecasting of common mode failures.

CONTEXT

Perspective of the problem

The benefits of processing power and flexibility of use provided by microprocessors mean that today an ever increasing number of these components are to be found in the control systems of machinery. However, problems do arise regarding the question of processing safety functions in addition to “normal” operating functions with these control systems. It is necessary, as a result, to consider the European standard EN 954-1 [EN954], the only reference available at the moment for the safety of machinery. This standard, although it is incomplete as it takes into account only those aspects linked to hardware faults (software is not dealt with), subdivides safety-related parts of control systems into five categories according to their behaviour in the presence of faults. For category 4, corresponding to the highest levels of risk, the designer must respect the following requirements:

The safety-related parts of control systems shall, as a minimum, be designed, constructed, selected, assembled and combined in accordance with the relevant standards so that they can withstand the expected influences.

The use of well-tried safety principles must be applied.

The safety-related parts of control systems shall be designed in such a way that:

- *a single fault in any of these safety-related parts does not lead to the loss of the safety function, and*
- *the single fault is detected at or before the next demand upon the safety function. If this is not possible, an accumulation of faults shall not lead to the loss of the safety function.*

By laying down such a requirement, the standard implicitly orientates the designer towards redundant structures.

> Remark: Although redundancy is the natural route chosen to respect the requirements of category 4, single channel solutions can also be envisaged. The coded monoprocessor developed for transport-related applications is one example. However, its specific nature and the difficulties attached to employing it make it poorly adapted to the safety of machinery. It will therefore not be gone into in this document.

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 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.

Taking common mode phenomena into account

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.

Work schedule

The work initially foreseen for aspect 3 of WP 1.2 of STSARCES should have encompassed software diversification only. However, the bibliography consulted on this subject has demonstrated that this technique quickly finds its limitations, the advantages provided - simplification of tests, lower version reliability - being minimised by the restrictions on the true independence of the different versions of a software product. This observation, added to the difficulty of finding a redundant system in the field of machine safety, motivated a widening of the study initially planned. Problems posed by common mode phenomena are therefore treated at a global hardware / software system level, without focusing on aspects linked solely to software diversity.

Various definitions are provided in the interest of adopting a common and unambiguous language. Some bibliographic data on the nature of phenomena at the root of common mode failures, together with a classification allowing the original causes of common mode failures to be listed, are then presented.

The following chapters concern the action to be taken to fight against these phenomena, following the methodology proposed by J.C. LAPRIE [LAP95] to construct the dependability of a system, namely :

- fault tolerance, a set of methods and techniques intended to provide a service that ensures the functions of the system despite faults affecting its components, its design or its interactions with man or other systems ;

- <u>fault avoidance</u>, a set of methods and techniques intended both to reduce the presence and to avoid the introduction of faults (in number and severity) . Fault elimination and fault prevention are parts of fault avoidance.

- <u>fault forecasting</u>, set of methods and techniques intended to estimate the presence, the creation, and the consequences of faults.

General measures are proposed for each of these headings, then measures specific to phenomena caused by common modes. Although closely linked, software and hardware are treated separately.

A check-list is provided to organise the points to be checked at the different phases of the life cycle of a product, and hence to ensure the arrangements made regarding the tolerance, avoidance, and forecasting of common mode failures.

<u>Important remark</u>: This document is, to a great extent, the product of theoretical analysis based on bibliographical investigations. In order to confirm its relevance and usefulness, certain methods have been applied to the SAFELOC prototype developed by the Swedish institute IVF.

COMMON MODE FAILURES

1.1. DEFINITIONS

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[1] 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. [LAP95] and [MDCI] are in agreement in defining the failure as being the transition from correct service to incorrect service. The failure occurs when the service provided no longer conforms to the specification. The deemed or supposed cause of a failure is a fault [MDCI], a definition that constitutes a shortcut compared to that given by [LAP95] 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 [PECH]. To all intents and purposes, [PECH], [MDCI], and [LAP95] define 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 given by [TAYL] clearly synthesises the different wordings found in the bibliography [VILL], [LAP95]. 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. [EDWA] 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 [VILL].

[1] Common Mode Failure: CMF

Remark : - When referring to this problem, [HSE] and [VILL] employ the vocabulary of common cause failure. Although apparently more explicit, this definition will not be retained as it is seldom employed.

- [STRIGINI] introduces the idea of coincident failure caused by faults relative to the input domains. This notion has been taken up by [LYU] with regard to software. Coincident failures affect two (or more) functionally equivalent software components subject to the same input.

Common mode failure appearance mechanism

The definition given by J.C. LAPRIE [LAP95] 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 1) is therefore obtained :

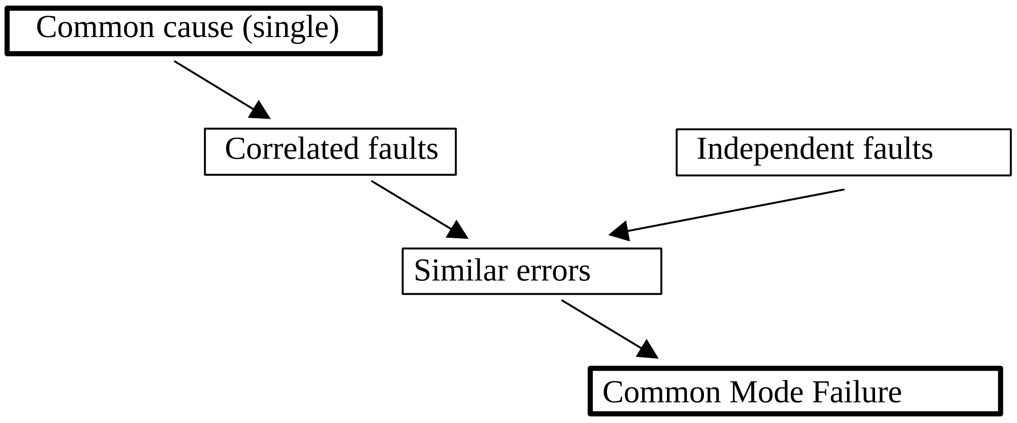

Figure 1: Common mode failure appearance mechanism

The standard " production " way of common mode failures is that which starts from a common cause. In certain cases, it may be that independent faults lead to similar errors. [LAP95]. On account of their low probability of appearance, these faults will not be gone into in the remainder of this document.

1.2. IDENTIFYING COMMON MODE FAILURES

In very general terms, [TAYL] points out that the causes of common mode failures can be a common property, a common process, a common environment or a common external event. These common influences [EWIC] can affect the system before it is put into operational service : design, manufacture, installation. In this case, the manifestations are not immediate and the system is " prone " to failure. They can also affect the system during operation, for example in the case of erroneous design or environmental disturbances .

In practice, common mode failures only appear if there is :

AN INITIATING SOURCE

This is the common cause capable of causing correlated faults, which is to say one or several internal or external events causing multiple failures of elementary systems. The former point has been confirmed by [BUCH] who points out an activation factor to trigger common mode failures, as well as the accumulation of systematic faults within the system. This latter point can be excluded from analyses if sufficiently frequent checks have been executed to detect any accumulation. Looking again at the terminology proposed by J.C. LAPRIE, several types of initiators can be distinguished :

Degradation of electronic components

The faults are internal accidental physical faults, whether permanent or temporary. It is, however, difficult to make the distinction between a fault appearing over the course of development (wrongly dimensioned components) and in operation (random failure of several components).

External perturbations

The increasing level of integration of electronic components means that they are very sensitive to external, accidental, operational physical faults such as interference from electromagnetic sources.

External interference can cause either transient faults, characteristic of interference due to the physical environment, or permanent faults, which involve interference arising from the operating environment. The limits separating these faults are, however, difficult to determine.

Human faults introduced at the different stages of the product life cycle

These are internal, accidental design faults linked to the development of systems. On account of the systematic faults that they generate, human faults are most certainly the largest source of common mode failures. Software faults are a typical example of human faults.

AND

A CORRELATION BETWEEN ELEMENTARY SYSTEMS OR COMPONENTS

As the definition suggests, there can only be a common mode failure if a correlation exists between several hardware or software “components” subject to an initiator. The correlation can exist between elementary systems or between components, the difference being in the level of description chosen [VILL]. They can be linked to equipment used in common, to physical interactions or to human intervention. [EDWA] distinguishes between two types of correlation:

Type 1:

– failure of an elementary system or of a component common to all the channels of a redundant system.

Type 2:

- coincidence of failure of two or more than two identical components of separated channels of a redundant system due to a common cause, the term components having a very broad meaning ;
- failure of one or several components of a different type having a single initial cause.

