

Environment Project FIS5-1999-00355

CEMSIS:  
Cost Effective Modernisation of Systems Important to Safety

**Assessment and analysis guidelines  
for Off-The-Shelf Product-based Systems Important for Safety**

v04

## Authors

In alphabetical order: J.-B. Chabannes, M. Neyret, N. Richer and N. Thuy (EDF).

## Revisions

v1	Draft for comment	
v2	Draft after EDF and Adelard comments	
v3	Draft after FANP comments	
v4	Final version	

## Circulation

Public

## Contents

1.	Introduction.....	5
1.1.	Context.....	5
1.2.	Document Framework .....	5
2.	References.....	7
3.	Glossary and abbreviations .....	8
3.1.	Glossary.....	8
3.2.	Abbreviations.....	9
4.	Off-the-shelf products.....	10
5.	Properties essential to safety .....	11
5.1.	Characterisation .....	11
5.1.1.	Identification.....	11
5.1.2.	Description.....	11
5.1.3.	Integrity.....	12
5.2.	Functional adequacy to real safety needs.....	12
5.3.	Correctness.....	13
5.4.	Robustness .....	13
5.5.	Maintenance of adequacy to safety .....	14
6.	Demonstrations that safety properties are satisfied.....	15
6.1.	Types of demonstration.....	15
6.1.1.	Systematic proof .....	15
6.1.2.	Demonstration by sampling .....	15
6.1.3.	Demonstration by inspection .....	15
6.1.4.	Demonstration by development process .....	15
6.2.	Credibility of a demonstration .....	16
6.2.1.	Principles of the demonstration .....	16
6.2.2.	Objects on which the demonstration is performed.....	17
6.2.3.	Tools supporting the demonstration.....	18
6.2.4.	Implementation of the demonstration .....	18
6.3.	Certification .....	18
6.4.	Example .....	18
6.4.1.	Principles .....	19
6.4.2.	Objects and environment .....	19
6.4.3.	Tools .....	20
6.4.4.	Implementation .....	20
7.	Optimised Safety justification strategy for OTSP-based SIS .....	21
7.1.	Pre-qualification of OTS product.....	21
7.1.1.	Functional assessment.....	21
7.1.2.	Dependability assessment .....	21
7.2.	Safety justification of the SIS .....	21
7.3.	The functional assessments in renovation projects .....	22
8.	Pre-qualification: Functional assessment of OTS product.....	24
8.1.	Taxonomy .....	24
8.2.	Task 1: Functional modelling .....	24
8.2.1.	Overall approach.....	24
8.2.2.	Customisation of the generic functional model .....	24
8.2.3.	Investigation objectives and principles .....	25
8.3.	Task 2: Functional description of OTS products .....	25
8.3.1.	Customisation of investigation principles.....	25
8.3.2.	Analysis of information and documentation.....	25
8.3.3.	Complementary investigations.....	26
8.3.4.	Reporting of investigations .....	26
9.	Pre-qualification: Dependability assessment of OTS products.....	27
9.1.	Taxonomy for dependability assessment .....	27

9.1.1.	Key characteristics .....	27
9.1.2.	Safety class.....	27
9.1.3.	Taxonomy and strategies for dependability assessments.....	28
9.2.	Properties of OTS products.....	30
9.2.1.	Properties contributing to the characterisation of the SIS.....	30
9.2.2.	Properties contributing to the correctness of the SIS .....	30
9.2.3.	Properties contributing to the robustness of the SIS .....	30
9.2.4.	Approaches for the dependability assessment of OTS products .....	31
9.3.	Use conditions.....	31
10.	Safety justification: OTS product matching with user requirements .....	32
10.1.	Task 3: User requirements regarding OTS products.....	32
10.1.1.	Selection of candidate architectural concepts .....	33
10.1.2.	Specification of product requirements .....	34
10.1.3.	Classification of product requirements into investigation groups.....	34
10.2.	Task 4: Matching OTS products with corresponding user requirements.....	34
10.2.1.	Comparison of product documentation and user requirements.....	34
10.2.2.	Complementary investigation .....	35
10.2.3.	Feedback of supplier .....	35
10.2.4.	Effective selection of products.....	35
11.	Safety justification: Causes and possible defences against Common Cause Failures .....	36
12.	Safety justification: Notion of OTS product criticality.....	37
12.1.	Criticality C2.....	37
12.2.	Criticality C1.....	37
12.3.	Criticality C0.....	38
12.4.	General strategy of criticality justification.....	38
13.	Conclusion .....	40
ANNEX I	: Approaches for the dependability assessment of OTS product .....	41
I.1	High complexity or medium complexity component in A1 approach, e.g. I&C platform .....	41
I.2	High complexity component in B1 approach, e.g. I&C platform.....	45
I.3	Medium-complexity component in B1 approach e.g. communication equipment .....	48
I.4	Medium-complexity component in B2 approach e.g. communication equipment .....	52
I.5	Simple devices in A2 or B2 approach.....	54
ANNEX II	: .....	56
II.1	Error Propagation.....	56
II.1.1	Propagation of incorrect data .....	56
II.1.2	Incorrect synchronisation in data exchanges .....	57
II.1.3	Aggression by another software.....	58
II.1.4	Denial of common resources .....	59
II.2	Common modules .....	60
II.2.1	Failure in common modules.....	60
II.2.2	Recommendation .....	60
II.3	Similarities in development processes .....	61
II.3.1	Incorrect common specifications .....	61
II.3.2	Error in common design & implementation methods.....	61
II.3.3	Fault in identical software modules .....	61
II.3.4	Fault in common development methods .....	62
II.3.5	Mistakes repeated by common staffing and management .....	63
II.3.6	Overall Recommendations.....	63
II.4	Exceptional conditions.....	64
II.4.1	Sensitivity to plant demands .....	64
II.4.2	Sensitivity to date.....	64
II.4.3	Recommendation .....	64
II.5	Maintenance activities .....	65

# 1. Introduction

## 1.1. Context

There are many nuclear power installations within Europe which require maintenance and modernisation. These installations contain many I&C systems that are regarded as “Systems Important to Safety” (SIS). In that context, CEMSIS aims at developing methods for replacing and modernising SIS. In the past, SIS were especially design for the nuclear industry and were often implemented using simple analogue, relay or discrete logic technologies that were relatively easy to analyse. Nowadays, SIS are becoming heavily reliant on computer-based systems and this notably means that replacement I&C systems will probably be based on Off-The-Shelf products (OTSP) that are not specifically design for the nuclear industry. In this context, the CEMSIS project is organised on three work axes:

- WP1 of CEMSIS proposes a framework for the safety justification of Off-The-Shelf-Product-based Systems Important to Safety (SIS). This framework is based on a claim-argumentation-evidence approach, where each claim made regarding the adequacy of the SIS to safety is justified by a structured argumentation. An argumentation is composed either of a set of evidences that are taken for granted and agreed upon by all parties involved, or of a set of sub-claims, each having its own argumentation.
- WP2 of CEMSIS proposes a process for the capture of requirements for the renovation of a SIS. One of the main objectives of this process is to ensure that the specified requirements represent completely and correctly the real safety needs that the SIS must satisfy.
- WP3 develops cost effective approaches and techniques to demonstrate in an efficient, valid and convincing manner that a given system containing OTS products is suitable for supporting functions important to safety.

This document is the public deliverable D3.4 of the CEMSIS project that synthesises the results of the WP3 works.

## 1.2. Document Framework

After some considerations on the term “OTS product” in chapter 4, WP3 proposes in chapter 5 a new approach for the safety justification based on the demonstration that properties essential to safety are satisfied by SIS. The current practice to justify the safety of SIS is to assess the compliance of a given system to the defined rules. One issue of this practice is that it does not consider the particular features of SIS, so that SIS are evaluated in a similar way without taking into account their specific features. This practice may then lead to some unnecessary efforts and cost overruns. Moreover, does a given system’s compliance to generic rules strictly demonstrate safety? To address these issues, chapter 5 identifies the properties that are essential to safety for OTSP-based SIS and chapter 6 compares the strengths and weaknesses of the possible types of demonstration to provide evidence that those properties are satisfied.

As a given OTS product is likely to be used in several SIS, Chapter 7 proposes to factorise, whenever possible, the justification effort regarding OTS products by decomposing the safety justification of the OTSP-based SIS as follow:

- The pre-qualification of OTS products that gathers:
  - The functional assessment of OTS product (chapter 8);
  - The dependability assessment of OTS product (chapter 9);
- The overall safety justification of the SIS.

Chapter 8 develops a functional taxonomy of OTS products and provides a method to guarantee a functional description level of OTS products suitable for a correct OTS products selection.

Chapter 9 develops taxonomy and strategies for the dependability assessment of OTS products. It identifies the safety properties of OTS products that need to be claimed and justified during pre-qualification

(independently of specific renovation project) so that they can contribute later to the overall safety justification of the SIS.

Regarding the safety justification of SIS, the present document does not intend to present a full process but to focus on the OTS product concerns of the safety justification. Consequently, it presents the following aspects of the safety justification:

- OTS product matching with user requirements (see chapter 10);
- The causes and the possible defences against Common Cause Failures (see chapter 11).
- The notion of OTS product criticality (see chapter 12);

The two first points are presented in the form of engineering process for OTS product matching with user requirements and in the form of possible practices for the defences against the potential causes of CCF. The scope of the safety justification includes notably the assessment of these engineering processes and practices.

The last point introduces the notion the criticality of OTS products in the SIS. This notion may be notably worthwhile to assess the overall dependability of SIS when the embedded OTS products do not intrinsically have dependability properties.

## 2. References

- [1] IEC 60880: International Electrotechnical Commission – Software for Computers in the Safety Systems of Nuclear Power Stations (Edition from 1986, presently in revision)
- [2] IEC 61513: International Electrotechnical Commission – Nuclear Power Plants – Instrumentation and control systems important to safety – General requirements for systems (Edition from 1993, presently in revision)
- [3] IEC 61226: International Electrotechnical Commission – Nuclear Power Plants – Instrumentation and control systems important for safety – Classification
- [4] IAEA Safety Guide n° 50-SG-D3: International Atomic Energy Agency – Protection system and related features in nuclear power plants (Subsided by Safety Guide NS-G-1.3)
- [5] IEC 61069: International Electrotechnical Commission – Industrial process measurement and control – Evaluation of system properties for the purpose of system assessment
- [6] IEC 60880-2: International Electrotechnical Commission – Software for Computers in the Safety Systems of Nuclear Power Stations – Software aspects of Defence against Common Cause Failures, Use of Software Tools and of Pre-existing Software (Edition from 2001 which will to be integrated into IEC 60880 during current revision)
- [7] IEC 62340: Nuclear power plants – Instrumentation and control for systems important to safety – Common Cause Failure (CCF) (1st CD circulated 12/2003)
- [8] IAEA Safety Glossary, Terminology used in Nuclear, Radiation, Radioactive Waste and Transport Safety, Version 1, April 2000.

## 3. Glossary and abbreviations

### 3.1. Glossary

<b>Adequacy</b>	Property of an item to be adapted to a use or purpose.
<b>Black Box</b>	Assessment approach where some information regarding internal design, implementation and development is available and accessible but the source code is not available or accessible.
<b>Clarity</b>	Property of an item to be easily understood by informed observers.
<b>Common Cause Failure</b>	Failure of a number of devices or components to perform their functions as a result of a single specific event or cause. <i>According to [5] definition 393-08-19.</i>
<b>Completeness</b>	Property of an item that has all necessary parts, elements, or steps.
<b>Consistency</b>	Absence of contradiction.
<b>(Level of...) Criticality C0</b>	The errors in the OTS product cannot lead to an unsafe situation.
<b>(Level of...) Criticality C1</b>	An error in the OTS product can lead to an unsafe situation only if another error occurs in some other part of the SIS.
<b>(Level of...) Criticality C2</b>	An error in the OTS product may lead to an unsafe situation.
<b>Currency</b>	Property of an item to be up to date.
<b>Error</b>	Discrepancy between a computed, observed or measured value or condition and the true, specified or theoretical value or conditions.
<b>Failure</b>	Failure occurs when the delivered service deviates from the intended service.
<b>Grey Box</b>	Assessment approach where detailed information regarding internal design, implementation and development is available and accessible and some bits of the source code are available and accessible.
<b>Identification</b>	Property that guarantees that all specimens with the same identification are identical.
<b>Integrity</b>	Condition of an item being unimpaired.
<b>Maintainability</b>	Ability of an item to restore to nominal conditions after failure or decline.
<b>Modifiability</b>	Ability to perform software modifications in safe conditions during the full SIS lifecycle.
<b>Modularity</b>	Degree to which an item is composed of discrete components such that a change to one component has minimal impact on other components (adapted from IEEE 610.12.1990).
<b>Perenniality</b>	Degree to which modifications and upgrades can still be performed on an item, even after a very long period.
<b>Precision</b>	Degree of exactness with which an information is given, so as to preclude diverging interpretations.
<b>OTS Product</b>	Product that already exists and is available as a commercial or proprietary product and is being considered for use in a computer-based system.
<b>Simplicity</b>	Degree to which an item is straightforward and easy to understand, either in its functionality or in its design and implementation.
<b>White box</b>	Assessment approach where the source code and detailed information regarding internal design, implementation and development are available and accessible.

### **3.2. Abbreviations**

<b>CCF</b>	Common Cause Failure
<b>COTS</b>	Commercial Off-The-Shelf
<b>EUC</b>	Equipment Under Control
<b>HMI</b>	Human Machine Interface
<b>I&amp;C</b>	Instrumentation & Control
<b>NPP</b>	Nuclear Power Plant
<b>OTS</b>	Off-The-Shelf
<b>OTSP</b>	Off-The-Shelf Product
<b>PLC</b>	Programmable Logic Controller
<b>SIS</b>	System Important to Safety
<b>WP</b>	Work Package

## 4. Off-the-shelf products

There are many definitions and different understandings for term “COTS” (Commercial Off The Shelf). In order to avoid unnecessary controversy, for the purpose of CEMSI and notably of WP3, it is proposed to use instead term “Off The Shelf Product” (OTSP).

IEC 60880-2 [1] defines “pre-existing software” as “*software, which already exists, is available as commercial or proprietary product and is being considered for use in a computer-based system*”. Similarly, “Off The Shelf Product” could be defined as “product which already exists, is available as commercial or proprietary product and is being considered for use in a computer-based system”. Pre-existing software is thus a particular kind of off the shelf product.

An OTS product can be used in several systems and is not tailored to the specific needs of one particular system. In general, it may benefit from experience in operation, and its supplier may, or may not be willing to contribute to the safety qualification process.

The notion of OTS product covers a very wide range. E.g., an OTS product may be:

- An equipment family, i.e., as defined in IEC 61513 [2], a set of hardware and software components that may work co-operatively in one or more defined architectures and to perform project specific functions;
- A dedicated device such as a sensor;
- A software component.

## 5. Properties essential to safety

WP3 proposes to base the safety justification on the demonstration that properties essential to safety are satisfied at the SIS level. This chapter identifies the properties that are essential to safety for OTSP-based SIS.

The top-level safety property that needs to be justified for the SIS is the **adequacy of the SIS over its lifetime to satisfy the real safety needs**. This main property may be decomposed into the following first level properties:

- The characterisation of the SIS (including its description);
- The functional adequacy of the SIS to the real safety needs;
- The correctness of the SIS with respect to its description;
- The robustness of the SIS with respect to postulated internal and external non-nominal conditions;
- The maintenance of the adequacy to safety during the lifetime of the SIS.

### 5.1. Characterisation

The overall objective of characterisation is to know with precision the system we are dealing with. For this purpose, the characterisation of a SIS is divided into three sub-properties:

- The identification of the SIS and of its main sub-systems and components;
- The description of the SIS;
- The integrity of the SIS.

#### 5.1.1. Identification

The overall objective of the identification of an item is to put a unique identity that distinguishes this item without any ambiguity. When software aspects are concerned, uniqueness means that two items with the same identification are identical under all aspects.

The identity of an item needs to be known with precision. This implies the knowledge of information such as name, version and status (e.g., approved or not). This may also imply the knowledge of the identification of its main sub-items and / or components.

#### 5.1.2. Description

The overall objective of the description of an item is to give an accurate and systematic description of all what needs to be known to assess the item and to select and / or use it in safe conditions. For a SIS, this includes a functional description and a design description.

