MINISTÈRE
CHARGÉ
DES TRANSPORTS

Liberté
Égalité
Fraternité

dgac direction générale de l'Aviation civile

GUIDANCE FOR WRITING GOOD SAFETY REQUIREMENTS

TECHNICAL GUIDE

MINISTÈRE
CHARGÉ
DES TRANSPORTS

Liberté
Égalité
Fraternité

direction
générale
de l'Aviation
civile

GUIDANCE FOR WRITING GOOD SAFETY REQUIREMENTS

TECHNICAL GUIDE

Civil Aviation Technical Service

Environment, Systems and Operations Safety, Planning Department

1st edition

V1R3 of 12/01/2021

AUTHORS

Laurent **PLATEAUX**
Head of Program
(DGAC/DSAC)

Laurence **MORIN**
Head of Program
(DGAC/STAC)

PROOFREADING COMMITTEE

André **BARKAT**
Head of Division "Air Navigation"
(DGAC/STAC/Department Environment, Safety of Systems and Operations, Planning)

Victor **BOULANGER**
Project Manager, Airport Safety and Capacity Division
(DGAC/STAC/Department Environment, Systems and Operations Safety, Planning)

Romain **BUFFRY**
Head of Division "Equipment"
(DGAC/STAC/Department Security, Equipment)

Guillaume **ROGER**
Scientific and International Advisor - International Affairs
(DGAC/STAC/Directorate)

Abstract

The Commission Implementing Regulation (EU) n°2017/373, known as IR ATM/ANS, integrates in its regulatory requirements and means of compliance all the principles developed in the current system engineering standards. Indeed, as these standards aim to ensure the control of systems complexity, the proposed approaches are perfectly adapted to complex socio-technical systems such as air traffic control functional systems.

By incorporating all system engineering key concepts, the IR ATM/ANS puts the quality of the requirements at the core of the argumentation of safety assessments or safety support assessments.

This guide is intended to accompany this evolution by compiling some good practices in terms of requirements writing. The aim of these good practices is to facilitate the management of requirements, to ensure that they are properly considered when designing a new system or as part of a change, and to ensure that they are effectively and completely validated and verified.

The different types of requirements are discussed as well as their properties and the appropriate verification methods. Some writing rules are also proposed.

This guide may be used by any person or organisation whose vocation is to develop safety assessments or safety support assessments, as well as anyone involved in the specification of socio-technical systems.

Keywords

Air navigation, safety studies, requirements, writing, good practices, guide, guidance, standards

CONTENTS

TABLE OF FIGURES

1. PREAMBLE

1.1. REMINDER OF THE CONTEXT

This guide is intended to provide guidance to air navigation service providers or any other aviation stakeholder for writing clear and efficient requirements. It does not in any way replace the regulations in force if it contradicts them. It is intended for any person contributing to the requirements specification process, whether for writing or for the verification of the requirements. Although the principles of requirements writing developed in this guide are easily approachable, the contextual elements as well as the justifications for certain practices require an awareness of the elementary principles of systems engineering. This concerns in particular the notion of the engineering tier, the role of different processes and the different typical engineering data produced during a complete system engineering process.

Within the framework of safety assessments, or safety support assessments, the service provider must define requirements. **Their fulfilment** will guarantee compliance with safety criteria or objectives and thus ensure a level of safety or service in line with regulatory and operational needs.

These requirements, known as "safety" or "safety support" requirements, play a key role in the regulatory demonstration of change control, as required under EU Regulation n°2017/373. The quality of the requirements is a key point essential to the development of the argument and requires the application of good practices. The implementation of the latter ensures both a good understanding of the objectives sought and an appropriate demonstration.

It shall be noted that even if the application of the most advanced engineering standards is sought, the terminology used in this guide may sometimes be different to keep a usual, understandable and more widely used vocabulary in this domain.

1.2. PURPOSE OF THE DOCUMENT

This guide proposes some rules of good practices for writing the safety and safety support requirements needed to control changes to the functional systems of air navigation service providers.

Although the purpose of this guide is to provide a framework for writing safety requirements defined within the framework of regulatory assessments, it is applicable to all types of requirements for all types of projects.

1.3. CONTENTS OF THE GUIDE

Chapters 1 and 2 frame the document and define the main terms used.

Chapter 3 presents a theoretical part on the definition of the notion of requirement and the main principles for writing requirements.

In Chapter 4, some examples of requirements are provided with do's and don'ts.

This guide will be enriched over time in terms of examples or new practices.

For more details on the processes for defining, writing or managing requirements, as well as on the general principles of system engineering, everyone can refer to the standards listed in §1.4.

1.4. REFERENCE DOCUMENTS AND STANDARDS

[Ref 1] ISO/IEC/IEEE 15288:2015 - Systems and software engineering - System life cycle processes.

[Ref 2] ISO/IEC/IEEE 29148:2018 - Systems and software engineering - Life cycle processes - Requirements engineering.

[Ref 3] ED-153 - Guidelines for ANS Software Safety Assurance - Issued in August 2009.

[Ref 4] ED-109A - Software Integrity Assurance Considerations for Communication and Navigation and Surveillance and Air Traffic Management (CNS/ATM) Systems - Issued in January 2012.

[Ref 5] SEBoK Editorial Board. 2020. The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 2.2, R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society.

[Ref 6] specief.org - Society for the Promotion and Certification of French Language Requirements Engineering - http://www.specief.org

2. DEFINITIONS

NEEDS

Needs express what a user or an organisation needs to have in order to accomplish a given mission. They are expressed from the perspective of the end user and usually in natural language. They can be written in a more or less formal way, such as the User Stories frequently used in agile methods.

OPERATIONAL CONCEPT (OF A SYSTEM)

"Statement of the assumptions and intentions of an organisation with regard to an operation or series of operations of a cooperating system or set of systems", SEBoK [Ref 5].

Note: Note that [Ref 5] distinguishes between the "operational concept" which is specific to a given system or set of systems and the "concept of operations" which has a broader scope, at the organisational level. The term CONOPS is rather associated with the latter notion, however, in the context of the development of a given system one will often allow oneself to call CONOPs the operational concept of the system under consideration. Indeed, from the perspective of a given system, the concept of operations and the operational concept are equivalent. The concept of CONOPS in this guide, and particularly in the abbreviations, will refer to the operational concept of a system and not to the concept of operations of an organisation.

CONCEPT OF OPERATIONS

SEBoK [Ref 5], SEBoK "Outlines the assumptions and intentions of an organization with respect to a transaction or series of transactions" [Ref 5].

REQUIREMENT

"Expression that defines the property or constraint of a system, product or process that is unambiguous, clear, unique, consistent, complete, verifiable and deemed necessary to meet an operational need", SEBoK [Ref 5].

"Characteristic observable from outside a given entity", Alan Davis ["201 principles of software development"].

SYSTEM ENGINEERING

System engineering is a structured and interdisciplinary scientific approach. The aim of which is to formalise and apprehend the design, validation and verification of complex systems. Its objective is to master and control the design of systems whose complexity does not allow a simple approach.

SPECIFICATION

The system specification contains all the functional and non-functional requirements and design constraints necessary for the accurate and complete description of a system. It constitutes the technical reference for the design, verification and validation of the system.