With these identification criteria, figure 1 becomes, for Type 1:

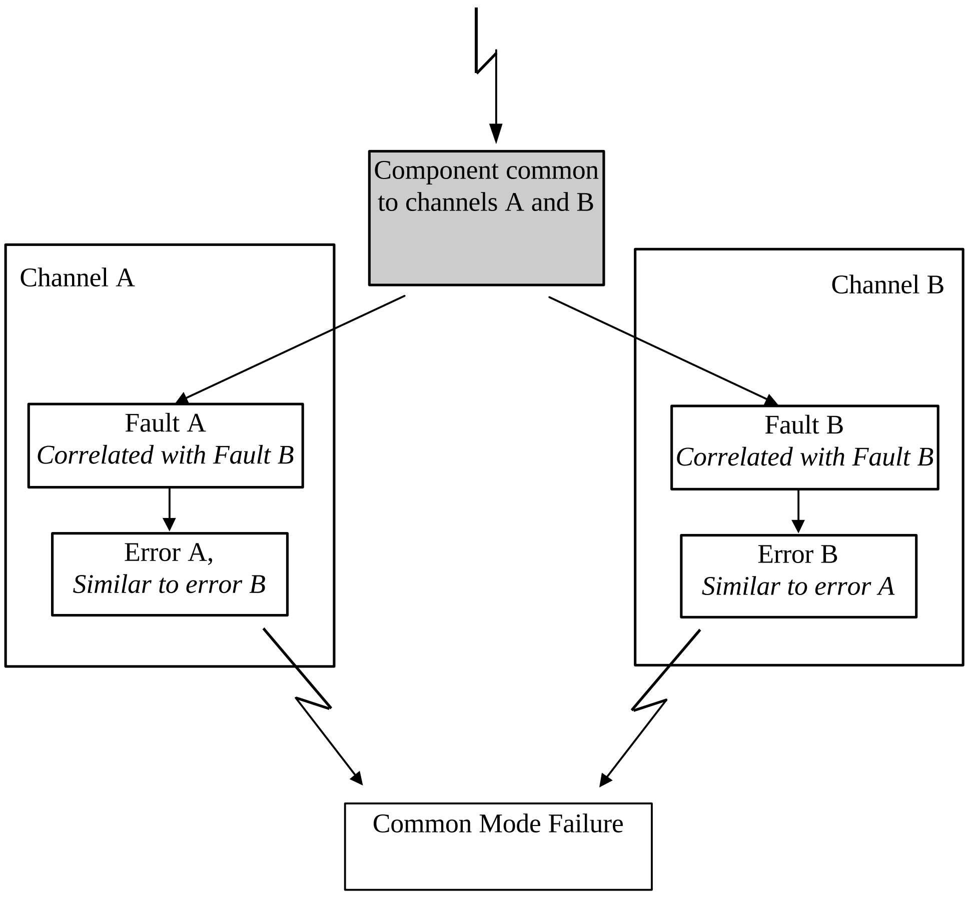
Component common to channels A and B
Channel A
Channel B
Fault A *Correlated with Fault B*
Fault B *Correlated with Fault B*
Error A, *Similar to error B*
Error B *Similar to error A*
Common Mode Failure

Figure 2: Type 1 correlation

Whereas, for type 2, it becomes

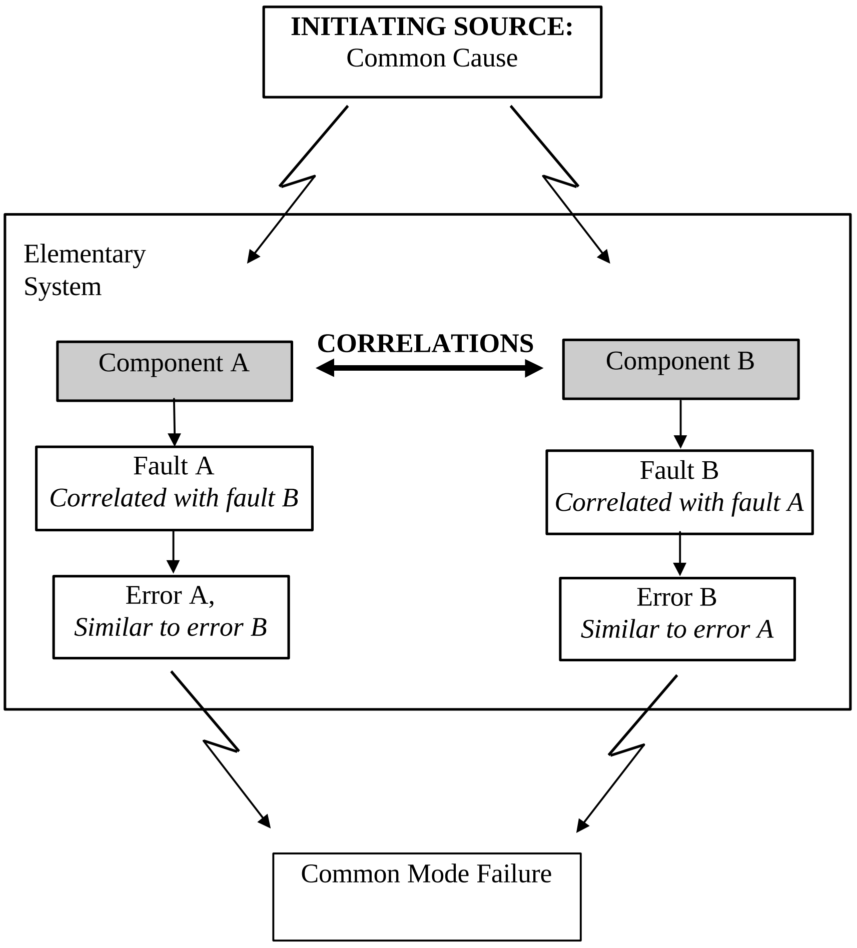

Figure 3: Type 2 correlation

Example of correlated faults: compilation faults

Let us consider two homogeneous redundant structures - identical source programme and hardware - the former developed by means of a single compiler C1, the latter with two different compilers C1 and C2. All the correlated faults generated by compilation at executable programme level of each microprocessor can be represented by the figure 4 :

Case of a single compiler C1 | Case of two compilers C1 and C2

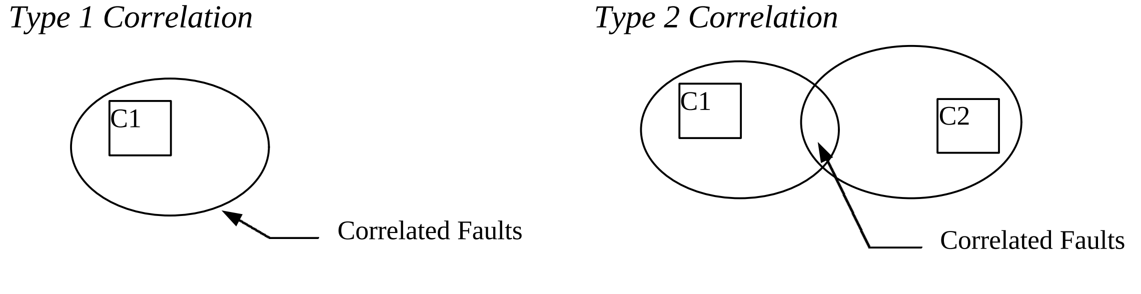

Figure 4: Example of correlated faults

The initiator is the man, who introduced the compilation faults. In the first case, the correlation between the executable programmes is total, whereas in the second case it is only at the intersection of the faults introduced by each of the compilers.

It can therefore be noted that:

No diversity $\longrightarrow$ $\boxed{\text{The entity (in this case compiler C1) is the potential cause of correlated faults for the redundant structure in question}}$

Diversity $\longrightarrow$ $\boxed{\text{Only the part common to both compilers is likely to cause correlated faults}}$

In the case where two compilers are employed, the analysis of the common mode failures therefore requires determination of all the errors relative to each of the faults of compiler C1. For each error produced, it must then be ensured that there is no corresponding fault in C2.

1.3. FIELD EXPERIENCE

The first way that should be considered to grasp a given phenomena is to try to determine its possible causes. Firstly, faults likely to cause failures within microprocessor-based devices must therefore be looked for.

When tackling causes of common mode failures, the authors of the consulted bibliography systematically mention field experience, the only apparent means of obtaining information on these phenomena.

Remark: Articles covering the causes and consequences of common mode failures are few and far between, manufacturers certainly hesitating to publish their results and users only rarely undertaking to understand the exact cause of failure of their equipment. The field experiences mentioned in these papers are specific to the field of application from which they stem, and report information about given equipments working in well defined operating conditions. In addition, the only available elements mainly concern systems whose complexity has no common ground with that of machine control systems. The results stemming from them are therefore very “ specific ” and difficult to extend to any type of application.

The interest of the information of this paragraph therefore lies more in the typology of the fault encountered than in the figures provided, which more often than not cannot even be considered as orders of magnitude.

1.3.1. General

The various causes of failures given by [MILL] for the field experience stemming from the nuclear sector are very general. 14 causes have been identified including operating, maintenance and test, design error, error regarding manufacturing, construction or quality control, faulty procedures, extreme environment, electrical or mechanical malfunction, and component failure or drift.

In the same field of activity, the analysis of the field experience provided by [TAYL] leads to the following distribution:

– 36% due to design,

- 19% due to component failure,
- 12% due to maintenance or installation errors,
- 11% due to operating errors,
- 10% due to administrative or procedural errors,
- 12% of unknown origin.

In a paper covering the data stemming from field experience over 20 years in the avionics industry, [PECH] was able to subdivide the failures into the following classes:

- components,
- interconnections,
- mechanical and electrical design of the system,
- non repetition or disappearance of problems after reset (Could Not Duplicate),
- environment stresses,
- utilisation.

The results relative to components are more difficult to interpret because of the lack of precision in the definition of these components. It can, however, be seen that the number of faults affecting the central processing unit is very low (21 out of 2486), and that they are mainly due to electrical malfunctions or to component failure. It would also appear that the faults mainly affect the input/output or interface components.

