IEC 62279

Edition 2.0 2015-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems

Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Logiciels pour systèmes de commande et de protection ferroviaire

IEC 62279:2015-06(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 15 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 60 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 62279

Edition 2.0 2015-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems

Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Logiciels pour systèmes de commande et de protection ferroviaire

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 45.060

ISBN 978-2-8322-2741-1

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

RAILWAY APPLICATIONS – COMMUNICATION, SIGNALLING AND PROCESSING SYSTEMS – SOFTWARE FOR RAILWAY CONTROL AND PROTECTION SYSTEMS

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62279 has been prepared by IEC technical committee 9: Electrical equipment and systems for railways.

This standard is based on EN 50128:2011.

This second edition cancels and replaces the first edition, issued in 2002. It constitutes a technical revision.

The main technical changes with respect to the previous edition are as follows:

- requirements on software management and organisation, definition of roles and competencies, deployment and maintenance have been added;
- a new subclause on tools has been inserted in 6.7, based on IEC 61508-2:2010;
- tables in Annex A have been updated;
- a new Annex B on key software roles and responsibilities has been introduced;

- a new Annex C on document control summary has been introduced;
- Annex B on Bibliography of techniques has been revised and updated as new Annex D.

The main changes with respect to EN 50128:2011 are listed below:

- the subclause on tools in 6.7 has been updated;
- Annex B on key software roles and responsibilities has been modified.

The text of this standard is based on the following documents:

FDIS	Report on voting
9/2023/FDIS	9/2046/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

This Standard should be read in conjunction with IEC 62278:2002, *Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS).*

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

INTRODUCTION

This Standard is part of a group of related standards. The others are IEC 62278:2002, *Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS)* and IEC 62425:2007, *Railway applications – Communication, signalling and processing systems – Safety related electronic systems for signalling*.

IEC 62278:2002 addresses system issues on the widest scale, while IEC 62425:2007 addresses the approval process for individual systems which can exist within the overall railway control and protection system. This Standard concentrates on the methods which need to be used in order to provide software which meets the demands for safety integrity which are placed upon it by these wider considerations.

This Standard provides a set of requirements with which the development, deployment and maintenance of any safety-related software intended for railway control and protection applications should comply. It defines requirements concerning organisational structure, the relationship between organisations and division of responsibility involved in the development, deployment and maintenance activities. Criteria for the qualification and expertise of personnel are also provided in this Standard.

The key concept of this Standard is that of levels of software integrity. This Standard addresses five software safety integrity levels where SIL 0 is the lowest and SIL 4 the highest safety related integrity levels. The higher the risk resulting from software failure, the higher the software safety integrity level will be.

This Standard has identified techniques and measures for the five levels of software integrity. The required techniques and measures for software Safety Integrity Levels 0 to 4 are shown in the normative tables of Annex A. In this standard, the required techniques for level 1 are the same as for level 2, and the required techniques for level 3 are the same as for level 4. This Standard does not give guidance on which level of software integrity is appropriate for a given risk. This decision will depend upon many factors including the nature of the application, the extent to which other systems carry out safety functions and social and economic factors.

It is within the scope of IEC 62278 and IEC 62425 to define the process of specifying the safety functions allocated to software.

This Standard specifies those measures necessary to achieve these requirements.

IEC 62278 and IEC 62425 require that a systematic approach be taken to:

a) identify hazards, assessing risks and arriving at decisions based on risk criteria,

b) identify the necessary risk reduction to meet the risk acceptance criteria,

c) define an overall System Safety Requirements Specification for the safeguards necessary to achieve the required risk reduction,

d) select a suitable system architecture,

e) plan, monitor and control the technical and managerial activities necessary to translate the Safety Requirements Specification into a Safety-Related System of a validated safety integrity.

As decomposition of the specification into a design comprising safety-related systems and components takes place, further allocation of safety integrity levels is performed. Ultimately this leads to the required software safety integrity levels.

The current state-of-the-art is such that neither the application of quality assurance methods (so-called fault avoiding measures and fault detecting measures) nor the application of

software fault tolerant approaches can guarantee the absolute safety of the software. There is no known way to prove the absence of faults in reasonably complex safety-related software, especially the absence of specification and design faults.

The principles applied in developing high integrity software include, but are not restricted to

– top-down design methods,
– modularity,
– verification of each phase of the development lifecycle,
– verified components and component libraries,
– clear documentation and traceability,
– auditable documents,
– validation,
– assessment,
– configuration management and change control, and
– appropriate consideration of organisation and personnel competency issues.

The System Safety Requirements Specification identifies all safety functions allocated to software and determines their safety integrity level. The successive functional steps in the application of this Standard are shown in Figure 1 and are as follows:

a) define the Software Requirements Specification and in parallel consider the software architecture. The software architecture is where the safety strategy is developed for the software and the software safety integrity level (7.2 and 7.3);
b) design, develop and test the software according to the Software Quality Assurance Plan, software safety integrity level and the software lifecycle (7.4 and 7.5);
c) carry out software/software and software/hardware integration on the target hardware and verify functionality (7.6);
d) accept and deploy the software (7.7 and 9.1);
e) if software maintenance is required during operational life then re-activate this Standard as appropriate (9.2).

A number of activities run across the software development. These include testing (6.1), verification (6.2), validation (6.3), assessment (6.4), quality assurance (6.5) and modification and change control (6.6).

Requirements are given for support tools (6.7) and for systems which are configured by application data or algorithms (Clause 8).

Requirements are also given for the independence of roles and the competence of staff involved in software development (5.1, 5.2 and Annex B).

This Standard does not mandate the use of a particular software development lifecycle. However, illustrative lifecycle and documentation sets are given in 5.3, Figure 3 and Figure 4 and in 7.1.

Tables have been formulated ranking various techniques/measures against the software safety integrity levels. The tables are in Annex A. Cross-referenced to the tables is a bibliography giving a brief description of each technique/measure with references to further sources of information. The bibliography of techniques is in Annex D.

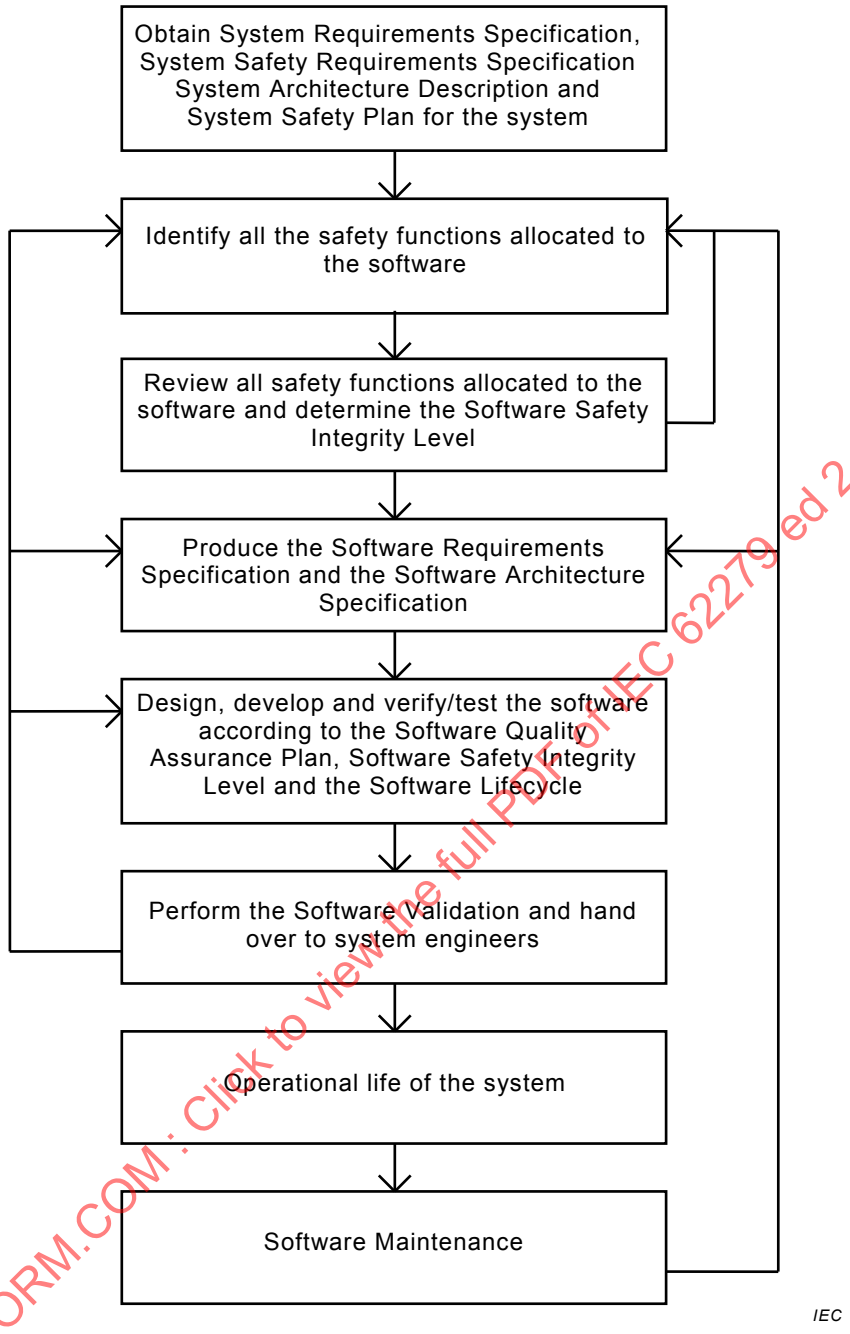


Figure 1 – Illustrative software route map

RAILWAY APPLICATIONS – COMMUNICATION, SIGNALLING AND PROCESSING SYSTEMS – SOFTWARE FOR RAILWAY CONTROL AND PROTECTION SYSTEMS

1 Scope

1.1 This International Standard specifies the process and technical requirements for the development of software for programmable electronic systems for use in railway control and protection applications. It is aimed at use in any area where there are safety implications. These systems can be implemented using dedicated microprocessors, programmable logic controllers, multiprocessor distributed systems, larger scale central processor systems or other architectures.

1.2 This Standard is applicable exclusively to software and the interaction between software and the system of which it is part.

1.3 This Standard is not relevant for software that has been identified as having no impact on safety, i.e. software of which failures cannot affect any identified safety functions. The concept of SIL 0 is introduced because uncertainty is present in the evaluation of the risk, and even in the identification of hazards. At least the SIL 0 requirements of this Standard are fulfilled for the software part of functions that have a safety impact below SIL 1.

1.4 This Standard applies to all safety related software used in railway control and protection systems, including

– application programming,
– operating systems,
– support tools,
– firmware.

Application programming comprises high level programming, low level programming and special purpose programming (for example: Programmable logic controller ladder logic).

1.5 This Standard also addresses the use of pre-existing software and tools. Such software may be used, if the specific requirements in 7.3.4.7 and 6.5.4.16 on pre-existing software and for tools in 6.7 are fulfilled.

1.6 Software developed according to any version of this Standard will be considered as compliant and not subject to the requirements on pre-existing software.

1.7 This Standard considers that modern application design often makes use of generic software that is suitable as a basis for various applications. Such generic software is then configured by data, algorithms, or both, for producing the executable software for the application. The general Clauses 1 to 6 and 9 of this Standard apply to generic software as well as for application data or algorithms. The specific Clause 7 applies only for generic software while Clause 8 provides the specific requirements for application data or algorithms.

1.8 This Standard is not intended to address commercial issues. These should be addressed as an essential part of any contractual agreement. All the clauses of this Standard will need careful consideration in any commercial situation.

1.9 This Standard is not intended to be retrospective. It therefore applies primarily to new developments and only applies in its entirety to existing systems if these are subjected to major modifications. For minor changes, only 9.2 applies. The assessor analyses the

evidences provided in the software documentation to confirm whether the determination of the nature and scope of software changes is adequate. However, application of this Standard during upgrades and maintenance of existing software is highly recommended.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62278:2002, *Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS)*

ISO/IEC 90003:2014, *Software engineering – Guidelines for the application of ISO 9001:2008 to computer software*

ISO/IEC 25010 series, *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*

ISO 9000, *Quality management systems – Fundamentals and vocabulary*

ISO 9001:2008, *Quality management systems – Requirements*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
assessment
process of analysis to determine whether software, which may include process, documentation, system, subsystem hardware and/or software components, meets the specified requirements and to form a judgement as to whether the software is fit for its intended purpose.

Note 1 to entry: Safety assessment is focused on but not limited to the safety properties of a system.

3.1.2
assessor
entity that carries out an assessment

3.1.3
commercial off-the-shelf (COTS) software
software defined by market-driven need, commercially available and whose fitness for purpose has been demonstrated by a broad spectrum of commercial users

3.1.4
component
a constituent part of software which has well-defined interfaces and behaviour with respect to the software architecture and design.

Note 1 to entry: A software component fulfils the following criteria:

– it is designed according to “Components” (see Table A.20);

– it covers a specific subset of software requirements;

– it is clearly identified and has an independent version inside the configuration management system or is a part of a collection of components (e.g. subsystems) which have an independent version.

3.1.5
configuration manager
entity that is responsible for implementing and carrying out the processes for the configuration management of documents, software and related tools including change management

3.1.6
customer
entity which purchases a railway control and protection system including the software

3.1.7
designer
entity that analyses and transforms specified requirements into acceptable design solutions which have the required safety integrity level

3.1.8
entity
person, group or organisation who fulfils a role as defined in this Standard

3.1.9
fault
abnormal condition that could lead to an error in a system

Note 1 to entry: A fault can be random or systematic.