SYSTEM

A set of components structured to accomplish one or more purposes.

The term "system" in this guide is taken in a broad sense and can refer to a technical system, a software, a socio-technical system, a system of systems, a management system, a process system, etc.

VALIDATION

All activities necessary to demonstrate that the specified requirements are correct and complete in relation to the operational need. Validation answers the question: "Have we specified and designed the right product?".

VERIFICATION

All activities necessary to demonstrate that the specified requirements are met by the system as implemented. The verification answers the question: "Does the system behave as specified?".

3. THEORETICAL CONSIDERATIONS

3.1. NOTION OF REQUIREMENT: DEFINITION AND OBJECTIVE

3.1.1. DEFINITION

One of the most commonly accepted definitions in the world of system engineering is that, for a given system, a requirement is the expression of a **property that this system must satisfy in order to comply with the needs for** which it is designed.

DEFINITION

A requirement is the expression of a **property that** a system must satisfy in order to **comply with the need for** which it is designed.

The term "property" can refer to different characteristics of the system such as functionalities, non-functional characteristics, constraints, performances, interfaces, etc. We will see below how the requirements for these different characteristics are expressed.

This definition is usually supplemented by a strong property that a requirement is a **characteristic observable from outside the system of interest**.

PROPERTY OF A REQUIREMENT

A requirement is a characteristic **observable from outside the** system of interest.

3.1.2. OBJECTIVE

A requirement has the difficult task of expressing what is expected from a system. It must express the "what", "what should the system do? "as opposed to the "how" which is a matter of system design. A requirement that expresses the "how" is generally not consistent with the property that a requirement must be "externally observable", however, as it will be seen later in §3.2.3, it is sometimes necessary to constrain the design.

Statistics on project failures and delays (Standish Group, for example) show that more than **50% of these failures are due to faulty requirements:** ambiguous, incomplete, inaccurate, forgotten, implicit, obsolete, etc.

Requirements play a key role in the system design process. Indeed these requirements are defined:

- For the system's sponsor, for evaluating the good understanding of its needs;
- For the developer/designer, for the actual realisation and implementation;
- For verifiers and validators, for assessing the relevance and completeness of verification and validation activities.

Due to the multitude of players involved in handling a requirement and the central role of the requirements in the system engineering activities, it is fundamental to attach great importance to it.

The achievement of all these objectives depends to a very large extent on the quality of the requirements.

3.1.3. THE CONCEPT OF SPECIFICATION

Combined with the requirements, it is common to encounter the notion of "**specification**".

A "specification" consists of a collection of requirements and often serves as a contractual or product repository to define all the properties of a system. While a requirement must be precise and complete while having a unitary character, the challenge of a specification is to be exhaustive, structured and consistent. Indeed, it must contain all the types of requirements necessary for the complete and consistent definition of the system.

DEFINITION

A specification is a **collection of requirements that** specifies all the properties of a system.

In addition to compiling the requirements, the specification generally carries a part of the explanation of the context and a part of the justification and understanding of the requirements. These elements provide a link with design choices as well as with the requirements or needs of the higher tier. They also include essential elements for the validation and verification of requirements. Furthermore, it is common for the specification to contain a chapter dedicated to traceability between the requirements or needs of the higher tier and those it contains.

3.1.4. THE NOTION OF NEEDS

There is also frequent reference to the notion of "**needs**" in engineering, with a particular link between "needs" and "requirements". Although there is also no absolute definition of the term "needs", it is commonly accepted that it is an expression of "For whom" and "For what" and is usually at the level of the operational use of the system. The needs are to be related to the expected operational missions and are generally expressed from the end-user's point of view and in a more natural and less formalised way than the requirements. Nevertheless, in some contexts they will be found in the form of fully formalised requirements. Like requirements that are compiled in a specification, needs are brought together in an **operational concept** or **expression of needs**.

The specification of the system(s) should cover all the needs identified in the operational concept.

3.2. CHARACTERISATION AND TYPOLOGY OF REQUIREMENTS

As seen above, a requirement is a property of a product or system that defines what is expected in relation to a particular need. It is quite common that for expressing a given need, several requirements are necessary as well as several **types** of requirements.

Three main families of requirements are usually defined:

- Functional requirements;
- Non-functional requirements;
- Constraints which may be design "constraints" or so-called "other" constraints

We will see that within these 3 families, there can be several other sub-families. However, it is essential to understand, first of all, what the purpose of classifying requirements is.

As mentioned, a requirement serves both the designer/developer of the system and the teams in charge of its verification or validation. However, these processes require to perform a variety of activities and it is precisely for this reason that it is useful to sort requirements: depending on the type, the activities to be carried out will be different and the expression of the requirement will be adapted to the activities to be implemented.

Classifying a requirement allows **to anticipate** the **design** and **verification/validation** activities to be carried out.

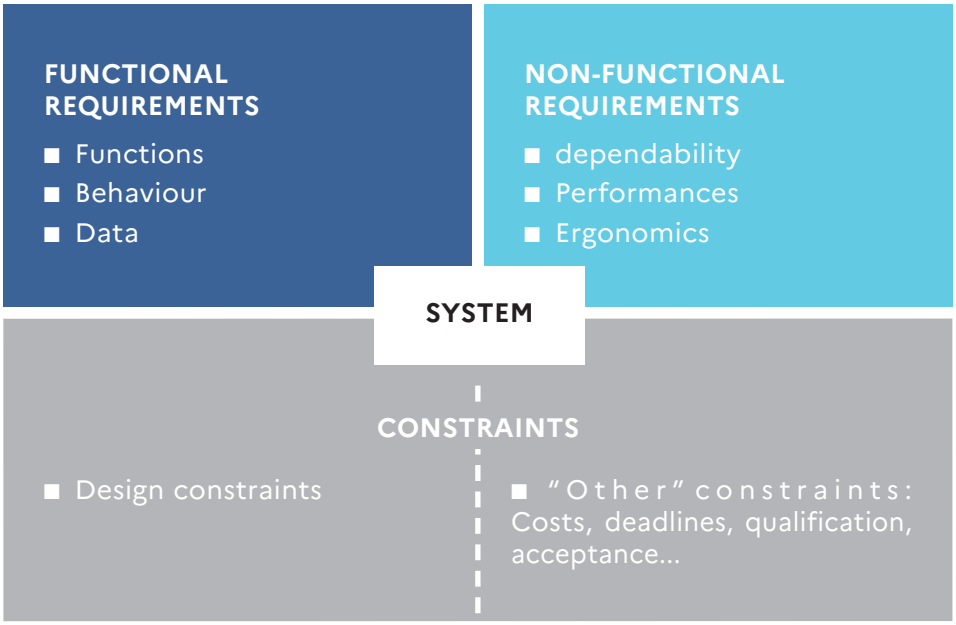

Figure 1: Typology of requirements.

3. THEORETICAL CONSIDERATIONS

3.2.1. FUNCTIONAL REQUIREMENTS

Functional requirements are the most common requirements for technical systems in particular. Their purpose is to specify the expected behaviour of the system and to answer the question "what should my system do". As we will see later in the writing rules, a functional requirement is often formulated in the form "System X must [do something] [with such performance]". Again, the focus should be on the "what" and not the "how".

