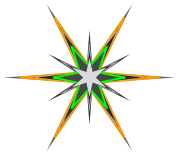

S T S A R C E S

Standards for Safety Related Complex Electronic Systems

Annex 2

Tools for Software fault avoidance

Task 1.1: Software Fault Avoidance through Quality

Final Report of WP1.2

Philippe Charpentier

INRS

European Project STSARCES
Contract SMT 4CT97-2191

Contents

SUMMARY:

This document outlines an optimal set of requirements for the production of embedded software used to ensure safety-related functions of the machinery sector. It does not replace the IEC 61508 Standard (especially part 3 "Software Requirements"), which served as a basis but provides a set of basic requirements, coherent with this standard, adapted to the machinery software products usually developed by SME/SMIs.

The requirements focus on the following points:

- Software product: Requirements describe the main characteristics which an embedded software should possess to guaranty its quality and safety.
- Software development process: Requirements are established for all technical activities associated with software development, for those involved in software design. These can then be used to guide the designer during the production of this type of software.
- Software verification: A reference framework is given for software evaluation. The requirements should allow the analyst to decide on the fitness of an embedded software to satisfy the safety requirements of the target system to be analysed.

1. INTRODUCTION

1.1. OBJECTIVES

This document outlines an optimal set of requirements for the production of software used to ensure safety-related functions within a given system.

The major objective dealt with here, through the application of these requirements, is the prevention of software failures and any other unexpected behaviour that might lead to the creation of dangerous faults in the target system.

In order to satisfy these objectives, the requirements discussed here focus on the following points:

- a description of the main characteristics which a finished software product should possess to guaranty its quality and safety (software product requirements),
- the establishment of the requirements imposed on all technical activities associated with software development, for those involved in software design. These can then be used to guide the designer during the production of this type of software (software development process requirements).
- a reference framework for software evaluation. These requirements should allow the analyst to decide on the fitness of a software product to satisfy the safety requirements of the target system to be analysed (software verification requirements).

1.2. TARGET AREAS

The requirements outlined in the present document concern software products that are part of a control system used to ensure safety functions.

They can be used for all types of machinery, regardless of the technology involved and whether or not the system parameters can be defined by the user, provided the safety functions are ensured by software and that these functions are not of an especially high criticality (e.g. safety functions in the area of nuclear power or aeronautics are excluded).

This document does not replace the IEC 61508 Standard (especially part 3 "Software Requirements"), which served as a basis. It provides a set of basic requirements, coherent with this standard, adapted to the machinery software products usually developed by SME/SMIs.

1.3. INSTRUCTIONS

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

determining the software requirement level, see Appendix B of the "Guide to evaluating software quality and safety requirements[1]").

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

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

It should be the system designer's responsibility to demonstrate to the analyst that all applicable requirements have been met in the software to be evaluated. Each of these elementary requirements should thus solicit an appropriate response, by means of either a document produced for the software or activities carried out during software development, results of which should be kept for evaluation by the analyst. For example, it is important to note that a validation testing activity such as informal debugging (with no paper or computational evidence) with a logic analyser does not constitute proof of a given test. This should not prevent the designer from carrying out this type of activity if it is needed during a specific development phase.

It is recommended that the system designer take these requirements into consideration from the outset of the software development procedure when the intervention of the analyst takes place before the beginning of the development itself. These requirements could give rise to contractual quality clauses when the system designer signs a contract with a software provider.

All texts in *italics* in the sections on requirements are informative comments intended to define the objectives of the requirements in question precisely, clarify the manner in which they are to be understood, and lay down any possible limitations on their application.

No.	Coordination with the analyst for software evaluation	Level	
		1	**2**
1.1	Any deviations from the requirements presented in this document should be pointed out by the applicant to the analyst and should be approved by the latter. *The applicant should request a meeting with the analyst.*	O	O

[1] GUIDE TO EVALUATING SOFTWARE QUALITY AND SAFETY REQUIREMENTS
STSARCES Project - WP 1.2 / Aspect 2 – Final Report - INRS – Fev 2000

1.4. OVERVIEW OF DOCUMENT

In addition to this introductory section, this document contains three major sections :

- Section 2: requirements concerning the software product :

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

- Section 3: requirements concerning the software development process:

  - Development process
  - Organisation
  - Documentation
  - Configuration and software modification management
  - Tools
  - External subcontracting
  - Reproduction, delivery

- Section 4: requirements concerning software verification and validation :

  - General verification and validation requirements

  - Verification requirements:

    - General verification requirements
    - Review requirements
    - Code verification (source code and data) requirements

  - Software test requirements:

    - General validation requirements
    - Software specification verification requirements: validation tests
    - Software design verification requirements: software integration tests
    - Detailed design verification requirements: module tests.

An appendix to this document contains coding rules, in particular for C and assembly language - the languages the most commonly used in the preparation of the type of software considered in this document.

2. SOFTWARE PRODUCT REQUIREMENTS

2.1. PRESENTATION

This chapter presents the requirements a software product should possess if it is to be fully safe in operation and of satisfactorily high quality. To obtain such a software product, a number of activities, a certain organisation and a number of principles must all be established. This should take place as early as possible in the development cycle (different requirements and recommendations concerning the development process are given in the following chapter).

Software product requirements should cover:

- Interface with the architecture of the system integrating the software
- Software specification
- Software that can be parametrized by the user
- Pre-existent software
- Software design
- Development languages
- Software coding (coding rules are presented in Appendix).