The functional description usually gives all necessary information regarding:

- The modes of behaviour (e.g., initialisation mode, nominal modes, down-graded modes, failure modes) and the transition conditions between modes;
- The functions provided in the identified modes of behaviour, and, possibly, what is to be guaranteed that the SIS does not do;
- The interfaces with other systems and equipment and with human beings, covering static and dynamic aspects;
- The configuration parameters, if any;
- The expectations and constraints regarding the conditions (notably the worst case conditions) that will be provided to the SIS by the environment, such as resources to be provided or maximum input loads;
- The performances, such as response times and accuracy, possibly depending on configuration, mode of behaviour and conditions provided;

- The dependability related information such as reliability, self-surveillance and management of failures.

The design description may be used in the assessment of the ability of the SIS to meet given constraints for safety (e.g., regulatory requirements). E.g., it may give information regarding:

- The overall organisation of the SIS and of its software, including information regarding redundancy and diversification, sub-systems and software units, components, internal interfaces and separations, etc. ;
- The overall internal functioning of the SIS and of its software, including information on algorithms, internal events, timing, synchronisation, resource management, error detection and signalling, etc.;
- The implementation, e.g., coding standards and rules, source code.

The description of a SIS may be based on part on the development information provided with the SIS (e.g., the SIS Requirements Specification and the SIS Design Specification). As this may sometimes be insufficient, complementary analyses and measurements on the real SIS may be necessary during the safety justification.

Some properties may be expected of the description of a SIS:

- Completeness with respect to the features of the SIS that are significant to safety;
- Correctness with respect to the real features of the SIS; this property is the same as the correctness of the SIS with respect to its description; and is developed in more detail in section 5.3;
- Accuracy, so as to reduce the uncertainty margins between the real features of the SIS and the corresponding description to values acceptable for the safety justification;
- Precision, so as to preclude any diverging interpretations by intended readers.

### **5.1.3. Integrity**

The integrity of a SIS is the property by which this SIS conforms to the model on which the safety justification is based. Whereas hardware may allow manufacturing margins, software usually does not: software integrity is always absolute.

## **5.2. Functional adequacy to real safety needs**

The SIS to be renovated must often fit in a complex environment where it interacts with other systems and equipment, with operators and other personnel, and with the physical process under control. All these may have very different states (nominal or failure states, start up, shut down, maintenance or normal operation) with possibly different behaviours and expectations with respect to the SIS. Thus, the adequacy of the SIS with respect to the real safety needs (which include functional and dependability needs) is often an important, difficult and critical issue. It is usually necessary to justify that:

- All the real safety needs have been identified and taken into account (completeness);
- All these identified needs have been correctly addressed (correctness).

In addition, and notably for systems supporting category A functions (see [3]), it is also necessary to justify that the SIS does not include functions that are not necessary to safety and that may have an adverse affect on safety (focus on safety).

WP2 proposes a process for the capture of the requirement for the renovated SIS. The present document complements this process with recommendations for the justification of the adequacy of what has been captured and implemented (see chapter 10).

### 5.3. Correctness

Correctness ensures that the SIS is and behaves as specified in its description. The correctness of the software of a SIS may be decomposed into a number of direct sub-properties:

- Freedom from intrinsic faults; intrinsic faults are faults that may be recognised as such independently of any requirements specification, e.g., division by zero, use of out of bound array indexes, use of illegal pointers, use of non-initialised variables, violations of assertions, permanent loss of resources, insufficient resources, dead locks, non-deterministic behaviour, non-controlled access to shared variables, etc.;
- Freedom from functional faults; functional faults are faults that may lead to inconsistencies with respect to specified requirements, e.g., incorrect or insufficiently accurate output values, incorrect timing or sequencing of outputs, etc.;
- Freedom from hardware / software incompatibilities; these faults result from discrepancies of one with respect to the other; e.g., insufficient memory (or too voluminous software), discrepancies in hardware-related operations (input – outputs, management of interrupts, memory management, etc.);
- Conformance to design & implementation documentation;
- Freedom from hidden functions or assurance that sub-functions which are not used for a specific SIS application can be blocked in a safe way.

The preceding sub-properties are the necessary components of software correctness. Others are more indirect, but they are often a good practice and facilitate the safety justification. Examples of indirect sub-properties are:

- Simplicity, clarity and modularity of architecture, i.e., clarity of the internal organisation;
- Simplicity, clarity and modularity of internal functioning, e.g., of algorithms, data flows, internal states;
- Quality of documentation,; e.g., clarity, completeness, consistency, pertinence, precision, etc.;
- Conformance to appropriate standards and rules;
- Testability of specifications, design and implementation.

### 5.4. Robustness

The overall objective of robustness is to guarantee a predictable behaviour of the SIS even in non-nominal conditions, so that safety is not jeopardised. The justification of the robustness of a SIS should identify and consider the different types of non-nominal conditions that may occur and affect the software, notably:

- Erroneous conditions in the software, such as erroneous data (software variables or software configuration parameters), dead locks, lack of resources (e.g., of memory or of operating system resources), erroneous timing or sequencing;
- Erroneous conditions in the rest of the SIS, e.g., in the hardware or in internal communications, protection against SIS internal failure propagation;
- Erroneous conditions in the environment of the SIS, e.g., partly incorrect input data;
- Specific considerations are required with respect to: excessive request rates, absence of inputs, extreme duration of continuous operation or unavailability of required external resources.

Whereas correctness is (or should be) an absolute objective, robustness is usually only a relative objective, as increased robustness may require increased complexity, and thus, increased risks of non-correctness and failure. Thus, robustness with respect to a given type of condition may or may not be claimed. If it is claimed, the justification may be based on intrinsic robustness or active robustness.

A SIS is intrinsically robust with respect to a given type of condition when, by design, this condition cannot possibly affect the SIS, or when the SIS does not need to perform any specific action to remain in, or return to, a nominal or safe mode of behaviour. E.g., a system that is not event-driven and that cyclically polls its inputs cannot be affected by input overload. A cyclic memory-less SIS will return to a nominal mode after a transient hardware fault.

Active robustness requires specific actions of the SIS and / or of its software. These actions are typically:

- The detection of non-nominal conditions;
- The reporting of non-nominal conditions, so as to allow appropriate and timely action by external means;
- The containment of the effects of non-nominal conditions, so as to avoid error propagation and to avoid them leading to system failure;
- Graceful degradation, from maintenance of nominal service to degradation to identified downgraded or failure modes;
- The safe restoration of the nominal service: after a failure, safe restoration includes the correct cleaning up of any corrupted data.

## **5.5. Maintenance of adequacy to safety**

The four first main safety properties (characterisation, adequacy to real needs, correctness and robustness) are usually justified during or at the end of the system development process. The fifth and last main safety property needs to be justified over the operation of the SIS. Though it is commented in this section, it is usually not addressed in the initial safety justification of a SIS.

As far as software is concerned, it may be decomposed into three sub-properties:

- The correctness of the software parameters during operation;
- The testability of the SIS during operation;
- The modifiability of the SIS and of its software.

The SIS and its software often need to be adapted to specific site conditions, either because they were designed and implemented so as to be used at different sites (e.g., in the case of a series of nearly identical plants), or because these conditions evolve in time (e.g., with the ageing of the fuel). Adaptations performed by changing the values of identified software parameters are usually not considered as SIS or software modifications and do not require an engineering-based justification, provided the updated values are within the range taken into consideration in the safety justification.

Some other adaptations cannot be performed by changing parameter values and require modifications of the SIS and / or of its software. Software modifications may also be needed to correct faults found in the specification or in the implementation. The key to the maintenance of the adequacy of the SIS to safety is then the ability to implement the software modifications that are necessary to safety during the complete lifetime of the SIS without diminishing the ability of the SIS to perform the required safety functions. Software modifiability may be divided in the following sub-properties:

- The longevity of the supporting infrastructure, i.e., the ability to use, during the required period of time, all the engineering services that are necessary to specify, implement, verify and validate modifications in the software and in the architecture of the SIS (e.g., the system and software engineering tools, and the development environments in which the tools can be operated); the processed software shall be identified unequivocally before and after any modification;
- The longevity of staff competency, i.e., the ability to have, during the required period of time, all the human competences that are necessary to specify, implement, verify and validate modifications in the software and in the architecture of the SIS;
- The longevity and accessibility of information, i.e., the ability to access, during the required period of time, all the information that is necessary to specify, implement, verify and validate modifications in the software and in the architecture of the SIS; in particular, this means that information exists, that its storage format is still readable, and that it can be retrieved even when immersed in vast amounts of irrelevant information.

Simplicity and modularity also contribute greatly to software modifiability.

## 6. Demonstrations that safety properties are satisfied

### 6.1. Types of demonstration

The framework for safety justification proposed by WP1 is based on a claim-argumentation-evidence approach, where each claim made regarding the dependability of the SIS is supported by a structured argumentation. An argumentation is composed either of a set of evidences which are taken for granted and agreed upon by all parties involved, or of a set of sub-claims, each having its own argumentation. Demonstration adopted to provide evidence may be categorised into four different types:

- Systematic proof;
- Demonstration by sampling;
- Demonstration by inspection by human experts;
- Demonstration by development process.

#### 6.1.1. Systematic proof

A systematic proof is based on rigorous principles and formal reasoning that provide an objective evidence that a claim is satisfied in all considered situations. It may be built in a priori (e.g., by the application of design rules precluding the item from violating the claim), or it may be achieved a posteriori (e.g., by a formal analysis of the item at the end of the development process).

Ideally, a claim should be supported by an objective and rigorous argumentation. Thus, systematic proofs should be the preferred approach for safety justification. However, while some claims can actually be supported by a systematic proof, others cannot be formally demonstrated in the current state of the technology and must be supported by the other types of demonstration. In addition, not all systematic proofs are absolutely airtight (see §6.2), and most need complementary demonstrations that are not necessarily systematic.

#### 6.1.2. Demonstration by sampling

A demonstration by sampling may provide a systematic evidence, but only on a finite and limited set of discrete samples. Demonstrations by sampling include tests, measurements, simulations, and experience in operation. The satisfaction of the claim on the full range of situations may be extrapolated if the set of samples is accepted as representative of the full range of situations. However, this extrapolation cannot usually be based on rigorous and objective reasoning, and prevents demonstrations by sampling from being completely objective.

#### 6.1.3. Demonstration by inspection

A demonstration by inspection is based on the capability of human experts to correctly assess the satisfaction of a given claim. Different types of inspection may be considered, e.g., informal re-reading, guided re-reading, critical design review and walkthrough. Inspection may be exhaustive (when applied to all documents concerned) or partial (when applied to a subset of the documents concerned). Although it may provide some level of confidence on aspects which are difficult to formalise and/or to prove formally, it is often highly subjective and difficult to repeat many times.

#### 6.1.4. Demonstration by development process

A demonstration by development process is indirect: it does not deal with the item on which the claim is made, but with development processes and lifecycles, quality assurance and with methods, rules and tools. Such types of evidence give confidence that the item and its future upgrades are produced in a coherent and controlled manner. As such, it gives confidence that the other results and measurements, performed as part of the justification, are valid. This type of demonstration is usually readily available and may constitute the first step of an assessment.

## 6.2. Credibility of a demonstration

Many claims made regarding the essential safety properties of the SIS, of its software or of an OTS product will be justified by demonstrations the credibility of which may also need to be justified. To this end, one should consider:

- The principles of the demonstration;
- The object on which the demonstration is performed, and the environment that is taken (or not taken) into account;
- The tools supporting the demonstration, if any;
- The implementation of the demonstration.

### 6.2.1. Principles of the demonstration

There should be an early agreement between the licensor and the licensee that the principles on which a demonstration is based are relevant to the claim made and applicable to the object of the claim. To this end, the principles of the demonstration and the assumptions made will usually need to be documented. The agreement may be formalised in the argumentation of the claim either as a hypothesis (which is not further justified) or by some form of justification.

#### 6.2.1.1. Systematic proof

The justification of the relevance of the principles of a systematic proof usually addresses the following topics:

- The soundness of the theoretical basis with respect to the claim and to the characteristics of the object on which the claim is made;
- The identification of the hypotheses and approximations;
- When necessary, the justification of the acceptability of these hypotheses and approximations.

#### 6.2.1.2. Demonstration by sampling

One difficulty with demonstrations by sampling is the justification that the set of samples allows a generalisation to the complete range of situations. Two types of demonstration by sampling may be considered separately: testing and experience in operation.

Regarding testing, different types of justification may be considered, either in isolation or in combination, notably:

- Justifications based on coverage criteria and levels of coverage; many coverage criteria have been proposed for software testing; functional coverage criteria are based on the specification, whereas structural criteria are based on the design and implementation;
- Justifications based on a statistical reasoning; this supposes that a probabilistic target is specified for the object and the claim; this also supposes the acceptance or the justification of the correctness of the underlying probabilistic reasoning, and of appropriateness of the probabilistic distribution of the samples with respect to the probabilistic distribution of the real operational events and conditions;
- Justifications based on fault injection, where the ability or inability of the fault detection programme to detect known injected faults is used to estimate the number of unknown residual faults; this supposes notably that the representativity of the faults injected is justified.

Regarding experience in operation, the volume, that is an argument on the weight of the demonstration, may be justified on statistical reasoning. The methods used for collecting information, condition of use during the experience and the significant facts (e.g., failures) need to be documented. Credibility of methods used for collecting experience must be justified so that the experience in operation strictly corresponds to reality (e.g., correctness and completeness in the detection and reporting of the significant facts). Justification must be provided that conditions of use in experience in operation covers the real conditions of use in the plant.

### **6.2.1.3. Demonstration by inspection**

The credibility of a demonstration by inspection relies essentially on human aspects such as:

- The soundness of the guidance and directives that are provided, with respect to the claim and to the characteristics of the object on which the claim is made;
- The competence of the inspectors with respect to the claim made;
- Their knowledge of the specifics of the project concerning the object on which the claim is made;
- Their dedication to the inspection task; this is related to the means, time and responsibility that are given to them;
- Their technical independence with respect to the designers of the object;
- Their ability to freely express their doubts or conclusions (organisational independence).

### **6.2.1.4. Demonstration by development process**

The justification of the relevance of the principles of a demonstration by development process usually addresses the following topics:

- The soundness of the process, methods, rules and tools used during development with respect to the claim and to the characteristics of the object on which the claim is made;
- The effective application of the process, methods, rules and tools, with the identification and justification of exceptions.

## **6.2.2. Objects on which the demonstration is performed**

### **6.2.2.1. Faithfulness**

A demonstration may be provided not on the object on which the claim is made, but on a representation of this object that is more suitable to the principles of the demonstration or to the limitations of the supporting tools. E.g., a demonstration based on the static analysis of software is usually provided on the source code or on intermediate code, whereas the operational representation is the executable binary code. In such cases, it may be necessary to justify the faithfulness of the representation on which the demonstration is performed with respect to the object on which the claim is made.

The translation may be from the representation to the object (e.g., when the source code is compiled into executable code) or from the object to the representation (e.g., when a simplified model is derived from the object). Several factors may reduce the faithfulness of the representation:

- Errors in the translation may lead to outright inconsistencies between the object and its representation;
- “Noise” or oversimplification in the translation process may lead to a representation that is not sufficiently accurate for the demonstration and may lead to an incorrect conclusion;
- The representation may be incomplete with respect to the object; e.g., some parts of the source code may be missing.

When it is necessary, the justification of faithfulness may be made in three main steps:

- The identification of the representation on which the demonstration is performed;
- The justification of the correctness of the representation with respect to the object, i.e. of the absence of outright inconsistencies;
- The identification of the nature of the differences between the representation and the object, and the justification of the acceptability of these differences, considering the principles of the demonstration; i.e. these differences must not affect the conclusion of the demonstration.

### **6.2.2.2. Coverage of the real operating conditions**

Even when the demonstration is provided directly on the object on which the claim is made, the conditions taken into consideration are not necessarily the real operational conditions of the object. (These conditions include the operational configuration of the object, the hardware constraints if the object is a software, the

interactions with other systems and equipment, the interactions with operators and maintenance staff.) E.g., some of the tests of the executable code may be performed on an engineering workstation or with a simulation of the process under control. Consequently, it may be necessary to justify the coverage by the demonstration of the real operating conditions.

The justification may be made in two steps:

- The description of the conditions used or simulated to provide evidence;
- The justification of coverage of the real operational conditions, considering the principles of the demonstration.