3.1.10
error
deviation from the intended design which could result in unintended system behaviour or failure

3.1.11
failure
unacceptable difference between required and observed performance

3.1.12
fault tolerance
built-in capability of a system to provide continued correct provision of service as specified, in the presence of a limited number of hardware or software faults

3.1.13
firmware
software stored in read-only memory or in semi-permanent storage such as flash memory, in a way that is functionally independent of applicative software

3.1.14
generic software
software which can be used for a variety of installations purely by the provision of application-specific data and/or algorithms

3.1.15
implementer
entity that transforms specified designs into their physical realisation

3.1.16
integration
process of assembling software and/or hardware items, according to the architectural and design specification, and testing the integrated unit

3.1.17
integrator
entity that carries out software integration

3.1.18
pre-existing software
software developed prior to the application currently in question, including COTS (commercial off-the shelf) and open source software

3.1.19
open source software
source code available to the general public with relaxed or non-existent copyright restrictions

3.1.20
programmable logic controller
solid-state control system which has a user programmable memory for storage of instructions to implement specific functions

3.1.21
project management
administrative and/or technical conduct of a project, including safety aspects

3.1.22
project manager
entity that carries out project management

3.1.23
reliability
ability of an item to perform a required function under given conditions for a given period of time

3.1.24
robustness
ability of an item to detect and handle abnormal situations

3.1.25
requirements manager
entity that carries out requirements management

3.1.26
requirements management
process of eliciting, documenting, analysing, prioritising and agreeing on requirements and then controlling change and communicating to relevant stakeholders.

Note 1 to entry: Requirement management is a continuous process throughout a project.

3.1.27
risk
combination of the rate of occurrence of accidents and incidents resulting in harm (caused by a hazard) and the degree of severity of that harm

3.1.28
safety
freedom from unacceptable levels of risk of harm to people

3.1.29
safety authority
body responsible for certifying that safety related software or services comply with relevant statutory safety requirements

3.1.30
safety function
a function that implements a part or whole of a safety requirement

3.1.31
safety-related software
software which performs safety functions

3.1.32
software
intellectual creation comprising the programs, procedures, rules, data and any associated documentation pertaining to the operation of a system

3.1.33
software baseline
complete and consistent set of source code, executable files, configuration files, installation scripts and documentation that are needed for a software release.

Note 1 to entry: Information about compilers, operating systems, pre-existing software and dependent tools is stored as part of the baseline. This will enable the organisation to reproduce defined versions and be the input for future releases at enhancements or at upgrade in the maintenance phase.

3.1.34
software deployment
transferring, installing and activating a deliverable software baseline that has already been released and assessed

3.1.35
software life cycle
activities occurring during a period of time that starts when software is conceived and ends when the software is no longer available for use.

Note 1 to entry: The software life cycle typically includes a requirements phase, design phase, test phase, integration phase, deployment phase and a maintenance phase.

3.1.36
software maintainability
capability of the software to be modified to correct faults, improve performance or other attributes, or to adapt it to a different environment

3.1.37
software maintenance
action, or set of actions, carried out on software after deployment with the aim of enhancing or correcting its functionality

3.1.38
software safety integrity level
classification number which determines the techniques and measures that have to be applied to software.

Note 1 to entry: Safety-related software has been classified into five safety integrity levels, where 0 is the lowest and 4 the highest.

3.1.39
supplier
entity that designs and builds a railway control and protection system including the software or parts thereof

3.1.40
system safety integrity level
classification number which indicates the required degree of confidence that an integrated system comprising hardware and software will meet its specified safety requirements

3.1.41
tester
entity that carries out testing

3.1.42
testing
process of executing software under controlled conditions as to ascertain its behaviour and performance compared to the corresponding requirements specification

3.1.43
tool class T1
generates no outputs which can directly or indirectly contribute to the executable code (including data) of the software

Note 1 to entry: T1 examples include: a text editor or a requirement or design support tool with no automatic code generation capabilities; configuration control tools.

3.1.44
tool class T2
supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software

Note 1 to entry: T2 examples include: a test harness generator; a test coverage measurement tool; a static analysis tool.

3.1.45
tool class T3
generates outputs which can directly or indirectly contribute to the executable code (including data) of the safety related system

Note 1 to entry: T3 examples include: a source code compiler, a data/algorithms compiler, a tool to change set-points during system operation; an optimising compiler where the relationship between the source code program and the generated object code is not obvious; a compiler that incorporates an executable run-time package into the executable code.

3.1.46
traceability
degree to which a relationship can be established between two or more products of a development process, especially those having a predecessor/successor or master/subordinate relationship to one another

3.1.47
validation
process of analysis followed by a judgment based on evidence to determine whether an item (e.g. process, documentation, software or application) fits the user needs, in particular with respect to safety and quality and with emphasis on the suitability of its operation in accordance to its purpose in its intended environment

3.1.48
validator
entity that is responsible for the validation

3.1.49
verification
process of examination followed by a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfils the requirements of that phase with respect to completeness, correctness and consistency

Note 1 to entry: Verification is mostly based on review of output documents (design, implementation, test documents etc.).

3.1.50
verifier
entity that is responsible for one or more verification activities

3.2 Abbreviations

ASR	Assessor
COTS	Commercial off-the-shelf
CGM	Configuration Manager
DES	Designer
HR	Highly Recommended
IMP	Implementer
INT	Integrator
JSD	Jackson System Development Method
M	Mandatory
MASCOT	Modular Approach to Software Construction, Operation and Test
NR	Not Recommended
PM	Project Manager
QAM	Quality Assurance Manager
R	Recommended
RAMS	Reliability, Availability, Maintainability and Safety
RQM	Requirements Manager
SDL	Specification and Description Language
SFC	Sequential Function Charts
SIL	Safety Integrity Level
SOM	Service Oriented Modeling
SSADM	Structured Systems Analysis & Design Methodology
TST	Tester
V&V	Verification and Validation
VAL	Validator
VER	Verifier

4 Objectives, conformance and software safety integrity levels

4.1 The allocation of safety-related system functions to software, as well as software interfaces, shall be identified in the system documentation. The system in which the software is embedded shall be fully defined with respect to the following:

– functions and interfaces;
– application conditions;
– configuration or architecture of the system;
– hazards to be controlled;
– safety integrity requirements;
– apportionment of requirements and allocation of SIL to software and hardware;
– timing constraints.

NOTE The allocation of safety integrity requirements could lead to different SIL for well-separated software and hardware parts of a subsystem. This allocation depends on the contribution of the software and hardware parts of the subsystem to the safety-related functions and on the mechanisms for the failure mitigation including the separation of function with different SIL.

4.2 The software integrity shall be specified as one of five levels, from SIL 0 (the lowest) to SIL 1 to 4 for Safety Integrity.

4.3 The required software safety integrity level shall be decided and assessed at system level, on the basis of the system safety integrity level and the level of risk associated with the use of the software in the system.

4.4 The SIL 0 requirements of this Standard shall be fulfilled for the software part of functions that have a degree of uncertainty about safety impact typically below SIL 1.

4.5 To conform to this Standard it shall be shown that the requirements defined in each subclause have been satisfied with respect to the software safety integrity level and techniques and methods defined in Annex A.

4.6 Where a requirement is qualified by the words "to the extent required by the software safety integrity level", this indicates that a range of techniques and measures shall be used to satisfy that requirement.

4.7 Where 4.6 is applied, tables from normative Annex A shall be used to assist in the selection of techniques and measures appropriate to the software safety integrity level. The selection shall be documented in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan. Guidance to these techniques is given in the informative Annex D.

4.8 If a technique or measure which is ranked as highly recommended (HR) in the tables is not used, then the rationale for using alternative techniques shall be detailed and recorded either in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan. This is not necessary if an approved combination of techniques given in the corresponding table is used. The selected techniques shall be demonstrated to have been applied correctly.

4.9 If a technique or measure is proposed to be used that is not contained in the tables then its effectiveness and suitability in meeting the particular requirement and overall objective of the subclause shall be justified and recorded in either the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan.

4.10 Compliance with the requirements of a particular sub-clause and their respective techniques and measures detailed in the tables shall be verified by the inspection of

documents required by this Standard. Where appropriate, other objective evidence, auditing and the witnessing of tests shall also be taken into account.

5 Software management and organisation

5.1 Organisation, roles and responsibilities

5.1.1 Objective

5.1.1.1 To ensure that all personnel who have responsibilities for the software are organised, empowered and capable of fulfilling their responsibilities.

NOTE Independence between roles could be required in order to reduce the possibility of people in different roles suffering from the same misconceptions or making the same mistakes. This form of independence can be achieved by employing different people in different roles but does not usually require the roles to be located in different parts of the organisation or in different companies (unless specifically required).It is also important that people in roles which involve making judgements about the acceptability of a product or process from the point of view of safety should not be influenced by pressure from their peers or supervisors, or by considerations of commercial gain. This form of independence is more likely to require different roles to be located in different parts of the organisation or to be located in a different company. In general, a greater degree of safety requires a greater degree of independence for various roles and organisation.

5.1.2 Requirements

5.1.2.1 As a minimum, the supplier shall implement the parts of ISO 9001:2008 dealing with the organisation and management of the personnel and responsibilities.

5.1.2.2 Responsibilities shall be compliant with the requirements defined in Annex B.

5.1.2.3 The personnel assigned to the roles involved in the development or maintenance of the software shall be named and recorded.

5.1.2.4 An Assessor shall be appointed by the supplier, the customer or the Safety Authority.

5.1.2.5 The Assessor shall be independent from the supplier. However, at the discretion of the Safety Authority, the assessor can be part of the supplier's organisation or of the customer's organization but independent from the development project.

5.1.2.6 The Assessor shall be independent from the project.

5.1.2.7 The Assessor shall be given authority to perform the assessment of the software.

5.1.2.8 The Validator shall give agreement/disagreement for the software release.

5.1.2.9 Throughout the Software Lifecycle, the assignment of roles to persons shall be in accordance with 5.1.2.10 to 5.1.2.14, to the extent required by software SIL.

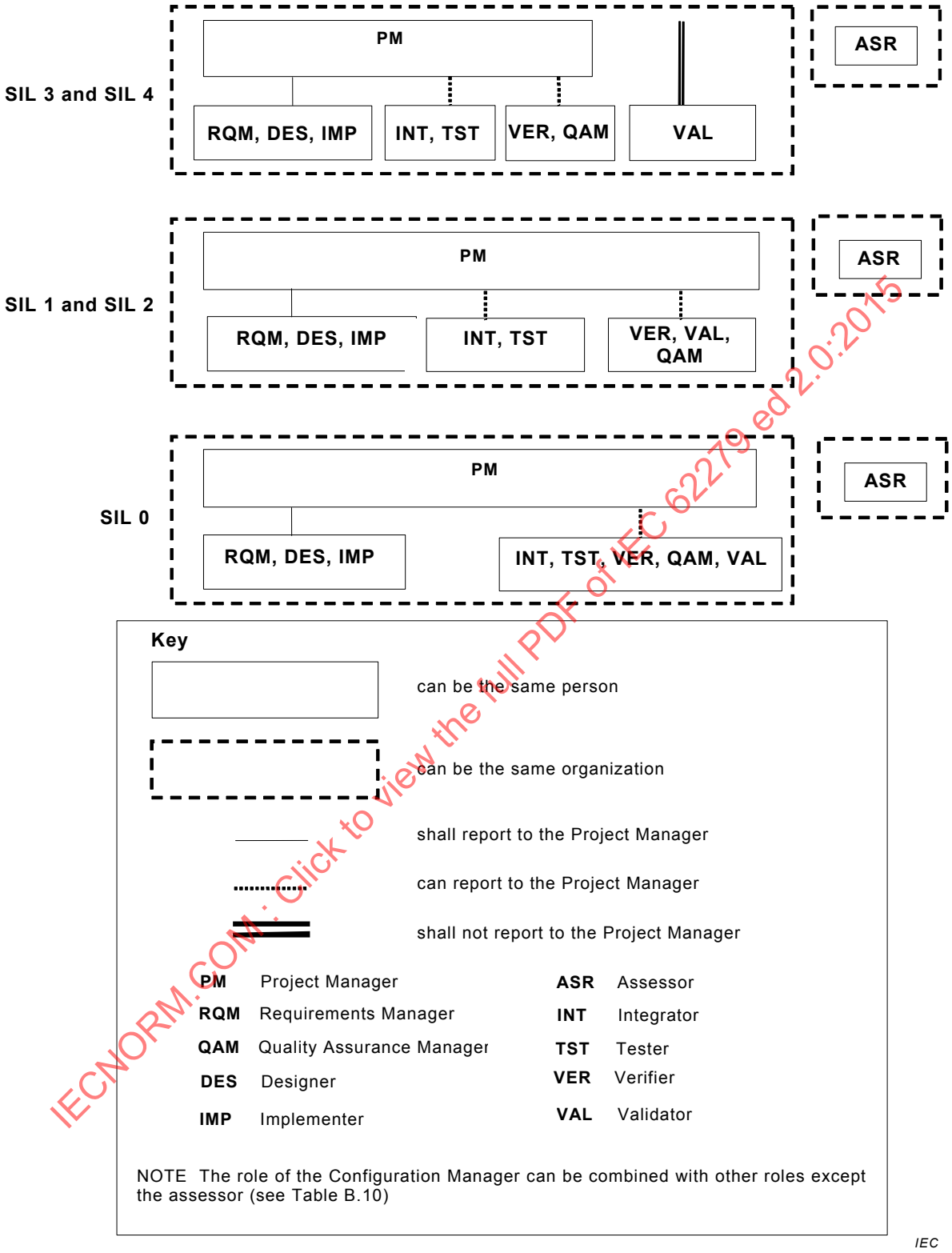


Figure 2 – Illustration of the preferred organisational structure

NOTE Figure 2 is only illustrative for the preferred organisational structure.