2.2. LIST OF REQUIREMENTS

2.2.1. Interface with system architecture

No.	Interface with system architecture	Level	
		1	**2**
1.2	Software safety requirements as well as the determination of expected events should arise from safety analyses at system, functional and hardware level, etc.	/	O
1.3	The list of constraints imposed by hardware architecture on software should be defined and documented. Consequences of any hardware/software interaction on the safety of the machine or system being monitored should be identified and evaluated by the designer, and taken into account in the software design. *Constraints such as :* *- protocols and formats,* *- input/output frequencies,* *- by rising and falling edge or by level,* *- input data using reverse logic etc.* *Listing these constraints allows them to be taken into account at the start of the development activity, and reduces the risk of incompatibilities between software and hardware when the former is installed in the target hardware.* *The consequences of any software errors should be studied at system level in particular.*	O	O

2.2.2. Software Specifications

No.	Software Specifications	Level	
		1	**2**
1.4	Software specifications should take the following points into account: a) safety functions with a quantitative description of the performance criteria (precision, exactness) and temporal constraints (response time), all with tolerances or margins when possible, b) non safety functions with a quantitative description of the performance criteria (precision, exactness) and temporal constraints (response time), all with tolerances or margins when possible, c) system configuration or architecture, d) instructions relevant to hardware safety integrity (programmable electronic systems, sensors, actuators, etc.), e) instructions relevant to software integrity and the safety of safety functions, f) constraints related to memory capacity and system response time, g) operator and equipment interfaces, h) instructions for software self-monitoring and for hardware monitoring carried out by the software, i) instructions that allow all the safety functions to be verified while the system is working (on-line testing). *The instructions for monitoring, developed taking safety objectives and operating constraints (duration of continuous operation ,etc.) into account, can include devices such as watch dogs, CPU load monitoring, feedback of output to input for software self-monitoring. For hardware monitoring, CPU and memory monitoring, etc. Instructions for safety function verification: for example, the possibility of periodically verifying the correct operation of safety devices should be included in the specifications.*	O	O
1.5	Functional requirements should be specified for each functional mode. The transition from one mode to the other should be specified *Functional modes can include:* *- nominal modes,* *- one or more degraded modes.* *The objective is to specify the behaviour in all situations in order to avoid unexpected behaviours in non-nominal contexts.*	O	O

2.2.3. Software that can be parametrized by the user

The following requirements concern the development of software products that are designed to allow end-users to set their own parameters.

The end-user may be either the person responsible for integrating the product into the system or else the user.

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

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

No.	Software that can be parametrized by the user	Level	
		1	**2**
1.6	The parameters should be formally specified (type, relations, …) in the form of memory arrays. Moreover, the software and the parameters should be capable of independent evolutions.	R	R
1.7	Software specifications should define mechanisms that can be used to prevent the possibility of any parameters set by the user can affect the system safety. In so far as modifiable parameters are concerned, these mechanisms should provide protection against : - undefined or invalid initial values, - values falling outside functional limits**,** - data alteration. The definition of software parameters by users should be kept within the limits established by the system specifications approved by the analyst. *In particular, test procedures should include different parameter values and different types of software behaviour. The designer should ensure that only those parameters which can be modified are accessible to the user.*	R	O

2.2.4. Pre-existent software

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

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

No.	Pre-existent Software	Level	
		1	**2**
1.8	The designer should indicate the use of pre-existent software to the analyst, and it is the designer's responsibility to demonstrate that pre-existent software has the same level as the present requirements. Such a demonstration should be done: - either by using the same verification activities on the pre-existent software as on the rest of the software, - or through practical experience where the pre-existent software has functioned on a similar system in a comparable executable environment (e.g. it is necessary to evaluate the consequences of a change of the compiler or of a different software architecture format). *The goal of indicating the use of pre-existent software is to open up consultations with the analyst as early as possible about any eventual difficulties that this type of software might cause.* *The integration of pre-existent source modules can be the cause of certain anomalies or unsafe behaviour if they were not developed with the same rigour as the rest of the software .*	O	O
1.9	Pre-existent software should be identified using the same configuration management principles that were applied to the rest of the software. *Perfect configuration control should be exercised over all the software components, regardless of their origin.*	O	O

2.2.5. Software design

No.	Software Design	Level	
		1	**2**
1.10	Description of the software design should include at the very least: - a description of the software architecture that defines the structure decided on to satisfy specifications, - a description of inputs and outputs (e.g. in the form of an internal and external data dictionary), for all the modules making up the software architecture, - sequencers and interruptions, - global data, - a description of each software module (inputs/outputs, algorithm, design particularities, etc.), - libraries used, - pre-existent software used.	O	O
1.11	Software should be modular in order to facilitate its maintenance: - each module or group of modules should correspond, if possible, to a function in the specifications - interfaces between modules should be as simple as possible. *The general characteristic of correct software architecture can be summed up in the following way: a module should possess a high level of functional cohesion and a simple interface with its environment*	O	O
1.12	Software should be designed to limit those parts associated with safety: - data/functional architecture: strict limitation of global variables, implementation of operators on state variables (visibility), - control of the layout of arrays in memory (risk of array overflows).	O	O

2.2.6. Development languages

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

- an assembly language, specific to the microprocessor or microcontroller employed,
- an advanced programming language such as C.