### **6.2.3. Tools supporting the demonstration**

Many demonstrations are possible in practice only with the assistance of supporting tools, such as static analysis tools, test benches, measurement tools or traceability tools. These tools have thus a strong impact on the demonstrations and their credibility. Regarding these tools, the important properties are:

- The identification of the tool: one must be able to know which tools in which versions were used to perform a demonstration, and how they were used;
- The adequacy of the tool with respect to the principles of the demonstration;
- The correctness of the tool results that are taken into consideration: tool errors must not lead to erroneous claims;
- The neutrality of the tool: a tool (most likely a test or simulation tool) must not alter the functioning of the object to the point where this can lead to erroneous claims.

### **6.2.4. Implementation of the demonstration**

The reality (has it been really implemented?) and the correctness (has it been correctly implemented?) of the implementation of a demonstration should be justifiable. Different or complementary approaches may be taken to this end, such as:

- The repeatability of the demonstration, so that an independent person can reproduce the demonstration;
- The auditability of the demonstration, so that an independent person can verify how the demonstration has been implemented.

## **6.3. Certification**

A certification attests that a given product conforms to some reference requirements. The cost-effectiveness of the pre-qualification of an OTS product may be increased by considering the available certifications, provided that, beyond the issue of the credibility of the certifier:

- The certified product is clearly identified;
- The reference requirements of the certification are documented and relevant and suitable for safety;
- The complete scope of tests and the related results which were performed for the certification are documented so that in case of additional safety features it would be possible to assess the effectiveness of supplementary tests;
- The certification is presented as a set of claims (i.e., the conformance of the product to the individual reference requirements) supported by auditable argumentation and evidence;

Thus, a certification is not taken as a whole: some of its claims may be accepted, while other may be rejected or need additional evidence. A certification may provide all four types of demonstration, and as described in §6.2, the four components of the credibility of each demonstration should be considered.

## **6.4. Example**

This section illustrates the notions of type of demonstration and of credibility of demonstration with the argumentation for the following claim:

- *K*: In the conditions specified, the executable code will not perform a division by zero.

According to the previous paragraphs dealing with the credibility of the demonstration, claim *K* may be decomposed in four sub-claims:

- *K1*: The used principles are appropriate for the demonstrations
- *K2*: The objects on which the demonstrations are made are faithful
- *K3*: The results of the supporting tools are correct
- *K4*: The implementation of the demonstrations is correct

#### **6.4.1. Principles**

The overall approach taken in the argumentation is a systematic proof by formal static analysis of the source code in *K* (based on Hoare's logic), and by showing that there is no division in the rest of the source code which is in assembly language. The initial claim *K* is supported by some assumptions related to the appropriateness of these principles:

- *K1.1*: Hoare's logic is an appropriate principle to systematically prove that there will be no division by zero
- *K1.2*: The absence of specific instructions in the assembly code systematically proves that there will be no division by zero

These assumptions may or may not be accepted by the regulators and need an early agreement in the justification process.

#### **6.4.2. Objects and environment**

The faithfulness of the objects on which the demonstrations are made is expressed in the following sub-claims:

- *K2.1*: The translation of the C and assembly programs that are considered produces the totality of the executable code for which claim *K* is made
- *K2.2*: The translation of the source programs that are considered is correct, i.e., it will not introduce faults that are not already in the source programs
- *A2.3*: The translation of the source programs into executable code does not introduce divisions that were not already in the source programs
- *K2.4*: The modifications made in the source code to facilitate the formal static analyses do not modify the conclusions of the analyses

Claim *K2.1* is supported by evidence based on the development process (notably on configuration management). It could also be based on a systematic proof (translation and bit by bit comparison with executable code on which claim *K* is made).

Claim *K2.2* is supported by evidences such as an extensive experience in operation of the translation tools (demonstration by sampling), testing of the translation tools (demonstration by sampling), use of these tools in narrow restricted conditions (demonstration by development process).

Assumption *A2.3* may or may not be accepted by the regulators.

Claim *K2.4* is supported by an argumentation based on inspection: all the transformations are documented so that their innocuousness can be audited.

The faithfulness of the environment taken into consideration is expressed in the following sub-claims:

- *K2.5*: The software parameters and the inputs have been exhaustively identified
- *K2.6*: Their ranges taken into consideration in the demonstration are those specified

Claim *K2.5* is supported by an argumentation based on the inspection of the design documentation and of the source code.

Claim *K2.6* is supported by an argumentation based on the inspection of the specification of the SIS and of its software.

### **6.4.3. Tools**

The correctness of the results of the supporting tools is expressed in the following sub-claim:

- *K3.1*: All the tools used in support of the demonstrations are fully identified
- *K3.2*: All the tools used in support of the demonstrations provide correct results

Claim *K3.1* may be verified by inspection.

Claim *K3.2* may be may be demonstrated by experience of the tools.

### **6.4.4. Implementation**

The correctness of the implementation of the demonstrations is expressed in the following sub-claims:

- *K4.1*: The tools have been correctly used and fed
- *K4.2*: Their outputs have been correctly analysed

These two claims may be verified by inspection by independent auditors, as the implementation of the demonstrations has been documented in detail.

## 7. Optimised Safety justification strategy for OTSP-based SIS

The safety justification of a SIS or the “system qualification” according to IEC 61513 [2] is the process that “provides assurance that an I&C system is capable of meeting, on continuing basis, the design basis performance requirements needed for the functions important to safety while subject to the specified environmental condition”. This process is performed in the context of the safety class of the system and of a specific set of safety requirements (from IAEA Safety Guide n° 50-SG-D3 [4] and IEC 61513 [2]). It determines whether this system is sufficiently safe for operational use.

The proposed safety justification strategy for OTSP-based SIS may be divided into two main phases:

- The pre-qualification of the OTS product embedded in the system;
- The effective safety justification of the SIS.

### 7.1. Pre-qualification of OTS product

A given OTS product is likely to be used in several SIS. The purpose of the pre-qualification is to factorise, whenever possible, the justification effort regarding this product, so as:

- To share the cost and to avoid unnecessary wastage of effort;
- To reduce uncertainties in system development and justification;
- To reduce the delays of system development and justification.

The fact that a given OTS product has been pre-qualified does not mean that it can be freely used in SIS. It just means that:

- The properties which can be assessed independently of any specific system have been assessed;
- The assessment has not identified any intrinsic contraindication for the use of this product in SIS, possibly provided that certain restrictive conditions are satisfied.

It is proposed to organise the pre-qualification of an OTS product in two main activities:

- A functional assessment;
- A dependability assessment.

#### 7.1.1. Functional assessment

The objective of the functional assessment of an OTS product is to assure that the functions, performances, interfaces, limitations and needs of the product are known to a level of detail that will allow an appropriate functional selection and a correct use in each target system. (see chapter 8)

#### 7.1.2. Dependability assessment

The objective of the dependability assessment of an OTS product is:

- To provide evidence that the product behaves as specified, possibly according to dependability figures;
- To provide evidence that it complies with all relevant regulatory or standard safety requirements;
- To identify its possible failure modes.

The dependability assessment may be performed in the framework of, or may recommend some restrictive use conditions for, the product, e.g., exclusion of some functions or some parts, maximum load, range of parameter values, protection against postulated failures, etc. (see chapter 9)

### 7.2. Safety justification of the SIS

The safety justification of the complete SIS still needs to be performed, even when all the OTS products it contains have been pre-qualified. This includes evidence that:

- The chosen OTS product match the functional and dependability needs set for them by the system requirements specifications and the system design;

- Each OTS product is used according to the conditions considered or recommended by its pre-qualification;
- In class A systems, the functions and parts of OTS products which are not strictly necessary to the system cannot affect its operation.

### 7.3. The functional assessments in renovation projects

For the cost effectiveness of the functional assessments, it is useful to make a distinction between project-independent and project-specific activities. Furthermore, the functional assessment for a given category of OTS products may be prepared independently of any specific OTS product, and then concretely adapted and applied to targeted OTS products. This reflection leads to the definition of four tasks which are summarised in Figure 1.

In the first task, a generic functional model for each main category of OTS products is defined. Its goal is to answer as exhaustively and accurately as possible the following question: “What are the functions and services that are generally expected from this category of products?”.

The objective of the second task is to provide a suitable description of the features of the OTS product that are considered for future projects. This description, for a given OTS product or category of OTS products, is organised according to the corresponding functional model established in the first task.

The third task formalises the user requirements of a specific project regarding each category of OTS products. These requirements are also organised according to the corresponding functional model.

Finally, the last task compares the results of the two preceding tasks, so as to decide, for a given project, which OTS product better fits the specified user requirements for a given category, possibly with specific conditions.

	<i>PRODUCT INDEPENDENT</i>	<i>PRODUCT DEPENDENT</i>
<i>PROJECT INDEPENDENT</i>	Task 1: Functional modelling, for each main category of OTS products	Task 2: Functional description of candidate OTS product
<i>PROJECT SPECIFIC</i>	Task 3: Specification of user requirements for each category of OTS products	Task 4: Matching of OTS products with corresponding user requirements specifications

Figure 1 : The four tasks of functional assessment.

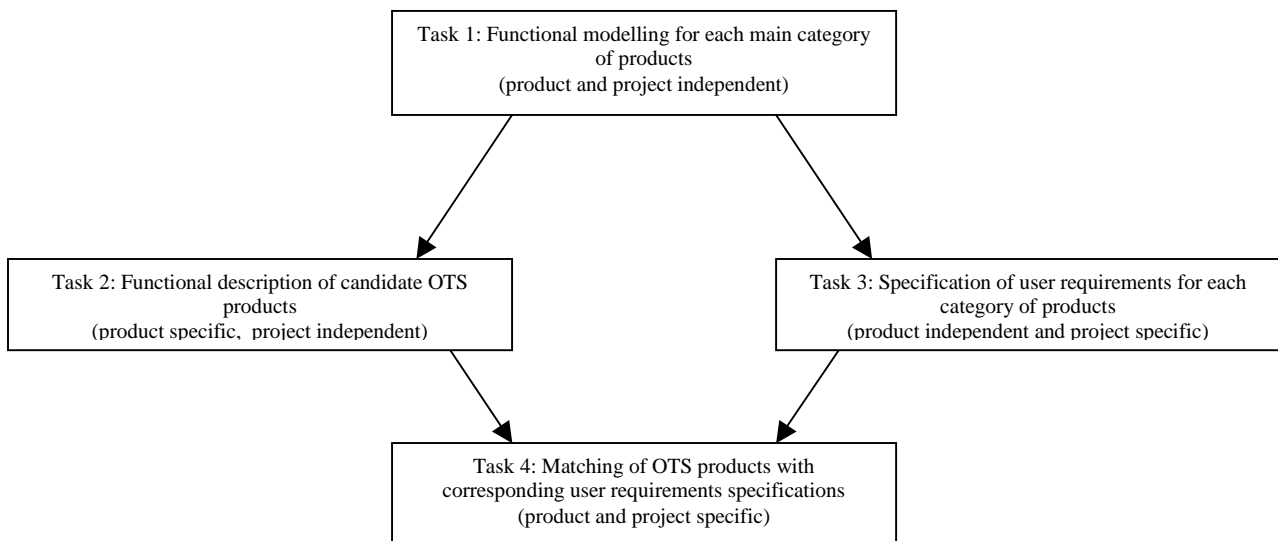


Figure 2 : Dependencies between the four tasks of functional assessment.

As task 1 is product and project independent, it can be performed once for all before any pre-qualification for each category of OTS products. Although the OTS product functional assessment in the pre-qualification phase actually corresponds to task 2, as the OTS product description has to be organised according to the corresponding functional model established in the task 1, this latter task is also described in this chapter.

Task 3 and 4 are parts of the safety justification of the SIS (see chapter 10).

## 8. Pre-qualification: Functional assessment of OTS product

### 8.1. Taxonomy

The OTS products embedding software that are classically found in the I&C of nuclear power plants can be organised according to the following functional taxonomy:

- *Dedicated devices* with very limited and targeted functionality, such as smart sensors, smart actuators, relays;
- *Automation* platforms such as Programmable Logic Controller (PLC) for multi-functions systems;
- Human – machine interface (*HMI*) platforms allowing operators to be informed on the state of the plant and of its systems and to issue all necessary manual controls;
- *Communication* products, which can be internal to a product family or gateways to external systems and products;
- *Software engineering tools*.

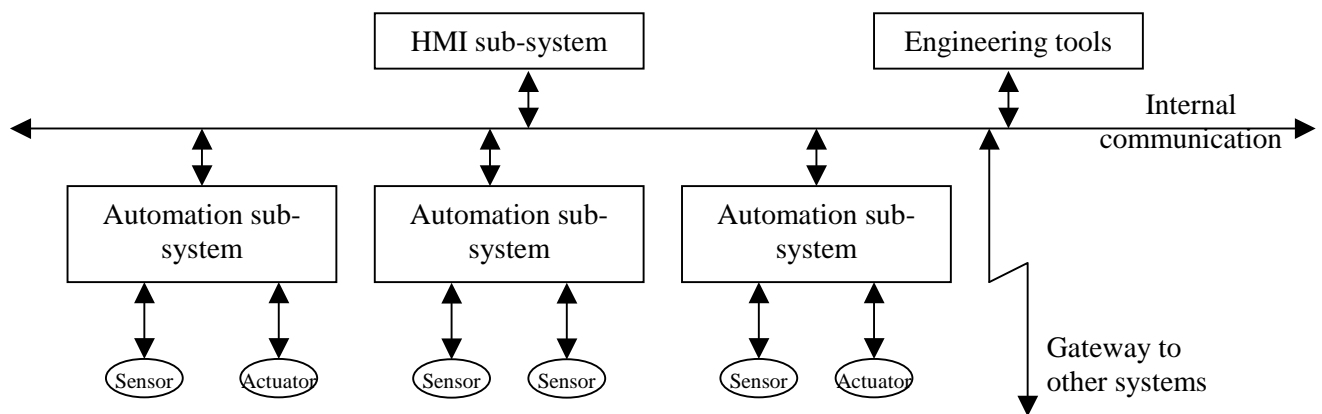


Figure 3 : Typical OTS products in the I&C of nuclear power plants

### 8.2. Task 1: Functional modelling

#### 8.2.1. Overall approach

This first task of the methodology aims at building a generic functional model for each main category of products. The model will be used to organise the specification of user requirements and the description of platforms. As this task is generic (project and product independent), it is carried out only once and can be applied to many projects and products afterwards.

The approach proposed to establish a functional model is:

- To identify the interactors, i.e., the entities that or who interact with the product; this will help to identify the functions to be generally expected from the product, and the types of interfaces to be provided;
- To identify the influencing conditions that may affect the behaviour and the performances of the product; this will help to establish guidelines for the specification of user requirements and for the description of real products.

#### 8.2.2. Customisation of the generic functional model

Due to technical constraints that may be set by the existing plants (notably when plants are built in series according to the same design), by industrial and strategic choices of the utility, or by the organisations and procedures set in place for engineering and operation, different users may have different functional

expectations for some of the investigation groups, and thus, may need to customise the generic functional model.

E.g., a user may wish to preserve the principles of the human-machine interface that are already in use in the plant to be renovated, or to enforce those that have been chosen for all future plants and renovations. This will be reflected in the customisation of investigation groups “Framework and capabilities for Human-Machine dialogue” and “Alarm management”.

### **8.2.3. Investigation objectives and principles**

An investigation group should have associated documentation that specifies:

- Its scope and boundaries (i.e., its relationships, dependencies and interfaces with the other investigation groups);
- The list of the items for which information regarding the product needs to be provided, and for each of them:
  - The nature and characteristics of the information expected;
  - The criticality of the item; i.e., more critical items will need a more thorough and credible investigation than less critical ones;
  - The investigation principles that may be used so as to obtain the desired information (e.g., inspection of documentation, analysis, test and measurement profiles).

## **8.3. Task 2: Functional description of OTS products**

The main objective of this task is to provide a suitable description of specific candidate products, so as to guarantee that one of the sub-properties of characterisation (see section 5.1), the description, will be satisfied. The overall process for the functional description of a product may be divided into the following activities:

- The customisation of investigation principles;
- The analysis of information and documentation available;
- The planning and implementation of complementary investigations;
- The reporting of investigations.

### **8.3.1. Customisation of investigation principles**

The investigation principles specified independently of any products may need to be adapted when a given product is considered, so as to take into consideration the information that is already available and the overall characteristics of the product. E.g., an investigation item may have a higher criticality in the framework of a given OTS product, or the investigation principles stated may be inapplicable or replaced by a another one that is more efficient and effective in this context.