1.3.2. Faults specific to common mode failures

The data provided by [VILL] stem for the most part from complex multi-technology systems (mechanic, hydraulic, electric). They are primarily qualitative and are not necessarily exclusive to common mode.

Environmental aggression: These causes are external to the system.

Design errors: These errors are difficult to predict as they are associated with the limits of know-how. They notably concern :

- the inability of a component or a system to fulfil its task ; the impossibility to conduct exhaustive tests or to simulate real conditions (for cost or planning reasons) can prevent their detection during tests ; this, for instance, is the case with software ;
- inadequate or harmful periodic tests like, for instance, poorly designed periodic tests likely to reveal common cause faults ;
- elementary system difficult to exploit ; this source is minimised at software level by writing simple, structured and modular programmes ;
- elementary system difficult to maintain ; this is the case with poorly designed or documented software products that cause problems at the modification phase ;
- poor optimisation regarding common cause failures ; the protection against one common cause failure can produce or favour another ; this is the case with certain

diversified structures which appear to protect against common mode failures while promoting the development of other failures of this type through slackness ;

– lacks during design studies, the answer to which may be quality control ; this is the main problem encountered with software.

Manufacturing errors: More often than not these are non conformities with the manufacturing technical specifications or technological errors ; they can be encountered on the software if the conditions of its “ manufacture ” (compiler or other) change ; in the case of hardware, they can appear if the technologies employed for the components are immature.

Exploitation errors: This is a significant source of error when “ system ” aspects are considered.

The results of [MILL] are provided for information purposes, and indicate that the faults at the source of common mode failures are caused by the personnel (57%), maintenance or procedures. A very small proportion (<5%) is due to extreme environmental conditions. However, no further details are given regarding these environments and their workings.

Finally, in the field of avionics, [TAYL] noted 83 coupled failures (or CMF) which have been subdivided into three main types:

– 16% of design failures, noted in certain seldom employed operating conditions,

– 77% of failures resulting from the increase in the failure rates of each separate components, which increases the probability of having several components in a failure state, broken down,

– 7% of failures due to operating maintenance errors on a group of components.

1.3.3. Faults common to diversified software

A number of academic experiments give an idea of the software faults at the root of common mode failures [KNIG], [BRIL], [ECK1]. They all concern N-versions programming researches, which consists of independently developing N programmes from the same specification. They were conducted on data algorithm processing for aerospace applications.

The authors of these experiments analysed the origins of common failures on at least two versions. The faults identified are closely linked to the application, and mostly concern a lack of understanding of the specifications. The following common faults were noted :

– programmers made the assumption of an equivalence between the comparison of cosines and of the corresponding angles; this fault cannot be put down to the specifications [KNIG],

– the calculation of a value of an angle was erroneous, primarily because of weaknesses in geometry on the part of the programmers [KNIG],

– the calculation of the angle formed by three aligned points was erroneous : certain programmers did not distinguish between 0 and π, others did not take into account the order of points. This is a typical example of faults that are in logic relation and caused correlated failures [BRIL],

– common lack of understanding of the various properties of non orthogonal co-ordinate systems ; the faults were not completely identical but led to coincident failures [ECK1],

– logic faults [ECK1],

– threshold calculation [ECK1],

– wrongly initialisation of a variable [ECK1].

It can be seen that none of the faults identified stems from a specification error, that they are closely linked to the application, and that for the most part are due a lack of understanding of the specifications. Other faults were listed which, however, did not cause common mode failures, for example the omission by the programmer to assign a value to a function or the use of an erroneous expression to index a table.

1.3.4. Conclusions on field experience

At the outcome of investigating data stemming from avionics, [PECH] made a number of remarks :

– what is a major cause of failure for one study is not for another ;

– no details can be given on the figures, which vary in some cases within a range from 25 to 75% of the number of failures found ;

– a study focusing on avionics shows that only 3 to 9% of system failures are due to components, the remainder being due to CND (Could Not Duplicate), to man, to an erroneous application, etc., which highlights the high and ever increasing reliability of components ;

– the causes of failures vary with time, certain being reduced by advances in technology or in manufacturing processes ;

– the types of failures also vary with time ; the major difficulty in understanding exactly the failure mechanisms often leads to an “ unknown ”, unverified or other type of failure typology ;

– the location and the mechanisms of failures must be examined for specific cases.

These remarks summarise the problems encountered when it is required to exploit rigorously the results stemming from field experience. Such exploitation is extremely delicate and can lead to premature generalisations. If the problem of confidentiality is added, these restrictions emitted on the field experience certainly explain the little bibliographic data available on this subject.

This lack of information is confirmed by the work relative to integrated-circuit failure modes conducted by the MCDI group of ISDF. This work indeed confirms the very great difficulty, if not the impossibility, of knowing exactly the failures that can affect integrated circuits.

The problem of common mode faults must therefore be studied without a perfect knowledge of the phenomena involved. It must therefore be based on “ suspicions ” stemming from certain field experience that the designer or analyst deems meaningful.

It is this reason that has led to the selective application [LALA] of techniques and methods to tolerate, eliminate and predict common mode failures.

1.4. CONSEQUENCES OF COMMON MODE FAILURES

The consequences of common mode failures do not fundamentally differ from failures due to single faults. To a great extent they depend on :

– the cause of these failures ; transient electromagnetic disturbance would certainly not have the same consequence as a software fault affecting both channels of a redundant structure ;

– the level at which these consequences is observed ; electromagnetic disturbance would have certain influences at component or bus level, which in turn would have consequences at device output level ;

– the structure of the device which, in certain cases, hides certain consequences which will have been able to be dangerous.

2. CORRELATED FAULT TOLERANCE

2.1. GENERAL

Fault tolerance has been defined by J.C. LAPRIE in [LAP95] as all the constructive methods and techniques intended to provide a service and to fulfil the function(s) of the system despite faults that may affect its components, its design or its interactions with man or other systems. These methods primarily concern the descending phase of the system life cycle, from the specification to the design. Aspects of interaction with the environment are beyond the scope of the work presented in this document.

Fault tolerance is obtained by employing constructive facilities capable of detecting faults that can affect the system. Once the faults have been detected, the system must adopt a safe behaviour, for example by recovery, pursuit or error compensation.

These latter types of behaviour allow the task to be carried on, often in a degraded mode. In the domain of machinery safety, the safe state is generally used, which leads to switching off the power to the actuators .

Depending on the types of faults that have to be taken into consideration, several possibilities are available to designers to produce a fault-tolerant system. If correlated faults are to be considered, [BOUR], [EDWA], and [EWIC] have set out a number of rules and techniques that should be applied during the design of an electronic device.

2.1.1. Fault detection mechanisms

Through construction it is possible to establish defence mechanisms against common mode failures within programmable electronic systems [61508]. These mechanisms are not unique to correlated faults, and are intended to detect faults before they cause failures, thus increasing the robustness of the systems in which they are installed.

These mechanisms [CHAR1] can be subdivided into two classes that depend on the time of detection:

Periodic test : These are generally software procedures, that test the correct operation of a system resource at a given time. The periodic test of a resource (processing or memory facilities or input/output peripheral device) is run on power on and possibly at periodic intervals during operation. As a result, it is generally unsuitable for detecting an operating anomaly due, for instance, to a transient fault or a systematic software fault, or to an error occurring between two tests. Periodic tests are primarily intended to detect potential accumulations of faults [HSE], [BUCH], [EN954], the assumed cause of certain common mode failures.

Memory and microprocessor checking are typical examples of periodic tests.

On-line monitoring : These are hardware and/or software mechanisms that permanently monitor the task of all or part of a system. Errors are detected instantaneously or within a very short period of time.

The watchdog is the first level of on-line monitoring. Combined with a effective system of comparison, redundancy is able to detect, more or less in real time, the faults that can appear in a microprocessor based system.

2.1.2. Separation

Separation [BOUR], [EDWA], [MOSL] is a constructive technique to stop failures propagating from one function to another, thereby limiting analyses to sensitive points. Certain facilities employed for separation are on the borderline between fault tolerance and avoidance. Two types of propagation are conceivable [UCRL].

Physical propagation

This involves the system hardware. The means of combating this type of propagation are :

- physical separation,
- electrical isolation of the different channels,
- power supply separation,
- electrical shieldering,
- no sharing of resources between safety functions and other functions,
- no use of components common with several channels,
- avoiding grouping too many cables or sub assemblies,
- ....

Logic propagation

This primarily concerns the software of the system. The means of combating this type of propagation are :

- physical isolation of the software modules, which are executed by two different microprocessors,
- no interactions through shared memories,
- no dual directional links between systems,
- employing safety protocols for transferring data by networks,
- .....

<u>Remark</u> : - Separation must be applied to each channel.

- As far as possible, separation is also applied during every phase of the

product life cycle : installation, manufacture, etc.

2.1.3. Fail-safe design

A system is "Fail-Safe" if all failures are, to an acceptable degree, minor failures [LAP95]. This is to say that their consequences are of the same magnitude as the advantages gained by the service provided in the absence of failures.

This definition allows for a high degree of subjective appreciation: "to an acceptable degree", "of the same magnitude". The definition given by the EN292-1 [EN292] standard is undoubtedly preferable. The notion of Fail-Safe is defined there as a theoretical condition which is attained if a safety function were to remain unchanged in case of a failure in the power supply or in a component contributing to this function. In practice, this condition is all the more satisfactorily met, the more the effect of failures on the safety function is reduced.