No.	Development Languages	Level	
		1	2
1.13	The selected programming language should correspond to the characteristics of the application, and should be fully and clearly defined or at least limited by clearly defined characteristics. *The characteristics of the application refer to such factors as size, type (industrial or scientific software, management, etc.), and any performance constraints. For example, COBOL does not satisfy the development requirements of a monitoring/control application on an industrial machine.* *Any deficiencies in the language can be avoided using appropriate coding rules.*	R	O

2.2.7. Coding

No.	Coding	Level	
		1	2
1.14	The source code should: a) be readable, understandable, and subject to tests, b) satisfy design specifications of the software module, c) obey the coding manual instructions.	O	O
1.15	The coding rules applicable to a given software product should be outlined in detail in a coding manual and used to develop the software. The coding manual should : - indicate what programming principles should be applied and prohibit any uncertain language aspects, - describe rules for source code presentation and documentation, - include conventions used in naming components, subroutines, variables and constants. *A set of rules are provided in Appendix A of this document. For example, indented presentation of different blocks of instructions, use of blank lines, contents of the source file header (author's name, input and output data, modified data, etc.).* *These conventions help improve software legibility and maintenance.*	R	O

3. SOFTWARE DEVELOPMENT PROCESS REQUIREMENTS

3.1. DEVELOPMENT PROCESS

3.1.1. Software Lifecycle

3.1.1.1. Presentation

A software project begins with the specification of the necessary and sufficient conditions for the design and production of a product. This organisation, specific to each project, forms the software lifecycle.

The activity schedule is defined in relation to the specific aspects of the project, for example, complexity of the system and the software under consideration, use of previously developed software products, production of a prototype or availability of test equipments.

The basic objectives of requirements relevant to software development planning (definition of the development phases, means required, etc.) are to orient development in such a way that it can satisfy the functional specification requirements of the target system and inspire the confidence necessary to satisfy safety constraints.

These requirements deal with the following points:

- **software lifecycle** that is used to define and coordinate software development activities, and all activities associated with development (documentation, configuration and modifications management, etc.) by adapting them to the system under consideration
- **choice of the development environment** and especially development tools,
- **planning of other activities** (verification, quality assurance, etc.),
- **certain additional requirements** that must be taken into account at the beginning of the development cycle (coordination with the analyst and external subcontracting).

3.1.1.2. Activities in the lifecycle

Software definition usually includes the following sequential development activities:

- specification
- design
- coding
- executable code production

These definition activities are complemented by software verification activities, the goal of which is:

- to verify that the software product satisfies specified requirements at each of the different stages of the development,

- to detect any errors that might have been introduced when developing the software.

The principal element of any software verification is testing. Other activities such as reviews or analyses (for example rereading the code) are possible aspects of software verification.

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

- module tests
- software integration tests
- software validation tests

3.1.2. Software lifecycle requirements

The goal of the following software lifecycle requirements is to obtain a formalised description of the organisation of software development and, in particular, the different technical tasks making up this development. This description promotes improved planning of the development activities and more thought being given to the optimal time schedule for this development. For the purposes of the accompanying documentation, this description can be grouped with the description of quality assurance steps.

No.	Software Lifecycle	Level	
		1	**2**
2.1	The software development lifecycle should be specified and documented (e.g in a Software Quality Plan). The lifecycle should include all the technical activities and phases necessary and sufficient for software development.	O	O
2.2	Each phase of the lifecycle should be divided into its elementary tasks and should include a description of: - inputs (documents, standards etc.), - outputs (documents produced, analytical reports, etc.), - activities to be carried out, - verifications to be performed (analyses, tests, etc.).	O	O

3.2. ORGANISATION

3.2.1. Software Quality Assurance Requirements

The goal of quality assurance activities is primarily to ensure that the software product, its documentation, and the lifecycle activities all conform to the present requirements and that any deficiencies are detected, evaluated, monitored and resolved.

No.	Software Quality Assurance	Level	
		1	2
2.3	The programme used to guarantee software quality should be well-documented (e.g in a Software Quality Plan) and include at least: - the organisation, the people who are responsible for quality assurance, development and tests, and the required independence, - the quality assurance activities included in the software lifecycle (*examples of methods, reviews, inspections*), - any documents produced (*reports, etc.).* *This documentation can be assembled by referencing any applicable internal documents used for an entire set of projects simply by specifying what has been used for the project in question .*	R	O

3.2.2. Safety supervision and management requirements

Safety supervision and management tasks are used to follow / monitor the setting up of safety measures as the project develops, and to ensure that all the expected activities take place. They should keep abreast of the availability of resources, the monitoring of critical junctures in the project and eventually, the follow-up of any subcontractors in so far as software safety is concerned. This should be done for all the information contained in the project schedule.

No.	Safety Supervision and Management	Level	
		1	2
2.4	Safety supervision should be a permanent activity while the software is being produced. *By continually monitoring the progress in software safety, it is possible to take any action on the conducted activities quickly. This is necessary to prevent safety from drifting .*	R	R

3.3. DOCUMENTATION

3.3.1. Documentation management requirements

Certain requirements specific to the documentation or to certain documents are presented below.

The aim of these requirements is to establish a common basis of reference for the applicant or organisation (designer in general) and the analyst who is evaluating the system.

It should be noted that all the information collected by the analyst during audits or through consultation of related documentation must always be treated as confidential.