5.1.2.10 The preferred organisational structure for SIL 3 and SIL 4 is:

a) Requirements Manager, Designer and Implementer for a software component can be the same person.

b) Requirements Manager, Designer and Implementer for a software component shall report to the Project Manager.

c) Integrator and Tester for a software component can be the same person.

d) Integrator and Tester for a software component can report to the Project Manager or to the Validator.

e) Verifier or Quality Assurance Manager can report to the Project Manager or to the Validator.

f) Validator shall not report to the Project Manager i.e. the Project Manager shall have no influence on the validator's decisions but the validator informs the Project Manager about his decisions.

g) A person who is Requirements Manager, Designer or Implementer for a software component shall neither be Tester nor Integrator for the same software component.

h) A person who is Integrator or Tester for a software component shall neither be Requirements Manager, Designer nor Implementer for the same software component.

i) A person who is Verifier or Quality Assurance Manager shall neither be Requirements Manager, Designer, Implementer, Integrator, Tester nor Validator.

j) A person who is Validator shall neither be Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester nor Verifier.

k) A person who is Project Manager can additionally perform the roles of Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester or Verifier providing that the requirements for the independence between these additional roles are respected.

l) Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.

m) The assessor shall be independent and organisationally independent from the roles of Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator.

However, the following options may apply:

n) A person who is Validator may also perform the role of Verifier, but still maintaining independence from the Project Manager. In this case the Verifier's output documents shall be reviewed by another competent person with the same level of independence as the Validator. This organisational option shall be subject to Assessor's approval.

o) A person who is Verifier or Quality Assurance Manager may also perform the role of Integrator and Tester, in which case the role of Validator shall check the adequacy of the documented evidence from integration and testing with the specified verification objectives.

5.1.2.11 The preferred organisational structure for SIL 1 and SIL 2 is:

a) Requirements Manager, Designer and Implementer for a software component can be the same person and shall report to the Project Manager.

b) Integrator and Tester for a software component can be the same person.

c) Integrator and Tester for a software component can report to the Project Manager or to the Validator.

d) Verifier, Quality Assurance Manager and Validator can be the same person.

e) Verifier, Quality Assurance Manager and Validator can report to the Project Manager.

f) A person who is Requirements Manager, Quality Assurance Manager, Designer or Implementer for a software component shall be neither Tester nor Integrator for the same software component.

g) A person who is Integrator or Tester for a software component shall neither be Requirements Manager, Designer nor Implementer for the same software component.

h) A person who is Verifier, Quality Assurance Manager or Validator shall neither be Requirements Manager, Designer, Implementer, Integrator nor Tester.

i) A person who is a Project Manager can additionally perform the roles of Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier or Validator provided that the requirements for the independence between these additional roles are respected.

j) Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.

k) The assessor shall be independent and organisationally independent from the roles of Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator.

However, the following options can apply:

l) A person who is Verifier or Quality Assurance Manager may also perform the role of Integrator and Tester, in which case the role of Validator shall include reviewing the Verifier's or Quality Assurance Manager's output documents hence maintaining two levels of checking within the project organisation.

m) A person who is Validator may also perform the role of Verifier, Quality Assurance Manager, Integrator and Tester. In this case the Verifier's or Quality Assurance Manager's output documents shall be reviewed by another competent person with the same level of independence as the Validator. This organisational option shall be subject to Assessor's approval.

5.1.2.12 The preferred organisational structure for SIL 0 is:

a) Requirements Manager, Designer and Implementer for a software component can be the same person and shall be managed by the Project Manager.

b) Integrator, Tester, Verifier, Quality Assurance Manager and Validator for a software component can be the same person.

c) Integrator, Tester, Verifier, Quality Assurance Manager and Validator can be managed by the Project Manager.

d) A person who is Requirements Manager, Designer or Implementer for a software component shall be neither Tester nor Integrator for the same software component.

e) A person who is Verifier, Quality Assurance Manager or Validator shall neither be Requirements Manager, Designer, nor Implementer.

f) A person who is Project Manager can additionally perform the roles of Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier or Validator providing that the requirements for the independence between these additional roles are respected.

g) Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.

h) The assessor shall be independent and organisationally independent from the roles of Project Manager, Requirements Manager, Quality Assurance Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator.

However, the following alternatives can apply:

i) Requirements Manager, Designer, Implementer, Integrator and Tester can be the same person.

j) The Validator ,Verifier and Quality Assurance Manager can also be the same person;

k) A person who is Verifier, Quality Assurance Manager or Validator shall neither be Requirements Manager, Designer, nor Implementer.

5.1.2.13 The roles Requirements Manager, Designer and Implementer for one component can perform the roles Tester and Integrator for a different component.

5.1.2.14 The roles of the Verifier, Quality Assurance Manager and the Validator shall be defined at the project level and shall remain unchanged throughout the development project.

5.2 Personnel competence

5.2.1 Objectives

To ensure that all personnel who have responsibilities for the software are competent to discharge those responsibilities by demonstrating the ability to perform relevant tasks correctly, efficiently and consistently to a high quality and under varying conditions.

5.2.2 Requirements

5.2.2.1 The key competences required for each role in the software development are defined in Annex B. If additional experience, capabilities or qualifications are required for a role in the software life cycle, these shall be defined in the Software Quality Assurance Plan.

5.2.2.2 Documented evidence of personnel competence, including technical knowledge, qualifications, relevant experience and appropriate training, shall be maintained by the supplier's organisation in order to demonstrate appropriate safety organisation.

5.2.2.3 Once it has been proved to the satisfaction of an assessor or by a certification that competence has been demonstrated for all personnel appointed in various roles, each individual will need to show continuous maintenance and development of competence. This could be demonstrated by keeping a logbook showing the activity is being regularly carried out correctly, and that additional training is being undertaken in accordance with ISO 9001 and ISO/IEC 90003:2014, 6.2.2 "Competence, awareness and training".

5.2.2.4 The organisation shall maintain procedures to manage the competence of personnel to suit appropriate roles in accordance to existing quality standards.

5.3 Life cycle issues and documentation

5.3.1 Objectives

5.3.1.1 To structure the development of the software into defined phases and activities.

5.3.1.2 To record all information pertinent to the software throughout the life cycle of the software.

5.3.2 Requirements

5.3.2.1 A life cycle model for the development of software shall be selected. It shall be detailed in the Software Quality Assurance Plan in accordance with 6.5.

Two examples of life cycle models are shown in Figure 3 and Figure 4.

5.3.2.2 The life cycle model shall take into account the possibility of iterations in and between phases.

5.3.2.3 Quality Assurance procedures shall run in parallel with life cycle activities and use the same terminology.

5.3.2.4 The Software Quality Assurance Plan, Software Verification Plan, Software Validation Plan and Software Configuration Management Plan shall be drawn up at the start of the project and maintained throughout the software development life cycle.

5.3.2.5 All activities to be performed during a phase shall be defined and planned prior to the commencement of the phase.

5.3.2.6 All documents shall be structured to allow continued expansion in parallel with the development process.

5.3.2.7 For each document, traceability shall be provided in terms of a unique reference number and a defined and documented relationship with other documents.

5.3.2.8 Each term, acronym or abbreviation shall have the same meaning in every document. If, for historical reasons, this is not possible, the different meanings shall be listed and the references given.

5.3.2.9 Except for documents relating to pre-existing software (see 7.3.4.7), each document shall be written according to the following rules:

– it shall contain or implement all applicable conditions and requirements of the preceding document with which it has a hierarchical relationship;

– it shall not contradict the preceding document.

5.3.2.10 Each item or concept shall be referred to by the same name or description in every document.

5.3.2.11 The contents of all documents shall be recorded in a form appropriate for manipulation, processing and storage.

5.3.2.12 When documents which are produced by independent roles are combined into a single document, the relation to the parts produced by any independent role shall be traced within the document.

5.3.2.13 Documents may be combined or divided in accordance with 5.3.2.12. Some development steps may be combined, divided or, when justified, eliminated, at the discretion of the Project Manager and with the agreement of the Validator.

5.3.2.14 Any life cycle and documentation structure chosen shall meet all the objectives and requirements of this Standard.