For designing a Fail-Safe system [BOUR], [EDWA], [EWIC], it is necessary, therefore, to at least ensure that short-circuits, open circuits and the stuck at 0 or 1 on the cables, tracks, discret or low integrated components do not alter the safety function. Secondly, dynamic signals are to be preferred to static signals.

2.2. 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 [BOUR], [EDWA], [EWIC]. 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 [UCRL].

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 [EWIC].

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

Remarks :
- The contribution of diversity, whatever the type, is difficult to grasp.
- Recourse to diversity must not constitute an argument to reduce the quality level of a system.

Six types of diversity must be considered [UCRL]:

2.2.1. Human diversity

The effects of man at every stage of the life cycle of a system are very variable and at the source of many accidents. Managed correctly, human diversity is therefore an advantage for the safety of a system.

Example : Different designers designing functionally different safety systems have little chance of making the same design errors.

The factors increasing human diversity are, in a decreasing order of efficiency :

- different organisations for the design (two different firms),
- different teams from the same firm managing the project,
- different designers or programmers,
- different test, installation and certification staff.

2.2.2. Design diversity

This involves the use of different approaches (hardware and software) to resolve an identical or similar problem. It is assumed that different designs will not be affected by common influences.

The factors increasing design diversity are in decreasing order of efficiency:

- different technologies (analogue/ digital), creating different responses to the same cause of fault.
- different approaches with identical technology (AD converter/ DA converter),
- different architectures (component assembly and connection).

Remark: This order (proposed by [UCRL]) remains open to discussion. Architecture diversity may indeed be a means of avoiding correlated faults arising from a human origin as it is necessary to undertake distinct prior study for each part of the architecture.

2.2.3. Software diversity

This 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) [LYU]. 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].

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 5).

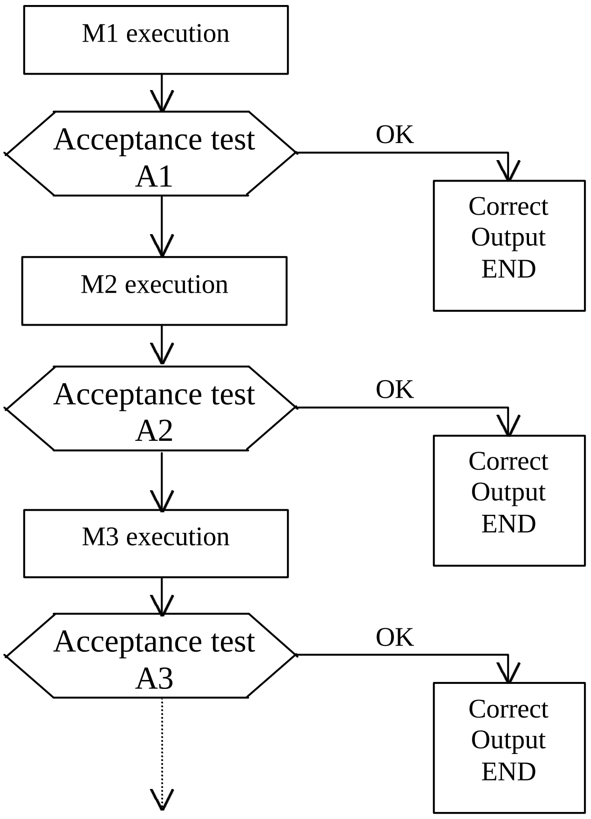

Figure 5: Example of recovery blocks

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 [LYU]. One of the problems posed by this method in a monoprocessor environment is found in the sequential nature of the execution of the versions [LYU].

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 6)

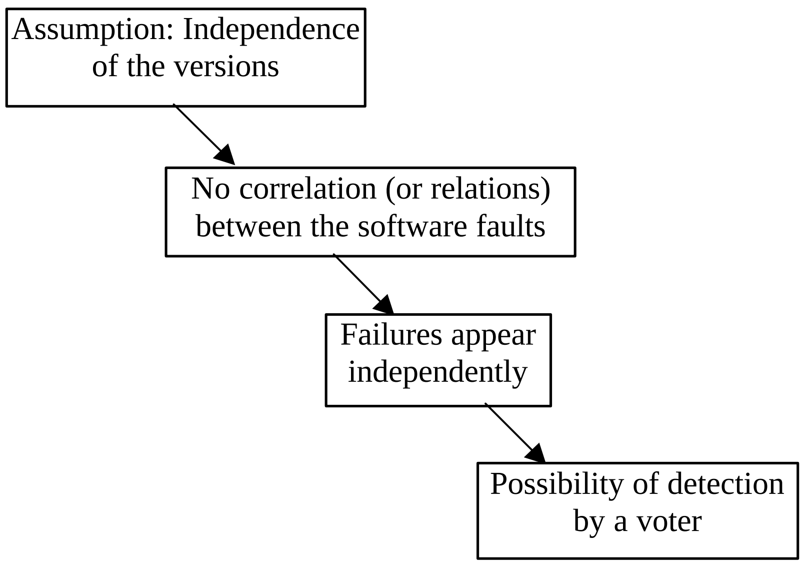
Assumption: Independence of the versions
No correlation (or relations) between the software faults
Failures appear independently
Possibility of detection by a voter

Figure 6: Assumptions for N-version programming

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

- ✓ 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.

Remark: these rules concern fault avoidance essentially and they are applied regardless of the type of software structure installed.

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 [BRIL], [KNIG]. Experience and probabilistic calculations have shown that there is no true independence between the different versions developed [KNIG], [ECK2]. 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 [BRIL].

– 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 [BRIL].

– The results are relative to the experiments carried out. Extension to any type of application is therefore difficult [KNIG].

– 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 [BRIL].

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. [KNIG] advises, 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 [ECK1].

– 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 [KNIG].

– 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 [KNIG].

2.2.4. Functional diversity

Functional diversity consists in creating different physical functions with similar safety actions. In the case of microprocessor system, such diversity can lead for instance to compute a signal or its complement or to test a result and its opposite ,. The designer should, nevertheless, establish with certainty whether or not this technique is indeed efficient in order to avoid the creation of any artificial diversity which does not correspond to a real problem.

The factors increasing functional diversity are, in decreasing order of efficiency :

- difference in the subjacent mechanisms,
- difference in function, control logic or means of activation,

- difference in the response time scales.

2.2.5. Signal diversity

This involves the use of signals coming from different sensors that must be independently capable of indicating abnormal conditions even if the other sensors have failed.

The factors increasing signal diversity are, in decreasing order of efficiency :

- differences in the physical effects measured,
- differences in the parameters measured, with the same principle employed,
- redundancy of identical sensors.

2.2.6. Equipment diversity

This consists in using different equipment to ensure similar safety functions ; the differences should be sufficiently great to lessen vulnerability to CMF. Attention must be paid to false diversities (components sold by two different sources). Computer diversity can have a beneficial effect on software diversity.

The factors increasing equipment diversity are, in decreasing order of efficiency :

- different suppliers of fundamentally different products,
- same suppliers of fundamentally different products,
- different suppliers of similar products,
- different versions of the same product.

In addition, on a highly technical level :

- different architectures for the microprocessors, this difference produces differences at compiler and link editor level amongst others,
- different versions of a microprocessor,
- different printed circuits,
- different bus structures.

3. CORRELATED FAULT AVOIDANCE

3.1. GENERAL

J.C. LAPRIE [LAP95] defines fault avoidance as the application of methods and techniques intended to obtain, as far as possible, a system that is free of fault. These techniques are essentially related to both hardware and software testing. Unlike fault tolerance, fault avoidance primarily encompasses the organisational aspects and tests linked to system development.

Fault avoidance methods are intended to reduce the overall probability of systematic failure to a level that the experts of Standard IEC 61508 deem equivalent to the probability of random hardware faults.

3.2. METHODS OF HARDWARE FAULTS AVOIDANCE

♦ ***Protection against environmental aggressions***

Environmental aggression is reputed to be a potential source of common mode failures in electronic equipment.

Certain authors do take these aggressions into account and propose a number of measures to limit their effects, for example avoiding the use of components such as line amplifiers which are vulnerable to external perturbations [BOUR], [EWIC].

♦ ***Other means***

- use the standards and a well-tried design;
- avoid unnecessary complexity or difficulties;
- correctly size components, memory capacities and processing times;
- employ components below their limits of use.

3.3. METHOD OF SOFTWARE FAULTS AVOIDANCE

Table 1, as well as the experiments (cf. chapters 1.3.3 et 2.2.3) concerning the correlation between the faults of various versions of the software, reveals, were it not already known, that software faults do exist and must be taken into account, this being true whatever the structure employed (homogeneous or heterogeneous).

These faults (single or common mode) are, in the majority of cases, due to the difficulties of the designers to control the development of increasingly complex software. They are, as a result, difficult to avoid and they require actions at each stage of the software life cycle. Experience demonstrates that common mode software faults cannot be distinguished from simple software faults [LYU]. There are, therefore, no specific avoidance methods associated with this category of fault.

A means of software faults avoidance is to draw up quality requirements for each of the lifecycle stages. The document “Software Quality and Safety Requirements” [QUAL] drawn up by INRS within the framework of the STSARCES project adapts the requirements laid down in the IEC 61508 Standard to the domain of machine safety:

- to set, with respect to the designer, requirements encompassing all the technical activities linked to software development, and to guide him or her in the activities to be undertaken to develop the software,

- to serve as a frame of reference to evaluate software under development, and to know the aptitude of a software product to satisfy the safety requirements applicable to the system being analysed.