No.	Documentation Management	Level	
		1	2
2.5	The list of documents to be produced should be defined at the outset of the project (e.g in a Software Quality Plan) *The creation of such a list ensures that the corresponding tasks are also planned and therefore constitutes a guarantee that all the expected documentation exists.*	O	O
2.6	Each document should at least: - be identified in a unique way (*reference, version, revision index*), - be dated, - carry a title that indicates the scope of its content and that sets the document in the context of the documentation as a whole. (*specification, design, etc.*), - be written in the language mutually agreed by the applicant and the analyst. Furthermore, any subsequent changes to the documents should follow established guidelines (management of revision indices, etc.), and all documents should be available in their definitive version when the final software evaluation is undertaken by the analyst. *The requirement concerning the langage in which the documents are to be written enables the analyst to carry out his evaluation with documents that the personnel can understand. If this is not the case, the applicant runs the risk of having the evaluation refused.* *The requirement concerning subsequent changes to documents is intended to maintain the quality level achieved with the initial version as new documentation versions appear.* *Provisional versions of the documents can be provided with the agreement of the analyst if he or she intervenes during the development phases.*	O	O
2.7	The necessary documentation should be established at each phase of the lifecycle to facilitate verification and validation, and the software safety requirements should be traceable and capable of being verified at each stage of the process (traceability matrix for each definition document) *This will avoid a situation where the only available documentation is the source code because the documents that should have been prepared were not (deadlines too tight, project manager transferred to another contract, etc.).*	R	O

3.4. CONFIGURATION AND SOFTWARE MODIFICATION MANAGEMENT

Management of the configuration and therefore of the version is indispensable to any development which requires approval. Indeed, approval is only valid where a given configuration has been identified. Configuration management includes configuration identification activities, modification management, the establishment of reference points and the archiving of software products, including the associated data (documents, records of tests,

etc.). Throughout the entire project lifecycle, the principal objectives of these requirements are to:

- *provide a defined and controlled software configuration that can be used to reproduce an executable code coherently (with future software production or modification in mind).*
- *provide a recognised reference basis for progress evaluation and for modifications management,*
- *provide a means of control which guarantees that any problems are properly analysed, and that the approved modifications are carried out,*
- *guarantee physical archiving and reproduction possibility of the software*

3.4.1. Configuration and archiving management requirements

No.	Configuration and Archiving Management	Level	
		1	**2**
2.8	A procedure for configuration management and modifications management should be defined and documented. This procedure should, as a minimum, include the following items: - articles managed by the configuration, at least : . software specification, . preliminary and detailed software design, . source code modules, . plans, procedures and results of the validation tests. - identification rules (of a source module, of a software version, etc.), - treatment of modifications (recording of requests, etc.). For each article of configuration, it is necessary to be able to identify any changes that may have occurred and the versions of any associated elements. *The purpose of this is to be able to trace the historical development of each article: what modifications have been made, why, and when?*	O	O
2.9	Software configuration management should allow a precise and unique software version identification to be obtained. Configuration management should associate all the articles (and their version) making up a software version . *The aim of this requirement is to avoid errors such as modification of a module that is not in the latest code version or delivery of an untested code version to the client, etc.*	O	O

No.	Configuration and Archiving Management	Level	
		1	2
2.10	All articles in the software configuration should be covered by the configuration management procedure before being tested or being requested by the analyst for final software version evaluation. *The objective here is to guarantee that the evaluation procedure is performed on software with all elements in a precise state. Any subsequent change will require revision of the software and will thus be identifiable by the analyst.*	O	O
2.11	Procedures for the archiving of software and its associated data should be established (methods for storing backups and archives). *These backups and archives can be used to maintain and modify software during its functional lifetime.*	O	O

3.4.2. Software modifications management

No.	Software Modifications Management	Level	
		1	2
2.12	Any software modification is subject to the rules established for modification and configuration management, and requires that the development process be recommenced at the highest "upstream" point needed to take the modification into account. *In particular, the documentation should also be updated, and all necessary verification activities carried out. This guarantees that the software will retain all its initial properties after any modification.*	O	O
2.13	The description of software modifications should include details of each modification made. This should include at least the following items for each modification: a) the modification request, b) the report detailing the analysis of the impact of the software modification, the decisions made in this respect and their justification, c) the version of the software to be modified as well as the configuration articles and their version. *This minimal degree of formalism ensures that modifications are introduced by the different individuals working on the project using properly controlled procedures and in a concerted, systematic fashion. By so doing , it is possible to define the tests to be carried out precisely - something that would be impossible if the exact state of the modifications and their impact had not previously been established.*	R	O

3.5. TOOLS

3.5.1. Development tool requirements

No.	Development Tools	Level	
		1	**2**
2.14	Optimisation of object code performance options are forbidden. *Any optimisation by the compiler might modify the size of the resultant code and its speed of execution. Since the instructions generated differ according to the options selected, test representiveness might be jeopardised if options are used.*	R	O
2.15	If a new compiler or a new linker is used during the development procedure, the validity of the testing activities already performed should be analysed by the designer. *The instructions generated might be different, and test representiveness might be altered. Only an analysis of the situation will remove the uncertainty caused by this.*	R	O
2.16	Tools used during the development procedure (compiler, linker, tests, etc.) should be identified (name, reference, version, etc.) in the documentation associated with the software version (e.g. in the Version Sheet). *Different versions of tools do not necessarily produce the same results. Precise identification of tools thus directly demonstrates the continuity of the process of generation of an executable version in the event that a version is modified.*	O	O

3.6. EXTERNAL SUBCONTRACTING

The aim of these requirements is to guarantee that the entire software respects the present set of requirements, whether it is subcontracted to several different firms or not. The software designer who signs a contract with external subcontractors should take all the necessary steps (e.g. software quality clauses) to ensure that any software not directly developed by his or her firm respects the requirements applicable to the entire software version integrated into the system under consideration.