Functional requirements should be formulated to describe only one behaviour as far as possible. This behaviour must be described at the system boundaries in such a way that the requirement respects the property of being observable from outside the system. Although they must be unitary, they must also be complete, i.e. they must define the expected behaviour completely, within the requirement itself and mention:

- The system which has to implement the "requirement";
- The context in which the requirement is valid, the prerequisites or preconditions ("in mode X", "when this condition is met", "in this technical or operational context", etc.), and the conditions that must be met in order for the requirement to be "valid";
- The expected behaviour (the event needed, the action or the result of the action, etc.);
- The precise interfaces that come into play (human trigger, network interface, button, mouse, switch, relay, protocol, etc.);
- Possible post-conditions (change of mode, temporary unavailability, etc.);
- The expected performance of the action (execution time, flow rate, power consumption, response time, jitter, latency, height, power, distance, etc.) and/or the result of the action (accuracy, tolerance, duration, force, temperature, etc.).

Robustness requirements, although often considered as a separate topic, are nothing more than functional requirements which are intended to define the expected behaviour in the event of unexpected or unsuitable situations (syntactic error, unexpected value, excessive flow rate, lack of data, untimely push of a button, action detection). The main difficulty in formulating a robustness requirement is to define the conditions under which the behaviour is expected. Just as it is easy to define the data and conditions for which we expect a precise behaviour, it can be extremely tedious to define all the unforeseen or unexpected conditions for which a robustness action will be necessary. In all cases, care should be taken to avoid negative wording and to specify the expected behaviour (and not the forbidden one!) in the event of abnormal situations. Several examples are proposed in this guide in paragraph 4.2.

Robustness requirements are primarily **functional requirements** and **shall** also be specified for validation and verification.

All functional requirements shall be translated into a **functionality** to be realised/accomplished by one or more system elements: a function of a software component, a mechanical or electronic behaviour, a human action, any activity.

Functional requirements are always translated into one or more **functions** to be implemented in one or more system elements.

In terms of validation and verification activity, in the vast majority of cases, the functional requirements will be:

- **Validated**, a priori, by simulation, analysis, equivalence or prototyping, or, a posteriori, by test, demonstration, operational evaluation, or other means, **and**;
- **Verified** by tests.

The specification of functional requirements requires precise knowledge of the input data handled, the output data, the types of possible triggers or interactions and their characteristics. This information is generally formalised within **interface requirements**.

Interface requirements must **comprehensively** specify all **interfaces between the system itself and the outside world**. They are indispensable input data for writing functional requirements.

These make it possible to define all the interfaces involved in exchanges between the **system of interest and the outside world**. Just as a functional requirement must be observable from the outside, an interface requirement must only concern interfaces used for interaction with the outside of the system. Please note that interfaces between system elements, i.e. internal to the system, will be defined during the system design and formalised in the specification of lower level components.

The challenges of interface description are completeness and accuracy. While deficient requirements are the cause of more than 50% of project failures, interface problems, although often easier to solve and rarely leading to project abandonment, are responsible for a large part of project redesign and rework. Since these requirements can only be verified in a simulated or even operational context, i.e. very late in the process, the cost of taking over these anomalies is often very high. While it can be extremely laborious for the person specifying the system to identify and list the interfaces exhaustively, it is important to be aware that any lack or imprecision will be a great source of error in the implementation because no one is better placed than the specifier of the requirement to characterise the target system's operating environment.

As far as possible, these interface requirements should be as precise as possible in order to identify, on the one hand, the nominal operating domain and, on the other hand, the degraded modes or those requiring robustness.

Concerning the verification of interface requirements, this is done from 2 persperctives. Firstly, it involves proofreading, ensuring that the interface requirements are all covered by the functional requirements and correctly referenced. In particular, all output interfaces must be addressed by at least one functional requirement.

Secondly, it consists in checking their correct implementation. However, this is done indirectly through the verification of the functional requirements based on the interfaces. There is therefore no actual interface tests, but the functional requirements must be tested to effectively exercise all interfaces.

3.2.2. NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements include all requirements that are not intended to express behaviour. They usually reflect a **general characteristic** of the system (the following list is not exhaustive):

- Form factor (size, volume, weight);
- Ergonomics;
- General performance in relation to available resources (power consumption, CPU/Memory use, number of operators, maximum throughput, etc.);
- Operational and technical environment (temperature, humidity, vibrations, noise, seismicity, etc.);
- Intrinsic performance of the system (operational dependability [reliability, availability, maintainability, testability], safety, security, etc.).

3. THEORETICAL CONSIDERATIONS

The formulation of these requirements is quite different from that of the functional requirements since there is no specific action expected but rather a general property of the system. Nevertheless, these requirements will have to be precise and complete, just like the functional requirements, and will have to define:

- The system concerned by the requirement;
- The context of validity of the requirement
- The non-functional characteristic addressed in the requirement;
- The quantitative or qualitative "objective";
- The tolerance or accuracy with which the objective must be achieved (an absolute objective is often unrealistic).

Non-functional requirements in most cases influence the system **architecture** and not directly the functionality of the system components.

In contrast to functional requirements, which are directly translated into functions on the system components until they are finally implemented, non-functional requirements generally influence **design choices and architecture**. This is particularly well illustrated by an availability type requirement which will not correspond to any particular function at the level of the system of interest but which is likely to constrain the architecture to achieve the expected level of availability, for example by using redundancy of system elements. These non-functional requirements may nevertheless require the implementation of additional elements for the needs of the architecture and thus generate (not decline!) functional requirements on the system elements. For example, in the case of redundancy, it is likely that this will induce requirements on failover, on the restoration of functions, on the synchronisation of states, etc.

In terms of validation and verification, the activities carried out are generally different from the ones for functional requirements and most of the time require different means, longer durations, analysis or synthesis of several tests. Certain characteristics, which are difficult to assess within acceptable test durations, will be verified by analysis and confirmed during prolonged use or will require means to speed up observations. This is the case, for example, for performances such as reliability, which can be verified a priori by analysis or simulation and confirmed during use. Similarly, a requirement specifying the prohibition of any single cause of failure will have to be verified by analysis of the architecture.

Performance requirements need a specific focus. The attentive reader will have noted that performance is addressed for both functional and non-functional requirements. It is important to dissociate, on the one hand, the performance associated with a **given function** and, on the other hand, the **general** performance of **a system**. The former must be specified within the functional requirements concerned so that it is clearly identified for the implementation of the requirement as well as for its verification which will have to measure this performance.

The **performance of a function** must be specified in **the requirement dealing with that function**, while an **overall performance** must be the subject of a **specific performance requirement**. The framework for their verification will be quite different.

For example, within the same requirement, it should be specified: "When the operator pushes button X, system Y shall display the position of object Z in less than 1 second with an accuracy of 5 metres". Functionality achieved with degraded performance will not be deemed to meet this requirement.

The second type of performance requirements are attributes of the overall system and can be formulated in a more general way, without referring to a specific functionality. Generally, specific tests (load, endurance, environment, etc.) will have to be implemented, beyond a simple functional test. It will therefore be possible to specify "system X must have an operational availability of 360 days out of 365" or "the memory occupation of system X must always be less than 80% of its total capacity".