These requirements concern: the software product, its development process, its verification and validation as these are in turn divided into sub-sections. Two requirement levels are to be distinguished (1 and 2) in relation to the criticity of the software in hand. The application of a requirement can be obligatory (O), Recommended (R) or left to the discretion of the designer.

Table 1 provides the objective of each topic along with examples of the faults that it can prevent. Due to hardware/ software interactions, it is to be noted that certain faults, which have been avoided, are not typical of the software, such as, for example, the interruption processing .

TOPIC AND OBJECTIVE OF THE REQUIREMENTS	EXAMPLE OF FAULTS AVOIDED
Software product Objective: *Set up activities, organisation, principles etc. as early as possible in the development cycle in order to obtain a software product which satisfies quality and safety requirements.*	Faults due to: – interfaces with the system architecture which integrate the software – the software specifications (limit cases, case of error, self tests, etc.) – the design of the software (erroneous algorithm) – the software coding, – the production of the executable (uncontrolled patch). – poor reuse of already existing software – erroneous parameter setting by the user
Software development process Objective: *to ensure that all the stages leading to production of the executable have been undertaken correctly.*	Faults due to: – poor management of versions – software regression subsequent to modifications – inadequate documentation that causes misunderstandings – poor synchronisation or incoherence between the different stages of the life cycle (design started before termination of the specification, coordination with the certified body, etc) – compiler and test facilities
Software verification and validation Objectives: *to demonstrate that the software products stemming from a phase of the development cycle conform both to the specifications established during the preceding phases and to the applicable rules or standards.* *To detect and take account of errors that may have been introduced over the course of the software development.*	Faults due to: – an incorrect interruption processing mechanism, – non respect of execution time requirements, – incorrect software response in transitory operation (switching on, input flow, switching to degraded operation, etc.), – resource access conflict or memory organisation problems, – inability of the integrated tests to detect breakdowns, – software/hardware interface errors, – stack overflow, – incorrect initialisation of variables and constants, – errors in parameter transfer , – deterioration of data, in particular global data, – inadequate numerical resolution from end to end – incorrect sequencing of events and operations. – incapacity of an algorithm to satisfy a software specification, – incorrect loop operations, – incorrect logic decision, – inability to process valid input data combinations correctly, – incorrect responses to missing or degraded input data, – violation of the limits of tables, – incorrect calculation sequence,

	– inadequate accuracy, correctness or performance of an algorithm.

Table 1: Faults avoided by safety and quality requirements

4. CORRELATED FAULT FORCASTING

4.1. GENERAL

Fault forecasting is one of the aspects that must be encompassed to construct the dependability of a system. Along with fault avoidance, it constitutes one of the components of fault avoidance. It consists in evaluating the behaviour of the system with respect to the appearance of faults, and is based on a set of methods and techniques to estimate **the presence, the creation,** and **the consequences** of faults

The forecasting of correlated faults at the origin of common mode failures will be dealt following the classification given in chapter 1.2 (existence of an initiating source and of a correlation between two sub-systems or components).

The **"fault presence and creation"** aspects of the forecasting techniques allow the determination of the initiators and correlation at the source of common mode failures, whereas the **"consequence"** aspect is employed to ascertain the exact nature of these failures.

Assessment of system behaviour in relation to the appearance of accidental faults, something which is necessary to fault forecasting, may be either qualitative or quantitative [LAPR]:

- ***Qualitative evaluation***

Qualitative evaluation is intended to identify and classify failures or the methods employed to avoid them. It should be considered in relation to the determinist requirements laid down in EN 954 [EN954], which requires the analyst to verify the behaviour of a system in the presence of predetermined faults. It raises the question of the definition of the faults to be taken into account for the evaluation, whether this is carried out on the system in question or on a model. This problem is more acute the more complex the components employed.

Indeed, although the faults arising on discrete or low integrated electronic components (short circuit, open circuit, stuck at 0 or 1) are known to a satisfactory degree, this is not the case for complex microprocessor type components, and even less so for software. In such case, if we want to carry out analyses at the level of the component, assumptions regarding the failure modes of these circuits have to be made. As such assumptions quickly reveal their limitations, the solution generally adopted is to go to a higher level and to consider the failures on a functional level.

Paragraph 4.3 describes several methods suited to the qualitative evaluation of the dependability of an electronic system with complex components.

- ***Quantitative evaluation***

The " determinist " evaluation provides an appreciation of the level of safety of a device employing electromecanic, discrete or low integrated components whose failure modes are known to a satisfactory degree. On the other hand, its capabilities are more limited when highly integrated components are employed, as it is then impossible to ascertain the failure modes of these components both thoroughly and with certainty.

Other approaches must be envisaged, like the quantitative evaluation recommended in Draft IEC 62061 Standard, which takes into account random hardware faults and, to a certain extent, common mode faults. Several results can be obtained by means of these calculations : reliability, availability, safety. Given the scope of this document, namely the safety of machinery, only the calculations necessary to determine the probability of dangerous failure are gone into.

4.2. ANALYSES PRIOR TO THE EVALUATION

The investigations to be undertaken prior to any evaluation, whether qualitative or quantitative, are common to any analysis, whatever the types of faults in question. The hazard and risk concept and analysis steps are taken from the IEC 61508 standard :

♦ ***Knowledge of the system***

This stage - still termed familiarisation by Mosleh [MOSL] or "concept" in the CEI 61508 standard - is not dedicated to fault forecasting. It consists in acquiring a general understanding of the system to be analysed, the functions that it must fulfil, and the physical environment in which it must evolve.

The different limit conditions will be examined in particular: the system's physical and functional limits, correlations and functional interfaces with other systems.

In the case of correlated faults, the elements of design, operation, maintenance and test procedures likely to increase the chances of multiple component failures are looked for in this phase.

♦ ***Hazard and risk analysis***

Firstly, system level hazards and the events causing them are identified. This analysis must be conducted for all reasonably foreseeable situations including failures and incorrect use. The risks associated with the dangerous events identified are then determined. This stage, carried out prior to any analysis, is closely related to the application in which the system will evolve.

Methods to assist these analyses are indicated in [1050] and [EN954]. They are based on a qualitative analysis that encompasses the seriousness of possible damage and the probability of such damage occurring: frequency and duration of exposure, probability of a dangerous event occurring, and possibility of avoiding or limiting this damage. In addition to these qualitative approaches, the CEI 61508 standard suggests two quantitative methods for risk determination.

♦ ***Preparation for modelisation***

The complexity of systems today means that modelisation is a necessary stage for evaluation. Preparation in this respect consists in gathering information about:

- the processes and technologies employed,
- the procedures and the frequency of the tests,
- the failure modes and rates,
- the coverage rates provided by the diagnostics,
- the maintenance and repair procedures.

In certain cases, the analyst can seek information on the following points to complete his or her knowledge:

- the type and the manufacturer of the components,
- the utilisation mode of the components,
- the internal and external (environmental) conditions of the components,
- the limits of use of the components and system interfaces,
- the location of components,
- the initial states of the components and their operating characteristics.

The information collected will serve as the basis to highlight certain correlation, and to identify the principle groups of common cause components [MOSH]. This preliminary identification may be achieved by seeking, for example by means of check lists, the attributes common to several components and the failure mechanisms likely to cause common mode failures (in fact, the weak points regarding common causes or potential correlation of the system). The following indications may be of assistance to the analyst :

- the identical, active, and functionally non diversified components used for redundancy should be considered as a common cause group ;
- diversified components that have identical redundant parts cannot be considered as independent.

Remark: The susceptibility of a group of common cause components does not depend solely on their degree of similarity. It also depends on the existence and the efficiency of the measures to protect against common modes.

For reasons of model complexity and the time required for the analyses, a selection could be made to suppress common cause events that have little influence on the result at system level, and to retain only what is significant. This selection can be based on a quantitative evaluation, for example by means of a fault tree.

4.3. QUALITATIVE EVALUATION

Two methods are generally employed to predict common mode faults [VILL].

4.3.1. Fault Tree Analysis (FTA)

- ***Overview of the method***

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.

The Fault Tree models the influences of the failure of one component or a series of components on a dangerous event. The elementary events are connected by AND and OR gates to view all the paths leading to the dangerous 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.

To avoid any error of interpretation and to ensure that the basic events have been correctly processed, the following must be verified :

– the dangerous events taken into account,

– the failure modes of components,

– the combinations of basic events,

– the paths leading to the failure,

– the common mode failures and systematic failures,

– the possible interactions.

The Fault Tree Analysis is currently in widespread use to deal with hardware aspects. It can also be applied to the evaluation of software faults, in particular to look for critical functions or modules.

- ***Determination of the initiators and correlation***

The Fault Tree Analysis allows modelling of the correlation between components [VILL]. The initiators, for their part, are obtained by extending the corresponding branches until the component degradation, the external perturbation or the human faults is reached.

- ***Determination the consequences***

The deductive character of the Fault Tree method means that it is not, a priori, a method to look for the consequences of faults, whether their origins be common cause or not.