3.6.1. External subcontracting requirements

No.	External Subcontracting	Level	
		1	**2**
2.17	In the event that any part (even partially) of the software development is subcontracted to a third party, the present requirements should also apply to the subcontractor. They may possibly be adapted to reflect the importance and nature of the subcontracted tasks.	O	O
2.18	The designer should ensure and demonstrate that the requirements have been respected by the subcontractor(s).	O	O

3.7. REPRODUCTION, DELIVERY

3.7.1. Executable code production requirements

The objective of the following activity is to enable the production of an executable object code that will be loaded into the target system.

No.	Executable Code Production	Level	
		1	**2**
2.19	Any option or change in the generation, during the software production should be recorded (e.g. in the Version Sheet) so that it is possible to say how and when the software was generated.	O	O

3.7.2. Software installation and exploitation requirements

No.	Software Installation and Exploitation	Level	
		1	**2**
2.20	All failures linked to safety and dependability functions brought to the attention of the designer of the system should be recorded and analysed. *This means that the designer is aware of any safety and dependability-related failures that are communicated to him and that he takes the appropriate action (e.g. warning other users, software modification, etc.).*	O	O

4. SOFTWARE VERIFICATION AND VALIDATION REQUIREMENTS

4.1. PRESENTATION

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

Software verification is not simply a series of tests, even though this is the predominant activity for the relatively small software considered in this text**. Other activities such as reviews and analyses, whether associated with these tests or not, are also considered to be verification activities**. In certain cases they can replace some tests (e.g. in the event that a test cannot be carried out because it would cause deterioration of a hardware component).

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

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

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

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

- **validation tests** allow verification that the software installed in the hardware satisfies the functional specifications, especially by verifying hardware/software interfaces, general performance levels, real-time operation, general functions, use and allocation of resources.

The following chapters present the different requirements concerning:

General verification and validation requirements

- Verification requirements :

  - General verification requirements

  - Review requirements

  - Code verification (source code and data) requirements

- Software test requirements :

  - General validation requirements

  - Software specification verification requirements: Validation tests

  - Software design verification requirements: Software integration tests

  - Detailed design verification requirements: Module tests

4.2. GENERAL VERIFICATION AND VALIDATION REQUIREMENTS

No.	General Verification and Validation Requirements	Level	
		1	**2**
3.1	The analyst should be able to carry out the evaluation of software conformity to the present requirements by conducting any audits or expertises deemed useful during the different software development phases. All technical aspects of software lifecycle processes are subject to evaluation by the analyst. The analyst must be allowed to consult all verification reports (tests, analyses, etc.) and all technical documents used during software development. *The intervention of the analyst **at the specification phase** is preferable to an a posteriori intervention since it should limit the impact of any decisions made. On the other hand, financial and human aspects of the project are not subject to evaluation.* *It is in interest of the applicant to provide proof of all activities carried out during software development.* *The analyst should have all the necessary elements at his or her disposal in order to formulate an opinion. Subcontracted software should not be left out of the evaluation procedure.*	O	O

No.	General Verification and Validation Requirements	Level	
		1	2
3.2	Evaluation of software conformity to the present requirements is performed for a specific, referenced software version. Any modification of previously evaluated software which has received a final opinion from the analyst should be pointed out to the latter in order that any additional evaluation activities can be carried out to update this opinion . *Any modification can modify software behaviour; the opinion delivered by the analyst can therefore only be applied to a precise software version.*	O	O

4.3. VERIFICATION REQUIREMENTS

As well as tests, verification activities can also include techniques such as reviews, inspections, checks, cross-readings and code verification.

4.3.1. General verification requirements

No.	General Verification Requirements	Level	
		1	2
3.3	A verification report should be produced for each verification activity, and should identify and document all distortions (non-conformities) with respect to: - the corresponding specifications, - rules or standards (design, coding), - any quality assurance procedures that may exist. *The goal of this requirement is to record any identified non-conformities in order to be able to correct them (either immediately in the case of non-conformities that are unacceptable from an operational or safety point of view or at a later date on another software version if the non-conformity is only minor).*	R	O

4.3.2. Review requirements

Internal reviews at key points in the development process allow the designer to ensure that the product will achieve the objectives set.

No.	Review Requirements	Level	
		1	2
3.4	An external specification review (with the analyst) should be held at the end of the software specification phase. Activities involving analysis and software specification verification should: - verify the exhaustiveness and adequacy of the software specifications with respect to the system specifications, - verify the traceability with respect to the system specifications. *The software specification review should ensure that the real needs were taken into account in the specifications, that the technical risks have been identified, resolved and reduced by appropriate choices, and that it has been verified that the software specifications satisfy the system specifications.* *Analysis activities ensure that software specifications are coherent with system specifications, that they are complete, and that the connection between the software and system specifications has been clearly established.*	R	O
3.5	Analysis activities and software design verification should verify the conformity to specifications. *Here the purpose is to ensure that the software specification and design (both detailed and preliminary) are coherent.*	O	O
3.6	An external validation review (with the analyst) should be held at the end of the validation phase. *This can be used to ascertain whether or not the product satisfies the specifications* *If the validation of the software is not subject to the expression of any reserves , development is considered to be at an end.*	O	O
3.7	The result of each review should be documented and archived. It should include a list of all actions decided on in the review process, and the review conclusion (decision on whether or not to move on to the next activity). The activities defined in the review should be monitored and treated.	O	O

4.3.3. Code verification (source code and data) requirements

This verification corresponds to the first step in the verification of the actual code once it has been written. This is a "static" verification in so far as it is based on cross-readings, inspections, etc. It is only after this point that dynamic verification procedures (module tests, integration, validation) will make up the principal verification methods.