Within the framework of performance requirements, it is also possible to find human performance requirements relating to the expected efficiency of the tasks performed, reaction times or other.

3.2.3. CONSTRAINTS

Among the constraints, it is possible to distinguish 2 categories. The first concerns so-called design constraints which have a deliberate and immediate influence on the design choices of the system. The second concerns constraints that are more related to the development or production environment of the system and does not immediately constrain the architecture. It can constrain the design of the system but indirectly.

Whichever category we are interested in, the fundamental aspect of constraints is that they must in no way contradict functional or non-functional requirements. If this were to happen, it would ultimately mean that we would have feasibility issues for functional or performance aspects. The difficulty is this incompatibility may not be detectable until very late in the development cycle. Therefore, it is important not to over-specify any constraints whatsoever and to prefer an approach that favours the specification of the right needs in a precise and complete manner.

3.2.3.1. DESIGN CONSTRAINTS

Design constraints are special requirements in the sense that they constrain the design of a system not by functional needs but by imposed design choices. These design constraints may be the result of regulatory constraints, norms and standards applicable to a given field or feedbacks and experience on similar systems. These constraints consist in constraining the solution.

Design constraints constrain the architecture of the system of interest, in its organisation or in the choice of its components. It is necessary to ensure the **compatibility of the design constraints with the functional requirements** that carry the operational needs.

For example, a design constraint could be: "The TEST system must be based on the Windows 7 update x operating system. ». This requirement de facto constrains the product architecture vis-à-vis the operating system and further constrains, indirectly, the performance, functionalities, and hardware platform. In this case, it will be necessary to be vigilant about functional requirements requiring hard real time, incompatible with a technology such as Windows, performance requirements in terms of power consumption that are incompatible with the hardware platforms supported by Windows 7, the use of applications that are incompatible with it, etc. Despite the risks or constraints that this brings, such a design constraint may be justified by a need to harmonise the computer population for its administration, by staff training or competence issues or by cost issues. It will therefore be necessary to ensure that it is consistent with the functional requirements of the product.

Design constraints have an immediate impact on the architecture of the system: either in terms of the organisation of the components, or in terms of the choice of components themselves. If the use of design constraints can be fully justified, it will be preferable to specify the real need imposing this choice. This allows:

- To open up the possibilities in terms of technical solutions;
- To avoid the risks of inconsistency between a functional or performance requirement and a design constraint;
- To better capture the real need for further developments;
- Not having to revise the specification in case of obsolescence management.

Verification of design requirements is usually done by analysis or inspection. Indeed, as they only specify architectural constraints or design choices, they do not correspond to a specific behaviour. However, it will be necessary to ensure during the verification tests that the design constraints do not prevent the achievement of a functional requirement.

Design constraints can be diverse and concern issues of architecture, design, regulatory compliance, application of design standards, reuse of existing material, etc. In most cases, they are linked to the operational and technical environment of the system.

3.2.3.2. “OTHER" CONSTRAINTS

The second category of constraints, the so-called "other" constraints, are not related to an operational or technical need of the system as such, but rather to the development context or environment. These constraints generally relate to the phase of the life cycle of the system that corresponds to the development or production of the system and not to its use.

These constraints include, for example, maximum development cost, time constraints, specific processes to be applied, regulatory constraints, production environment, etc.

Verification of these constraints will also not be done by testing and will instead consist of inspection or review of development plans, project planning elements or demonstration of regulatory compliance.

Like design constraints, these "other" constraints should not contradict the functional requirements of the system to be designed. If this were the case, it would constitute an inconsistency and could lead to a feasibility problem. For example, developing a highly complex system implementing extremely innovative functionalities with such strong cost and time constraints that they become incompatible with the desired functional objectives. One could also consider a constraint on a test environment that is incompatible with the performance levels sought and to be measured.

While the "design" constraints are generally formalised in the system specifications, given their immediate influence on the design of the system, the "other" constraints are more likely to be included in the project scoping documents.

3.3. SAFETY: A TYPOLOGY AND AN ATTRIBUTE

We have seen in the previous paragraphs different typologies of requirements that allow a given system to be described as exhaustively and precisely as possible. All these requirements have their place in the design of a system, either in the architecture (non-functional requirements) or in the functionality of the components (functional requirements).

In the field of air traffic control and, more generally, in so-called critical or safety areas, there is a type of requirement that plays a very special role. These are the **"safety requirements"**.

Among the safety requirements, a clear distinction must be made between non-functional safety requirements, which set the safety objectives to be achieved, and requirements with a safety attribute, which aim to achieve the safety objectives.

It is important to distinguish between two types of safety requirements:

- The first is effectively a typology of requirement: this is a type of requirement that belongs to the non-functional requirements and which generally aim to set the objectives or criteria for the safety of the system: failure rate per operational hour, operational availability, maximum severity of a failure, etc. These criteria are often derived directly from risk acceptability matrices or methods such as HAZOP (Hazard and Operability analysis), ALARP (As Low As Reasonably Practical), FHA (Functional Hazard Analysis);
- As for the second type, it is rather an attribute that cuts across the typologies seen above. Indeed, we will encounter both functional safety requirements and non-functional safety requirements. This attribute is the result of safety assessments identifying certain properties of the system necessary to ensure a given level of safety. This attribute facilitates the monitoring of requirements that contribute to the achievement of safety objectives or criteria but does not constitute a typology as such. For example, these requirements will include functional requirements for safety functions ("If a hardware failure is detected, system X shall send a [FAIL_MAT] message to the supervisory system within 5 seconds", "If no message is received for more than 5 seconds, system X will display a red [Loss of Connection] banner as defined in the HMI definition ref [YYYY].", "The operational availability of system X shall be greater than 363 days out of 365 with total downtime of less than 4 hours", "System X shall not have a single cause of failure", "System X shall have 2 redundant functional channels. ", etc.).