4.3.2. Failure Mode and Effect Analysis (FMEA)

- ***Overview of the method***

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. When considering function failures the following approach is employed :

- <u>Determination of the failure modes of each of the functions considered for the analyses.</u> This is conducted after examining the specifications of these functions, more often than not distinguishing three classes : function not fulfilled, incorrect function, and result different from that expected. Certain authors [SERV] consider five failure modes without giving further details.

- <u>Determination of the local effects of failures</u>. This is generally limited to stating the values taken by the outputs in function of the elementary failure modes.

- <u>Determination of the global effects</u>. The local effects of failures are propagated to system level.

- Classification of global effects according to criticity. This classification is made using the results of the risk and hazard analysis carried out beforehand.
- Determination of dangerous failures. A criticity threshold is set, the failures producing effects above this threshold being considered dangerous.

FMEA is in widespread use for hardware systems. This technique can also be encountered for software analysis (Software Errors Effects Analysis) [THIR].

- ***Determination of the initiators and correlations***

FMEA is not a direct method of determining initiators and correlation. In order to contribute significantly to highlighting common mode failures, the FMEA must be used in association with other methods capable of determining the initiators and the correlations between components and sub-systems.

C. HOURTOLLE [HOUR], for example, suggests combining AMDE with Fault Trees to determine the critical software functions. Knowledge of these functions enables greater focus in investigations aiming to detect common mode software faults.

- ***Determination of the consequences***

Provided the correlated faults have been characterised correctly, FMEA is a method that can be employed to look for the consequences of these faults.

4.4. QUANTITATIVE EVALUATION

Introducing common modes into the quantitative evaluation depends on the knowledge the analyst has of them :

- The correlation have been clearly identified (environment, human errors). They can then be modelled, for example, in the fault tree of the system.
- The correlation have already been encountered on similar systems, a comparison is then carried out.
- The original causes of the phenomena are difficult to identify. They are then processed by parametric models relative to the common causes.

Paragraphs 4.4.1 and 4.4.2 look at the third case.

4.4.1. Modelling common mode faults due to hardware

The proposed models only take into account common modes produced by random hardware faults.

Among the different models that can be envisaged, the model in most widespread use, particularly for single redundancies, is the β factor [NUREG], [61508], [ISA], a single parameter model intended to represent hardware-related common causes. For more accurate analyses on high degrees of redundancy, multiple parameter models can be employed (multiple Greek letter, α factor and binomial failure rate). In the remainder of this section, both β factor and multiple Greek letter modelling are described.

b factor

This model corresponds to applications where it is possible to isolate one (or several) group(s) of common cause components. For a group of three components A, B, and C, the total failure rate λ of a component of this group is, strictly :

$$\lambda = \lambda i + 2.\lambda 2 + \lambda d$$

with :

- λi the rate of independent random failures,
- λ2 the failure rate due to the influence of components taken two by two (AB and AC are to be considered for component A), by adopting the hypothesis that the influences between components A, B and C taken two by two are identical,
- λd the failure rate due to three-component common causes (ABC is to be considered).

The β factor model assumes that all the components of a group of common cause components have failed, which leads to retaining only the failures affecting the three components, i.e. λd. Those affecting only two components are ignored, cancelling the corresponding λ2 factor. A fraction β of the component failure rate is associated with common cause events of the other components of the group, and the following is obtained :

$$\lambda i = (1 - \beta) \cdot \lambda$$

$$\lambda d = \beta \cdot \lambda$$

<u>Use of β for calculation of the failure rate associated with common cause faults.</u>

The IEC 61508 Standard describes a method of determining the failure rate associated with common cause faults that is adapted to calculations using Markov graphs. This rate takes account of the diagnostic capabilities offered by the microprocessors which leads to the breaking down of the preceding factor into two distinct parts: the first attached to undetected dangerous failures (β), the second to detected dangerous failures (β_D). The λ_{CMF} rate (or Dependent Failure Rate λd) is thus expressed by the following formula:

$$\lambda_{CMF} = \beta \bullet \lambda_{DU} + \beta_D \bullet \lambda_{DD}$$

Where :

λ_{DU} is the rate of undetected dangerous failures of a single channel,

λ_{DD} is the rate of detected dangerous failures of a single channel,

The factors β and β_D are determined empirically, primarily making use of field experience and taking into account:

- <u>the defence measures against the occurrence of common mode failures</u>, by distinguishing those whose contribution is improved by using diagnostic tests (X) from those whose contribution is not improved by these same tests (Y). A weighting Xi and Yi is assigned to each measure. These weightings are available in a table. The values de X and Y are given by:

$$X = \sum Xi \text{ , } Y = \sum Yi$$

- tests designed to detect the faults in a channel, both from the detection capability and test execution frequency points of views, which then gives :

$$\beta = f(X+Y) \text{ et } \beta_D = f((Z+1) \bullet X + Y)$$

The value of Z is given by a table. It depends on the coverage of the diagnostic facilities and on the time between two diagnostic tests. As soon as this time exceeds five minutes, the value of Z for the control logic is zero, which gives $\beta = \beta_D$. The value of Z is zero for the sensors and actuators as soon as the time between two tests goes beyond one week.

Remark : The application of the method proposed to determine β and β_D could be discussed. Indeed, it can lead to the following conclusion:

- For a homogeneous redundant structure executing a diagnostic test satisfactorily, i.e. a test executed at least once per minute with a coverage of 99%, the following is obtained :

$$\lambda^1_{CMF} = 0{,}02 \bullet \lambda_{DU} + 0{,}01 \bullet \lambda_{DD}$$

- For a diversified redundant structure executing a mediocre diagnostic test, i.e. a test executed less than once per minute with a coverage of 60%, the following is obtained :

$$\lambda^2_{CMF} = 0{,}02 \bullet \lambda_{DU} + 0{,}02 \bullet \lambda_{DD}$$

With λ_{DU} and λ_{DD} identical, such equality means that with respect to random hardware common mode faults, diversified redundancy can less efficient than homogeneous redundancy. In practice, such an observation would appear highly unlikely, in particular if all the components of one channel differ from those of another.

Multiple Greek letters for a defined series of components

This is an extension of β factor. It involves the possible influences of one component on the other components of the same common cause group. In this case, the multiplication factors of $\lambda 2$ are no longer zero.

For a triple redundancy structure, the parameters of the model are :

λ = probability of total failure due to independent and common cause events.

β = Conditional probability that the common cause of the failure of a component is at the origin of the failure of a second component.

γ = Conditional probability that the common cause of the failure of one components is at the origin of the failure of two other components.

We then have : $\lambda i = (1 - \beta) . \lambda$

$\lambda 2 = (1/2) . \beta . (1 - \gamma) . \lambda$

$\lambda d = \beta . \gamma . \lambda$

4.4.2. Modelling common mode faults due to software

Recent literature published by B. Littlewood [LITT] shows that, by contrast with random hardware faults, software faults are not subjected to modelisation with the same ease.

"… Clearly there remain large gaps in our understanding of some of the basic issues here. We know that we would like to have independence of the failure behaviour of the components in a redundant or diverse system, because this would allow us to carry out quite simple calculations to determine system reliability. If we cannot claim independence, then we need to estimate the degree of dependence achieved for the particular system under examination in order to compute its reliability. Unfortunately, means of using information about system design in order to estimate this dependence are very poor; and direct empirical evidence of common failure is, by its nature, extremely sparse. The poor general understanding is illustrated well in literature concerning software diversity for fault tolerance, where until recently, it was common to use words like 'independent' and 'diverse' quite loosely: for example, it was said that 'independent development' of 'diverse' versions was a mean to obtaining 'independent' failure behaviour in the versions. Before we can develop theories and models that will allow us to predict system reliability in the presence of common cause failures, we need to have a basic understanding of these fundamental concepts and their relationships. We need to answer questions such as: What is diversity? Are there designs more diverse than those? How diverse are these two designs what diversity can I expect by allowing designers complete freedom, but forbidding their communication with one another? …." [LITT]

This observation is confirmed by hypotheses concerning independence worked out in order to attempt the modelisation of the reliability of a multi-version structure [LYU]. By contrast with hardware faults, the techniques for the modelisation of common mode software faults are insufficiently mature to be introduced in the evaluation of the probability of a dangerous system failure.

4.4.3. Quantification using Markov graphs

Quantification by means of Markov graphs consists firstly in modelling the operation of the system in the form of a graph. Common mode failures are taken into account by adding the corresponding transitions to this graph. The probabilities of dangerous failures are then determined using matrix calculations from the failure rates determined by means of the formulae given in paragraph 4.4.1 [ISA], [61508], [DEACBr], [BOUS].