### **8.3.2. Analysis of information and documentation**

#### **8.3.2.1. Collection**

An important source of information is the documentation provided by the supplier of the OTS product. This includes at least the User documentation, but may also include other documents such as design documents or guidelines for safety applications.

The records and documents resulting from experience in operation, and notably the error and failure reports, may also provide very useful information regarding failure modes, discrepancies with the User documentation, dependability, frequent human mistakes, etc.

Product certifications (e.g., of conformance to given industrial standards) may also be considered.

#### **8.3.2.2. Assessment of relevance**

The purpose of this activity is to verify that the documents and information that have been collected are relevant and applicable to the OTS product considered, as some of them may concern versions or conditions

of use very different from those intended. Irrelevant information and documentation should be discarded or used with caution and caveats.

### **8.3.2.3. Analysis**

The purpose of this activity is to collect the bits of information present in the selected documentation and that are relevant for an investigation item. In many cases, the credibility of the collected information will need to be assessed, notably when there are inconsistencies, unusually “ambitious” claims, or when the documents are of dubious origin or quality.

### **8.3.3. Complementary investigations**

The information collected in the preceding activity is usually not sufficient for a good functional description of a complex OTS product like an I&C platform. Complementary investigations are then necessary. Many reasons may justify or require complementary investigation, e.g.:

- The high criticality of some investigation items;
- The low credibility of some information elements;
- The absence of some information.

The purpose of the planning of complementary investigations is to specify:

- Which investigation items will need further investigation;
- The objectives to be reached for each of these investigation items;
- The overall approach to be applied, considering the information that is already available and the degree of credibility that is needed;
- The methods, means and criteria to be used, and the tests, measurements, analyses and inspections to be performed.

### **8.3.4. Reporting of investigations**

The reporting of investigations (analysis of documents and complementary investigations) should be documented and structured consistently for all products of the same category, preferably according to the functional model resulting from task 1, so as to allow comparisons.

When possible, the functional description so obtained should be confirmed by the supplier so as to limit the risk of incorrect description due to misunderstanding of the documentation, incorrect assumptions or out-of-date products or information.

## 9. Pre-qualification: Dependability assessment of OTS products

### 9.1. Taxonomy for dependability assessment

#### 9.1.1. Key characteristics

A cost effective dependability assessment of an OTS product must usually take into account the key characteristics of the product and of the SIS in which it will be embedded. It is proposed to consider mainly:

- The *safety class* of the SIS in which the product will be embedded;
- The *functional complexity* of the OTS product;
- The availability of information regarding the *development of the OTS product*;
- The availability of information regarding *experience in operation*.

#### 9.1.2. Safety class

The safety class of a SIS reflects its importance with respect to safety. Many different classification schemes are used in the nuclear industry. This document considers the first two safety classes of I&C systems defined by IEC 61226 [3]:

- Systems of safety class A play a principal role in the achievement or maintenance of safety; they prevent design basis events from leading to significant sequences of events, or mitigate their consequences; thus, the highest levels of dependability and understanding are required of these systems;
- Systems of safety class B play a complementary role in the achievement or maintenance of safety; the operation of class B systems may avoid the need to operate class A systems; they may complement class A systems in mitigating a design basis event, so that plant or equipment damage or activity release may be avoided or minimised; class B also includes systems whose failure could initiate or worsen the severity of a design basis event; because class A systems provide the ultimate prevention or mitigation, the safety requirements for class B systems need not be as high as those for class A systems.

##### 9.1.2.1. Functional complexity

The functional complexity of an OTS product measures the difficulty of guaranteeing that all possible uses of the OTS product have been covered. E.g., it depends notably on:

- The number of independent functions provided by the OTS product;
- The number and on the range of the parameters accepted by the OTS product;
- The level of programming accepted by the OTS product;
- The number of external factors which influence the functioning of the OTS product.

Thus, a programmable logic controller (PLC), which can be programmed to support a wide variety of applications, which can be organised in a seemingly infinite number of hardware and software configurations, and the functioning of which depends on a wide number of external factors such as values and load of inputs, is usually considered as functionally more complex than a dedicated non programmable sensor. The evaluation of the complexity of an OTS product may also take into account the safety class of the system (e.g., the criteria for evaluating functional complexity may be different for class A and for class B) and the extent of the use conditions to be considered for the OTS product.

This document considers three levels of functional complexity for pre-existing products:

- *High* functional complexity (e.g., programmable logic controllers);
- *Medium* functional complexity (e.g., communication interfaces);
- *Low* functional complexity (e.g., dedicated devices).

### **9.1.2.2. Availability of development information**

The development documentation of an OTS product provides information on how the product was specified, designed, implemented, tested, verified, validated. It may concern the OTS product itself, or the development and quality assurance processes.

The dependability assessment of an OTS product usually benefits from the availability of development information. However, this information is not always available or accessible to assessors. E.g., the supplier of the OTS product may want to protect its know-how, or may not consider safety applications sufficiently important in terms of business to bother. The information may also be difficult to analyse by those who are not expert in the technologies involved.

This document considers three levels of availability of development information:

- For a *white-box* OTS product, the source code and detailed information regarding internal design, implementation and development are available and accessible;
- For a *grey-box* OTS product, detailed information regarding internal design, implementation and development is available and accessible and some bits of the source code are available and accessible;
- For a *black-box* OTS product, some information regarding internal design, implementation and development is available and accessible but the source code is not available or accessible.

Caveat : *The terms “white box”, “grey box” and black box” refer here to the availability of the source code and may differ from the common understanding of those terms.*

### **9.1.2.3. Availability of information on experience in operation**

Experience in operation may provide valuable information for the dependability assessment of an OTS product, provided that this information is credible and sufficiently precise. E.g., the assessor must be sure that:

- The information concerns the version of the OTS product being assessed (and not another product or another version of the product);
- All relevant facts in the experience taken into consideration are detected and reported;
- The conditions of use of the OTS product during the experience taken into consideration are comparable to, or more severe than, those in the target SIS.

This document considers two levels of availability of information on experience in operation:

- No experience in operation or no information available on the experience in operation;
- Justified experience in operation.

### **9.1.3. Taxonomy and strategies for dependability assessments**

Not all combinations of characteristics are likely to be acceptable for safety application. The following table indicates the combinations that will be considered by CEMSIS. Dark cells indicate combinations that are not considered. These combinations are not necessarily forbidden, but if they occur, they will require a case by case approach. White cells are grouped and labelled according to possible assessment strategies.

Availability of the development information:		White-box		Grey-box		Black-box	
		Experience in operation					
Safety class	Functional complexity	No	Yes	No	Yes	Yes	No
Class A	High	A1		[Hatched]			
	Medium						
	Low	A1 / A2		[Hatched]	A2		[Hatched]
Class B	High	B1				[Hatched]	[Hatched]
	Medium	B1 / B2				B2	
	Low					B2	

Figure 4 : Strategies for dependability assessments.

Caveat : *Figure 4 presents the possibility to use black boxes for SIS of safety class A. In order to avoid unnecessary controversy, it may be necessary to emphasize that, for safety class A SIS, at least some information regarding internal design, implementation and development needs to be available and accessible on the embedded OTS products. As defined in §9.1.2.2, black boxes are meant in this document as source code black boxes.*

*Moreover, figure 4 presents the dependability assessments strategies for OTS products pre-qualification. Whenever the information available on an OTS product is not considered as sufficient to provide appropriate confidence on its intrinsic dependability, it is then possible, during the safety justification of the complete SIS, to consider the measures that have been specifically applied in the SIS to prevent the failures of the OTS product (see §12.2).*

**9.1.3.1. Strategy A1**

Strategy A1 is a white-box approach for OTS products intended for class A systems. It is based on the systematic analysis of detailed development information, and on white and black-box testing. Experience in operation, when available, is a complementary means. For OTS products of high or medium functional complexity intended for class A systems, only this approach gives an appropriate confidence.

**9.1.3.2. Strategy A2**

Strategy A2 is a black-box approach for OTS products intended for class A systems. It is based on intensive black-box testing, on large experience in operation, and on recommendations for use so as to mitigate possible failures. The analysis of development information, when available, is a complementary means.

**9.1.3.3. Strategy B1**

Strategy B1 is a grey-box approach for OTS products intended for class B systems. It is based on a combination of analysis of available development information, and grey and black-box testing. Experience in operation, when available, is a complementary means.

**9.1.3.4. Strategy B2**

Strategy B2 is a black-box approach for OTS products intended for class B systems. It is essentially based on intensive black-box testing. The analysis of development information and of experience in operation, when available, is a complementary means.

## 9.2. Properties of OTS products

The CEMSIS strategy for a cost-effective safety justification is based on the pre-qualification of OTS products independently of projects. The properties of an OTS product that need to be claimed and justified during pre-qualification are those that can contribute to the safety justification of future SIS, and that can be justified independently of any specific renovation project. E.g., an appropriate identification of the SIS requires the identification of its main components, and in particular, of its main OTS products.

The SIS properties that are essential to safety and the justification of which may be facilitated by a pre-qualification of the OTS products are:

- The characterisation of the SIS;
- The correctness of the SIS with respect to its description;
- The robustness of the SIS with respect to postulated internal and external erroneous conditions.

As it fully depends on each renovation project, no OTS product property has been identified to support the justification of the functional adequacy of the SIS to the real safety needs.

The modifiability of the SIS and of its software may benefit from some properties of its OTS products, but as it is not in the scope of the initial safety justification, this will not be further discussed in this document.

### 9.2.1. Properties contributing to the characterisation of the SIS

The OTS product properties supporting the characterisation of the SIS are:

- The identification of the OTS product, so as to allow the complete identification of the SIS;
- The description of the OTS product, so as to facilitate the description of the SIS;
- The integrity of the OTS product, which is an integral part of the integrity of the SIS.

In addition, the identification and the description of an OTS product are necessary for the justification of its correctness.

### 9.2.2. Properties contributing to the correctness of the SIS

The correctness of an OTS product with respect to its description is necessary to justify the correctness of the SIS. However, the argumentation of correctness of the OTS product will strongly depend on the information that is available regarding its internal design, its implementation, its development and its experience in operation.

According to the strategies for dependability, the argumentation for correctness may be a white, a grey box, or a black box argumentation. A white or grey box argumentation can rely on the same sub-properties of correctness as described in §5.3.

The dependability strategies consider a black box argumentation for the justification of correctness of OTS products used for both categories A and B, but the acceptability of the functional complexity of the OTS product must be justified:

- For category A, the OTS products must be of low functional complexity;
- For category B, the OTS products must be of low or medium functional complexity.

In addition, for this type of argumentation, a separate argumentation for the different types of faults as presented in §5.3 is rarely possible. In that case, correctness may be addressed as a whole.

### 9.2.3. Properties contributing to the robustness of the SIS

In some cases, the robustness of the SIS is totally or nearly totally dependent on the robustness of the OTS product. E.g., this is the case when the OTS product is the I&C platform. In such cases, the sub-properties expected for the robustness of the SIS are also expected of the OTS product. (see §5.4).

In other cases, the robustness of the SIS is obtained by means that are external to the OTS product. In such cases, the robustness of the OTS product is not a necessity and may be replaced by the completeness of the characterisation of the failure modes of the OTS product, so that appropriate actions may be taken in the rest

of the SIS. Appropriate actions aim to prevent a failure propagation that would lead to an overall SIS failure and shall guarantee that the OSTP failures will not lead to unsafe and not-specified SIS behaviours.

However, in most cases, the robustness of the SIS will be based on the joint capabilities of the OTS product (e.g., detection of the failures of the OTS product) and of the rest of the SIS (e.g., signalling and containment).

#### **9.2.4. Approaches for the dependability assessment of OTS products**

It is not the purpose of this document to give very detailed requirements for the dependability assessments, notably because such requirements could become rapidly obsolete with technical advances in products, technologies and assessment techniques. The guidelines are given in the form of tables in ANNEX I where the main safety properties relevant to dependability assessment are crossed with the relevant activities of the lifecycle. Recommendations are given regarding the types of the demonstrations that can be reasonably and cost-effectively proposed.

ANNEX I contains 5 tables, corresponding to:

- I&C platform or communication equipment for category A systems, assessed in a white box approach (I.1);
- I&C platform for category B systems, assessed in a grey box approach (I.2);
- Communication equipment for category B systems, assessed in a grey box approach (I.3);
- Communication equipment for category B systems, assessed in a black box approach (I.4);
- Sensors and actuator controls for systems of both categories assessed in a black box approach (I.5).

#### **9.3. Use conditions**

The dependability assessment of an OTS product may be facilitated by taking into account specific and restricted use conditions. However, if the OTS product is submitted to use conditions outside the range considered in the dependability assessment, this assessment may have to be reconsidered. E.g., a class A SIS may be designed so as to have a limited number of functioning modes and to provide narrow and stable conditions of use to its OTS products. A dependability assessment of an OTSP based on these conditions of use may be not adequate if the OTS product is to be used in another system in a different range of conditions, even if the SIS is of class B.

## 10. Safety justification: OTS product matching with user requirements

The evaluation of a product consists in judging, on the basis of concrete elements, its capability of fulfilling a specific objective or class of objectives [5]. Thus, **the ultimate purpose of the evaluation of an OTS product is to determine its ability to achieve qualitatively and quantitatively the objectives that are defined and formalised in the user requirements specifications set by a renovation project for this product.**

The functional assessment that is to be assessed during the safety justification of OTSP-based SIS should provide one of three verdicts regarding the OTS product: suitable, unsuitable (whatever the configuration of the product) or suitable with defined reservations or quantified risks. The third verdict being in all likelihood the most frequent, it is to be expected that the functional assessment will allow three types of feedback:

- The capability for acting on the offer; that may be an acceptable option in the context of a medium or long term policy with suppliers who take into account the expectations of their clients;
- The re-examination of the user requirements specifications, notably for feasibility reasons.

The possibility, once the strengths and weaknesses of the product are known in sufficient details, of defining a set of rules and procedures that correct or bypass the noted weak points.

According to §7.3, this section focuses on the task 3 and 4 of functional assessments of the renovation projects.

### 10.1. Task 3: User requirements regarding OTS products

WP2 proposes a process for establishing the user requirements specifications for a renovation project. These requirements concern the I&C system. The user requirements that concern the I&C platform and the other OTS products integrated in the I&C system will be derived in part from the system user requirements, in part from industrial considerations specific to each organisation (for cost-effectiveness, those organisation specific considerations should be integrated once and for all, when relevant, to the detailed functional model), and in part from considerations regarding the system architecture and the role it assigns to each OTS product.

In many cases, several architectural concepts are possible for the same set of system user requirements. In particular, some technological choices (e.g., use or not of “smart” instrumentation and actuation, use or not of field buses) may lead to specific architectural concepts, which may in turn lead to specific user requirements on some categories of OTS products. Figure 5 illustrates a possible approach for allocating the system user requirements specifications to the different OTS products.

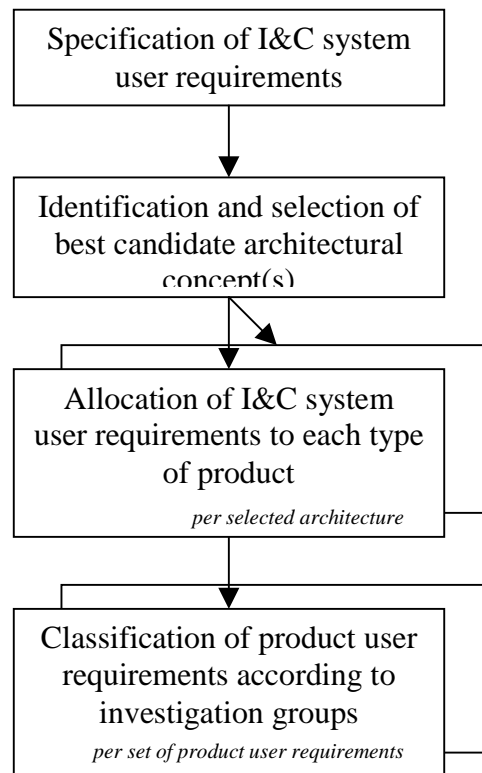


Figure 5 :Allocation of system user requirements to types of OTS products.