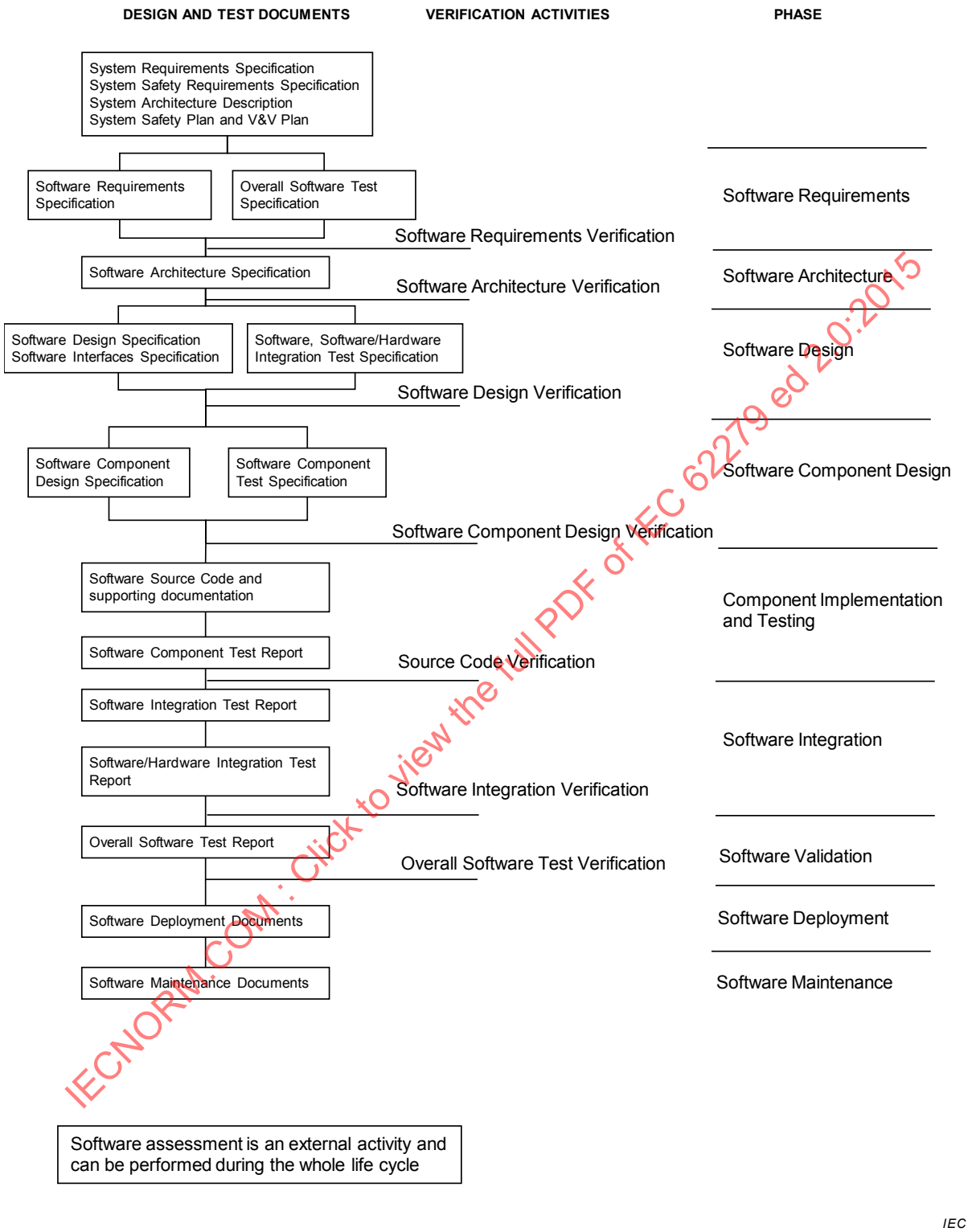


Figure 3 – Illustrative Development Life cycle 1

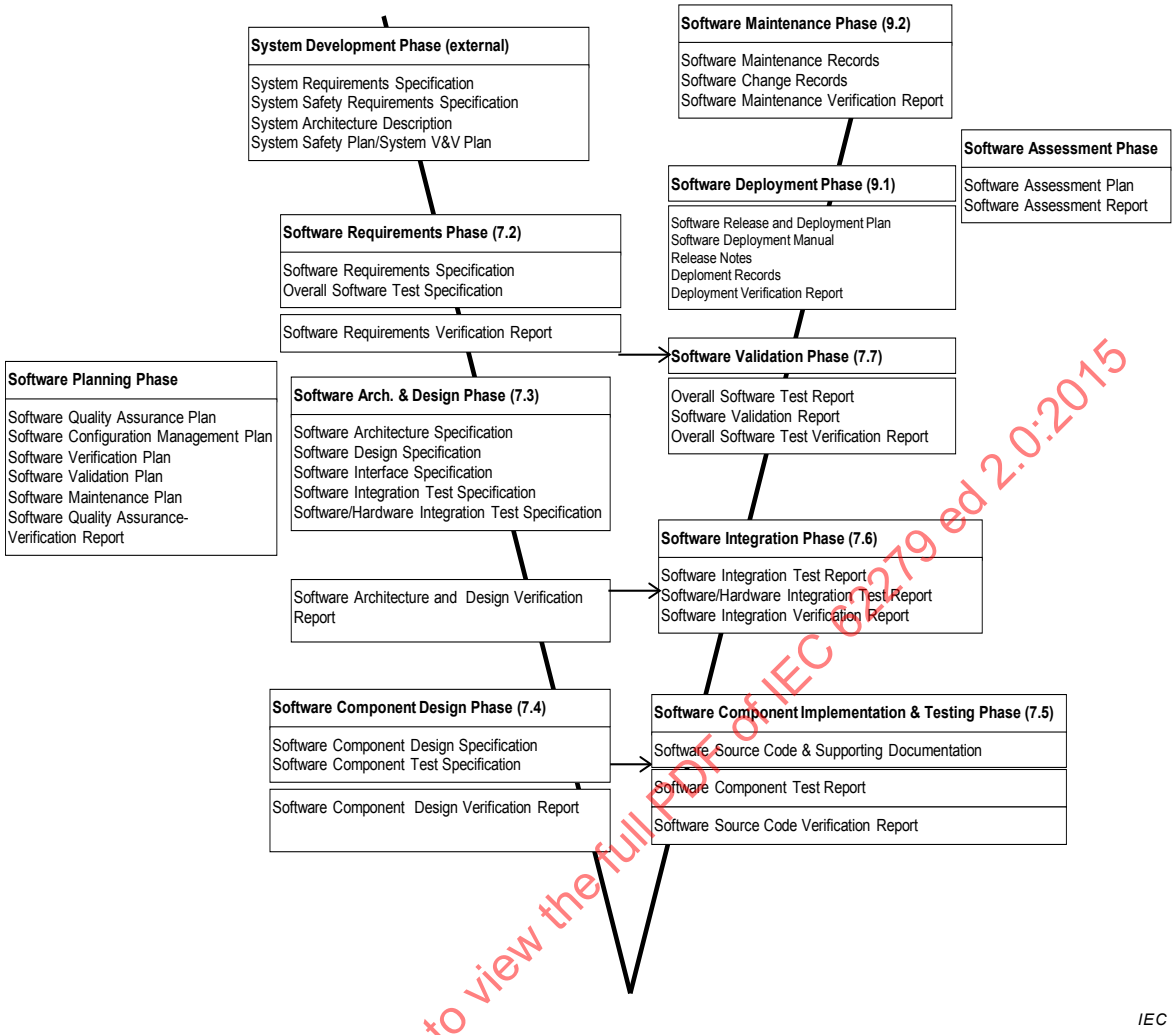


Figure 4 – Illustrative Development Life cycle 2

6 Software assurance

6.1 Software testing

6.1.1 Objective

The objective of software testing, as performed by the Tester and/or Integrator, is to ascertain the behaviour or performance of software against the corresponding test specification to the extent achievable by the selected test coverage.

6.1.2 Input documents

All necessary System, Hardware and Software Documentation as specified in the Software Verification Plan.

6.1.3 Output documents

a) Overall Software Test Specification

b) Overall Software Test Report

c) Software Integration Test Specification

d) Software Integration Test Report

e) Software/Hardware Integration Test Specification

f) Software/Hardware Integration Test Report

g) Software Component Test Specification

h) Software Component Test Report

6.1.4 Requirements

6.1.4.1 Tests performed by other parties such as the Requirements Manager, Designer or Implementer, if fully documented and complying with the following requirements, may be accepted by the Verifier.

6.1.4.2 Measurement equipment used for testing shall be calibrated appropriately. Any tools, hardware or software, used for testing shall be shown to be suitable for the purpose.

6.1.4.3 Software testing shall be documented by a Test Specification and a Test Report, as defined in the following.

6.1.4.4 Each Test Specification shall document the following:

a) test objectives;

b) test cases, test data and expected results;

c) types of tests to be performed;

d) test environment, tools, configuration and programs;

e) test criteria on which the completion of the test will be judged;

f) the criteria and degree of test coverage to be achieved;

g) the roles and responsibilities of the personnel involved in the test process;

h) the requirements which are covered by the test specification;

i) the selection and utilisation of the software test equipment.

6.1.4.5 A Test Report shall be produced as follows:

a) the Test Report shall mention the Tester names, state the test results and whether the test objectives and test criteria of the Test Specification have been met. Failures shall be documented and summarized;

b) test cases and their results shall be recorded, preferably in a machine-readable form for subsequent analysis;

c) tests shall be repeatable and, if practicable, be performed by automatic means;

d) test scripts for automatic test execution shall be verified;

e) the identity and configuration of all items involved (hardware used, software used, equipment used, equipment calibration, version information of the software tested, as well as version information of the test specification) shall be documented;

f) an evaluation of the test coverage and test completion shall be provided and any deviations noted.

6.2 Software verification

6.2.1 Objective

The objective of software verification is to examine and arrive at a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfil the requirements and plans with respect to completeness, correctness and consistency. These activities are managed by the Verifier.

6.2.2 Input documents

All necessary System, Hardware and Software Documentation.

6.2.3 Output documents

a) Software Verification Plan

b) Software Verification Report(s)

c) Software Quality Assurance Verification Report

6.2.4 Requirements

6.2.4.1 Verification shall be documented by at least a Software Verification Plan and one or more (process-related) Verification Reports.

6.2.4.2 A Software Verification Plan shall be written, under the responsibility of the Verifier, on the basis of the necessary documentation.

Requirements from 6.2.4.3 to 6.2.4.9 refer to the Software Verification Plan.

6.2.4.3 The Software Verification Plan shall describe the activities to be performed to ensure proper verification and that particular design or other verification needs are suitably provided for.

6.2.4.4 During development (and depending upon the size of the system) the plan may be sub-divided into a number of child documents and be added to, as the detailed needs of verification become clearer.

6.2.4.5 The Software Verification Plan shall document all the criteria, techniques and tools to be used in the verification process. The Software Verification Plan shall include techniques and measures chosen from Table A.5, Table A.6, Table A.7 and Table A.8. The selected combination shall be justified as a set satisfying 4.8, 4.9 and 4.10.

6.2.4.6 The Software Verification Plan shall describe the activities to be performed to ensure correctness and consistency with respect to the input to that phase. These include reviewing, testing and integration.

6.2.4.7 In each development phase it shall be shown that the functional, performance and safety requirements are met.

6.2.4.8 The results of each verification shall be retained in a format defined or referenced in the Software Verification Plan.

6.2.4.9 The Software Verification Plan shall address the following:

a) the selection of verification strategies and techniques (to avoid undue complexity in the assessment of the verification and testing, preference shall be given to the selection of techniques which are in themselves readily analysable);

b) selection of techniques from Table A.5, Table A.6, Table A.7 and Table A.8;

c) the selection and documentation of verification activities;

d) the evaluation of verification results gained;

e) the evaluation of the safety and robustness requirements;

f) the roles and responsibilities of the personnel involved in the verification process;

g) the degree of the functional based test coverage required to be achieved;

h) the structure and content of each verification step, especially for the Software Requirement Verification (7.2.4.22), Software Architecture and Design Verification (7.3.4.41, 7.3.4.42), Software Components Verification (7.4.4.13), Software Source Code Verification (7.5.4.10) and Integration Verification (7.6.4.13) in a way that facilitates review against the Software Verification Plan.

6.2.4.10 A Software Quality Assurance Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 6.2.2.

The requirement in 6.2.4.12 refers to the Software Quality Assurance Verification Report.

6.2.4.11 The Verifier has to make sure that the verification of the Software Verification Plan is carried out by a person who is competent in this matter. In order to be in line with good practice and independence rules of this standard the Verifier shall not verify the Software Verification Plan himself, in case he is the author of it.

6.2.4.12 Once the Software Verification Plan has been established, verification shall address

a) that the Software Verification Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.2.4.3 to 6.2.4.9,

b) the internal consistency of the Software Verification Plan.

The results shall be recorded in a Software Quality Assurance Verification Report.

6.2.4.13 Any Software Verification Reports shall be written, under the responsibility of the Verifier, on the basis of the input documents. These reports can be partitioned for clarity and convenience, and shall follow the Software Verification Plan. The requirement in 6.2.4.14 refers to the Software Verification Reports.

6.2.4.14 Each Software Verification Report shall document the following:

a) the identity and configuration of the items verified, as well as the Verifier names;

b) items which do not conform to the specifications;

c) components, data, structures and algorithms poorly adapted to the problem;

d) detected errors or deficiencies;

e) the fulfilment of, or deviation from, the Software Verification Plan (in the event of deviation the Verification Report shall explain whether the deviation is critical or not);

f) assumptions if any;

g) a summary of the verification results.

6.3 Software validation

6.3.1 Objective

6.3.1.1 The objective of software validation is to demonstrate that the processes and their outputs are such that the software is of the defined software safety integrity level, fulfils the software requirements and is fit for its intended application. This activity is performed by the Validator.

6.3.1.2 The main validation activities are to demonstrate by analysis and/or testing that all the software requirements are specified, implemented, tested and fulfilled as required by the applicable SIL, and to evaluate the safety criticality of all anomalies and non-conformities based on the results of reviews, analyses and tests.

6.3.2 Input documents

All system, hardware and software documentation as specified in this Standard.

6.3.3 Output documents

a) Software Validation Plan

b) Software Validation Report

6.3.4 Requirements

6.3.4.1 The Software Validation activities shall be developed and performed, with their results evaluated, by a Validator with an appropriate level of independence as defined in 5.1.

6.3.4.2 Validation shall be documented with, at least, a Software Validation Plan and a Software Validation Report, as defined in the following.

6.3.4.3 A Software Validation Plan shall be written, under the responsibility of the Validator, on the basis of the input documents.

Requirements from 6.3.4.4 to 6.3.4.6 refer to the Software Validation Plan.

6.3.4.4 The Software Validation Plan shall include a summary justifying the validation strategy chosen. The justification shall include consideration, according to the required software safety integrity level, of

a) manual or automated techniques or both,

b) static or dynamic techniques or both,

c) analytical or statistical techniques or both,

d) testing in a real or simulated environment or both.

6.3.4.5 The Software Validation Plan shall identify the steps necessary to demonstrate the adequacy of any Software Specification in fulfilling the safety requirements set out in the System Safety Requirements Specification.

6.3.4.6 The Software Validation Plan shall identify the steps necessary to demonstrate the adequacy of the Overall Software Test Specification as a test against the Software Requirements Specification.

6.3.4.7 A Software Validation Report shall be written, under the responsibility of the Validator, on the basis of the input documents.

Requirements from 6.3.4.8 to 6.3.4.11 refer to the Software Validation Report.

6.3.4.8 The results of the validation shall be documented in the Software Validation Report.

6.3.4.9 The Validator shall check that the verification process is complete.

6.3.4.10 The Software Validation Report shall fully state the software baseline that has been validated.

6.3.4.11 The Validation Report shall clearly identify any known deficiencies in the software and the impact these may have on the use of the software.

6.3.4.12 The entity responsible for Validation shall provide review of the output documents from 6.3.3.

6.3.4.13 Once the Software Validation Plan has been established, the review of this document shall address:

a) that the Software Validation Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.3.4.4 to 6.3.4.6,

b) the internal consistency of the Software Validation Plan.

6.3.4.14 Once the Software Validation Report has been established, the review of this document shall address:

a) that the Software Validation Report meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.3.4.8 to 6.3.4.11 and 7.7.4.8 to 7.7.4.12,

b) the internal consistency of the Software Validation Report.

6.3.4.15 The Validator shall be empowered to require or perform additional reviews, analyses and tests.

6.3.4.16 The software shall only be released for operation after authorisation by the Validator.

6.3.4.17 Simulation and modelling may be used to supplement the validation process.

6.4 Software assessment

6.4.1 Objective

6.4.1.1 To evaluate that the lifecycle processes and their outputs are such that the software is of the defined software safety integrity levels 1 to 4 and is fit for its intended application.

6.4.1.2 For SIL 0 software, requirements of this standard shall be fulfilled but where a certificate stating compliance with ISO 9001 is available, no assessment will be required.

6.4.2 Input documents

a) System Safety Requirements Specification

b) Software Requirements Specification

c) All other documents necessary to carry out the assessment process.

6.4.3 Output documents

a) Software Assessment Plan

b) Software Assessment Report

6.4.4 Requirements

6.4.4.1 The assessment of the software shall be carried out by an Assessor who is independent as described in 5.1.2.6 and 5.1.2.7.

6.4.4.2 Software with a Software Assessment Report from another Assessor does not have to be an object of a new assessment. The assessor shall check that the software is fit for its intended use within the intended environment, and that the former assessment stated the software has achieved a safety integrity level at least equal to the required level.