3. THEORETICAL CONSIDERATIONS

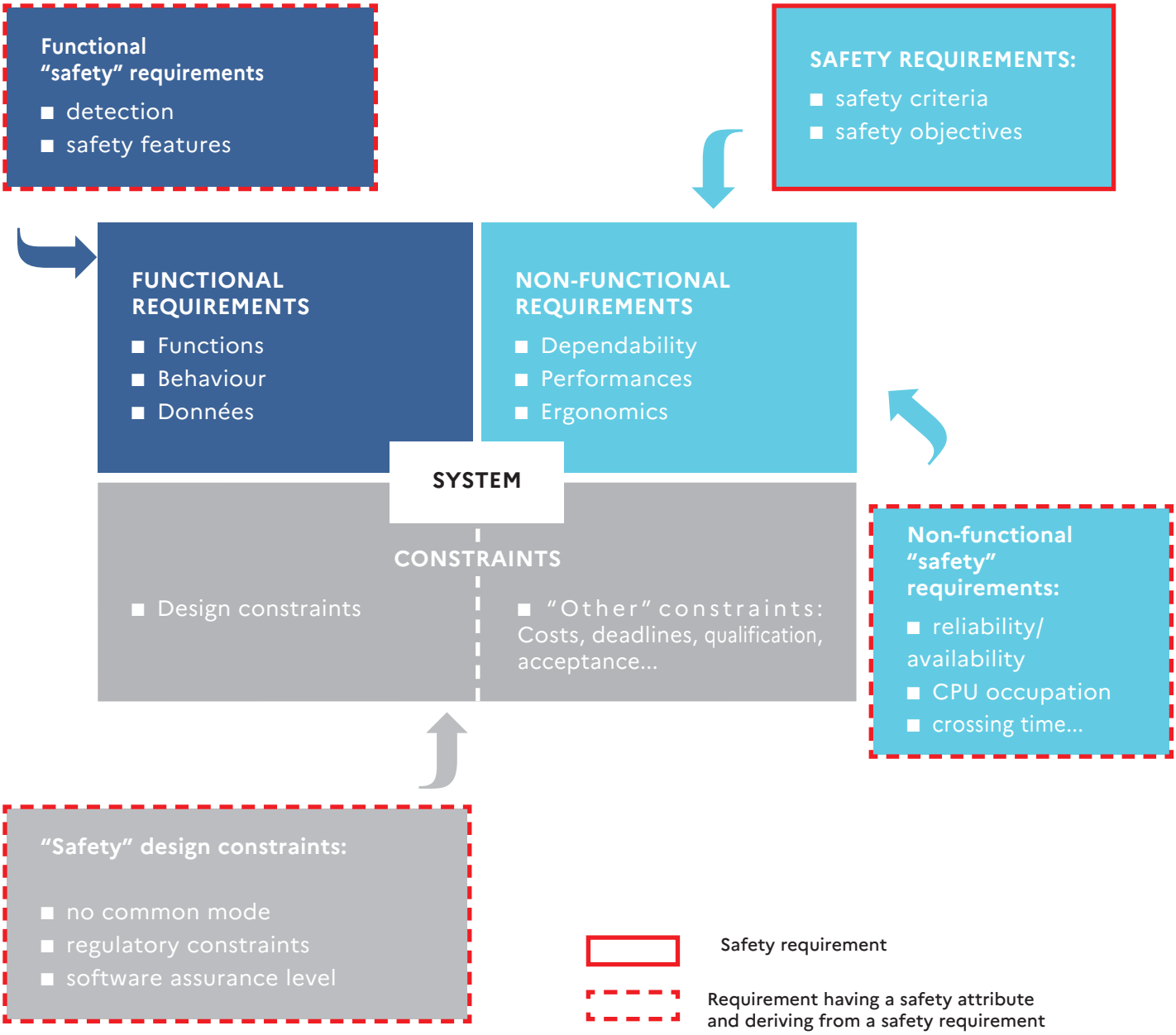

Figure 2 : Safety typology and attribute.

In the context of safety requirements, it is also common to encounter requirements that relate more to processes, derived from contractual or engineering constraints. Although these requirements are important in terms of meeting safety objectives and criteria, they generally have no place in a system specification, as their verification is part of more global activities than those involved in the verification of a system. They must therefore be formalised in the development plans, in the processes and/or the project management part of a contract, etc. A typical case of such a safety requirement is "System X software must be developed with a development assurance level of ED109A/AL4 or ED153/SWAL3". This requirement does not provide any information on what the software should do but constrains the development processes to achieve the functionality specified by the functional and non-functional requirements. Only the implementation of development processes and a suitable organisation will ensure that this requirement is met. Compliance with this safety requirement must always be associated with a given functional baseline for the system and the software .

The EU regulation n°2017/373 highlights the importance of the safety requirements which, if met, make it possible to demonstrate the fulfilment of the safety criteria of a change for a so-called ATS provider. The safety requirements referred to in the regulation bring together both safety requirements in the strict sense (safety objectives/criteria) and functional/non-functional requirements/safety constraints, the latter generally deriving from the former. It is the **demonstration that all these requirements have been met that** will give the **the argument validity**.

This EU regulation n°2017/373 also defines the notion of "Safety Support Requirements" for certified services of so-called non-ATS providers. This notion of "safety support" is close to the attribute of safety and does not in itself constitute a typology of requirement. These requirements may be functional, non-functional or constraints. They are described as "safety support" to highlight their importance in the provision of the final non-ATS service, given their direct contribution to the level of safety of the ATS service using it.

3. THEORETICAL CONSIDERATIONS

3.4. LEVEL OF REQUIREMENTS AND TRACEABILITY

The notion of level of **requirements** is used to define the **level or tier of engineering** under consideration. When a requirement is expressed, it is imperative to identify the level of the system at which one is situated. Therefore, we will see that a requirement must be formulated in a form such as "system X must...". Understanding this notion of level is a prerequisite if we want to respect the property which says that a requirement must be observable at the system's boundary.

It is not possible in this guide to further develop the notion of requirement levels and the role that the design process plays in this notion, however this aspect is fundamental to achieve correct requirement formulations and to understand how to ensure a correct flow of requirements from one level to another. For this purpose, it will be useful to refer to works or standards (see §1.4) which deal in detail with the subject of system engineering.

For the purposes of this guide, it should be kept in mind that an important activity in requirements verification is to ensure that the coverage of the requirements of one level is fully covered by the requirements of the lower level. This ensures, step by step, that all operational requirements are well implemented or realised and available. It also identifies possible limitations when certain requirements are not met or when performance is not achieved. These considerations make even more sense when dealing with safety requirements.

Associated with the level of requirements, we systematically find the notion of traceability. This traceability, which links together the upstream and downstream requirements, is precisely intended to facilitate the evaluation of the coverage of requirements from one level to another and we will see that this need for traceability requires favouring on the one hand the uniqueness of the behaviour described in a requirement and on the other hand the unique identification of the requirements. Thanks to these properties, traceability will allow in particular:

- For downstream traceability:
  - To check that requirements above are covered by the requirements below;
  - To carry out impact assessments whether a change in upstream requirements occurs.
- For upstream traceability:
  - To verify/justify the need for a lower-level requirement (unwanted function);
  - Carry out an impact analysis on the system service when changing a component.

3.5. REQUIREMENTS, TESTING AND TRACEABILITY

As seen above, requirements constitute the applicable reference framework for defining verification and validation activities. For each level of requirement, it is possible to associate a level of verification and/or validation, from the most unitary component to the complete integrated system. The relevance of the verification and validation is all the greater as it is possible to establish the completeness of the tests regarding the different behaviours expected from the system in all conceivable configurations.

Traceability between the requirements on the one hand and the **verification and validation activities** on the other hand provides the **indispensable support to demonstrate the completeness** of the verification and validation.

The demonstration of this completeness will usually be based on traceability between requirements and verification/validation activities. This could classically be a traceability link between a requirement and one or more tests within the framework of a verification activity but also a traceability link between a design requirement and an analysis activity demonstrating the satisfaction of the requirement or a traceability link between a requirement and a demonstration report within the framework of a requirement validation activity.

This need for traceability implies the uniqueness of the requirements as well as clear identification.

3.6. SYNTHESIS

The set of requirement typologies seen earlier and the flow of requirements between the different engineering activities are summarised in the Figure 3. The V-cycle representation allows to clearly identify the different traceability relationships as well as the different flows. However, this representation does not presume the dynamics of the applied development or lifecycle, nor the form that requirements, traceability links or data transfers may take.

This representation is perfectly valid for a development cycle of the Cascade, Spiral or Scrum type.