### 10.1.1. Selection of candidate architectural concepts

After the preliminary or high level I&C system user requirements are specified, candidate architectural concepts for I&C system must be identified, analysed and selected. These requirements constitute one of the inputs of the project viability analysis. They should express the overall objectives and needs of the user regarding the I&C system in a way that does not put unnecessary dependence upon, and constraints on, candidate technologies and solutions. E.g., preliminary user requirements regarding the I&C system may not specify the precise application functions to be provided, but only their main characteristics such as the numbers of different types of sensors and actuators, dimensioning information, overall dependability constraints. Information regarding the flexibility of requirements may also be provided so as to distinguish requirements that are mandatory and non negotiable from those that are desirable but that may be reconsidered if they lead to excessive cost, delay or uncertainty.

The identification of possible architectural concepts is in large part based on the experience and skills of system designers, and on their familiarity with the functional characteristics (including capabilities and limitations) of the potentially suitable OTS products. This familiarity may result from experience in operation and / or from functional assessments. One important objective of the design and description of an architectural concept is to allow:

- The identification of the parts constituting the architectural concept;
- The description of their interactions;
- The identification of the parts that may be assigned to given categories of OTS products;
- The specification of the corresponding product requirements;
- The verification that the non negotiable preliminary user requirements regarding the I&C system can be satisfied.

This should also be done at a high level so as to avoid overly restrictive product user requirements specifications.

In many cases, after a first technical and economical assessment of the identified candidate architectural concepts, only one is retained for the subsequent system engineering activities. However, and notably when

advanced technologies and products have matured in similar applications, the identification of an optimal solution may require a detailed comparison of candidate architectures (e.g., a comparison between a conservative, proven but functionally limited or expensive architecture, and an advanced, novel and more powerful or more economical one). For the cost-effectiveness of the system development process, the number of candidates to be analysed in detail should be strictly limited: usually, the maximum number will be 2 or 3.

### **10.1.2. Specification of product requirements**

The allocation of the high level or preliminary system user requirements specifications to OTS products is performed in the framework of the detailed analysis of a candidate architectural concept. One of the results of this activity is the specification of user requirements for the different parts constituting the candidate system architecture, and notably for the parts that might be assigned to off-the-shelf pre-existing products. This specification results from the allocation of the individual system user requirements to the architectural parts, and from considerations internal to the architecture itself (communication and synchronisation between architectural parts, surveillance of a part by another part, initialisation, ...).

### **10.1.3. Classification of product requirements into investigation groups**

The objective of this activity is to distribute the user requirements regarding a given category of products into the investigations groups corresponding to this category, so as to make sure that all these requirements will be taken into account, and to provide an input to the specification of the investigation programs.

## **10.2. Task 4: Matching OTS products with corresponding user requirements**

### **10.2.1. Comparison of product documentation and user requirements**

The first step of task 4 consists in the comparison, investigation group by investigation group, of the information that is available on the candidate products with the project specific user requirements for a given type of product (resulting from task 3, see §10.1).

At this stage, the information that is available regarding a given OTS product comes from two main sources:

- The project independent functional description of the product (resulting from task 2, see §8.3);
- The information specifically given by the supplier as an answer to the user requirements.

For a given user requirement, the result of the comparison may be:

- Yes, according to the information that is available, the product complies with the requirement;
- No, according to the information that is available, the product does not comply with the requirement;
- Require further investigation, the information that is available is not sufficient or is contradictory, and additional investigation is necessary.

Even when the conclusion of this first comparison is positive, a more thorough investigation may be considered if the requirement is identified as a critical requirement that must be satisfied. Figure 6 illustrates a typical decision table.

	Available information says requirement is satisfied	Available information says requirement is not satisfied	Available information is not sufficient to conclude whether requirement is satisfied or not
<b>Critical user requirement</b>	Assess credibility of available information, and consider the need for complementary investigation	Assess credibility of available information, and consider the need for, and cost-effectiveness of, complementary investigation	Consider complementary investigation if the product is not eliminated for other reasons
<b>Non-critical user requirement</b>			Consider complementary investigation if the product is not eliminated for other reasons

Figure 6 : Typical decision table for complementary investigation.

### 10.2.2. Complementary investigation

A Complementary investigation plan should be developed when complementary investigation is decided for a given product. This plan is preferably organised in the same investigation groups as identified by task 1 (see §8.2). It identifies the user requirements concerned by the complementary investigation, and what information needs to be obtained. Different means may be used to this end, e.g.,:

- Further analysis of the User documentation of the product;
- Questions to the supplier;
- Additional tests and analyses of the product.

After the implementation of the complementary investigation, the additional information collected should be integrated in the product description, and the compliance of the product to the user requirements concerned should be reassessed.

### 10.2.3. Feedback of supplier

Prior to the final conclusion, the supplier should have the possibility to comment on the conclusions regarding the compliance of the product to the user requirements, so as to have the opportunity to:

- Correct any misinterpretation by the system designers;
- Suggest possible uses of the product that may reverse a negative conclusion;
- Inform the system designers of any imminent evolution of the product that might modify a conclusion.

### 10.2.4. Effective selection of products

The process presented in this section is a technical process aiming at evaluating the conformance of a product to each individual functional or technical requirement. When several products are in competition, the usual case is that the ranking of products with respect to these requirements is not monotonous: a given product will be better than the competition on some requirements, but will be worse on some others. In addition, many factors need to be taken into account when selecting OTS products. It is not the purpose of this document to address this overall issue: its only goal is to provide as objective and clear information as reasonably achievable to decision makers.

## 11. Safety justification: Causes and possible defences against Common Cause Failures

Common Cause Failures (CCF) are not the solely concern of the SIS based on OTS products but the use of OTS products in SIS may increase the potential of CCF because it leads naturally to have more identical or similar components in different subsystems of the SIS. As CCF scope is large, this document mainly considers the CCF related to software. Moreover, in order to lighten the document body, all the detailed information regarding each type of CCF have been shifted in ANNEX II.

In ANNEX II, CCF are categorized according to their potential sources as follow:

1. CCF due to error propagation (see ANNEX II.1);
2. CCF due to common modules (see ANNEX II.2);
3. CCF due to similarities in development processes (see ANNEX II.3);
4. CCF due to exceptional conditions (see ANNEX II.4);
5. CCF due to maintenance activities (see ANNEX II.5).

ANNEX II takes many elements from the IEC 60880-2 standard [6], some elements from the IEC 61513 standard [2] and a few ideas from the IEC 62340 draft v3 [7]. For each source of CCF, it presents the possible defences with their respective benefits and drawbacks. Some defences are then recommended and some others are not. The recommendations are presented for designers as for developers but can also be viewed as a mean to assess a SIS regarding CCF. For the safety justification of a SIS, the same sources of CCF are to be considered between the different components or OTS product of a same SIS and particularly when the low criticality (see chapter 12) of a component needs to be justified.

## 12. Safety justification: Notion of OTS product criticality

Independently or not of the pre-qualification phase of the OTS products, the safety justification of SIS needs to provide proof elements on the dependability of the SIS. The dependability argumentation of SIS may rely on the dependability of its OTS products. However it is likely that SIS embed OTS products on which no conclusion on their level of dependability have been reached on the only basis of what is known of these OTS products. In such cases and considering that this is the dependability of the SIS that need to be justified; it may be worthwhile to consider the effective criticality of the OTS products within the SIS.

Moreover, it may be fully relevant to distinguish the critical parts of a SIS from the less critical other parts so as to increase the efficiency of the safety justification by really focusing on the points important to safety and not wasting time on irrelevant parts of the systems. Up to now, it seems that such a distinction within SIS of safety class A might not be fully adequate as these systems are designed so as to include only functions important to safety and so as to be as simple as possible. Thus, the current approach is to claim that all is critical in a safety class A SIS. Consequently, this approach addresses first the safety class B SIS.

Whereas not dedicated to SIS using OTS products, this approach fits perfectly with OTS product-based SIS, as the OTS products are functional elements easily distinguishable. Moreover, it is likely that the low amount of information available on some types of OTS products may rationally lead the designers to adopt specific methods of integration so as to prevent the failures of these products. The safety justification may lean on these specific methods to justify the safe use of the OTS products.

Criticality may be defined as “the potentiality that the occurrence of an OTS product failure may have reverse effect on the safety”. In the three following sections, three levels of criticality are proposed.

### 12.1. Criticality C2

**An error in the OTS product may lead to an unsafe situation.**

OTS products of criticality C2 are as critical as the SIS in which they are embedded. This is the highest level of criticality. It is by default the criticality level that is assumed for the OTS products when no claim on criticality is made for their justification.

### 12.2. Criticality C1

**An error in the OTS product can lead to an unsafe situation only if another error occurs in some other part of the SIS.**

According to the IAEA Safety Glossary [8], the single failure criterion is defined as “a criterion (or requirement) applied to a system such that it must be capable of performing its task in the presence of any single failure.” The notion of criticality C1 partly results from this principle of single failure criterion. Instead of requiring that system must be capable of performing its task in the presence of any single failure, the idea is to take advantage of the design or architecture measures that may have been applied so as to ease the justification of OTS products for a given SIS. Here, the overall objective is not the continuation of the system task but the guarantee to remain in safe situations.

The demonstrations that OTS products have a C1 criticality level rely, in most cases, on the architecture or the design of SIS. It requires that clear measures have been specifically applied in the SIS either to prevent the failures of the OTS product at stake or to maintain a safe state in spite of the failures of the OTS product. Redundancy, diversity and wrapping are the three main usable principles to achieve these goals.

Redundancy addresses the failures of C1 OTS products by using several C1 OTS products in parallel to insure the correct operation of a function. This approach raises the problem of the error likelihood and of the failure frequency of these products. According to the definition of C1 criticality, an error in the product may lead to an unsafe situation only if another error also occurs in some other parts of the SIS. When several OTS products of criticality C1 are used in parallel, the likelihood that two independent errors occur at the same time in two independent OTS products may not be negligible. Thus, in some case, it may be necessary to justify the acceptability of the situation notably on the basis of:

- The probability that an unsafe situation occurs;
- The class of safety function to be performed by the SIS.

In the one hand, whereas pure redundancy is very efficient to prevent failures due to hardware random faults, application of redundancy is a weak protection against software systematic faults. On the other hand, whereas diversity in the software implementation is a possibility to cope with CCF caused by systematic faults in software modules this approach is not recommended in ANNEX II.3.3.7 as it raises other problematic issues. Thus, the recommended approach is to apply diversity factors to redundancy, such as diversity in the usage of identical modules or the diversity of the input signal sources.

When SIS is designed so that several OTS products are used in parallel to guarantee the correct operation of a same functional element, the justification that these several OTS products are C1 leads to justify that:

- These OTS products are independent one from the other;
- These OTS products are independent regarding the potential CCF;
- The likelihood that these OTS products fail independently at the same time is low enough to be acceptable.

Using extensively the criticality C1 in the dependability demonstration of SIS may lead to difficulties such as evaluating the likelihood that two C1 components integrating software fail simultaneously.

In contrast to redundancy where the use of multiple C1 OTS products in parallel aims at guaranteeing the correct operation of a given function, wrapping principle consists in using C2 components (the wrappers) to prevent or detect the failures of the C1 OTS products. The wrapper may prevent the wrapped C1 component failures by filtering the calls to undesirable functions and / or by checking the consistency of the inputs and the outputs. In both cases, the wrapper aims at protecting the SIS from the unsafe situations for which the C1 component may be the source. For instance, the role of the wrapper may be to check the inconsistencies in the outputs or the non-performance of the service by wrapped OTS products and to act as necessary so as to lead to a safe situation (e.g., a safe fall-back position). The wrapping requires that the failure modes of the wrapped OTS product are correctly characterised and that the wrapper correctly prevents these failures modes.

For wrapping, as a component of higher criticality guarantees the leading to a safe situation whatever the failures of the OTS products then, the question of error likelihood or failure frequency of the OTS product is irrelevant to the safety demonstration.

### 12.3. Criticality C0

**An error in the OTS product cannot lead to an unsafe situation.**

This is the lowest level of criticality. With the implementation of the new technologies and notably the use of pre-existing I&C platform to design SIS, it is likely that some parts of the SIS do not participate in the safety functions. Moreover, some of these parts might actually have no adverse effect on any safety functions. For the safety justification efficiency, it is relevant to distinguish the critical sub-systems of the SIS from the not critical ones so as to really focus on the safety. However, the innocuousness of the non-critical parts on the critical ones needs to be justified.

Three cases of OTS product of C0 criticality can be considered within a SIS:

- **The product is completely separated from all the safety functions:** justification mainly based on the architecture of the SIS;
- **The product cannot influence any safety function:** justification based on the properties of the management of the resources shared between the OTS products and the rest of the SIS;
- **The product cannot have adverse influence on any safety function:** justification based on the defences implemented in the rest of the SIS so as to guarantee that the failures of the OTS products cannot jeopardize the correct operating of the safety functions.

### 12.4. General strategy of criticality justification

The justification of the low criticality (C1 or C0) of OTS product leads to assess through which ways the product may jeopardize the correct operating of the safety functions. The OTS product may interact with the SIS and some other systems. Thus, the strategy may be decomposed in two sets of sub-claims:

- The claims on the interaction mechanisms between the component and the rest of the SIS;
- The claims on the interaction mechanisms between the component and the other systems of the plant.

The interaction mechanisms are in relation with the resources shared between the systems or the sub-systems. For instance, regarding software, the resources that can be shared are:

- The communication channels (e.g., pipes, sockets);
- The memory (e.g., RAM, hard disk...);
- The CPU;
- The hardware configuration;
- Other resources.

Each interaction mechanism needs to be characterised so as to clearly identify the potential means of error propagation between the systems or the sub-systems. For instance, the characterisation of the interactions between different software tasks in relation to the CPU might consist notably in describing the mechanisms of CPU allocation and of interrupt management.

The second step of the criticality justification deals with the demonstration that correct provisions are applied according to the potential threats of error propagation. This is further detailed in ANNEX II.1.

## 13. Conclusion

The main innovative aspects of the approaches proposed to demonstrate that OTSP-based systems can be suitable for supporting functions important to safety are summarised below:

- **WP3 proposes to base the safety justification on the demonstration that SIS satisfy the properties that are essential to safety.**
  - The current practice to justify the safety of SIS is actually to assess the compliance of a given system to the defined rules. This practice does not consider the particular features of SIS, so that SIS are evaluated in a similar way without taking into account specific features of the SIS. Consequently, this practice may require some unnecessary efforts leading to cost overruns. The proposed approach aims at assessing the adequacy of the SIS over its lifetime to satisfy the real safety needs on the base of structured argumentation so as to demonstrate that safety properties are satisfied. This approach is flexible as necessary and meets the OTS products issues of the safety justification of SIS;
  - The top-level safety property that needs to be justified for the SIS is the **adequacy of the SIS over its lifetime to satisfy the real safety needs**. It may be decomposed into the following properties:
    - The characterisation of the SIS;
    - The functional adequacy of the SIS to the real safety needs;
    - The correctness of the SIS with respect to its description;
    - The robustness of the SIS with respect to postulated internal and external non-nominal conditions;
    - The maintenance of the adequacy to safety during the lifetime of the SIS.
  - WP3 studies the possible types of demonstration to prove the satisfaction of those properties. It focuses on the credibility elements of demonstrations.
- **WP3 proposes to pre-qualify OTS product independently of renovation project** so as:
  - To share the cost and avoid unnecessary wastage of effort;
  - To reduce uncertainties in system development and justification;
  - To reduce the delays of system development and justification.
- **WP3 proposes a taxonomy and several strategies for OTS products dependability assessment:**

WP3 proposes a taxonomy of OTS products on the base of the safety class of the SIS in which the products will be embedded, their functional complexity, the availability of information regarding their development and the availability of information regarding experience in operation. The developed strategies for dependability assessment take into account the specific features of OTS products.
- **WP3 focuses on the OTSP issues of the safety justification through:**
  - **The functional assessment of OTS products** so as to determine their abilities to achieve qualitatively and quantitatively the objectives that are defined and formalised in the user requirements specifications set by a renovation project for this product;
  - Since the use of OTS products in SIS may increase the potential of CCF, **the assessment that appropriate strategies are applied during the SIS development to make Common Cause Failures adequately unlikely;**
  - **The notion of criticality;** since it is likely that SIS embed OTS products on which no conclusion on their level of dependability have been reached on the only basis of what is known of these OTS products, it may be worthwhile to consider the effective criticality of the OTS products within the SIS so as to evaluate the overall dependability of SIS.