6.4.4.3 The Assessor shall have access to all project-related documentation throughout the development process.

6.4.4.4 A Software Assessment Plan shall be written, under the responsibility of the Assessor, on the basis of the input documents from 6.4.2. Where appropriate, an existing documented generic Software Assessment Plan or procedure may be used. The requirement in 6.4.4.5 refers to the Software Assessment Plan.

6.4.4.5 The Software Assessment Plan shall include the following scope:

a) aspects with which the assessment deals;

b) activities throughout the assessment process and their sequential link to engineering activities;

c) documents to be taken into consideration;

d) statements on pass/fail criteria and the way to deal with non-conformance cases;

e) requirements with regard to content and form of the Software Assessment Report.

6.4.4.6 The entity responsible for assessment shall provide review of the assessment on the basis of the input documents from 6.4.2.

Requirement 6.4.4.7 refers to the internal verification or peer review of the assessment.

6.4.4.7 Once the Software Assessment Plan has been established, verification shall address

a) that the Software Assessment Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.4.4.5,

b) the internal consistency of the Software Assessment Plan.

6.4.4.8 The Assessor shall assess that the software of the system is fit for its intended purpose and responds correctly to safety issues derived from the System Safety Requirements Specification.

6.4.4.9 The Assessor shall assess if an appropriate set of techniques from Annex A, suitable for the intended development, has been selected and applied in accordance to the required safety integrity level.

Moreover, the assessor shall consider the extent to which each technique from Annex A is applied, i.e. whether it is applied to all or to only part of the software, and also look for evidence that it is properly applied.

6.4.4.10 The Assessor shall assess the configuration and change management system and the evidences on its use and application.

6.4.4.11 The Assessor shall review the evidence of the competency of the project staff according to Annex B and shall assess the organisation for the software development according to 5.1.

6.4.4.12 For any software containing safety-related application conditions, the Assessor shall check for noted deviations, non-compliances to requirements and recorded non-conformities if these have an impact on safety, and make a judgment whether the justification from the project is acceptable. The result shall be stated in the assessment report.

6.4.4.13 The Assessor shall assess the verification and validation activities and the supporting evidence.

6.4.4.14 The Assessor shall agree the scope and contents of the Software Validation Plan. This agreement shall also make a statement concerning the presence of the Assessor during testing.

6.4.4.15 The Assessor may carry out audits and inspections (e.g. witnessing tests) throughout the entire development process. The Assessor may ask for additional verification and validation work.

NOTE It is of advantage to involve the Assessor early in the project.

6.4.4.16 A Software Assessment Report shall be written under the responsibility of the Assessor. Requirements from 6.4.4.17 to 6.4.4.19 refer to the Software Assessment Report.

6.4.4.17 The Software Assessment Report shall meet the requirements of the Software Assessment Plan and provide a conclusion and recommendations.

6.4.4.18 The Assessor shall record his/her activities as a consistent base for the Software Assessment Report. These shall be summarised in the Software Assessment report.

6.4.4.19 The Assessor shall identify and evaluate any non-conformity with the requirements of this Standard and judge the impact on the final result. These non-conformities and their judgments shall be listed in the Software Assessment Report.

6.5 Software quality assurance

6.5.1 Objectives

6.5.1.1 To identify, monitor and control all those activities, both technical and managerial, which are necessary to ensure that the software achieves the quality required. This is

necessary to provide the required qualitative defence against systematic faults and to ensure that an audit trail can be established to allow verification and validation activities to be undertaken effectively.

6.5.1.2 To provide evidence that the above activities have been carried out.

6.5.2 Input documents

All the documents available at each stage of the lifecycle.

6.5.3 Output documents

a) Software Quality Assurance Plan

b) Software Configuration Management Plan, if not available at system level

c) Software Quality Assurance Verification Report

NOTE The Software Quality Assurance Report is included in the Quality Management Report defined in IEC 62425.

6.5.4 Requirements

6.5.4.1 All the plans shall be issued at the beginning of the project and updated during the lifecycle.

6.5.4.2 The organisations taking part in the software development shall implement and use a Quality Assurance System compliant with ISO 9000, to support the requirements of this Standard. ISO 9001 certification is highly recommended.

6.5.4.3 A Software Quality Assurance Plan shall be written, under the responsibility of the Quality Assurance Manager, on the basis of the input documents from 6.5.2.

The requirements from 6.5.4.4 to 6.5.4.6 refer to the Software Quality Assurance Plan.

6.5.4.4 A Software Quality Assurance Plan shall be written and shall be specific to the project. It shall implement the requirements of 6.5.4.5.

6.5.4.5 As a minimum, the following items shall be specified or referenced in the Software Quality Assurance Plan.

a) Definition of the life cycle model:
   1) activities and elementary tasks consistent with the plans, e.g. Safety Plan, that have been established at the System level;
   2) entry and exit criteria of each activity;
   3) inputs and outputs of each activity;
   4) major quality activities;
   5) the entity responsible for each activity.

b) Documentation structure.

c) Documentation control:
   1) roles involved for writing, checking and approval;
   2) scope of distribution;
   3) archiving.

d) Tracking and tracing of deviations.

e) Methods, measures and tools for quality assurance according to the allocated safety integrity levels (refer to Annex A).

f) Justifications, as defined in 4.7 to 4.9, that each combination of techniques or measures selected according to Annex A is appropriate to the defined software safety integrity level.

Some of the Software Quality Assurance Plan required information may be contained in other documents, such as a separate Software Configuration Management Plan, a Maintenance Plan, a Software Verification plan and a Software Validation Plan. The subclauses of the Software Quality Assurance Plan shall reference the documents in which the information is contained. In any case the content of each subclause of the Software Quality Assurance Plan shall be specified either directly or by reference to another document.

The referenced documents shall be reviewed in order to ensure they provide all the required information and that they fully address the requirements of this Standard.

6.5.4.6 Quality assurance activities, actions, documents, etc. required by all normative subclauses of this Standard shall be specified or referenced in the Software Quality Assurance Plan and tailored to the specific project.

6.5.4.7 A Software Quality Assurance Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 6.5.2.

The requirement in 6.5.4.8 refers to the Software Quality Assurance Verification Report.

6.5.4.8 Once the Software Quality Assurance Plan has been established, verification shall address

a) that the Software Quality Assurance Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.5.4.4 to 6.5.4.6,

b) the internal consistency of the Software Quality Assurance Plan.

The results shall be recorded in a Software Quality Assurance Verification Report.

6.5.4.9 Each planning document shall have a paragraph specifying details about its own updating throughout the project: frequency, responsibility, method.

6.5.4.10 Each software document and deliverable shall be placed under configuration control from the time of its first release.

6.5.4.11 Changes to all items under Configuration Management Control shall be authorised and recorded.

6.5.4.12 In addition to software development, the Configuration Management System shall also cover the software development environment used during the full life cycle.

This extension, necessary for the reproducibility of the development and for the maintenance activities, shall include all the tools, translators, data and test files, parameterisation files, and supporting hardware platforms.

6.5.4.13 The supplier shall establish, document and maintain procedures for control of the external suppliers, including

– methods and relevant records to ensure that software provided by external suppliers adheres to established requirements. Previously developed software shall be assured to be compliant with the required software safety integrity level and dependability. New software shall be developed and maintained in conformity with the Software Quality Assurance Plan of the Supplier or with a specific Software Quality Assurance Plan prepared by the external supplier in accordance with the Software Quality Assurance Plan of the Supplier,

– methods and relevant records to ensure that the requirements provided to the External Supplier are adequate and complete.

6.5.4.14 Traceability to requirements shall be an important consideration in the validation of a safety-related system and means shall be provided to allow this to be demonstrated throughout all phases of the life cycle.

6.5.4.15 Within the context of this Standard, and to a degree appropriate to the specified software safety integrity level, traceability shall particularly address

a) traceability of requirements to the design or other objects which fulfil them,

b) traceability of design objects to the implementation objects which instantiate them,

c) traceability of requirements and design objects to the tests (component, integration, overall test) and analyses that verify them.

Traceability shall be the subject of configuration management.

6.5.4.16 In special cases, e.g. pre-existing software or prototyped software, traceability may be established after the implementation and/or documentation of the code, but prior to verification/validation. In these cases, it shall be shown that verification/validation is as effective as it would have been with traceability over all phases.

6.5.4.17 Objects of requirements, design or implementation that cannot be adequately traced shall be demonstrated to have no bearing upon the safety or integrity of the system.

6.6 Modification and change control

6.6.1 Objectives

6.6.1.1 To ensure that the software performs as required, preserving the software safety integrity and dependability when modifying the software.

6.6.1.2 These objectives are managed by the Configuration Manager.

6.6.2 Input documents

a) Software Quality Assurance Plan

b) Software Configuration Management Plan

c) All relevant design, development and analysis documentation

d) Change Requests

e) Change impact analysis and authorisation

6.6.3 Output documents

a) All changed input documents

b) Software Change records (see 9.2.4.11)

c) New Configuration records

6.6.4 Requirements

6.6.4.1 The Change Management Process shall define at least the following aspects:

a) the documentation needed for problem reporting and/or corrective actions, with the aim of giving feedback to the responsible management;

b) analysis of the information collected in the problem reports to identify its causes;

c) the practices to be followed for reporting, tracking and resolving problems identified both during the development phase and during software maintenance;

d) the specific organisational responsibilities with regard to development and software maintenance;

e) how to apply controls to ensure that corrective actions are taken and that they are effective;

f) impact analysis of the effect of the changes on the software component under development or already delivered;

g) impact analysis shall state the re-verification, re-validation and re-assessment necessary for the change;

h) where multiple changes are applied, the impact analysis shall consider the cumulative impact;

NOTE Several changes could cumulatively require a complete re-test.

i) authorisation before implementation.

6.6.4.2 All changes shall initiate a return to an appropriate phase of the lifecycle. All subsequent phases shall then be carried out in accordance with the procedures specified for the specific phases in accordance with the requirements in this Standard.

6.7 Support tools and languages

6.7.1 Objectives

The objective is to provide evidence that potential failures of tools do not adversely affect the integrated toolset output in a safety related manner that is undetected by technical and/or organisational measures outside the tool. To this end, software tools are categorised into three classes namely, T1, T2 and T3 respectively (see definitions in 3.1).

When tools are used as a replacement for manual operations, the evidence of the integrity of tools output can be adduced by the same process steps as if the output was done in manual operation. These process steps might be replaced by alternative methods if an argumentation on the integrity of tools output is given and the integrity level of the software is not decreased by the replacement.

6.7.2 Input documents

Tools specification or manual.

6.7.3 Output documents

Tools validation report (when needed see 6.7.4.4 or 6.7.4.6).

6.7.4 Requirements

6.7.4.1 Software tools shall be selected as a coherent part of the software development activities.

NOTE Appropriate tools to support the development of software are used in order to increase the integrity of the software by reducing the likelihood of introducing or not detecting faults during the development. Examples of tools relevant to the phases of the software development lifecycle include

a) transformation or translation tools that convert a software or design representation (e.g. text or a diagram) from one abstraction level to another: design refinement tools, compilers, assemblers, linkers, binders, loaders and code generation tools,

b) verification and validation tools such as static code analysers, test coverage monitors, theorem proving assistants, simulators and model checkers,

c) diagnostic tools used to maintain and monitor the software under operating conditions,

d) infrastructure tools such as development support systems,

e) configuration control tools such as version control tools,

f) application data tools that produce or maintain data which are required to define parameters and to instantiate system functions e.g. function parameters, instrument ranges, alarm and trip levels, output states to be adopted at failure, geographical layout.

The selected tools are able to cooperate. In this context, tools cooperate if the outputs from one tool have suitable content and format for automatic input to a subsequent tool, thus minimizing the possibility of introducing human error in the reworking of intermediate results.

Tools are usually selected and demonstrated to be compatible with the needs of the application.

The availability of suitable tools to supply the services that are necessary over the whole lifetime of the software is considered as well.

6.7.4.2 The selection of the tools in classes T2 and T3 shall be justified (see 7.3.4.12). The justification shall include the identification of potential failures which can be injected into the tools output and the measures to avoid or handle such failures.

6.7.4.3 All tools in classes T2 and T3 shall have a specification or manual which clearly defines the behaviour of the tool and any instructions or constraints on its use.

6.7.4.4 For each tool in class T3, evidence shall be available that the output of the tool conforms to the specification of the output or failures in the output are detected. Evidence may be based on the same steps necessary for a manual process as a replacement for the tool and an argument presented if these steps are replaced by alternatives (e. g. validation of the tool). Evidence may also be based on

a) a suitable combination of history of successful use in similar environments and for similar applications (within the organisation or other organisations),

b) tool validation as specified in 6.7.4.5,

c) diverse redundant code which allows the detection and control of failures resulting in faults introduced by a tool,

d) compliance with the safety integrity levels derived from the risk analysis of the process and procedures including the tools,

e) other appropriate methods for avoidance or detection and control of failures introduced by tools.

NOTE 1 A version history could provide assurance of maturity of the tool, and a record of the errors / ambiguities associated with its use in the environment.

NOTE 2 The evidence listed for T3 could also be used for T2 tools in judging the correctness of their results.

6.7.4.5 The results of tool validation shall be documented covering the following results:

a) a record of the validation activities;

b) the version of the tool manual being used;

c) the tool functions being validated;

d) tools and equipment used;

e) the results of the validation activity; the documented results of validation shall state either that the software has passed the validation or the reasons for its failure;

f) test cases and their results for subsequent analysis;

g) discrepancies between expected and actual results.