These verifications include code and data verifications.

No.	Code Verification (source code and data)	Level	
		1	2
3.8	Code verification (static analysis) should ensure that the code conforms to : - the software design documents, - coding rules. *Here the purpose is to ensure that the design is up to date and coherent with the code (source and data)*	R	O

4.4. SOFTWARE TEST REQUIREMENTS

4.4.1. General validation requirements

Before writing the first test sheets, it is important to establish a test strategy in a Test Plan. This strategy indicates the approach adopted, the objectives that have been set in terms of test coverage, the environments and specific techniques used, the success criteria to be applied, etc.

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

No.	General Validation Requirements	Level	
		1	2
3.9	The software verification strategy used at the different software development steps and the techniques and tools used for this verification should be described in a Test Plan before being used. This description should, as a minimum, include: - identification of the software and its safety-related components that will be submitted to validation procedure before use, - organisation of the verification activities (integration, validation, etc.) and any interfaces with other development activities, - independence of the verification (if applicable): the verification strategy should be developed and implemented, and the test results should be evaluated independently (by an individual, department, or organisation) in relation to the size of the development team, - verification methods and tools used (*types of tests, etc.),* - environment of the verification (*test equipment, emulators, etc.),* - manner in which test results were verified, - a traceability matrix demonstrating the correspondance between the tests to be undertaken and the objectives of the tests defined. *A single document (e.g. plan of the test) can satisfy all the planning requirements of several verification activities (module testing, integration, validation). These documents can, if necessary, refer to general procedures or instructions that are applicable to all software projects in addition to the specific measures taken for the project.* *The aim of formalising this strategy is to further ensure the reproducibility of the activity .* *The advantage of independent evaluation lies in the introduction of individuals not involved in development phases, and who therefore do not know how the software has been developed. In general, this ensures that the tests are performed more efficiently.*	R	O

No.	General Validation Requirements	Level	
		1	2
3.10	Verification of a new software version should include non-regression tests. *Non-regression tests are used to ensure that the modifications performed on the software have not modified the behaviour of the software in any unexpected way.*	O	O
3.11	Directives for drawing up test procedures should include : - a description of the input data to be used (*value*), - a description of the expected output (*value),* - criteria on which test results will be judged acceptable (*tolerance*). *This requirement implies that the tests carried out will be documented (using a format defined by the designer). The test documentation can be used to optimise the tests carried out (by avoiding repeating the same test several times), and to redo the tests at a later date (non regression tests of a new software version, or for another similar project for which software routines have been reused).*	R	O
3.12	The tests formalised in reports should be able to be carried out again (e.g., in the presence of the analyst). *Certain tests requiring specific means can only be carried out again with a sufficient warning. This requirement allows the analyst to guarantee the reality and exactness of all test results presented during evaluation.*	R	O

4.4.2. Software specification verification requirements: Validation tests

The purpose of these verifications is to detect errors associated with the software in the target system environment. Errors detected by this type of verification include:

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

Validation tests are the principal component of software specification verification.

No.	Software Specification Verification	Level	
		1	2
3.13	The test coverage should be made explicit in a traceability matrix and respect the following requirements: - each element of the specification, including safety mechanisms, should be covered by a validation test, - it should be possible to verify the real-time behaviour of the software in any operational mode. Furthermore, the validation should be carried out in conditions representative of the operational conditions of the system. *This requirement guarantees that the software reacts as expected in operation. It applies only to cases where the test conditions can be destructive for hardware (e.g., physical fault of a component that cannot be simulated).* *To be significant, validation should be performed in the operational conditions of the system (i.e. with the final versions of software and hardware, and the software installed in the target system.). Any other combination could decrease the efficiency of the test and require analysis of its representiveness.*	O	O
3.14	Validation results should be recorded in a validation report that should cover at least the following points: - the versions of software and system that were validated, - a description of the validation tests performed (inputs, outputs, testing procedures), - the tools and equipments used to validate or evaluate the results, - the results showing whether each validation test was a success or failure, - a validation assessment: identified non-conformities, impact on safety, decision as to whether or not to accept the validation. A validation report should be made available for each delivered software version and should correspond to the final version of each delivered software product. *This report can be used to provide proof that tests were indeed carried out, and that the results were correct (or contained explainable deviations). It can also be used to redo tests at a later date, for a future software version or for another project.* *The second requirement provides a guarantee that each delivered version has been validated in its final form. On the other hand, this requirement does not impose a complete validation of each modification of an existing code - an impact analysis can, in certain cases, justify partial validation.*	O	O

4.4.3. Software design verification requirements: Software integration tests

This verification focuses on the correct assembly of software modules and on the mutual relationships between software components. It can be used to reveal errors of the following kind:

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

Software integration tests form the principal component of this verification.