## ANNEX I : Approaches for the dependability assessment of OTS product

This annex presents different approaches for the dependability assessment of OTS products in the form of tables. For each step of the safety lifecycle of OTS products, the tables present the type of demonstration that can be realised to demonstrate the properties that are essential to safety (left column). Grey cells mean that no demonstration is expected at the corresponding step of the safety lifecycle. Empty cells mean that the types of demonstration expected for the sub-property are those expected for the property at the corresponding step of the safety lifecycle. The demonstration of the main safety properties is often supported by the demonstration of their sub-properties.

### I.1 High complexity or medium complexity component in A1 approach, e.g. I&C platform

The dependability assessment of an I&C platform for category A applications is necessarily a white box approach. The overall strategy may be summarised as follows:

- Indirect demonstrations by development process (e.g., quality assurance, application of general rules);
- Whenever practically possible, evidence of correctness of software should be based on credible systematic proofs; these demonstrations may be given a priori by design, or a posteriori by formal static analysis of the implemented software;
- However, since in the current state of the technology these analyses do not always take into account the real executable code and all its interactions with its environment (notably the supporting hardware), complementary evidence by sampling (most usually tests, and possibly experience in operation) is still needed;
- Evidence by inspection is difficult to replace for properties like absence of hidden functions.
- The dependability assessment of medium-complexity components (e.g. communication equipment) in a A1 approach must be performed in the same manner.

<i>Properties of an AI I&amp;C platform or communication equipment</i>	<i>Specification</i>	<i>Design</i>	<i>Implementation</i>	<i>Unit Testing</i>	<i>Integration Testing</i>	<i>Validation testing</i>	<i>Operation</i>	<i>Pre-qualification</i>	
<b>Characterisation</b>									
• Identification	Demonstration by development process, e.g., by configuration management, by quality assurance Systematic proof, e.g., by intrinsic branding of main components							Demonstration by inspection, e.g. of the configuration management process, of the quality assurance plan	
• Description									
• <b>Completeness</b> with respect to product features								Demonstration by development process: see section 7 on Functional assessment	
• <b>Correctness</b> with respect to real product	See Correctness with respect to description								

<p>•Precision</p>							<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>• use of glossaries for terms having a specific meaning</li> <li>• use of languages with well-defined syntax and semantics</li> </ul> <p>Demonstration by inspection of description</p>	
<p>•Accuracy, depending on intended requirements</p>							<p>Demonstration by inspection of description</p>	
<p>• Integrity</p>	<p>Systematic proof, e.g., by auto-diagnostic of integrity</p>						<p>Demonstration by inspection Demonstration by sampling, e.g. testing for presence of auto-diagnostic function</p>	
<p>Correctness with respect to description</p>		<p>Demonstration by development process, e.g.,:</p> <ul style="list-style-type: none"> <li>• application of general design rules, notably for clarity and testability of architecture and behaviour, and possibly from relevant standards</li> <li>• quality of design documentation</li> <li>• rigorous and documented selection of OTS components, and cautious use of these components</li> <li>• design reviews and / or walkthroughs</li> <li>• traceability between specified</li> </ul>	<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>• application of general coding rules for clarity and testability, possibly from relevant standards</li> <li>• systematic proof of conformance to some coding rules</li> <li>• code reviews and / or walkthroughs</li> <li>• rigorous version and configuration management</li> </ul>	<p>Demonstration by sampling, e.g., unit testing with structural and functional coverage criteria Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>• reporting of tests</li> <li>• rigorous version and configuration management</li> </ul>	<p>Demonstration by sampling, e.g., integration testing with coverage criteria with respect to design provisions Demonstration by development process, e.g.,:</p> <ul style="list-style-type: none"> <li>• planning and reporting of tests</li> <li>• use of selected verification tools</li> <li>• independence of verifiers</li> </ul>	<p>Demonstration by sampling, e.g., validation testing with coverage criteria with respect to specification, or with statistical criteria Demonstration by development process, e.g.,:</p> <ul style="list-style-type: none"> <li>• planning and reporting of tests</li> <li>• use of selected verification tools</li> <li>• independence of verifiers</li> </ul>	<p>Demonstration by sampling, e.g., analysis of experience in operation</p>	<p>Demonstration by sampling, e.g., complementary tests Demonstration by inspection of development and quality assurance documentation, e.g.,:</p> <ul style="list-style-type: none"> <li>• guided design reviews or walkthroughs</li> <li>• assessment of quality assurance plans, integration plan, validation plan</li> </ul>

		requirements and design provisions							
<ul style="list-style-type: none"> <li>Freedom from intrinsic SW faults</li> </ul>		<p>Systematic proof: by specific design rules avoidance of some types of intrinsic faults, e.g.,:</p> <ul style="list-style-type: none"> <li>use of statically allocated resources to avoid faults in resource management and risks of running short of resources,</li> <li>mono-tasking and absence of interrupts to avoid faults in internal synchronisation</li> </ul>	<p>Systematic proof by formal static analysis to prove freedom from some types of intrinsic faults, e.g.,:</p> <ul style="list-style-type: none"> <li>division by zero</li> <li>out of bound array indexes and variables</li> <li>use of non initialised variables</li> </ul>					<p>Demonstration by sampling (e.g., by testing) for intrinsic faults and for parts not covered by systematic proofs</p>	<p>Systematic proofs or complementary systematic proofs, e.g. static analysis of the source code</p>
<ul style="list-style-type: none"> <li>Freedom from functional faults</li> </ul>		<p>Systematic proof by specific design rules to guarantee, e.g.,:</p> <ul style="list-style-type: none"> <li>timeliness</li> <li>sequencing</li> </ul>	<p>Systematic proof by formal static analysis to prove some functional claims, e.g., correctness of simple Boolean computations</p>						<p>Systematic proofs or complementary systematic proofs</p>
<ul style="list-style-type: none"> <li>Freedom from SW / HW incompatibility</li> </ul>					<p>Systematic proof that some HW/SW constraints are satisfied, e.g.,:</p> <ul style="list-style-type: none"> <li>compatibility of available HW memory and SW memory requirements</li> </ul> <p>Demonstration by sampling (testing) for other HW/SW constraints, e.g.,:</p> <ul style="list-style-type: none"> <li>inputs and outputs</li> </ul>				<p>Systematic proofs or complementary systematic proofs</p>
<ul style="list-style-type: none"> <li>Conformance to design &amp; implementation documentation</li> </ul>		<p>Demonstration by development process, e.g., quality assurance</p>							<p>Demonstration by inspection</p>

<ul style="list-style-type: none"> <li>Freedom from hidden functions</li> </ul>		Demonstration by inspection of design and source code					Demonstration by inspection of design and source code	
<b>Robustness</b>	Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>systematic identification of non-nominal conditions</li> <li>analysis of past experience</li> <li>analysis of tradeoffs</li> <li>specification of robustness</li> <li>critical review of specification of robustness</li> </ul>	Systematic proof, e.g., <ul style="list-style-type: none"> <li>intrinsic robustness by design to given non-nominal conditions</li> <li>use of redundancy, diversification, independence and separation</li> </ul> Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>documentation of design provisions for active robustness</li> <li>critical design review of provisions for active robustness</li> </ul>	Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>use of defensive programming</li> <li>conformance to implementation rules aiming at robustness</li> </ul>		Demonstration by sampling, e.g., injection of different types of faults	Demonstration by sampling (e.g., fault injection)	Demonstration by sampling	Demonstration by sampling (e.g., fault injection)

## I.2 High complexity component in B1 approach, e.g. I&C platform

The dependability assessment of an I&C platform for category B applications is usually a grey box approach. The overall strategy may be summarised as follows:

- Indirect demonstrations by development process (e.g., quality assurance, application of general rules);
- Evidence of correctness of software is mainly based on demonstration by sampling in all stages following the implementation (i.e., unit testing, integration testing, validation testing, experience in operation and testing during pre-qualification); whenever practically possible, systematic proofs a priori may be required to provide more confidence in correctness ( e.g., application of design rules);
- Evidence by inspection is difficult to replace for properties like absence of hidden functions.

<i>Properties of an B1 I&amp;C platform</i>	<i>Specification</i>	<i>Design</i>	<i>Implementation</i>	<i>Unit Testing</i>	<i>Integration Testing</i>	<i>Validation testing</i>	<i>Operation</i>	<i>Pre-qualification</i>
<b>Characterisation</b>								
• Identification	Demonstration by development process, e.g., by configuration management, by quality assurance Systematic proof, e.g., by intrinsic branding of main components							Demonstration by inspection, e.g. of the configuration management process, of the quality assurance plan
• Description								
• <b>Completeness</b> with respect to product features								Demonstration by development process: see section 7 on Functional assessment
• <b>Correctness</b> with respect to real product	See Correctness with respect to description							
• <b>Precision</b>								Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>• use of glossaries for terms having a specific meaning</li> <li>• use of languages with well-defined syntax and semantics</li> </ul> Demonstration by inspection of description
• <b>Accuracy</b> , depending on intended requirements								Demonstration by inspection of description

<ul style="list-style-type: none"> <li>Integrity</li> </ul>	Systematic proof, e.g., by auto-diagnostic of integrity							Demonstration by inspection Demonstration by sampling, e.g. testing for presence of auto-diagnostic function
<b>Correctness</b> with respect to description		Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>Application of general design rules, notably for clarity and testability of architecture and behaviour, and possibly from relevant standards</li> <li>quality of design documentation</li> <li>rigorous and documented selection of OTS components, and cautious use of these components</li> <li>design reviews and / or walkthroughs</li> <li>traceability between specified requirements and design provisions</li> </ul>	Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>application of general coding rules for clarity and testability, possibly from relevant standards</li> <li>systematic proof of conformance to some coding rules</li> <li>code reviews and / or walkthroughs</li> <li>rigorous version and configuration management</li> </ul>	Demonstration by sampling, e.g., unit testing with structural and functional coverage criteria Demonstration by development process, e.g., reporting of tests rigorous version and configuration management	Demonstration by sampling, e.g., integration testing with coverage criteria with respect to design provisions Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>planning and reporting of tests</li> <li>use of selected verification tools</li> <li>independence of verifiers</li> </ul>	Demonstration by sampling, e.g., validation testing with coverage criteria with respect to specification, or with statistical criteria Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>planning and reporting of tests</li> <li>use of selected verification tools</li> <li>independence of verifiers</li> </ul>	Demonstration by sampling, e.g., analysis of experience in operation	Demonstration by sampling, e.g., complementary tests Demonstration by inspection of development and quality assurance documentation, e.g.: <ul style="list-style-type: none"> <li>guided design reviews or walkthroughs</li> <li>assessment of quality assurance plans, integration plan, validation plan</li> </ul>
<ul style="list-style-type: none"> <li>Freedom from intrinsic SW faults</li> </ul>		Systematic proof: by specific design rules avoidance of some types of intrinsic faults.		Demonstration by sampling (e.g., by testing) for intrinsic faults and for parts not covered by systematic proofs				
<ul style="list-style-type: none"> <li>Freedom from functional faults</li> </ul>		Systematic proof by specific design rules to guarantee, e.g.: <ul style="list-style-type: none"> <li>Timeliness</li> <li>Sequencing</li> </ul>						
<ul style="list-style-type: none"> <li>Freedom from SW / HW incompatibility</li> </ul>					Demonstration by sampling (testing) for other HW/SW constraints, e.g.:			

					<ul style="list-style-type: none"> <li>inputs and outputs</li> </ul>			
<ul style="list-style-type: none"> <li>Conformance to design &amp; implementation documentation</li> </ul>		Demonstration by development process, e.g., quality assurance						Demonstration by inspection
<ul style="list-style-type: none"> <li>Freedom from hidden functions</li> </ul>		Demonstration by inspection of design and source code						Demonstration by inspection of design and source code
<b>Robustness</b>	<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>systematic identification of non-nominal conditions</li> <li>analysis of past experience</li> <li>analysis of tradeoffs</li> <li>specification of robustness</li> <li>critical review of specification of robustness</li> </ul>	<p>Systematic proof, e.g., intrinsic robustness by design to given non-nominal conditions</p> <p>use of redundancy, diversification, independence and separation</p> <p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>documentation of design provisions for active robustness</li> <li>critical design review of provisions for active robustness</li> </ul>	<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>use of defensive programming</li> <li>conformance to implementation rules aiming at robustness</li> </ul>		Demonstration by sampling, e.g., injection of different types of faults	Demonstration by sampling (e.g., fault injection)	Demonstration by sampling	Demonstration by sampling (e.g., fault injection)

### I.3 Medium-complexity component in B1 approach e.g. communication equipment

Medium-complexity OTS product intended to be integrated in SIS of class B shall be restricted to components that perform a limited, well-defined and well-understood set of functions. The narrow scope of the functionality and of the input space allows series of tests that will span the device input space and increase our confidence in the component.

The dependability assessment of communication equipment for category B applications may follow a grey box approach. The overall strategy may be summarised as follows:

- Indirect demonstrations by development process (e.g., quality assurance, application of general rules);
- Evidence of correctness of software is mainly based on demonstration by sampling in all stages following the implementation (i.e., unit testing, integration testing, validation testing, experience in operation, testing during pre-qualification);
- Demonstration by inspections may be difficult to achieve due to lacks of information in the development phases of the product. Whenever possible, access to the manufacturer for clarification of specific topics may be achieved, (e.g. information about filters that “remember” previous operational states would be required if statistical testing is to be undertaken).

<i>Properties of B1 medium-complexity component</i>	<i>Specification</i>	<i>Design</i>	<i>Implementation</i>	<i>Unit Testing</i>	<i>Integration Testing</i>	<i>Validation testing</i>	<i>Operation</i>	<i>Pre-qualification</i>
<b>Characterisation</b>								
• Identification	Demonstration by development process, e.g., by configuration management, by quality assurance Systematic proof, e.g., by intrinsic branding of main components							Demonstration by inspection, e.g. of the configuration management process, of the quality assurance plan
• Description								
• <b>Completeness</b> with respect to product features								Demonstration by development process: see section 7 on Functional assessment
• <b>Correctness</b> with respect to real product	See Correctness with respect to description							
• <b>Precision</b>							Demonstration by development process, e.g.,	<ul style="list-style-type: none"> <li>• use of glossaries for terms having a specific meaning</li> <li>• use of languages with well-defined syntax and semantics</li> </ul> Demonstration by inspection of description

<ul style="list-style-type: none"> <li>• <b>Accuracy</b>, depending on intended requirements</li> </ul>							Demonstration by inspection of description	
<ul style="list-style-type: none"> <li>• Integrity</li> </ul>	Systematic proof, e.g., by auto-diagnostic of integrity						Demonstration by inspection and sampling (testing for presence of auto-diagnostic function).	
<b>Confirmation of medium functional complexity status</b>							Confirmation by review based on criteria such as number of available functions, number of configuration parameters, number of inputs and outputs, presence or absence of volatile memory.	
<b>Correctness</b> with respect to description		Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>• Application of general design rules, notably for clarity and testability of architecture and behaviour, and possibly from relevant standards</li> <li>• quality of design documentation</li> <li>• rigorous and documented selection of OTS components, and cautious use of these components</li> <li>• design reviews and / or walkthroughs</li> <li>• traceability between specified requirements and design provisions</li> </ul>	Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>• application of general coding rules for clarity and testability, possibly from relevant standards</li> <li>• systematic proof of conformance to some coding rules</li> <li>• code reviews and / or walkthroughs</li> <li>• rigorous version and configuration management</li> </ul>	Demonstration by sampling, e.g., unit testing with structural and functional coverage criteria Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>• reporting of tests</li> <li>• rigorous version and configuration management</li> </ul>	Demonstration by sampling, e.g., integration testing with coverage criteria with respect to design provisions Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>• planning and reporting of tests</li> <li>• use of selected verification tools</li> <li>• independence of verifiers</li> </ul>	Demonstration by sampling, e.g., validation testing with coverage criteria with respect to specification, or with statistical criteria Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>• planning and reporting of tests</li> <li>• use of selected verification tools</li> <li>• independence of verifiers</li> </ul>	Demonstration by sampling, e.g., analysis of experience in operation	Demonstration by sampling, e.g., complementary tests (e.g. testing effects of timing and memory constraints, boundary value testing) Demonstration by inspection of development and quality assurance documentation, e.g.: <ul style="list-style-type: none"> <li>• guided design reviews or walkthroughs</li> <li>• assessment of quality assurance plans, integration plan, validation plan</li> </ul>