6.7.4.6 Where the conformance evidence of 6.7.4.4 is unavailable, there shall be effective measures to control failures of the executable safety related software that result from faults that are attributable to the tool.

NOTE 1 An example is the generation of diverse redundant code which allows the detection and control of failures resulting in faults introduced by a translator.

NOTE 2 As an example, the fitness for purpose of a non-trusted compiler can be justified as follows.

The object code produced by the compiler has been subjected to a combination of tests, checks and analyses which are capable of ensuring the correctness of the code to the extent that it is consistent with the target Safety Integrity Level. In particular, the following applies to all tests, checks and analyses.

– Testing has been shown to have a sufficiently high coverage of the implemented code. If there is any code unreachable by testing, it has been shown by checks or analyses that the function concerned is executed correctly when the code is reached on the target.

– Checks and analyses have been applied to the object code and shown to be capable of detecting the types of errors which might result from a defect in the compiler.

– No more translation with the compiler has taken place after testing, checking and analysis.

– If further compilation or translation is carried out, all tests, checks and analyses will be repeated.

6.7.4.7 The software or design representation (including a programming language) selected shall

a) have a translator which has been evaluated for fitness for purpose including, where appropriate, evaluated against the international or national standards,

b) match the characteristics of the application,

c) contain features that facilitate the detection of design or programming errors,

d) support features that match the design method.

A programming language is one of a class of representations of software or design. A Translator converts a software or design representation (e.g. text or a diagram) from one abstraction level to another level. Examples of Translators include: design refinement tools, compilers, assemblers, linkers, binders, loaders and code generation tools.

The evaluation of a Translator may be performed for a specific application project, or for a class of applications. In the latter case all necessary information on the tool regarding the intended and appropriate use of the tool shall be available to the user of the tool. The evaluation of the tool for a specific project may then be reduced to checking general suitability of the tool for the project and compliance to the "specification or manual" (i.e. proper use of the tool). Proper use might include additional verification activities within the specific project.

A validation suite may be used to evaluate the fitness for purpose of a Translator according to defined criteria, which shall include functional and non-functional requirements. For the functional Translator requirements, dynamic testing may be a main validation technique. If possible an automatic testing suite shall be used.

6.7.4.8 Where 6.7.4.7 cannot be fully satisfied, the fitness for purpose of the language, and any additional measures which address any identified shortcomings of the language shall be justified and evaluated.

NOTE See NOTE 2 in 6.7.4.6.

6.7.4.9 Where automatic code generation or similar automatic translation takes place, the suitability of the automatic Translator for safety-related software development shall be evaluated at the point in the development life cycle where development support tools are selected.

6.7.4.10 Configuration management shall ensure that for tools in classes T2 and T3, only justified versions are used.

6.7.4.11 Each new version of a tool that is used shall be justified (see Table 1). This justification may rely on evidence provided for an earlier version if sufficient evidence is provided that

a) the functional differences (if any) will not affect tool compatibility with the rest of the toolset,

b) the new version is unlikely to contain significant new, unknown faults.

NOTE Evidence that the new version is unlikely to contain significant new unknown faults could be based on a credible identification of the changes made, and on an analysis of the verification and validation actions performed.

6.7.4.12 The relation between the tool classes and the applicable subclauses is defined within Table 1.

Table 1 – Relation between tool class and applicable subclauses

Tool class	Applicable subclauses
T1	6.7.4.1
T2	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.10, 6.7.4.11
T3	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.4, 6.7.4.5 (or 6.7.4.6), 6.7.4.7, 6.7.4.8, 6.7.4.9, 6.7.4.10, 6.7.4.11

NOTE The selection and application of tools are usually aligned with the safety integrity of the products under development. The relation between the tool classes and assurance methodologies appropriate to specific SILs for the products under development where the tools are employed is illustrated within Table 2.

Table 2 – Illustrative Relation between tool class and product SIL

Tool Class	Tool Assurance Methodology	SIL of Product under Development
T1	Not required	Not necessary
T2 and T3	1-Evidence of successful use in similar environments or 2-Run a library of test cases with deterministic outcomes to ascertain functional integrity	1-2
	1-Follow the full requirements of this standard pertinent to the specific application SIL to the development or acquisition of the tool or 2-Run a library of broadly recognised test cases/test suites with deterministic outcomes to ascertain functional integrity or 3-Apply diverse tools to the system and compare performance of the work product, checking for differences	3-4

7 Generic software development

7.1 Life cycle and documentation for generic software

7.1.1 Objectives

To provide a description of the software itself, from the higher levels of abstraction down to the detailed refinements, in order to create a frame for the demonstration of the achieved safety as well as for future maintenance actions.

7.1.2 Requirements

7.1.2.1 To the extent required by the software safety integrity level, the documents listed in Table A.1 shall be produced for a generic software.

7.1.2.2 The sequence of deliverable documents as they are described in Table A.1 reflects an ideal linear waterfall model. This model is however not intended to be a reference in the sense of schedule and linkage of activities, as it would usually be difficult to achieve a strict compliance in practice. Phases can overlap but verification and validation activities shall demonstrate the consistency of inputs and outputs (documents and software) within and between the phases.

However, the main purpose of the documentation foreseen is to provide a description of the software itself, from the higher levels of abstraction down to the detailed refinements, in order to create a frame for the demonstration of the achieved safety as well as for future maintenance actions.

7.2 Software requirements

7.2.1 Objectives

7.2.1.1 To describe a complete set of requirements for the software meeting all System and Safety Requirements related to software and provide a comprehensive set of documents for each subsequent phase.

7.2.1.2 To describe the Overall Software Test Specification.

7.2.2 Input documents

a) System Requirements Specification

b) System Safety Requirements Specification

c) System Architecture Description

d) External Interface Specifications (e.g. Software/Software Interface Specification, Software/Hardware Interface Specification)

e) Software Quality Assurance Plan

f) Software Validation Plan

7.2.3 Output documents

a) Software Requirements Specification

b) Overall Software Test Specification

c) Software Requirements Verification Report

7.2.4 Requirements

7.2.4.1 A Software Requirements Specification shall be written, under the responsibility of the Requirements Manager, on the basis of the input documents from 7.2.2.

The requirements from 7.2.4.2 to 7.2.4.15 refer to the Software Requirements Specification.

7.2.4.2 The Software Requirements Specification shall express the required properties of the software being developed. These properties, which are all (except safety) defined in ISO/IEC 25010 series, shall include

a) functionality (including capacity and response time performance),

b) robustness and maintainability,

c) safety (including safety functions and their associated software safety integrity levels),

d) efficiency,

e) usability,

f) portability.

7.2.4.3 The software safety integrity level shall be derived as defined in Clause 4 and recorded in the Software Requirements Specification.

7.2.4.4 To the extent required by the software safety integrity level, the Software Requirements Specification shall be expressed and structured in such a way that it is

a) complete, clear, precise, unequivocal, verifiable, testable, maintainable and feasible,

b) traceable back to all the input documents.

7.2.4.5 The Software Requirements Specification shall include modes of expression and descriptions which are understandable to the responsible personnel involved in the life cycle of the software.

7.2.4.6 The Software Requirements Specification shall identify and document all interfaces with any other system, either within or outside the equipment under control, including operators, wherever a direct connection exists or is planned.

7.2.4.7 All relevant modes of operation shall be detailed in the Software Requirements Specification.

7.2.4.8 All relevant modes of behaviour of the programmable electronics, in particular failure behaviour, shall be documented or referenced (e.g. system level documentation) in the Software Requirements Specification.

7.2.4.9 Any constraints between the hardware and the software shall be documented or referenced (e.g. system level documentation) in the Software Requirements Specification.

7.2.4.10 To the extent required by the description of system documentation, the Software Requirements Specification shall consider the software self-checking and the hardware checking by the software. Software self-checking consists of the detection and reporting by the software of its own failures and errors.

7.2.4.11 The Software Requirements Specification shall include requirements for the periodic testing of functions to the extent required by the System Safety Requirements Specification.

7.2.4.12 The Software Requirements Specification shall include requirements to enable all safety functions to be testable during overall system operation to the extent required by the System Safety Requirements Specification.

7.2.4.13 All functions to be performed by the software, especially those related to achieving the required system safety integrity level, shall be clearly identified in the Software Requirements Specification.

7.2.4.14 Any non-safety functions which the software is required to perform shall be clearly identified in the Software Requirements Specification.

7.2.4.15 The Software Requirements Specification shall be supported by techniques and measures from Table A.2. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.2.4.16 An Overall Software Test Specification shall be written, under the responsibility of the Tester, on the basis of the Software Requirements Specification.

The requirements from 7.2.4.17 to 7.2.4.19 refer to the Overall Software Test Specification.

7.2.4.17 The Overall Software Test Specification shall be a description of the tests to be performed on the completed software.

7.2.4.18 The Overall Software Test Specification shall choose techniques and measures from Table A.7. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.2.4.19 The Overall Software Test Specification shall identify for each required function the test cases including

a) the required input signals with their sequences and their values,

b) the anticipated output signals with their sequences and their values,

c) the test success criteria, including performance and quality aspects.

7.2.4.20 A Software Requirements Verification Report shall be written, under the responsibility of the Verifier, on the basis of the System Safety Requirements Specification, Software Requirements Specification, Overall Software Test Specification and Software Quality Assurance Plan.

Requirements from 7.2.4.21 to 7.2.4.22 refer to the Software Requirements Verification Report.

7.2.4.21 The Software Requirements Verification Report shall be written in accordance to the generic requirements established for all the Verification Reports (see 6.2.4.14).

7.2.4.22 Once the Software Requirements Specification has been established, verification shall address

a) the adequacy of the Software Requirements Specification in fulfilling the requirements set out in the System Requirements Specification, the System Safety Requirements Specification and the Software Quality Assurance Plan,

b) that the Software Requirements Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 7.2.4.2 to 7.2.4.15,

c) the adequacy of the Overall Software Test Specification as a test against the Software Requirements Specification,

d) the definition of any additional activity in order to demonstrate the correct coverage of not testable requirements,

e) the internal consistency of the Software Requirements Specification,

f) the adequacy of the Software Requirements Specification in fulfilling or taking into account the constraints between hardware and software.

The results shall be recorded in a Software Requirements Verification Report.

7.3 Architecture and Design

7.3.1 Objectives

7.3.1.1 To develop a software architecture that achieves the requirements of the software.

7.3.1.2 To identify and evaluate the significance of hardware/software interactions for safety.

7.3.1.3 To choose a design method if one has not been previously defined.

7.3.1.4 To design software of a defined software safety integrity level from the input documents.

7.3.1.5 To ensure that the resultant system and its software will be readily testable from the outset. As verification and test will be a critical element in the validation, particular consideration shall be given to verification and test needs throughout the implementation.

7.3.2 Input documents

Software Requirements Specification

7.3.3 Output documents

a) Software Architecture Specification

b) Software Design Specification

c) Software Interface Specifications

d) Software Integration Test Specification

e) Software/Hardware Integration Test Specification

f) Software Architecture and Design Verification Report

7.3.4 Requirements

7.3.4.1 A Software Architecture Specification shall be written, under the responsibility of the Designer, on the basis of the Software Requirements Specification.

Requirements from 7.3.4.2 to 7.3.4.14 refer to the Software Architecture Specification.

7.3.4.2 The proposed software architecture shall be established and detailed in the Software Architecture Specification.

7.3.4.3 The Software Architecture Specification shall consider the feasibility of achieving the Software Requirements Specification at the required software safety integrity level.

NOTE The Software Architecture aims at minimising the size and complexity of the safety part of the application.

7.3.4.4 The Software Architecture Specification shall identify, analyse and detail the significance of all hardware/software interactions.

7.3.4.5 The Software Architecture Specification shall identify all software components and for these components identify

a) whether these components are new or existing,

b) whether these components have been previously validated and if so their validation conditions,

c) the software safety integrity level of the component.

7.3.4.6 Software components shall

a) cover a defined subset of software requirements,

b) be clearly identified and independently versioned inside the configuration management system.

7.3.4.7 The use of pre-existing software shall be subject to the following restrictions.

a) For all software safety integrity levels the following information shall clearly be identified and documented:
   - the requirements that the pre-existing software is intended to fulfil;
   - the assumptions about the environment of the pre-existing software;
   - interfaces with other parts of the software.

b) For all software safety integrity levels the pre-existing software shall be included in the validation process of the whole software.

c) For software safety integrity levels SIL 3 or SIL 4, the following precautions shall be taken:
   - an analysis of possible failures of the pre-existing software and their consequences on the whole software shall be carried out;
   - a strategy shall be defined to detect failures of the pre-existing software and to protect the system from these failures;
   - the verification and validation process shall ensure
     1) that the pre-existing software fulfils the allocated requirements,
     2) that failures of the pre-existing software are detected and the system where the pre-existing software is integrated into is protected from these failures,
     3) that the assumptions about the environment of the pre-existing software are fulfilled.

d) The pre-existing software shall be accompanied by a sufficiently precise (e.g. limited to the used functions) and complete description (i.e. functions, constraints and evidence). The description shall include hardware and/or software constraints of which the integrator shall be aware and take into consideration during application. In particular it forms the vehicle for informing the integrator of what the software was designed for, its properties, behaviour and characteristics.

NOTE Statistical evidence could be used in the validation strategy of the pre-existing software.

7.3.4.8 The use of existing verified software components developed according to this Standard in the design is to be preferred wherever possible.