4.4.4. Quantification using Fault Tree Analysis

Fault trees are easily quantifiable if the basic events are independent. In such cases, it is possible to transform the tree into a Boolean expression, with as the rule:

Basic event $\rightarrow$ Boolean variable

AND gate $\rightarrow$ Boolean product of the Boolean variables of the input events

OR gate $\rightarrow$ Boolean sum of the Boolean variables of the input events

The basic event independence condition requires special treatment for common mode failures [ISA]. The following example illustrates how these failures are introduced:

Example:

Given an event E, resulting from the combination of (A1, A2), two dependent input modules and (B1, B2), two dependent output models but independent of (A1, A2):

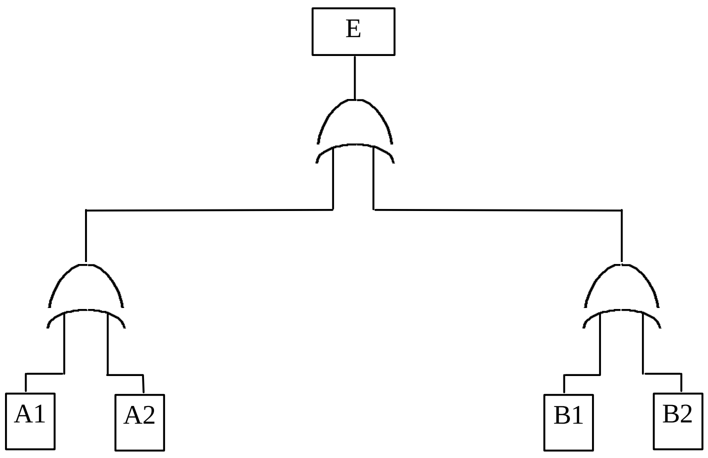

Figure 7: Fault Tree without CMF

The Boolean expression of E is:

$$E=A1+A2+B1+B2$$

which leads to the probability:

$$P(E)=P(A1+A2+B1+B2)$$

$$P(E) = P(A1) + P(A2) + P(B1) + P(B2)$$

$$- P(A1.A2) - P(A1.B1) - P(A1.B2) - P(A2.B1) - P(A2.B2) - P(B1.B2)$$

$$+ P(A1.A2.B1) + P(A1.A2.B2) + P(A1.B1.B2) + P(A2.B1.B2)$$

$$- P(A1.A2.B1.B2)$$

Given the independence of the input and output modules, the preceding expression can be simplified as:

$$P(E)=P(A1) + P(A2) + P(B1) + P(B2) - P(A1.A2) - P(B1.B2)$$

P(A1.A2) represents the part common to A1 and A2 and can be assimilated to a "common mode" event A_{CMF} (as is the case for P(B1.B2)).

In practice, it is therefore possible to rewrite the preceding equation in the form of a tree by introducing an event A_{CMF} which renders the events A1 and A2 independent (this is also true of the output module). The tree thus becomes:

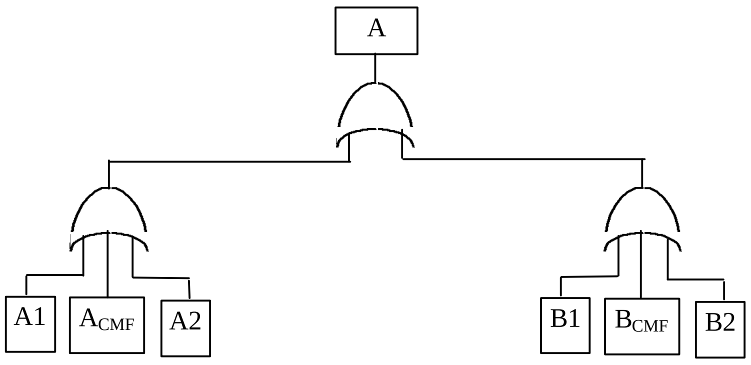

Figure 8: Fault Tree with CMF

Characterising the common mode failures between (A1, A2) and (B1, B2) by factors β_A and $\beta_{B,}$ the failure rate associated with the basic events of the tree will be:

$(1-\beta_A).\lambda_A$ & $(1-\beta_B).\lambda_B$ for A1 and A2 & for B1 and B2

$\beta_A.\lambda_A$ & $\beta_B.\lambda_B$ for A_{CMF} & B_{CMF}

4.5. OVERVIEW

As previously indicated, common mode fault forecasting is an estimation of the **presence and the creation** of these faults as well as of their **consequences** (the common mode failures).

Table 2 provides an overview of the main results relative to the forecasting of faults in order to give the capabilities of each method for the “ creation / presence ” aspects of faults and for their consequences.

The possibilities of quantitative and qualitative evaluation of common mode faults by means of Fault Trees would appear to make it a method well adapted to the forecasting of common mode faults. This capability should be confirmed by one or several applications on concrete cases.

EVALUATION METHOD ESTIMATED ASPECT	QUALITATIVE EVALUATION : FT, FMEA	QUANTITATIVE EVALUATION : FT, MARKOV
Presence and creation of CMF Must cover: • the initiators at the source of CMFs *Deterioration of electronic components, external perturbations, human faults introduced into the different stages of the life cycle of the product* And • the correlation at the source of CMFs *Correlation between elementary systems or between components*	Initiators at the source of CMFs Yes, whatever the method. All that is required is to extend the analysis until the component deterioration, the external perturbations or the human faults are reached The FT explicitly reveals the initiators whereas FMEA must start out from these initiators.	Initiators at the source of CMFs In every case, the evaluation of failure rates requires knowledge of the source initiators. FT: same possibilities as for use in qualitative evaluation. Markov: No direct possibility of highlighting the source initiators.
	Correlations at the source of CMFs Graphic representation by Fault Tree allows a rapid view of the principle correlations likely to produce CMFs	Correlations at the source of CMFs The correlations are introduced into the model of the system. They are therefore not revealed by a quantitative evaluation, but the calculations of the probability of dangerous failures take account of this.
Consequences of CMFs	The consequences of CMFs are clearly highlighted by the inductive methods (FMEA). The deductive character of Fault Trees means that on first approach, this technique is not intended to look for the consequences of CMFs.	Calculation of the probability of dangerous failures [1508]. Possibility of modifying the structure, the components, and the self-test period to improve the results.

Table 2: Overview of the methods for Common Mode Fault forecasting

5. CHECK-LISTS

The check-list organises the points to be verified during the different phases of the system life cycle, and thus ensures the arrangements made regarding the tolerance, avoidance and forecasting of common mode failures [HSE]. The questioning serves as a framework for the analyst or designer to take account of CMFs during these different activities. It is intended to highlight certain sensitive points, which should then be the subject of detailed analyses.

The aim of check-lists is to promote a critical analysis with respect to CMFs for all the following aspects :

- safety requirement specifications,
- specification, design, manufacture, test, maintenance and modification of the hardware,
- system test,
- operational,
- specification, design, coding, test, maintenance and modification of the software.

These various phases of the system life cycle are explored to inform the analyst on :

- a minimum design quality, design reviews, and quality control;
- the establishment of a structure to co-ordinate the maintenance, test, and utilisation activities;
- a verification, from the design stage onwards, of the reliability obtained.

The following convention is employed to complete the column relative to the results of the evaluation of each item ; the maximum score is obtained in the case of maximum diversity.

Example : For the question " Have the safety-related specifications been written in different forms ? ", the score would be 5 if the languages employed to draw up the specifications are totally different and 0 if they are identical.

Certain items in the following check lists have been taken from [HSE].

SYSTEM SPECIFICATIONS

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Have the safety specifications been written and developed by different people ?, If no, what are the links between these people? *The specification languages can be different..* *It is very difficult, if not impossible, to introduce real diversity at system specification level, links often being necessary between the teams responsible for drawing up the specifications.*	☐☐☐☐☐	
	Have the safety-related specifications been verified by different people ? If no, what are the links between these people? *This verification allows detection of specification errors (CMF and others). As the specifications generally constitute the common point of any development, even diverse, any fault at this level will be passed on to all the channels. The comparison systems being incapable of detecting such faults, it is important to keep their number to a minimum.*	☐☐☐☐☐	
	Have the initiating sources of common mode faults likely to affect the system been determined ? *CMF determination is vital, from the design stage onwards, to take these phenomena into account correctly.*	☐☐☐☐☐	

HARDWARE SPECIFICATIONS

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Have the hardware specifications been written and 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 hardware specification level, links often being necessary between two teams responsible for drawing up the specifications.*	☐☐☐☐☐	
	Have the common mode faults likely to affect the hardware been determined ? What	☐☐☐☐☐	

	are the potential sources of CMF at the hardware level? *CMF determination is vital to take these phenomena into account correctly.* *In particular, the possible influences of any external perturbations, systematic faults and possible common points are to be analysed.*		
	What is the level of human diversity at work? • different organisations involved in the design process (2 different companies), • different teams from the same company running the project, • different designers or programmers, • different testers, installers or certification personnel.	 ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	

HARDWARE DESIGN

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Have the different channels been developed by different people ? If no, what are the links between these people? *This diversity overcomes, in particular, design faults of human origin*	☐☐☐☐☐	
	What are the physical means introduced to avoid the physical propagation of faults: • physical separation? • electrical isolation of the different channels? • Separate power supply ? • use of electrical shielding? • attribution of emergency equipment to this function alone (no resource shared with other parts of the system)? • non-use of components common to several channels? *Due to the fact that they represent a common point, the links are potential sources of CMF as they are*	 ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	

	liable to propagate failures. *Links between channels are normally devoted to information exchanges and synchronisations: shared memories, serial or parallel buses.*		
	What are the means which have been used to avoid the logical propagation of faults: • physical isolation of software modules executed by two different microprocessors? • interactions through shared memories? • absence of bidirectional links between systems? • existence of protocols rendering safe the transfer of information through buses?	☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	
	Where diversity in the use of equipment is to be found, this is based on: • different suppliers for fundamentally different products? • the same suppliers for products that are fundamentally different? • different suppliers for similar products? • different versions of the same product?	☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	
	In the case where functional diversity exists, this is based on a difference : • in underlying mechanisms? • in functions, control logic or means of activation? • in response time ranges? *The most efficient functional diversity has recourse to different underlying mechanisms.*	☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	
	Where diversity in the signals is to be found, this is based on: • difference between the measured physical effects? • difference between the measured parameters with the use of the same principles?	☐☐☐☐☐ ☐☐☐☐☐	