No.	Software Design Verification	Level	
		1	2
3.15	Software integration tests should be able to verify: - correct sequencing of the software execution, - exchange of data between modules, - respect of the performance criteria, - non-alteration of global data. The test coverage should be given explicitly in a traceability matrix demonstrating the correspondence between the tests to be undertaken and the objectives of the tests defined.	R	O
3.16	Any modification of the software during its integration should be analysed to identify the impact on the relevant modules and to ascertain whether certain verifications should be repeated. *All tests carried out should be representative of the final software version. Code modifications during tests might invalidate the results of previous tests.*	R	O
3.17	Integration test results should be recorded in a software integration test report, which should, as a minimum, contain the following points: - the version of the integrated software - a description of the tests performed (inputs, outputs, procedures), - the integration tests results and their evaluation.	R	O

4.4.4. Detailed design verification requirements: Module tests

Module tests focus on software modules and their conformity with the detailed design. This activity is indispensable for large and complex software products, but is only recommended for the relatively small software products dealt with here. This conformity can also be demonstrated using static techniques (e.g. re-reading the code).

This phase of the verification procedure allows detection of the following type of errors:

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

No.	Detailed Design Verification	Level	
		1	2
3.18	Each software module should be submitted to a series of tests to verify, using input data, that the module fulfils the functions specified at the detailed design stage. The test coverage should be given explicitly in a traceability matrix that demonstrates the correspondence between the tests to be undertaken and the objectives of the tests defined.	/	R
3.19	Module test results should be recorded in a report that contains at least the following points: - the version of the module tested, - the input data used, - expected and observed results, - an evaluation of the results (positive or otherwise). *The aim of this requirement is to ensure that each module is verified in its final version and that tests will be reproducible (for use in non-regression tests for example).*	/	R

5. CONCLUSION

The major objective dealt with this document, through the application of safety and quality requirements, is the prevention of embedded software failures and any other unexpected behaviour that might lead to the creation of dangerous faults in a system dedicated to the machinery sector.

In order to satisfy this objective, requirements are given and organised into two software requirement levels (1,2) according to the criticity of the functions ensured by the software. The requirements focus on:

- Software product. Requirements are established in order to obtain an embedded software that is fully safe in operation and of satisfactorily high quality.
- Software development process. The basic objectives of these requirements (definition of the development phases, means required, etc.) are to orient development in such a way that it can satisfy the functional specification requirements of the target system and inspire the confidence necessary to satisfy safety constraints.
- Software verification: The purpose of verification requirements is to demonstrate that software products stemming from a given phase of the development cycle conform to the specifications established during the previous phases and to any applicable standards or rules. They also serve as a means of detecting and accounting for any errors that might have been introduced during embedded software development.

The requirements outlined in this document can be used for all types of machinery, regardless of the technology involved and whether or not the system parameters can be defined by the user, provided the safety functions are ensured by software.

It is recommended that the system designer take these requirements into consideration from the outset of the embedded software development procedure, when the intervention of the analyst takes place before the beginning of the development itself.

6. GLOSSARY

- **Aliasing:** access to the same data by means of different names.
- **Audit:** methodical and independent examination intended to determine whether the activities and the results relative to quality and safety satisfy the previously-established measures, and if these measures have been applied efficiently and in such a way as to ensure the achievement of the objectives set (NXF50-120).
- **Coding**: activity of source code production, representation in languages such as assembly language or C, which can then be accepted by an assembler or compiler for the production of executable instructions for use by the target processor.
- **Design**: software architecture construction activity (usually called "preliminary design") which allows the implementation of software specifications and the construction of detailed algorithms (activity usually referred to as "detailed design").
- **Executable code production**: activity in which the different code components are grouped (by link editing especially) into a form that allows the generation of an executable code to be loaded into the target system.
- **Version Sheet:** document defining a software version in terms of its constituent parts and the tools required to produce them.
- **Configuration Management:** all the activities (manual and automatic) allowing the constituent parts of a software product and the relations existing between these parts to be identified and defined. Such activities consist in controlling changes to a software product during its life cycle, following up the state of application of these changes, archiving each of the successive states and checking that each state is complete and coherent.
- **Test traceability matrix**: matrix intended to establish the correspondence between the tests to be carried out and the objectives of the tests defined, and allowing the evaluation of test coverage.
- **Module tests**: can be used to demonstrate that the lowest level software module carries out the entire function foreseen at the detailed design stage, and only this function.
- **Software quality plan**: document laying down the operating modes, resources and sequences of activities linked to quality regarding a product, service contract or particular project. (NFX 50-120). It is a project management tool.

- **Review**: activity organised at a key point ("milestone") in the development cycle (e.g. specification, validation) in which specialists external to the development team participate. The review is intended to ensure that all the activities required during a given phase have been undertaken, that the problems to be dealt with have been taken into account and that the solutions proposed allow the transition to the next development phase. It is not a control operation. It judges the work undertaken in a given phase or stage in terms of the justifications offered and making recommendations.

- **Software integration tests**: are used to verify the relationships between the different software components and the temporal sequencing of treatment. They are used to verify that units composed of several software modules function correctly.

- **Software validation tests**: are the ultimate step in the software development process. The purpose of these tests is to verify the correct operation of the software with respect to the specifications in the environment of the target system .

- **Specification**: activity that consists in describing the expected software functionalities, and that takes into account inputs and any applicable constraints.

- **Validation**: the test and evaluation of the integrated computer system (hardware and software) to ensure compliance with the functional, performance and interface requirements.