7.3.4.9 Where the software consists of components of different software safety integrity levels then all of the software components shall be treated as belonging to the highest of these levels unless there is evidence of independence between the higher software safety integrity level components and the lower software safety integrity level components. This evidence shall be recorded in the Software Architecture Specification.

7.3.4.10 The Software Architecture Specification shall describe the strategy for the software development to the extent required by the software safety integrity level. The Software Architecture Specification shall be expressed and structured in such a way that it is

a) complete, consistent, clear, precise, unequivocal, verifiable, testable, maintainable and feasible,

b) traceable back to the Software Requirements Specification.

7.3.4.11 Measures for handling faults shall be included in the Software Architecture Specification in order to achieve the balance between the fault avoidance and fault handling strategies.

7.3.4.12 The Software Architecture Specification shall justify that the techniques, measures and tools chosen form a set which satisfies the Software Requirements Specification at the required software safety integrity level.

7.3.4.13 The Software Architecture Specification shall take into account the requirements of 8.4.8 when the software is configured by applications data or algorithms.

7.3.4.14 The Software Architecture Specification shall choose techniques and measures from Table A.3. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.3.4.15 The size and complexity of the developed software architecture shall be balanced.

7.3.4.16 Prototyping may be used in any phase to elicit requirements or to obtain a more detailed view on requirements and their consequences.

7.3.4.17 Code from a prototype may be used in the target system only if it is demonstrated that the code and its development and documentation fulfils this Standard.

7.3.4.18 A Software Interface Specification for all Interfaces between the components of the software and the boundary of the overall software shall be written, under the responsibility of the Designer, on the basis of the Software Requirements Specification and the Software Architecture Specification.

The requirement in 7.3.4.19 refers to the Software Interface Specification.

7.3.4.19 The description of interfaces shall address

a) pre/post conditions,

b) definition and description of all boundary values for all specified data,

c) behaviour when the boundary value is exceeded,

d) behaviour when the value is at the boundary,

e) for time-critical input and output data:

   1) time constraints and requirements for correct operation,

   2) management of exceptions,

f) allocated memory for the interface buffers and the mechanisms to detect that the memory cannot be allocated or all buffers are full, where applicable,

g) existence of synchronization mechanisms between functions (see e).

All data from and to the interfaces shall be defined for the whole range of values defined by the type of the data, including the ranges which are not used when processed by the functions:

h) definition and description of all equivalence classes for all specified data and each software function using them,

i) definition of unused or forbidden equivalence classes.

NOTE The data type includes the following:

a) input parameters and output results of functions and/or procedures;

b) data specified in telegrams or communication packets;

c) data from the hardware.

7.3.4.20 A Software Design Specification shall be written, under the responsibility of the Designer, on the basis of the Software Requirements Specification, the Software Architecture Specification and the Software Interface Specification.

Requirements from 7.3.4.21 to 7.3.4.24 refer to the Software Design Specification.

7.3.4.21 The input documents shall be available, although not necessarily finalised, prior to the start of the design process.

7.3.4.22 The Software Design Specification shall describe the software design based on a decomposition into components with each component having a Software Component Design Specification and a Software Component Test Specification.

7.3.4.23 The Software Design Specification shall address

a) software components traced back to software architecture and their safety integrity level,

b) interfaces of software components with the environment,

c) interfaces between the software components,

d) data structures,

e) allocation and tracing of requirements on components,

f) main algorithms and sequencing,

g) error reporting mechanisms.

7.3.4.24 The Software Design Specification shall choose techniques and measures from Table A.4. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.3.4.25 Coding standards shall be developed and specify

a) good programming practice, as defined by Table A.12,

b) measures to avoid or detect errors which can be made during application of the language and are not detectable during the verification (see 7.5 and 7.6). Such failures are derived by analysis over all features of the language,

c) procedures for source code documentation.

7.3.4.26 The selection of a coding standard shall be justified to the extent required by the software safety integrity level.

7.3.4.27 The coding standards shall be used for the development of all software and be referenced in the Software Quality Assurance Plan.

7.3.4.28 In accordance with the required software safety integrity level the design method chosen shall possess features that facilitate

a) abstraction, modularity and other features which control complexity,

b) the clear and precise expression of

1) functionality,

2) information flow between components,

3) sequencing and time related information,

4) concurrency,

5) data structure and properties,

c) human comprehension,

d) verification and validation,

e) software maintenance.

7.3.4.29 A Software Integration Test Specification shall be written, under the responsibility of the Integrator, on the basis of the Software Requirements Specification, the Software Architecture Specification, the Software Design Specification and the Software Interface Specifications.

The requirements from 7.3.4.30 to 7.3.4.32 refer to the Software Integration Test Specification.

7.3.4.30 The Software Integration Test Specification shall be written in accordance with the generic requirements established for a Test Specification (see 6.1.4.4).

7.3.4.31 The Software Integration Test Specification shall address the following:

a) it shall be shown that each software component provides the specified interfaces for the other components by executing the components together;

b) it shall be shown that the software behaves in an appropriate manner when the interface is subjected to inputs which are out of specification;

c) the required input data with their sequences and their values shall be the base of the test cases;

d) the anticipated output data with their sequences and their values shall be the basis of the test cases;

e) it shall be shown which results of the component test (see 7.5.4.5 and 7.5.4.7) are intended to be reused for the software integration test.

7.3.4.32 The Software Integration Test Specification shall choose techniques and measures from Table A.5. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.3.4.33 A Software/Hardware Integration Test Specification shall be written, under the responsibility of the integrator, on the basis of the System Design Description, the Software Requirements Specification, the Software Architecture Specification and the Software Design Specification.

The requirements from 7.3.4.34 to 7.3.4.39 refer to the Software/Hardware Integration Test Specification.

7.3.4.34 A Software/Hardware Integration Test Specification should be created early in the development life cycle, in order that integration testing may be properly directed and that particular design or other integration needs may be suitably provided for. Depending upon the size of the system, the Software/Hardware Integration Test Specification may be subdivided during development into a number of child documents and be naturally added to, as the hardware and software designs evolve and the detailed needs of integration become clearer.

7.3.4.35 The Software/Hardware Integration Test Specification shall distinguish between those activities which can be carried out by the supplier on his premises and those that require access to the user's site.

7.3.4.36 The Software/Hardware Integration Test Specification shall address the following:

a) it shall be shown that the software runs in a proper way on the hardware using the hardware via the specified hardware interfaces;

b) it shall be shown that the software can handle hardware faults as required;

c) the required timing and performance shall be demonstrated;

d) the required input data with their sequences and their values shall be the basis of the test cases;

e) the anticipated output data with their sequences and their values shall be the basis of the test cases;

f) it shall be shown which results of the component test (see 7.5.4.5) and of the software integration test (see 7.6.4.3) are intended to be reused for the software/hardware integration test.

7.3.4.37 The Software/Hardware Integration Test Specification shall document the following:

a) test cases and test data;

b) types of tests to be performed;

c) test environment including tools, support software and configuration description;

d) test criteria on which the completion of the test will be judged.

7.3.4.38 The Software/Hardware Integration Test Specification shall be written in accordance with the generic requirements established for a Test Specification (see 6.1.4.4).

7.3.4.39 The Software/Hardware Integration Test Specification shall choose techniques and measures from Table A.5. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.3.4.40 A Software Architecture and Design Verification Report shall be written, under the responsibility of the Verifier, on the basis of the Software Requirements Specification, Software Architecture Specification, Software Design Specification, Software Integration Test Specification and Software/Hardware Integration Test Specification.

The requirements from 7.3.4.41 to 7.3.4.43 refer to the Software Architecture and Design Verification Report.

7.3.4.41 The Software Architecture and Design Verification Report shall be written in accordance with the generic requirements established for a Verification Report (see 6.2.4.14).

7.3.4.42 After the Software Architecture, Interface and Design Specifications have been established, verification shall address

a) the internal consistency of the Software Architecture, Interface and Design Specifications,

b) the adequacy of the Software Architecture, Interface and Design Specifications in fulfilling the Software Requirements Specification with respect to consistency and completeness,

c) that the Software Architecture Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.16 as well as the specific requirements in 7.3.4.1 to 7.3.4.14,

d) that the Software Interface Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.16 as well as the specific requirements in 7.3.4.18 to 7.3.4.19,

e) that the Software Design Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.16 as well as the specific requirements in 7.3.4.20 to 7.3.4.24,

f) the adequacy of the Software Architecture Specification and the Software Design Specification in taking into account the hardware and software constraints.

The results shall be recorded in a Software Architecture and Design Verification Report.

7.3.4.43 After the Software Integration and Software/Hardware Integration Test Specifications have been established, verification shall address

a) that the Software Integration Test Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.16, as well as the specific requirements in 7.3.4.29 to 7.3.4.32,

b) that the Software/Hardware Integration Test Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.16, as well as the specific requirements in 7.3.4.33 to 7.3.4.39.

The results shall be recorded in a Software Architecture and Design Verification Report.

7.4 Component design

7.4.1 Objectives

7.4.1.1 To develop a software component design that achieves the requirements of the Software Design Specification to the extent required by the software safety integrity level.

7.4.1.2 To develop a software component test specification that achieves the requirements of the Software Component Design Specification to the extent required by the software safety integrity level.

7.4.2 Input documents

Software Design Specification

7.4.3 Output documents

a) Software Component Design Specification

b) Software Component Test Specification

c) Software Component Design Verification Report

7.4.4 Requirements

7.4.4.1 For each component, a Software Component Design Specification shall be written, under the responsibility of the Designer, on the basis of the Software Design Specification.

Requirements from 7.4.4.2 to 7.4.4.6 refer to the Software Component Design Specification.

7.4.4.2 For each software component, the following information shall be available

– author,

– configuration history, and

– short description.

The configuration history shall include a precise identification of the current and all previous versions of the component, specifying the version, date and author, and a description of the changes made from the previous version.

7.4.4.3 The Software Component Design Specification shall address

a) identification of all lowest software units (e.g. subroutines, methods, procedures) traced back to the upper level,

b) their detailed interfaces with the environment and other components with detailed inputs and outputs,

c) their safety integrity level without any further apportionment within the component itself,

d) detailed algorithms and data structures.

Each Software Component Design Specification shall be self consistent and allow transforming into code of the corresponding components.

7.4.4.4 Each Software Component Design Specification shall be readable, understandable and testable.

7.4.4.5 The size and complexity of each developed Software Component shall be balanced.

7.4.4.6 The Software Component Design Specification shall choose techniques and measures from Table A.4. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.4.4.7 For each component, a Software Component Test Specification shall be written, under the responsibility of the Tester, on the basis of the Software Component Design Specification.

The requirements from 7.4.4.8 to 7.4.4.10 refer to the Software Component Test Specification.

7.4.4.8 The Software Component Test Specification shall be written in accordance with the generic requirements established for a Test Specification (see 6.1.4.4).

7.4.4.9 A Software Component Test Specification shall be produced against which the component shall be tested. These tests shall show that each component performs its intended function. The Software Component Test Specification shall define and justify the required criteria and degree of test coverage to the extent required by the software safety integrity level. Tests shall be designed so as to fulfil three objectives:

a) to confirm that the component performs its intended functions (black box testing);

b) to check how the internal parts of the component interact to carry out the intended functions (black/white box testing);

c) to confirm that all parts of the component are tested (white box testing).

7.4.4.10 The Software Component Test Specification shall choose techniques and measures from Table A.5. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

7.4.4.11 A Software Component Design Verification Report shall be written, under the responsibility of the Verifier, on the basis of the Software Design Specification, Software Component Design Specification and Software Component Test Specification.

Requirements from 7.4.4.12 to 7.4.4.13 refer to the Software Component Design Verification Report.

7.4.4.12 The Software Component Design Verification Report shall be written in accordance with the generic requirements established for a Verification Report (see 6.2.4.14).

7.4.4.13 After each Software Component Design Specification has been established, verification shall address

a) the adequacy of the Software Component Design Specification in fulfilling the Software Design Specification,

b) that the Software Component Design Specification meets general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17, as well as the specific requirements in 7.4.4.1 to 7.4.4.6,

c) the adequacy of the Software Component Test Specification as a set of test cases for the Software Component Design Specification,

d) that the Software Component Test Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17, as well as the specific requirements in 7.4.4.7 to 7.4.4.10,

e) the decomposition of the Software Design Specification into software components and the Software Component Design Specification with reference to

   1) feasibility of the performance required,

   2) testability for further verification, and

   3) maintainability to permit further evolution.

The results shall be recorded in a Software Component Design Verification Report.

7.5 Component implementation and testing

7.5.1 Objectives

To achieve software which is analysable, testable, verifiable and maintainable. Component testing is also included in this phase.

7.5.2 Input documents

a) Software Component Design Specification

b) Software Component Test Specification

7.5.3 Output documents

a) Software Source Code and supporting documentation

b) Software Component Test Report

c) Software Source Code Verification Report

7.5.4 Requirements

7.5.4.1 The Software Source Code shall be written under the responsibility of the Implementer on the basis of the Software Component Design Specification. Requirements from 7.5.4.2 to 7.5.4.4 refer to the software source code.

7.5.4.2 The size and complexity of the developed source code shall be balanced.

7.5.4.3 The Software Source Code shall be readable, understandable and testable.

7.5.4.4 The Software Source Code shall be placed under configuration control before the commencement of documented testing.