<ul style="list-style-type: none"> <li>Freedom from intrinsic SW faults</li> </ul>		<p>Systematic proof: by specific design rules avoidance of some types of intrinsic faults.</p>		<p>Demonstration by sampling (e.g., by testing) for intrinsic faults and for parts not covered by systematic proofs</p>				<p>Demonstration by sampling, e.g. testing over sets of input values representative of the full input range, testing with input failures and with input values out of range</p>
<ul style="list-style-type: none"> <li>Freedom from functional faults</li> </ul>		<p>Systematic proof by specific design rules to guarantee, e.g.,:</p> <ul style="list-style-type: none"> <li>Timeliness</li> <li>Sequencing</li> </ul>						<p>Demonstration by sampling of specific features e.g. for a communication module, testing concentrated on time sequencing of a series of messages, receiving and passing on messages, handling large volumes of messages</p>
<ul style="list-style-type: none"> <li>Freedom from SW / HW incompatibility</li> </ul>					<p>Demonstration by sampling (testing) for other HW/SW constraints, e.g.,:</p> <ul style="list-style-type: none"> <li>inputs and outputs</li> </ul>			<p>Demonstration by sampling, e.g. testing for normal operation, testing for recovery from supply interruption and restoration, testing for incorrect functioning because of radio frequency interference.</p>
<ul style="list-style-type: none"> <li>Conformance to design &amp; implementation documentation</li> </ul>		<p>Demonstration by development process, e.g., quality assurance</p>						<p>Demonstration by inspection</p>
<ul style="list-style-type: none"> <li>Freedom from hidden functions</li> </ul>		<p>Demonstration by inspection of design and source code</p>						<p>Demonstration by inspection of design documentation and operational records</p>
<p><b>Robustness</b></p>	<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>systematic identification of non-nominal conditions</li> <li>analysis of past experience</li> <li>analysis of tradeoffs</li> <li>specification of robustness</li> <li>critical review of specification of robustness</li> </ul>	<p>Systematic proof, e.g.,</p> <ul style="list-style-type: none"> <li>intrinsic robustness by design to given non-nominal conditions</li> <li>use of redundancy, diversification, independence and separation</li> </ul> <p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>documentation of design provisions for active robustness</li> </ul>	<p>Demonstration by development process, e.g.,</p> <ul style="list-style-type: none"> <li>use of defensive programming</li> <li>conformance to implementation rules aiming at robustness</li> </ul>		<p>Demonstration by sampling, e.g., injection of different types of faults</p>	<p>Demonstration by sampling (e.g., fault injection)</p>	<p>Demonstration by sampling</p>	<p>Demonstration by sampling (e.g., fault injection, avalanche/stress testing, sets of input values outside the normal range)</p>

- |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|
|  |  | <ul style="list-style-type: none"><li>critical design review of provisions for active robustness</li></ul> |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|

### I.4 Medium-complexity component in B2 approach e.g. communication equipment

The dependability assessment shall be performed in a similar manner to assessments of simple devices in the A2 or B2 approach (see section 9.1.3). For such an approach to be valid in the case of medium-complexity equipment, the data on experience in operation must be of high quality, relating to the same component version as that under investigation, and relating to an application that is very similar to the proposed application.

The dependability assessment of medium complexity components for category B applications may follow, whenever the conditions described just above are satisfied, a black box approach. The overall strategy may be summarised as follows:

- Evidence of correctness of software is based on demonstration by sampling, mainly by experience in operation and testing during pre-qualification;
- Characterisation and confirmation of medium complexity of the component is mainly supported by demonstration by inspection of the description.

<i>Properties of B2 medium-complexity component</i>	<i>Device development</i>	<i>Operation</i>	<i>Pre-qualification</i>
<b>Characterisation</b>			
• Identification	Demonstration by development process, e.g. by configuration management, by quality assurance		Demonstration by inspection, e.g. of the configuration management process, of the quality assurance plan
• Description			
• <b>Completeness</b> with respect to product features		Demonstration by sampling, e.g. identification of failure modes	Functional assessment (see section 7)
• <b>Correctness</b> with respect to real product			See Correctness with respect to description
• <b>Precision</b>			Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>• use of glossaries for terms having a specific meaning</li> <li>• use of languages with well-defined syntax and semantics</li> </ul> Demonstration by inspection of description
• <b>Accuracy</b> , depending on intended requirements			Demonstration by inspection of description
• Integrity	Systematic proof, e.g., by auto-diagnostic of integrity		Demonstration by inspection and sampling (testing for presence of auto-diagnostic function)
<b>Confirmation of medium functional complexity status</b>			Demonstration by inspection of the description performed by experts, e.g. guided review based on technical criteria (such as number of configuration parameters, of I/O, of interfaces handled, absence of volatile memory). Also inspection of the operational data to determine that it is of sufficiently high quality to compensate for lack of information about the development process and internal design.
<b>Correctness</b> with respect to description		Demonstration by sampling, e.g., analysis of experience in operation (so as to evaluate the probability of failure per demand, of failure during a certain period of time and of error containment)	<ul style="list-style-type: none"> <li>• Demonstration by sampling</li> <li>• analysis of operational experience</li> <li>• testing with functional coverage and statistical criteria (functional coverage and statistical criteria may depend on the functional complexity)</li> <li>• testing effects of timing and memory constraints</li> </ul>

			<ul style="list-style-type: none"> <li>• boundary value testing</li> <li>• specific features e.g. for a communications module this might be correct time sequencing of messages; correctly receiving and passing on messages so that none are lost; handling large volumes of messages.</li> </ul> <p>Demonstration by development process, e.g.:</p> <ul style="list-style-type: none"> <li>• quality assurance</li> <li>• development rules</li> </ul>
<b>Robustness</b>		<p>Demonstration by sampling, e.g.: analysis of experience in operation (e.g. failure analyses)</p>	<p>Demonstration by sampling, e.g.:</p> <ul style="list-style-type: none"> <li>• fault injection</li> <li>• avalanche/stress testing</li> <li>• sets of input values outside the normal range</li> </ul> <p>Demonstration by development process, e.g. rules of development.</p>

## I.5 Simple devices in A2 or B2 approach

The dependability assessment of simple devices for category A & B applications is usually a black box approach. The overall strategy may be summarised as follows:

- Evidence of correctness of software is based on demonstration by sampling, mainly by experience in operation and testing during pre-qualification;
- Characterisation and confirmation of low complexity of the device is mainly supported by demonstration by inspection of the description.

<i>Properties of A2 or B2 simple devices</i>	<i>Device development</i>	<i>Operation</i>	<i>Pre-qualification</i>
<b>Characterisation</b>			
• Identification	Demonstration by development process, e.g. by configuration management, by quality assurance		Demonstration by inspection, e.g. inspection of the configuration management development process, of the quality assurance plan
• Description			
• <b>Completeness</b> with respect to product features		Demonstration by sampling, e.g. identification of failure modes	Functional assessment (see section 7)
• <b>Correctness</b> with respect to real product			See Correctness with respect to description
• <b>Precision</b>			Demonstration by development process, e.g., <ul style="list-style-type: none"> <li>• use of glossaries for terms having a specific meaning</li> <li>• use of languages with well-defined syntax and semantics</li> </ul> Demonstration by inspection of description
• <b>Accuracy</b> , depending on intended requirements			Demonstration by inspection of description
• Integrity	Systematic proof, e.g., by auto-diagnostic of integrity		Demonstration by inspection of description, so as to ascertain the existence of a function of auto-diagnostic of integrity
<b>Confirmation of low functional complexity status</b>			Demonstration by inspection of the description performed by experts, e.g. guided review based on technical criteria (such as number of configuration parameters, of I/O, of interfaces handled, absence of volatile memory)
<b>Correctness</b> with respect to description		Demonstration by sampling, e.g., analysis of experience in operation (so as to evaluate the probability of failure per demand, of failure during a certain period of time and of error containment)	<ul style="list-style-type: none"> <li>• Demonstration by sampling</li> <li>• testing with functional coverage and statistical criteria (functional coverage and statistical criteria may depend on the functional complexity)</li> <li>• testing of influence of different HW constraints on SW (e.g. memory constraints)</li> <li>• boundary value testing</li> </ul> Demonstration by development process, e.g.: <ul style="list-style-type: none"> <li>• quality assurance</li> <li>• development rules</li> </ul>
<b>Robustness</b>		Demonstration by sampling, e.g.: analysis of experience in operation (e.g. failure analyses)	Demonstration by sampling, e.g.: <ul style="list-style-type: none"> <li>• avalanche/stress testing</li> </ul>

			<ul style="list-style-type: none"><li>• out of boundary value testing</li></ul> Demonstration by development process, e.g. rules of development.
--	--	--	--

## ANNEX II : Causes and possible defences against Common Cause Failures

### II.1 Error Propagation

Two cases are to be considered in this part:

- The errors of a safety-related system that can propagate SIS;
- The errors of systems not important for safety that can propagate SIS.

Error in a system may propagate to a SIS causing its failure. There are four possible ways for an error to be propagated:

- The propagation of incorrect data;
- The incorrect synchronisation in data exchanges;
- The aggression by another software;
- The denial of common resources.

#### II.1.1 Propagation of incorrect data

When a SIS communicates with other systems, a system where an error has occurred may propagate incorrect data to SIS that can receive those data and may cause its failure. Four defences have been identified to avoid the propagation of incorrect data:

##### II.1.1.1 *Communications avoidance*

The most obvious defence against the propagation of incorrect data is to avoid communication between the SIS and the other systems. This defence is efficient and give a strong argument for the safety demonstration but it is not always applicable when the systems really need to communicate. Moreover the absence of communication may be difficult to establish. For instance, the absence of communication link between SIS and others systems is not sufficient to prove that incorrect data cannot be propagated as data may be exchanged by many other means such as diskette or CDROM.

##### II.1.1.2 *One-way links*

Another defence to protect a SIS that need to communicate with another system from the propagation of incorrect data is to use a unidirectional link. In this way, the SIS that send data through the unidirectional link cannot be disturbed by the system which receive them. However, the sender can still have problem caused by incorrect synchronisation in the unidirectional link. This problem and the possible defences are discussed in section II.1.2.

The advantage with the unidirectional link mechanism is that it narrows the demonstration field to be done on the unidirectionality of the link what is normally easier but may still be difficult if the link is a complex object. On the other hand, this defence is useful when only the sender has to be protected, as it offers no protection to the receiver.

##### II.1.1.3 *Plausibility checks*

Plausibility checks is a generic name for different sort of checks that can be performed on the communicated data. This includes basic mechanisms such as range checks and checksum as well as more complicated mechanisms as correlations. Correlation consists in analysing several data values and in determining if some values are not coherent according to the knowledge of the process, e.g.:

- "Correlation in time", i.e. the data values sent by a device like a sensor are compared with the ones it has previously sent;
- "Correlation in space", i.e. the analyse of data values sent by several redundant devices allows to detect incorrect data;
- "Complex correlation", i.e. the analysis of data values sent by different devices that are functionally diversified allows to detect more incorrect data.

Plausibility checks can efficiently detect some incorrect data but are inoperative against incorrect but plausible data. The more complex are the correlation mechanisms the more incorrect data they are able to detect, but the difficulties to implement and prove them increase as well.

**II.1.1.4 Non disturbance of the functions important to safety**

Another approach to cope with CCF caused by propagation of incorrect data is to demonstrate that the functions important to safety cannot be disturbed by any kinds of incorrect data that they can receive. The assessment of the architecture, implementation and validation of the SIS may provide proof elements but do not have as strength as the formal demonstration such as static analyses of the software performed by adequate tools.

This approach is interesting because the proof is external to the SIS and thus does not increase SIS complexity. On the other hand, proof may be difficult to do.

**II.1.1.5 Recommendations**

Having no communication is the easiest way to avoid propagation of incorrect data but it is of no use when communication between systems is absolutely required. However, it is worthwhile to mention it to keep in mind that SIS have to communicate only by absolute necessity. In this case, one-way links may be used in the situation where a SIS has to communicate data to other systems that are not important to safety. The plausibility checks and the proof of non-disturbance needs to be used almost systematically in the situation where SIS have to communicate and when their applications are possible.

Possible defences	Benefits	Drawbacks
No communication	Absolute proof	Not always applicable
One-way links	Proof need to be focused on the unidirectionality of the link	If the link is a complex object, the proof may become difficult The unidirectionality protect only the sender, not the receiver
Plausibility checks on exchanged data: - Range - Checksum - Correlation, e.g.: * Correlation in space * Correlation in the time * Complex correlation	Simple to implement as long as simple correlation mechanisms are used	Inoperative against incorrect but plausible data; Complex correlation mechanisms may become difficult to implement and to prove
Non disturbance of the functions important to safety by any incorrect data (proof by static analysis)	Proof external to system	Proof may be difficult to do

**Table 1: Recommended defences against propagation of incorrect data**

**II.1.2 Incorrect synchronisation in data exchanges**

When a SIS exchanges some information with another systems, the situation may happen where the receiver is not able to receive the data when the sender sends them or the sender doesn't send the data when the receiver expects them. The exchanged data are perfectly correct but the synchronisation between the senders and the receivers are not and this may cause the failure of several systems. The most obvious defence is to have no communication between systems (see §II.1.1.1), but considering that systems absolutely need to communicate, two defences may be used to avoid incorrect interactions.

**II.1.2.1 Simple protocols with no synchronisation**

Incorrect synchronisation can be avoided if simple communication protocols are used where SIS can send or received data to/from others systems at any time without synchronisation. Those simple communication protocols are typically implemented by a hardware register, which guarantees that data may be written and read to/from the register at any time.

The advantage with this defence is that the demonstration has to be done on the simple hardware register, what is easier than on complex communication protocols.

**II.1.2.2 Fail safe protocols**

Another possibility is to design communication protocols in such a way that they are fail-safe in regard of safety and they have a reduced sensitivity to incorrect synchronisation (for instance by using specific mechanisms like message queuing).

The problem with this defence is that protocols that implement mechanisms like message queuing are complex and thus demonstrating that those complex protocols are fail-safe may be difficult.

**II.1.2.3 Recommendation**

The preferred defence against the incorrect synchronisation in data exchanges has to be the use of simple communication protocols where there is no need of synchronisation as long as those protocols are sufficient. However, simple protocols may be unsuitable for some needs so the use of more complex protocols cannot be excluded. In that case, the protocols have to be designed in such a way that they are fail-safe. However, demonstrating that complex protocols are fail-safe may be difficult.

Possible defences	Benefits	Drawbacks
Simple protocols with no synchronisation (guaranteed by an hardware register)	Proof is not very difficult as it has to be done on the simple hardware register	Not suitable for all needs
Fail safe protocols	Allows the use of more complex communication protocols that are sometimes needed	Demonstrating that complex protocols are fail safe may be difficult

**Table 2: Recommended defences against incorrect synchronisation in data exchanges**

**II.1.3 Aggression by another software**

In the situation where two or more applications that perform functions important to safety share some resources like memory spaces or IO boards, an application where an error has occurred may cause the failure of the others applications by writing in their memory spaces or IO boards. Two defences are possible against this kind of aggression.

**II.1.3.1 Guaranteed physical isolation**

The most obvious defence against the aggressions is to ensure the physical separation of resources. This defence is clearly efficient, but it may increase the overall complexity if the different functions need to communicate.

**II.1.3.2 Guaranteed logical isolation**

The other possibility to protect the applications against aggressions is to provide a logical separation of resources, generally ensured by an operating system. The main drawback of this defence is that demonstrating its efficiency is very difficult if not unfeasible as the mechanisms to implement it in the operating systems are very complex.

**II.1.3.3 Recommendation**

Physical isolation offers the best possible defence against aggressions by another software but it may lead to increase the complexity in the situation where the functions need to communicate. Logical isolation may increase the reliability when the safety functions become more complex, typically in SIS of safety class B. However

proving its efficiency is generally difficult due to the intrinsic complexity of the implementation. It is recommended to use physical isolation when independence between systems is very important. When independence is less important and functions become more complex, logical isolation may be used.