- **Verification**: the process of determining whether or not the product of each phase of the digital computer system development process fulfils all the requirements imposed by the previous phase.

7. APPENDIX: CODING RULES

7.1. INTRODUCTION

Coding requirements are presented in this Appendix in the form of coding rules for software written in C or in assembly language (the most widely used languages in the areas considered in this text).

The interest of these rules is to reduce the risk of programming errors and to make re-reading the code and integrating and maintaining the software much easier. It is not our intention to restrict the software developer, merely to attract his or her attention to possible sources of error during programming activities or to difficulties that are likely to occur during the integration and maintenance steps. For example:

Forbidding the assignment of variables in the Boolean expression in C is intended to detect programming errors:

if (my_variable = my_constant) ...

The intention of the programmer is to compare a variable to a constant and not to assign my_variable to be equal to my_constant. There is thus a typing error, with "=" instead of

"= =".

Rules limiting structural complexity (limitation of nesting, restrained use of break and continue, etc.) are intended to facilitate integration and maintenance. It is indeed somewhat delicate to test or to modify structurally complex sources.

The rules proposed here are applicable to any requirement level set for the software.

Note: there may be times when certain of these rules may be unsuited to a given project. It is preferable to avoid imposing a strict ban on a given point but rather to allow periodic neglect of a given rule. Nevertheless, it is necessary to add a comment explaining why the rule was not respected (and not simply forgotten), and to justify the action.

These rules are presented in three parts:

- general rules applicable to both C and assembly language. These rules are identified by the prefix "G".
- rules specific to C. These rules are identified by the prefix "C".
- rules specific to assembly language. These rules are identified by the prefix "A".

Each part is composed of rules applicable to comments and declarative parts, followed by rules applicable to executable instructions.

7.2. GENERAL RULES

7.2.1. General rules: Comments and declarative parts

No.	General Rules: Comments and Declarative Parts
G-1	Use a standard header for each module and subroutine.
G-2	Use explicit identifiers.
G-3	Limit subroutines to 150 lines.
G-4	Comment all data and parameters declarations.
G-5	Comment source code instructions using the following principles: - comments that should be coherent with the code, - one comment for each block of significant instructions and for each complex decision, - use non-trivial comments, i.e. comments that add something as compared to simply reading the code, - comments for the abnormal ends of modules.
G-6	Aliasing forbidden**.**

7.2.2. General Rules: Instructions

No.	General Rules: Instructions
G-7	Use a single entry and single exit point for each subroutine.
G-8	Do not use recursive structures.
G-9	Limit the nesting of control structures (less than five levels).
G-10	All instructions should be reachable by executing the code (no dead code).
G-11	Do not use literal numerical values in the source code (except for trivial values).
G-12	Do not declare unused variables/constants.
G-13	Initialise each variable before use.
G-14	Limit the use of pointers.
G-15	Limit the use of interruptions.
G-16	Branch in the loops forbidden.
G-17	Recursivity, if employed, should be explained in detail.

7.3. RULES FOR CODING IN C

7.3.1. Rules for coding in C: Comments and declarative parts

No.	Rules for coding in C: Comments and declarative parts
C-1	Define the upper limit of each array with a symbolic constant (the lower limit is implicitly set to 0 in C).
C-2	Test the limits of arrays when performing operations on array indices in order to avoid violating the array size limitations.
C-3	Indent the source code in relation to the nesting of control structures.
C-4	Characterise each array.
C-5	Initialise arrays when they are declared.

7.3.2. Rules for coding in C: Instructions

No.	Rules for coding in C: Instructions
C-6	Explain each conversion of type (casting).
C-7	Transfer data using only parameters.
C-8	Do not transfer expressions in function calls.
C-9	Do not redefine imported data locally.
C-10	Reserve the use of the "**static**" clause in a C function to state variables.
C-11	Set a feedback test on the code returned from each function.
C-12	Limit the data exported by a C module.
C-13	Expressions should be independent of the order in which they are evaluated. Do not use "++" or "--" in complex expressions.
C-14	Do not use "**!=**" and "==" instructions for real numbers.
C-15	Set the end of the programme (if it exists) in the **main programme**, and only with an **exit** statement.
C-16	Do not perform any assignments in Boolean expressions.
C-17	Limit the use of "**goto**" to the treatment of errors.
C-18	Limit the use of "**continue**" or "**break**" instructions. Use at most one "**continue**" or one "**break**" per loop.
C-19	Do not modify the counter of the "**for**" loop in the body of the loop.
C-20	Use the "**sizeof**" instruction whenever possible.
C-21	Forbid the use of dynamic allocations ("**malloc**" calls)
C-22	By default, reserve the treatment of a "**switch**" to the treatment of errors.
C-23	Associate a "**default**" to each "**switch**".
C-24	Terminate all cases of "**switch**" by a "**break**".

7.4. RULES FOR CODING IN ASSEMBLY LANGUAGE

7.4.1. Rules for coding in assembly language: Comments and declarative

No.	Rules for coding in Assembly language: Comments and declarative parts
A-1	Define the modes of use of all registers.
A-2	Define the mode of representation of the different types of variables.
A-3	Define the internal representation of the Boolean operators TRUE/FALSE
A-4	Define the mode of representation of negative integers.
A-5	Define the initialisation of the registers.
A-6	Define the mode of called parameter transfer: stack or register.
A-7	Comment all tests carried out.
A-8	Comment logic masks.
A-9	Comment branch points.
A-10	Comment indirect addressing operations.