3. THEORETICAL CONSIDERATIONS

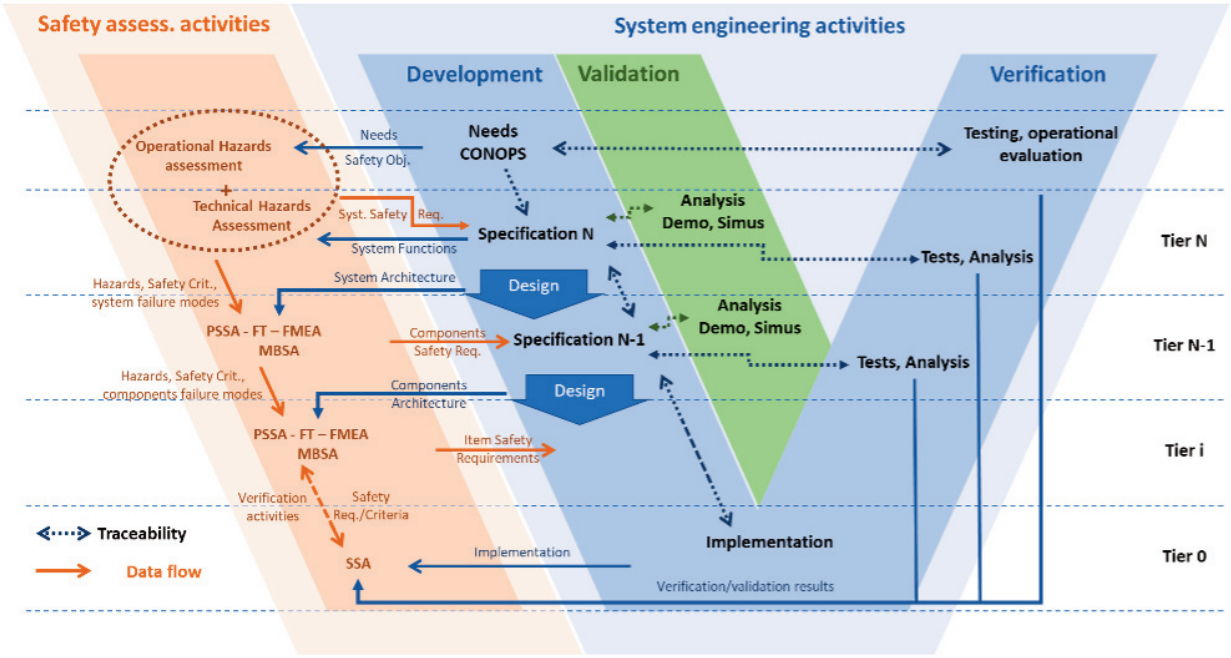

Figure 3: Summary of requirement flows and traceability.

Glossary of the figure:

- FT: Fault Tree
- FMEA: Failure Mode, Effects Analysis
- CONOPS: Concept of Operations
- MBSA: Model Based Safety Assessment
- PSSA: Preliminary System Safety Assessment
- SSA: System Safety Assessment

4. GOOD PRACTICES FOR WRITING REQUIREMENTS

4.1. THE GOOD PROPERTIES OF A REQUIREMENT

As seen above, the requirements are first a collaborative tool that allows exchanges on the system's expectations between the prescriber of the need, the implementer and the verifier. Indeed, the requirements are written to:

- Establish unambiguously and in technical terms what is to be achieved;
- Provide the reference for carrying out:
  - The system design;
  - Validation and verification activities, including the establishment of tests.

For this purpose, the requirements must follow writing rules to limit the risk of ambiguity and ensure that they are understandable by all stakeholders.

4.1.1. THE 11 CHARACTERISTICS OF A GOOD REQUIREMENT

A requirement must be...	This means...	Justification
Identified	The requirement must have a unique identifier.	This identification enables traceability to be established between requirements, between requirements and tests, and also during impact and development analyses.
Useful, Necessary	The requirement must reflect the expressed need, and only the expressed need (linked to traceability see below).	Additional and optional features add risks of inconsistency or additional failure modes. In terms of safety, particular care must be taken to limit unnecessary functionality.
Concise, unambiguous	The requirement describes the expected feature in a simple, short and clear manner. It must be easily readable and understandable by all stakeholders. Care should be taken to limit implicit considerations. It does not contain any explanation, reasoning, or justification. These additions may be added as comments or may be the subject of footnotes. The same word must have only one meaning.	Ambiguity can come from specific business vocabulary, vague words or implicit information. In the case of very present domain semantics, acronyms or technical terms, a glossary and a list of definitions should be drawn up. The ambiguity of the requirements could be improved by having them reviewed by different actors.
Simple, Unique	The requirement specifies only one behaviour. However, this behaviour must be fully defined.	The uniqueness of the requirement facilitates traceability between a need (or several needs) and the requirement, as well as between this requirement and lower-level requirements.
Independent of implementation	The requirement indicates what is to be done, but not how it is to be done. It should not describe how this need is to be realised or implemented unless there is a constraint from the customer, the end-user, or the environment, in which case it should be classified as a "constraint".	A requirement specifying how a particular function should be performed may create inconsistencies with other functional requirements and undermine the feasibility of the product.

A requirement must be...	This means...	Justification
Complete, self-sufficient	The requirement must be self-sufficient and contain all the information necessary for its implementation and verification. It recalls the context if necessary (no implicit)	The requirement indicates the system concerned, interfaces, execution conditions, expected behaviour, prerequisites, etc.
Non-redundant	There is no overlap with any other requirement.	Redundant requirements can lead to inconsistencies in the long run if they change. If redundancy is detected, the 2 requirements concerned must be reworded or one of the requirements must be eliminated.
Consistent, compatible with other requirements	A requirement must not contradict another requirement. All requirements must be compatible with each other.	It is a feature that relates to a complete specification or set of requirements.
Verifiable	The requirement is sufficiently well described to identify the criteria for successful audits and to define effective means of verifying the requirement. The various means available are: inspection, analysis, demonstration and testing (IADT).	The verifiable aspect of the requirements must be sought as soon as the requirement is written. It is advisable to involve, in the proofreading of the requirements, the people in charge of carrying out the tests.
Achievable, feasible	A requirement must be realistic. There is a satisfactory technical means of meeting the requirement within the budget and time limits.	
Traceable	Any requirement must be traceable to a higher-level need or requirement in order to be able to trace its origin.	This traceability also implies respect for the uniqueness and unique identification of a requirement.

4. GOOD PRACTICES FOR WRITING REQUIREMENTS

4.1.2. THE MUST: MEASURABLE/UNIQUE/SIMPLE/TRACEABLE

MUST is one of the simplest mnemonics for retaining the essential characteristics of a requirement:

- **Measurable**: the requirement contains quantitative and/or qualitative criteria for assessing the level to be achieved. There is a method for verifying the system against the requirement (inspection, analysis, demonstration, test etc.).
  - For example:
    - "The radar runway display system must have good performance" => not measurable
    - "The radar track must appear in less than 500 ms in 95% of the time" => measurable.
- **Unique**: No redundancy in requirements: a unique identifier is required, a single piece of information, strictly necessary and precise.
- **Simple**: No unnecessary or redundant information, clear and precise wording, understandable by all stakeholders, one defined behaviour.
- **Traceable**: Traceability makes it possible to identify the origin of the requirement and to easily find its justification. It also makes it possible to trace to the tests.