	• redundancy of identical sensors?	☐☐☐☐☐	
	Have the channels been designed with: • different technologies ? *Digital / analogue, TTL or CMOS, etc..* • electronic components of differing architectures ? *RISC or CISC microprocessors , which require the use of different assemblers or compilers.* • electronic components of differing versions ? *Different components guard against certain design or utilisation faults.* • different printed circuits? • different bus structures?	☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐	
	Are there shielding s and protective devices to protect against electromagnetic perturbation ? *Shielding and protective devices form a barrier against common faults likely to be produced by external perturbation..*	☐☐☐☐☐	
	Do redundant channels have a common clock ? *Distinct clocks create a temporal diversity useful in avoiding CMF of external origin.*	☐☐☐☐☐	
	Are the different hardware resources tested periodically ? *Periodic tests avoid the accumulation of faults, and as a result have an effect on common mode fault avoidance.*	☐☐☐☐☐	
	Is there a power supply common to all the channels ?	☐☐☐☐☐	
	Have state of the art rules been followed during the design of the hardware ? *Respect of state of the art rules avoids the appearance of certain operational faults. These rules can be : employing standards and well-tried design, avoiding unnecessary complexity and difficulties, correctly sizing components, memory capacity and processing time, employing components within their limits of use.*	☐☐☐☐☐	

HARDWARE REALISATION

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Is there sufficient independence between the creation of the different channels ? *Sources of common faults due to hardware realisation are less numerous than for software.*	☐☐☐☐☐	
	Are the printed circuits of each channel different ? *A difference at component layout and signal routing level will improve immunity to common mode faults of external origin.*	☐☐☐☐☐	

HARDWARE TESTING

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Have the hardware tests been conducted by different people from those who specified and designed the hardware ? *Tests are part of fault avoidance (whether common mode or not).* *The people who specified and designed the hardware are not the best placed to detect faults during the test phase.* *As a minimum, the tests should be specified by people other than those who designed and specified the hardware.*	☐☐☐☐☐	
	Has there been sufficient independence between the tests of the different channels ? *Identical or similar tests can be inappropriate for hardware fault detection.*	☐☐☐☐☐	
	In the case of tests revealing a failure, have possible common mode related causes been looked for ?	☐☐☐☐☐	

SOFTWARE SPECIFICATIONS

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	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 ? Is 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.*	☐☐☐☐☐	

SOFTWARE DESIGN

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	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.*	☐☐☐☐☐	

SOFTWARE CODING

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	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 ?	☐☐☐☐☐	

SOFTWARE TESTING

N°	ITEM TO BE EVALUATED	RESULT	COMMENTS
	Have the testing specifications been developed by different people ? *Links are often necessary between two teams responsible for drawing up specifications.*	☐☐☐☐☐	
	Have the software tests been conducted by people different from those who specified and designed the software ?	☐☐☐☐☐	

6. CONCLUSION

The categorie 4 requirement of the EN 954 standard implicitly orientates the designer towards redundant structures, that poses the question of common mode failures, inherent to this concept.

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.

This document proposes actions to deal with these failures when constructing the dependability of a system (fault tolerance, fault avoidance and fault forecasting).

General measures, then measures specific to phenomena caused by common modes failures, are given for each of these topics.

A check-list organises the points to be checked at the different phases of the life cycle of a product, and hence to ensure the arrangements made regarding the tolerance, avoidance, and forecasting of common mode failures.

BIBLIOGRAPHIC REFERENCES

[EN292] **SECURITE DES MACHINES - Concepts de base, principes généraux de conception.**
Partie 1: Terminologie, méthodologie - CEN/TC114/SG N39,

[EN954] **SÉCURITÉ DES MACHINES - Parties des systèmes de commandes relatives à la sécurité**
Partie 1: Principe généraux de conception - Décembre 96

[61508] **SÛRETÉ FONCTIONNELLE : systèmes relatifs à la sûreté - Version 1998**
Partie 1 : Prescriptions générales
Partie 2 : Exigences pour les systèmes électriques / électroniques / électroniques programmables
Partie 3 : Prescriptions concernant les logiciels
Partie 4 : Définitions et abréviations
Partie 5 : Exemples de méthodes pour la détermination des niveaux d'intégrité de sécurité
Partie 6 : Guide pour l'application des parties 2 et 3
Partie 7 : Bibliographie des techniques et des mesures

[ISA] **SAFETY INSTRUMENTED SYSTEMS - SAFETY INTEGRITY LEVEL EVALUATION TECHNIQUES**
Part 3 : Determining the SIL of a SIS via Fault Tree Analysis
dTR84.0.02, March 98, Version 3

[BOUR] **DEFENCES AGAINST CMF IN REDUNDANCY SYSTEMS - A guide for management, designers and operators**
Bourne A.J., Edwards G.T., Hunns D.M., Poulter D.R. - Watson I.A.
UK Atomic Energy Authority Report SRD-R196 - 1981

[BOUS] **METHODES DE QUANTIFICATION DU NIVEAU DE SECURITE APPLIQUEES AU SAFELOC**
Bourges R.
Rapport de stage INRS. Juin 99. 88p

[BRIL] **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

[BUCH] **OCCURRENCE OF COMMON MODE FAILURE**
Buchner H.
Reliability engineering and system Safety, 1994, vol. 45, n°1-2, pp. 201, 204

[CHAR1] **CONCEPTION D’UN DISPOSITIF A MICROPROCESSEUR SÛR DE FONCTIONNEMENT : Règles et Méthodes**
Charpentier P.
Cahiers de Notes Documentaires, n°146, 1er trimestre 1992, pp 5, 14

[DEACBr] **QUANTIFICATION DE LA PROBABILITE DE DEFAILLANCE DANGEREUSE D’UN SYSTEME ELECTRONIQUE PAR GRAPHE DE MARKOV.**
Brandt C.
DEA ATNS. Rapport de synthèse. Sep 1998. 45 p.

[DOGB] **ETUDE DES ANOMALIES INTRODUITES PAR DES PERTURBATIONS ELECTROMAGNETIQUES PARVENANT SUR LES BUS D’INFORMATIONS RELIANT UN MICROPROCESSEUR A UNE MEMOIRE EXTERNE**
Dogbe K. E.
Thèse UST de LILLE, Avril 94

[ECK1] **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

[ECK2] **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

[EDWA] **A STUDY OF COMMON MODE FAILURES**
Edwards G.T., Watson I.A.
UK Atomic Energy Authority Report SRD-R146 - 1979

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

[HSE] **PES IN SAFETY RELATED APPLICATIONS GENERAL TECHNICAL GUIDELINES**
HEALTH AND SAFETY EXECUTIVE

[HOUR] **CONCEPTION DE LOGICIELS SURS DE FONCTIONNEMENT: ANALYSE DE LA SURETÉ DES LOGICIELS; MÉCANISMES DE DÉCISION POUR LA PROGRAMMATION EN N-VERSION**
Hourtolle C.
Thèse de docteur ingénieur, Toulouse, Octobre 87

[KNIG] **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, janv 1986, pp 96, 109

[LALA] **FAULT TOLERANCE IN EMBEDDED REAL-TIME SYSTEMS: IMPORTANCE AND TREATMENT OF COMMON MODE FAILURES**
Lala J.H., Harper R.E.
Congrès "Hardware and software architecures for fault-tolerance: Experience and perspectives, Juin 1994. Ed Springler-Verlag, pp. 263, 282

[LAP95] **GUIDE DE LA SURETE DE FONCTIONNEMENT**
Laprie J.C. et all
CEPADUES EDITIONS - 1995

[LITT] **THE IMPACT OF DIVERSITY UPON COMMON-MODE FAILURES**
Littlewood B.
Reliability Engineering and Sytem Safety, Vol. 51, N°1, 1996, pp 101, 113

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

[MDCI] **MODE DE DEFAILLANCE DES CIRCUITS INTEGRES - Constat des problèmes posés**
GROUPE DE TRAVAIL MDCI DE L'ISDF - 1994

[MILL] **DATA SUMMARIES OF LER - US NUCLEAR POWER PLANTS**
Millel C.F., Hubble W.H., Sams D.W., Moore W.E.
US Nuclear Regulatory Commission - EGG - EA - 5816 - 1982

[MOSL] **COMMON CAUSE FAILURES/ AN ANALYSIS METHODOLOGY AND EXAMPLES**
Mosleh A.
Reliability engineering and system safety _ 34 - 1991 - pp. 249-292

colloque international de fiabilité et de maintenabilité, St Malo, Octobre 1996, pp. 388, 396

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

[THIR] **MÉTHODOLOGIE D'ANALYSE DES EFFETS DES ERREURS LOGICIELLES (AEEL) APPLIQUÉE À L'ÉTUDE D'UN LOGICIEL DE HAUTE SÉCURITÉ**
Thireau Ph.
5ième colloque international de fiabilité et de maintenabilité, Biarritz, 1986, pp. 111, 117

[UCRL] **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

[VILL] **DÉFAILLANCES DÉPENDANTES ET DE CAUSE COMMUNE**
Villemeur A.
Sûreté de fonctionnement des systèmes industriels
Ed. Eyrolles, 1988, pp. 371, 410.