Possible defences	Benefits	Drawbacks
Guaranteed physical isolation (physical separation of resources, i.e.: memory, IO boards)	Absolute proof	Increase the overall complexity if the functions need to communicate
Guaranteed logical isolation (logical separation of resources, i.e.: memory, IO boards)	Increase the reliability when the safety functions become more complex, typically in safety systems of class B	Demonstrating the efficiency of logical isolation is generally difficult due to the overall complexity of its implementation

**Table 3: Recommended defences against aggression by another software**

#### II.1.4 Denial of common resources

After aggression, the other problem related to resources sharing between applications that perform functions important to safety is when an application where an error has occurred pre-empt the resources of the others applications and may cause their failures. Resources like CPU, memory, IO boards and network bandwidth may be pre-empted. The two possible defences are mostly the same than the defences against aggression presented in §II.1.3.

##### II.1.4.1 No common resources

The most obvious defence against CCF caused by denial of common resources is to have no common resources. This defence is clearly efficient and closely related to the physical isolation described in §II.1.3.1.

##### II.1.4.2 Guaranteed availability of resources

Guaranteeing the availability of resources is another way to avoid CCF caused by denial of common resources. This defence is related to the logical isolation presented in §II.1.3.2, but may be slightly easier to implement and to prove according to the type of resources. For instance, static memory allocation is a mechanism able to guarantee the availability of memory and its efficiency can be demonstrated. On the other hand, guaranteeing the availability of a shared CPU may be difficult.

##### II.1.4.3 Recommendation

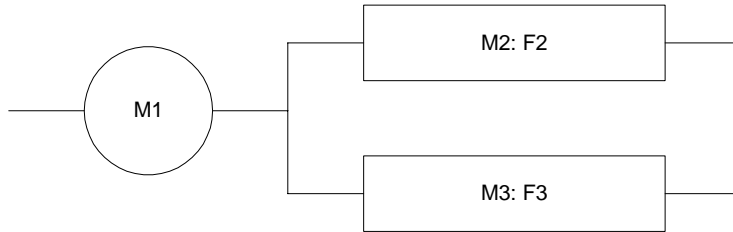
The overall guideline shall be to avoid common resources when independence between systems is very important. When independence is less important, mechanisms that can guarantee the availability of resources and thus prevent the resources pre-emption or exhaustion can be used. The efficiency of those mechanisms is more or less difficult to implement and to prove according to the type of the shared resource.

Possible defences	Benefits	Drawbacks
No common resources	Absolute proof	
Guaranteed availability of resources	Efficiency can be demonstrated for several resources (memory, network); Prevent resources pre-emption or exhaustion	Availability may be difficult to guarantee with resources like CPU

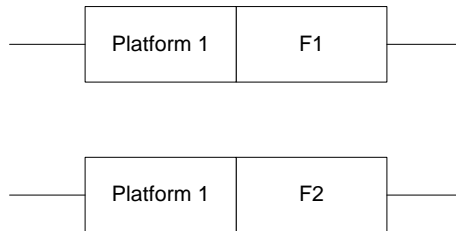
**Table 4: Recommended defences against denial of common resources**

## II.2 Common modules

Modules are said common when a single module is used by several systems such as in Figure 1: Common Module where module M1 is used by M2 to perform function F2 and by M3 to perform function F3. Modules are said identical when several modules from the same model are used by several systems such as in Figure 2: Identical Modules where the two identical platforms are used to perform function F1 and F2. CCF caused by failure in common modules are presented in section II.2.1 whereas CCF caused by fault in identical modules are considered as a problem of similarities in development processes and discussed in section II.3.3.



**Figure 1: Common Module**



**Figure 2: Identical Modules**

### II.2.1 Failure in common modules

Failure in any common modules leads systematically to failure of all the modules that required them.

The obvious and recommended defence against the failure in common modules is to avoid that kind of modules. In the rare situation where common modules cannot be avoided, they have to be developed with extremely rigorous development process and qualified with as rigorous validation procedures. Furthermore, all the consequences of the failure in common modules have to be carefully assessed to be able to guarantee that those failures are safe.

### II.2.2 Recommendation

Avoiding common modules must be the absolute rule for different lines of defence or redundancy channels. In the other cases, if common modules cannot be avoided, exceptional precautions have to be taken in their development process and also to guarantee that their failures are safe.

Possible defences	Benefits	Drawbacks
No common modules	Absolute proof	
Extremely rigorous development process and validation procedures with a careful assessment of the failure modes	May demonstrate that the safety principle of the plant are not jeopardise by the existence of common modules	Not applicable between different lines of defence or redundancy channels

**Table 5: Recommended defences against failure in common modules**

## **II.3 Similarities in development processes**

Several SIS may fail because of the same fault introduced by common factors in their development processes. The common factors identified by the IEC 60880-2[6] standard are the common specifications, design (architecture, algorithms), implementation methods, identical software modules (platform, tools), common development methods, staffing and management (including common human factors such as training, organization, thinking processes). These common factors are discussed from §II.3.1 to §II.3.5 and the possible defences, mostly based on diversity are presented. Finally, §II.3.6 summarize the defences that we recommend.

### **II.3.1 Incorrect common specifications**

Specifications or specification elements may be common for several software that performs functions important to safety. If those specifications are incorrect, incomplete or inaccurate this may lead to introduce the same kind of faults in those software. In some situations of the plant, the faults in the software may cause simultaneous errors and may lead to the failure of several safety functions.

The defence against incorrect common specifications is to apply functional diversity. The risk to introduce the same kind of fault in the software due to incorrect common specifications is reduced because few specifications may be common in diversified functions. Furthermore, functional diversity brings freely the diversity in the usage of identical modules (if identical modules are used) and also the signal diversity.

### **II.3.2 Error in common design & implementation methods**

Error in common design such as in the software architecture or in the algorithms used by several software that performed safety functions may lead to the failure of those functions. Error in common implementation methods such as the programming language may also lead to the failure of the safety functions performed by software written in the erroneous language.

A possible defence against error in common design and implementation methods is to diversify. Diversify the design approaches (software architectures, algorithms) and the implementation methods (programming languages). This defence may be useful especially when the specifications are sufficiently experienced and stable to be sure that they contain no error nor omit some elements. However, this defence may be difficult to apply as having different software architectures or algorithms for the same specifications has proven to be difficult. Furthermore, the algorithms used in safety functions of class A & B are so simple that they don't need to be diversified and no failure of SIS caused by error in software architecture or programming language has never been record.

### **II.3.3 Fault in identical software modules**

When identical software modules are used to perform several functions important to safety, a fault in the identical modules may lead to the failure of those functions. The software modules that are identical in the realization of safety functions are typically the automation platforms that include hardware, system software, compiler and libraries. The possible defences against CCF caused by fault in identical software modules are listed below.

#### **II.3.3.1 No identical modules**

The first defence is to have no identical modules by applying diversity in the implementations, i.e. using, for the same purpose, different modules with diversified implementation. Diversity in the implementation may be applied to:

1. The automation platform;
2. The application software that performs the safety functions in redundancy channels (n-version software).

Diversifying the implementation avoid that faults in the implementation of identical modules used in different safety systems failed simultaneously. It may be reinforced if the diversified implementations are done by diversified development teams (see §II.3.5). However, developing, maintaining and assessing the reliability of several automation platforms or application software is harder than to focus on a single ones so applying that diversity may lead to have diversified implementation of lower quality.

### **II.3.3.2 Rigorous development process and qualification**

Another way to avoid CCF caused by the use of faulty identical modules is to develop identical modules with a rigorous development process that aim to produce fault-free software and to qualify those identical modules with great care by extensive testing, extensive reviewing and / or static analysis.

### **II.3.3.3 Defensive programming and assessment of the failure modes**

Using defensive programming and assessing carefully the failure modes of identical modules to be able to guarantee that their failures are safe is another possible defence. However, the proof that all possible failure modes have been assessed may be difficult to do according to the complexity of the SIS. This defence has to be applied in conjunction with a rigorous development process and qualification.

### **II.3.3.4 Diversify the usage of identical modules**

Another possibility is to diversify the usage of identical modules to reduce the risk that a fault in those modules leads to a simultaneous failure of the functions that require them. As already mentioned in §II.3.1, this diversification comes naturally with functional diversity but it come also freely with the diversity in the implementation of application software.

### **II.3.3.5 Signal diversity**

According to the IEC 60880-2[6], the diversity in signals is the design of the channels or systems such that a coincident failure of two channels or systems is very unlikely because there is a demonstrable difference in the signal trajectories for the SIS. In practice the diversity in signal trajectories come naturally with the functional diversity but it is not the same for non-diversified functions. In fact, if a safety function needs information like the temperature as input signal, diversifying the sources that measure the temperature (sensors) will lead to signal trajectories for the temperature that will be only slightly different or not different at all. In any cases, that kind of difference will not be demonstrable as it results from small calibration differences between the sensors. Consequently, it seems that signal diversity offers no serious defence against CCF for non-diversified safety functions.

### **II.3.3.6 Asynchronous operation**

Designing the redundant channels using asynchronous operation is a possibility to show a defence when the same processors in different channels are subjected to identical trajectories. This defence is simple, can be applied without having any negative effect on the safety and aims to avoid that systems exposed to the same signal trajectories in two different channels failed exactly at the same time. However, if the erroneous conditions continue, both asynchronous systems will finally failed so this defence offers no serious arguments for the safety demonstration.

### **II.3.3.7 Recommendation**

Applying diversity in the implementation to avoid identical modules seems the natural way to cope with CCF caused by faults in those modules but it considerably increase the development needs and by the way may lead to have diversified modules of lower quality. Consequently, diversifying all the identical modules systematically is not recommended. For redundant channels, it is preferred to use rigorously developed and qualified identical modules for which appropriate techniques like defensive programming have been applied and failure modes have been carefully assessed to guarantee safe failures. It is recommended that the redundant channels run asynchronously even if it does not protect much against CCF because this defence can be easily implemented and has no negative impact on the safety. For different lines of defences, functional diversity can be applied so as to bring naturally diversity in the use of identical modules and to bring also signal diversity. In some very specific situations, like for the safety functions outside dimensioning, diversifying identical modules such as the automation platform is nevertheless a possibility.

## **II.3.4 Fault in common development methods**

According to the IEC 60880-2[6] standard, a fault in development method that are common to several software which perform functions important to safety may lead to the failure of those functions. Consequently, it proposes to diversify the design and that the two designs follow deliberately dissimilar development methods (preferably using different staff, see the next section). However, errors happen much more likely in the applying of development method than in the development method itself, so the diversity in development method is not

recommended. It is preferred to use a simple, well-mastered and applied development method and to ensure that expected results have been effectively achieved.

### **II.3.5 Mistakes repeated by common staffing and management**

The last common factors that may lead to CCF according to the IEC 60880-2[6] standard are the common staffing and management that include common human factors such as training, organisation and thinking process. The proposed defence is essentially to diversify the design with dissimilar development methods and different staffing with formal and restricted communication between the diversified staff. This defence aims to avoid mistakes reproduced by the same peoples and practices. However, it does not offer protection against common human factor such as training and thinking process as people in the different teams may have read the same books containing errors or followed the same erroneous training programs.

There are two tasks that are significantly different in the development process: design & implementation on the one hand, and tests & validation on the other hand. The two tasks may be performed by the same teams or by separate development team and validation team. The development team itself may be diversified, what leads necessarily to diversify the design and implementation. The validation team may also be diversified and this does not necessary leads to diversify the design and implementation.

Globally, CCF caused by mistakes repeated by common staffing and management are not specific to programmed systems and have to be considered for the design of the overall I&C system. It is recommended to use a validation team fully independent from the development team for the simple safety systems (class A). For the more complex safety systems (class B), it seems appropriate to integrate some people from the development team in the validation team, but to have still too different teams with fully independent responsible. Diversifying the development teams may be difficult as different qualified teams are not necessarily available. Moreover it will lead to high costs to solve few problems as most of the problems in SIS are not in the design and implementation (see §II.3.2).

### **II.3.6 Overall Recommendations**

Experience shows that software specifications are the most important source of systematic failures in systems important to safety. As a consequence, functional diversity is the only defence that address the most important source of systematic failure in SIS. Furthermore, functional diversity if implemented into independent lines of defence brings naturally the diversification in the use of identical modules and also in the signal trajectories. It is not recommended to systematically diversify the automation platform between different independent lines of defence as the use of the identical platform is diversified and as long as the platform itself is rigorously developed and qualified. However, the diversification of the automation platform cannot be excluded for some specific situations.

Between different redundant channels, it is preferable to use of a rigorous development process and qualification with some independence between the development team and the validation teams and to apply appropriate mechanisms such as defensive programming to guarantee safe failure. It is not systematically recommended to diversify the design and the implementation of the safety functions because those diversifications have a high cost to solve very few problems as no failure caused by problems in the design or implementation has never been deplored in SIS.

Possible defences	Benefits	Drawbacks
Rigorous development process and qualification	Leads opfully to obtain fault-free software	
Defensive programming and assessment of the failure modes	Guaranteed safe failures	The proof that all possible failure modes have been assessed may be difficult to do according to the complexity of the SIS
Functional diversity which bring naturally the diversification of the usage of identical modules and of the signals	Functional diversity is the only defence against incorrect specifications which are the most important sources of systematic failures in SIS	Not always applicable

**Table 6: Recommended defences against fault introduced by similarities in the development process**

## II.4 Exceptional conditions

The occurrence of exceptional conditions may lead to failures of several systems if those SIS are sensitive to those conditions. The most common sources of sensitivity are the sensitivity to plant demands and the sensitivity to dates, especially calendar dates.

### II.4.1 Sensitivity to plant demands

The concern about the sensitivity of the safety functions to the plant demands is that those safety functions may be overloaded by too much information and then several may fail at the same time.

The recommended method to avoid this kind of CCF of is to design safety functions to be independent of plant demands. This means that the safety functions must use only the strictly required information but especially that it must be up to them to query those information from the plant at their own rate, by mechanisms like polling.

### II.4.2 Sensitivity to date

The normal approach to avoid date problems in SIS is first to use dating mechanism only by absolute necessity and second to have dates reset regularly by short periods of time. This leads to have a cyclic behaviour where the safety functions are exactly in the same conditions by short periods of time and then may be more easily tested and assessed. Note that the proof of safety may still be difficult to do if many dating mechanisms, even reset regularly by short periods of time, are used in SIS.

The calendar dates must not be used in safety system. Most if not all the time, calendar dates are not really required in SIS themselves. However, it may be interesting to log the events that happen in the SIS and in this case, dating is clearly attractive. It is suggested that such kind of logging mechanisms has nothing to do in SIS but have to be implemented in external systems. The SIS may be, for instance, connected to those external systems by unidirectional links.

### II.4.3 Recommendation

The purpose of all the defences described in this section is to prevent the SIS from being sensitive to the exceptional conditions and all of them are recommended.

Possible defences	Benefits	Drawbacks
Design SIS in a way that they query themselves the strictly required information from the plant at their own rate	Ensure that safety functions can't be overflowed during demand situation in the plant	
Use dating mechanisms by absolute necessity and reset dates regularly by short periods	Dating mechanisms that are reset regularly are easier to test and assess so proving their safety is possible	The proof of safety may still be difficult to do if many dating mechanisms, even reset regularly by short periods of time, are used in SIS
Design SIS in a way that their operational behaviour is free from dependencies to calendar dates	Allow easy proof that safety functions can't be affected by any calendar dates	

**Table 7: Recommended defences against sensitivity to exceptional conditions**

## II.5 Maintenance activities

The maintenance activities are the actions conducted by the maintenance operators on the systems of the plant. Inappropriate maintenance activities may lead to jeopardize the overall safety of the plant. This may happen by putting too many systems in maintenance at the same time or by operator mistakes that may cause the failure of several systems in the plant. For instance, an operator may have to shut down the power supply of one system to perform maintenance on it, but by mistake shut down the power supply of several other systems as well causing the failure of those systems. Note that the CCF caused by faults that may have been introduced after some modifications performed either on the hardware or on the software used in the plant are not considered as CCF caused by maintenance activities but as CCF caused by similarities in the development process.

CCF caused by maintenance activities are not specific to programmed systems and shall be addressed globally for the whole plant so they will not be discussed here in depth. The definition of global and individual maintenance plans during the design process of the plant is the basis to avoid CCF caused by inappropriate maintenance activities. To enforce the strict application of the maintenance plans, specific mechanisms may be used. For instance locking mechanisms may be used to prevent the simultaneous access to systems that should not be in maintenance at the same time. Those locking mechanisms may be physical but with digital I&C systems, access control and online monitoring mechanisms may also be found. However, access control and online monitoring mechanisms are rather complex and thus integrating them in SIS would dramatically increase the system complexity. Consequently, it is recommended to exclude those mechanisms from the SIS.