4.1.3. SMART: SIMPLE/MEASURABLE/ATTAINABLE/ REALISTIC/TRACEABLE

SMART is a second mnemonic to estimate the quality of a requirement:

- **Specific/Simple**: No unnecessary or redundant information, clear and precise formulation, understandable by all stakeholders, one defined behaviour.
- **Measurable**: the requirement contains quantitative and/or qualitative criteria for assessing the level to be achieved. There is a method for verifying the system against the requirement (inspection, analysis, demonstration, test etc.).
  - For example:
    - "The radar runway display system must have good performance" => not measurable
    - "The radar track must appear in less than 500 ms in 95% of the time" => measurable.
- **Attainable**: the requirement is technically feasible;
- **Realistic**: the requirement is achievable within the given constraints (cost, resources, time, etc.).
- **Testable and Traceable**: traceability makes it possible to identify the origin of the requirement and to easily find its justification. It also allows you to trace back to the tests.

4.2. IN PRACTICE

4.2.1. DEFECTS FOUND AND HOW TO CORRECT THEM

- **Avoid vague adjectives**:

"Fault-tolerant", "faithful", "adaptable", "fast", "slow", "ergonomic", "user-friendly", "sufficient", "secure", "ad hoc", "robust", "relevant", "different", "good", "excellent", "efficient".

- **To be avoided**:

"etc...", "and/or", "one or more" (replace by "the"), "several".

- **Use action verbs**:

“provides”, “displays”, “calculates”, avoid vaguer verbs such as “manage”, “support”, “maximise”, “minimise”, “optimise”, “improve”, “accommodate”.

- **Use verbs in the present tense**:

"must" and not "should", "could".

- **Avoid ambiguous terms**:

"state of the art", "almost always", "approximately", "close to", "fairly", "often", "easily", "few", "many", "enough", "appropriate", "effective", "if possible", "when necessary", "if necessary", "but not limited to", "as far as possible".

- **Clarify terms that are too general**:

The "management", "the system", "the equipment", "the function", "the inputs", "the purpose", etc...

- **Proscribe the negative form in the requirements**:

• It is often tempting to formulate a requirement in a negative form when a behaviour or property has been identified that appears to be harmful or detrimental to the system: "the system must not do this" or "the XXX performance of the system must not be less than/above this value". However, this negative wording has several drawbacks:

  • Behaviours that a system should not have are in fact probably infinite. Even if some have more serious safety or operational impacts than others, it is not possible to characterise a system by what it should not do. This would call for too much implicitness. Characterising completely and precisely what it must do in the nominal and robustness cases makes it possible to anticipate all functions and malfunctions, including harmful ones.

  • A negative requirement cannot be verified objectively and completely. An equivalence class approach makes it possible to verify the behaviour specified in a positive requirement as much as the verification that there is not a situation in which the system will behave in such a way requires going through all potential operational situations. This is not generally feasible.

• Negative wording is often a matter of need and therefore expresses rather a justification of the requirements. It will then be necessary to explicitly define the requirements to meet this need. For example, the following are examples of such requirements:

  • "The implementation of this new functionality must not lead to regression" is **not** a requirement and translates into "The development of this new functionality will be carried out in accordance with the XXX development processes to ensure the non-regression of the system";

  • "CPU consumption must not exceed 80%" will translate into "the CPU consumption of the system must remain below 80% 90% of the time. "and "When the CPU consumption of the system exceeds 80%, the system must send an alert [CPU_ALERT_80] to the supervisor and must stop non-priority processes. ».

4.2.2. EXAMPLES

■ SYSTEM REQUIREMENTS

What not to do	Problem	Replace with (only as an example, some things have been added to complete)
In order to minimise the disk space required for archiving, the components on the server's operating system media should be limited as far as possible to those required for a server without a graphical user interface so that the installation image will fit on a CD (use of "minimal" iso image).	This requirement is more of a design constraint. Vague terms such as "minimise", "as far as possible". As long as it has to fit on a CD, we know the maximum size. The fact that the selected operating system does not contain the GUI is either a solution or another constraint. What if the size of the operating system with GUI is smaller than the size of a CD? "To minimise the disk space needed for archiving": this is the rationale for the requirement, not to put it in the requirement itself but in the justification or in the need generating the requirement.	**Contraint_1**: the installation image of the X software must fit on a single 700MB CD. **Contraint_2**: the installation image of the operating system of the X software must contain all the necessary elements for the installation of a server without a graphical interface. **Contraint_3**: All components that are not necessary for the operation of a server without a GUI for the X software must be removed from the installation image.
The choice of the format of the local time source must be configurable.	This is both a functional requirement and a design constraint that requires a number of things to be configurable. To be specified: at power-up? via a configuration file? online? Is there a default format? The reference to the interface definition folder is also missing. The possibility of configuration must be defined: which formats are available, definition of the interface.	**Exi_Fonc_1**: The system shall allow to set the format of the local time source as defined in the interface document ref XXX. **Contrainte_1**: The choice of the time source format is part of the off-line parameters defined in the ICD[1] ref YYY and must be taken into account when starting the software. Or **Exi_HMI_1**: the software's HMI, as defined in the HMI specification ref ZZZ, shall allow to modify the format of the local time source, which must be taken into account without restarting in less than 30 sec.

[1] ICD: Interface Control Document

What not to do	Problem	Replace with (only as an example, some things have been added to complete)
In order to be able to investigate possible problems, it must be possible to activate a recording of the received frames by parameterisation.	This is not a requirement and contains 2 considerations: The possibility to record frames and the fact that it is a setting. The objective of the investigation is the need and not the requirement itself.	**Exi_fonct_Enreg_1**: System X must enable and disable the recording of all IP frames according to the format defined in the document ref YYY. **Exi_perf_Enreg_1**: The increase in processor load due to the activation of the recording of received frames must be less than 10%. **Exi_perf_Enreg_2**: The increase in memory occupancy due to the activation of the recording of received frames must be less than 10%. **Contrainte_Enreg_1**: The X system must have an off-line parameter file as defined in the ICD ref ZZZ allowing the activation and deactivation of the frame recording and which must be taken into account when the software is started.
The server must have the necessary tools/methods to establish a time synchronisation performance monitoring (current offset, synchronisation distance, ...).	All performances to be monitored must be listed precisely, no suspension points should be used in the requirement. A choice must be made between putting a design constraint on the fact that the system contains specific tools (these will need to be listed and linked to particular needs) or putting a functional requirement that addresses system capabilities. The latter is preferable.	**Exi_function_Follow_Perf_1**: The X software must be able to present the current offset to the nearest millisecond. **Exi_HMI_Follow_Perf_1**: the HMI of the X software, as defined in the HMI ref ZZZ specification, must allow the current offset to be displayed to the nearest millisecond over a history of 30 min in steps of 10 sec. **Exi_function_Follow_Perf_2**: The X software must be able to present the current synchronisation distance to the nearest minute. **Exi_HMI_Follow_Perf_2**: The HMI of software X, as defined in the HMI ref ZZZ specification, must allow the current synchronisation distance to be displayed to the nearest minute with refresh times less than one minute.

4. GOOD PRACTICES FOR WRITING REQUIREMENTS

What not to do	Problem	Replace with (only as an example, some things have been added to complete)
The partitioning performed by the server installation must: • leave space on the hard disk; • separate (at least) the /var partition from the system partition (log saturation must not prevent server operation)	Terms too vague: "available space", "at least", use of negative trainer. "Leaving space on the hard disk" is not understandable. Available space for whom? for what? another partition? This is both a design requirement (separation /var and system) and an expression of need (leaving space available and preventing saturation).	**Exi_Install_1**: the installation of the system must ensure that the size of data related to logging is limited to x GB. Note on this requirement: this could be expressed as a logging duration. This requirement can also be expressed in the form of a constraint: **Contraint_Install_1**: The size of the /var partition must be at least xGB and the size of the system partition must be at least 3 times the size of the installed system. **Exi_Fonc_Logs_1**: If the size of log data exceeds 80% of the xGB allocated to the logs, the system must send an alert to the supervision according to ICD XXX and purge the oldest data.
The component allows you to configure a delay in relation to the time received from the time source.	What delay? What time source? Requirement that cannot be tested because we do not know what we have to observe.	**Exi_Fonct_Offset_1**: The software X must allow the time received by the time source to be distributed with a positive or negative [Time_Source_Offset] as defined in the XXX setting ICD. **Contraint_Offset_1**: The system X must have an offline parameter file as defined in the ICD ref YYY allowing the [Time_Source_Offset] parameter setting and which must be taken into account at the software startup. **In DCI XXX**: [Time_Source_Offset]: integer, unit: second, range [-60; 60], default value: 0.

What not to do	Problem	Replace with (only as an example, some things have been added to complete)
The DATEL frame is converted to a time structure used for NTP broadcasting.[2]	This requirement cannot be coded, cannot be tested. What should be observed?	**Exi_Fonct_DATEL_1**: The X server should broadcast the [DAT_Time] received by the [DATEL_Frames] to lower-tier NTP components with a period of less than 10 minutes. The time must be distributed in the format defined in ICD ref XXX and within 500ms of receipt of the actually broadcast DATEL frame. **Interface_Struct_NTP_1**: The [NTP_Time] must respect the XXXXXXX format defined in the RFCYY.
The status of the time chain (GO/NOGO, status word) and the consistency of the received time are checked.	Incomplete requirement: the condition is checked, so what? What should be observed?	Rather, the requirement should take the following form: **Exi_Fonct_Status_1**: On receipt of a DATEL time frame, if the "STATUS" field of the time synchronization chain (specified in ICD XXX) is different from the expected states, system X must send an alarm to the supervision and reject the frame. **Exi_Fonct_Coherence_1**: When a DATEL time frame is received, if the received time is not consistent with the current time, system X must send an alarm to the supervision and reject the frame. The consistency check will be done by applying the following algorithm: XXX.

[2] NTP: Network Time Protocol

4. GOOD PRACTICES FOR WRITING REQUIREMENTS

■ SAFETY REQUIREMENTS

What not to do	Problem	Replace with
(Non-regression) The addition of the Normal/Rescue toggle setting by component X for Site 1 must not interfere with the nominal operation of the component at sites other than Site 1.	Negative form Non-testable (specify what is meant by "must not interfere" and "nominal operation"?) Non-regression MUST be guaranteed by the application of engineering and safety processes (QMS and SMS) and cannot be considered as a requirement, even for safety.	Non-regression is the application of internal engineering processes and should not be a requirement. If it were to be specified (for a contract for example), it should take the following form: **Exi_Secu_Processus**: The development of system X must follow the development process of the XXX company in order to ensure the non-regression of the system after modification.
(Non regression) The evolutions of version V1.2 of the component do not lead to a significant increase in the CPU load of the component.	It is not a question of non-regression as such, but only a performance requirement. The negative form is not recommended. It is preferable to specify requirements in absolute terms, as even a small increase in an already saturated load is not acceptable. Moreover, the requirement is not testable: It is necessary to specify what a "significant increase" means?	**Exi_Perf_CPU_1**: The CPU load of software X must remain below 50% CPU load on average over nominal load scenarios as defined in document ref XXX and below 90% load over 60 second periods in loaded scenarios as defined in document ref XXX.
Validate the consistency of the maps displayed during a Main/Back-up toggle during the evaluation phase.	Too vague. Human activity that needs to be made explicit.	**Exi_Proc_Secu_Evalop_1**: During the evaluation phase, during a Main/Back-up switchover, the controller must check the consistency of the displayed maps by applying the XXX procedure. Or **Exi_Proc_Secu_Evalop_1bis**: During the evaluation phase, during a Main/Back-up switchover, the controller must check that the displayed maps are identical in all points (alternative: on the following points to be explained).

4. GOOD PRACTICES FOR WRITING REQUIREMENTS

What not to do	Problem	Replace with
Check that the changeover procedure is taken into account in the Manex.	This is not a requirement as such and does not specify a behaviour of the system or its components. It is a verification activity.	**Exi_Proc_Secu_Toggle_Operator** : In case of warning of the following malfunctions of the main system (list ALL concerned malfunctions), the operator must perform a toggle action on the backup system within 2 min. **Exi_Proc_Secu_Toggle_Manex**: The Operator Manex must contain the procedure for switching from the main system to the back-up system. **Exi_Proc_Secu_Takeover_Training**: Operator training must enable operators to perform a takeover operation from the main system to the back-up system in less than 2 min in 95% of cases. All these rather operationally oriented requirements must find their counterpart in the technical system: **Exi_Secu_Toggle_System**: When the operator initiates a toggle command and the operating system is the primary system, the operating system must become the backup system in less than 1 min. **Exi_Secu_Alarm_Default_Syst_Princ**: If the main system has one of the following faults (put the exhaustive list of faults concerned), the supervision system must issue an alarm to the operator and display an alarm on the supervision HMI in less than 30 sec.

• **Example of a safety requirement**:

Exi_Safety_Objective : The rate of occurrence of system failures leading to separation losses greater than 50% must be less than 10E-6 failures/operational hour.

• **Examples of tagged functional and non-functional requirements [safety]**:

Exi_supervision: [Safety] In the event of a hardware failure of the system, the supervision must report an alarm (as defined in the HMI_Superv ref XXX specification) to the supervision operators in less than 15sec.

Exi_Performance_Display: [Safety] all objects on the screen must be displayed with a relative accuracy of 1% and an absolute accuracy of less than 0.5Nm.

Conception : STAC/Division documentation et diffusion des connaissances

Couverture : © Marie-Ange FROISSART DGAC /STAC
© Richard METZGER DGAC /STAC

Crédit photos : © Fotolia page 7
© Marie-Ange FROISSART DGAC /STAC page 9
© Richard METZGER DGAC /STAC pages 20, 24

Février 2021

Direction générale de l'Aviation civile
service technique de l'Aviation civile
CS 30012 - 31 avenue du Maréchal Leclerc
94 385 Bonneuil-sur-Marne cedex FRANCE
Téléphone : 01 49 56 80 00
www.stac.aviation-civile.gouv.fr
www.ecologie.gouv.fr