7.5.4.5 A Software Component Test Report shall be written, under the responsibility of the Tester, on the basis of the Software Component Test Specification and the Software Source Code.

Requirements from 7.5.4.6 to 7.5.4.7 refer to the Software Component Test Report.

7.5.4.6 The Software Component Test Report shall be written in accordance with the generic requirements established for a Test Report (see 6.1.4.5).

7.5.4.7 The Software Component Test Report shall include the following features.

a) A statement of the test results and whether each component has met the requirements of its Software Component Design Specification.

b) A statement of test coverage shall be provided for each component, showing that the required degree of test coverage has been achieved for all required criteria. Any unreachable executable code shall be justified in accordance with the allocated SIL, see Table A.21.

7.5.4.8 A Software Source Code Verification Report shall be written, under the responsibility of the verifier, on the basis of the Software Component Design Specification, the Software Component Test Specification and the Software Source Code.

Requirements from 7.5.4.9 to 7.5.4.10 refer to the Software Source Code Verification Report.

7.5.4.9 The Software Source Code Verification Report shall be written in accordance with the generic requirements established for a Verification Report (see 6.2.4.14).

7.5.4.10 After the Software Source Code and the Software Component Test Report have been established, verification shall address

a) the adequacy of the Software Source Code as an implementation of the Software Component Design Specification,

b) the correct use of the chosen techniques and measures from Table A.4 as a set satisfying 4.8 and 4.9,

c) determining the correct application of the coding standards,

d) that the Software Source Code meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17, as well as the specific requirements in 7.5.4.1 to 7.5.4.4,

e) the adequacy of the Software Component Test Report as a record of the tests carried out in accordance with the Software Component Test Specification.

The results shall be recorded in a Software Source Code Verification Report.

7.6 Integration

7.6.1 Objectives

7.6.1.1 To carry out software and software/hardware integration.

7.6.1.2 To demonstrate that the software and the hardware interact correctly to perform their intended functions.

7.6.2 Input documents

a) Software/Hardware Integration Test Specification

b) Software Integration Test Specification

7.6.3 Output documents

a) Software Integration Test Report

b) Software/Hardware Integration Test Report

c) Software Integration Verification Report

7.6.4 Requirements

7.6.4.1 The integration of software components shall be the process of progressively combining individual and previously tested components into a composite whole in order that the components interfaces and the assembled software may be adequately proven prior to system integration and system test.

7.6.4.2 During software/hardware integration any modification or change to the integrated system shall be subject to an impact study which shall identify all components impacted and the necessary re-verification activities.

7.6.4.3 A Software Integration Test Report shall be written, under the responsibility of the Integrator, on the basis of the Software Integration Test Specification.

Requirements from 7.6.4.4 to 7.6.4.6 refer to the Software Integration Test Report.

7.6.4.4 The Software Integration Test Report shall be written in accordance with the generic requirements established for a Test Report (see 6.1.4.5).

7.6.4.5 A Software Integration Test Report shall be produced as follows:

a) a Software Integration Test Report shall be produced stating the test results and whether the objectives and criteria of the Software Integration Test Specification have been met. If there is a failure, the circumstances for the failure shall be recorded;

b) test cases and their results shall be recorded, preferably in machine readable form for subsequent analysis;

c) tests shall be repeatable and, if practicable, be performed by automatic means;

d) the Software Integration Test Report shall document the identity and configuration of all the items involved.

7.6.4.6 The Software Integration Test Report shall demonstrate the correct use of the chosen techniques and measures from Table A.6 as a set satisfying 4.8 and 4.9.

7.6.4.7 A Software/Hardware Integration Test Report shall be written, under the responsibility of the integrator, on the basis of the Software/Hardware Integration Test Specification.

Requirements from 7.6.4.8 to 7.6.4.10 refer to the Software/Hardware Integration Test Report.

7.6.4.8 The Software/Hardware Integration Test Report shall be written in accordance with the generic requirements established for a Test Report (see 6.1.4.5).

7.6.4.9 A Software/Hardware Integration Test Report shall be produced as follows:

a) the Software /Hardware Integration Test Report shall state the test results and whether the objectives and criteria of the Software/Hardware Integration Test Specification have been met. If there is a failure, the circumstances of the failure shall be recorded;

b) test cases and their results shall be recorded, preferably in a machine-readable form for subsequent analysis;

c) the Software/Hardware Integration Test Report shall document the identity and configuration of all items involved.

7.6.4.10 The Software/Hardware Integration Test Report shall demonstrate the correct use of the chosen techniques and measures from Table A.6 as a set satisfying 4.8 and 4.9.

7.6.4.11 A Software Integration Verification Report shall be written, under the responsibility of the Verifier, on the basis of the Software and Software/Hardware Integration Test Specifications and the corresponding test reports.

Requirements from 7.6.4.12 to 7.6.4.13 refer to the Software Integration Verification Report.

7.6.4.12 The Software Integration Verification Report shall be written in accordance with the generic requirements established for a Verification Report (see 6.2.4.14).

7.6.4.13 After the Software Integration Test Report and the Software/Hardware Integration Test Report have been established, verification shall address

a) the adequacy of the Software Integration Test Report as a record of the tests carried out in accordance with the Software Integration Test Specification,

b) whether the Software Integration Test Report meets the requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17, as well as the specific requirements in 7.6.4.3 to 7.6.4.6,

c) the adequacy of the Software/Hardware Integration Test Report as a record of the tests carried out in accordance with the Software/Hardware Integration Test Specification,

d) whether the Software/Hardware Integration Test Report meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17, as well as the specific requirements in 7.6.4.7 to 7.6.4.10.

7.7 Overall Software Testing / Final Validation

7.7.1 Objectives

To analyse and test the integrated software and hardware to ensure compliance with the Software Requirements Specification with particular emphasis on the functional and safety aspects according to the software safety integrity level and to check whether it is fit for its intended application.

7.7.2 Input documents

a) Software Requirements Specification

b) Overall Software Test Specification

c) Software Verification Plan

d) Software Validation Plan

e) All Hardware and Software Documentation including intermediate verification results

f) System Safety Requirements Specification

7.7.3 Output documents

a) Overall Software Test Report

b) Overall Software Test Verification Report

c) Software Validation Report

d) Release Note

7.7.4 Requirements

7.7.4.1 An Overall Software Test Report shall be written, under the responsibility of the Tester, on the basis of the Overall Software Test Specification.

Requirements from 7.7.4.2 to 7.7.4.4 refer to the Overall Software Test Report.

7.7.4.2 The Overall Software Test Report shall be written in accordance with the generic requirements established for a Test Report (see 6.1.4.5).

7.7.4.3 The Validator shall specify and perform supplementary tests at his discretion or have them performed by the Tester. While the Overall Software Tests are mainly based on the structure of the Software Requirements Specification, the added value the Validator shall contribute, are tests which stress the system by complex scenarios reflecting the actual needs of the user.

7.7.4.4 The results of all tests and analyses shall be recorded in a Overall Software Test Report.

7.7.4.5 An Overall Software Test Verification Report, which shall be written under the responsibility of the Verifier.

7.7.4.6 The software shall be exercised either by connection to real items of hardware or actual systems with which it would interface in operation, or by simulation of input signals and loads driven by outputs. It shall be exercised under conditions present during normal operation, anticipated occurrences and undesired conditions requiring system action. Where simulated inputs or loads are used it shall be shown that these do not differ significantly from the inputs and loads encountered in operation.

NOTE Simulated inputs or loads could replace inputs or loads which are only present at system level or in faulty modes.

7.7.4.7 A Software Validation Report shall be written, under the responsibility of the Validator, on the basis of the Software Validation Plan.

Requirements from 7.7.4.8 to 7.7.4.12 refer to the Software Validation Report.

7.7.4.8 The Software Validation Report shall be written in accordance with the generic requirements established for the Validation Report (see 6.3.4.7 to 6.3.4.11).

7.7.4.9 Once integration is finished and overall software testing and analysis are complete, a Software Validation Report shall be produced as follows:

a) it shall state whether the objectives and criteria of the Software Validation Plan have been met. Deviations to the plan shall be recorded and justified;

b) it shall give a summary statement on the tests results and whether the whole software on its target machine fulfils the requirements set out in the Software Requirements Specification;

c) an evaluation of the test coverage on the requirements of the Software Requirements Specification shall be provided;

d) an evaluation of other verification activities in accordance to the Software Verification Plan and Report shall be done together with a check that requirements tracing is fully performed and covered;

e) if the Validator produces own test cases not given to the Tester the Software Validation Report shall document these in accordance with 6.3.4.7 to 6.3.4.11.

7.7.4.10 The Software Validation Report shall contain the confirmation that each combination of techniques or measures selected according to Annex A is appropriate to the defined software safety integrity level. It shall contain an evaluation of the overall effectiveness of the combination of techniques and measures adopted, taking account of the size and complexity of the software produced and taking into account the actual results of testing, verification and validation activities.

7.7.4.11 The following shall be addressed in the Software Validation Report:

a) documentation of the identity and configuration of the software;

b) statement of appropriate identification of technical support software and equipment;

c) statement of appropriate identification of simulation models used;

d) statement about the adequacy of the Overall Software Test Specification;

e) collection and keeping track of any deviations found;

f) review and evaluation of all deviations in terms of risk (impact);

g) a statement that the project has performed appropriate handling of corrective actions in accordance with the change management process and procedures and with a clear identification of any discrepancies found;

h) statement of each restriction given by the deviations in a traceable way;

i) a conclusion whether the software is fit for its intended application, taking into account the application conditions and constraints.

7.7.4.12 Any discrepancies found including detected errors and non-compliances with this Standard or with any of the software requirements or plans, as well as constraints and limitations, shall be clearly identified in a separate subclause of the Software Validation Report, evaluated regarding the safety integrity level and included in any Release Note which accompanies the delivered software.

7.7.4.13 A Release Note which accompanies the delivered software shall include all restrictions in using the software. These restrictions are derived from

a) the detected errors,

b) non-compliances with this Standard,

c) degree of fulfilment of the requirements,

d) degree of fulfilment of any plan.

8 Development of application data or algorithms: systems configured by application data or algorithms

8.1 Objectives

8.1.1 A characteristic feature in many railway systems is the need to design each installation to meet the individual requirements for a specific application. A system configured by application data and/or by application algorithms allows approved generic software to be customised with the individual requirements for each specific application.

The objective for the development of application data is the correct deriving of the data from the given installation and the check of the intended behaviour, followed by an assessment of the used development process for that application data.

The requirements for the development of application algorithms are the same as the development of generic software as described in Clauses 1 to 7 and 9.

A typical example is a system whose generic software is pre-configured for a generic railway application by a set of application algorithms, and which is then further configured to each specific installation by instantiation and interconnection of the application algorithms and by a set of configuration data. For instance, the signalling principles of an interlocking system (e.g. signal management, point management) may be implemented by a set of application algorithms.

Application data typically take the form of parameter values or descriptions (identity, type, location, etc.) of external objects. Application algorithms may take the form of e.g. function block diagrams, state charts and relay ladder diagrams, which determine the desired response of the system according to its inputs, its current state and specific parameter values. Application algorithms include logical connections and operations to be executed.

The application data/algorithms are usually produced using dedicated tools. They may be expressed in tabular or diagrammatic formats, which can be interpreted or compiled into executable codes often after conversion into source codes handled via specialised languages (with syntax and semantics).

The customisation of systems through configurability gives the designer different degrees of control over the detailed software functionality.

8.1.2 The procedures and the tools used for their development shall be appropriate to the system safety integrity level as determined by the function for which they are developed.

8.1.3 The subclauses of 8.4 describe the requirements for the initial development of a configurable system and for the subsequent development of each set of application-specific data/algorithms.

8.2 Input documents

a) Software Requirements Specification of generic software

b) Software Architecture Specification of generic software

c) Application conditions of the generic software and application tools

d) User manuals of the generic software and application tools

8.3 Output documents

a) Application Preparation Plan

b) Application Requirements Specification

c) Application Architecture and Design

d) Application Test Specification

e) Application Test Report

f) Application Preparation Verification Report

g) Source Code of Application Data/Algorithms

h) Application Data/Algorithms Verification Report

8.4 Requirements

8.4.1 Application Development Process

8.4.1.1 An Application Preparation Plan shall be written, under the responsibility of the Requirements Manager or Designer, on the basis of the input documents from 8.2.

The requirements from 8.4.1.2 to 8.4.1.11 refer to the Application Preparation Plan.

8.4.1.2 An Application Preparation Plan shall be produced in order to define and detail the application development process, including all the activities, deliverables and roles in charge of them. It can be produced either for each specific application or for a class of specific applications, i.e. for a generic application.

8.4.1.3 The Application Preparation Plan shall define a documentation structure for the application preparation process.

8.4.1.4 The Application Preparation Plan shall choose techniques and measures from Table A.11. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

8.4.1.5 The Application Preparation Plan shall specify the procedures and application tools (with their classification based on 6.7) to be used in the application development process.

8.4.1.6 The Application Preparation Plan shall include verification and validation activities to ensure that the application data/algorithms are complete, correct and compatible with each other and with the generic application, and to provide evidence that the application conditions of the generic application are met. These verification and validation activities and evidence can be replaced by verification and validation performed on the tools that produce the application data/algorithms. The results are gathered together in the Application Preparation Verification Report and the Application Test Report.

8.4.1.7 The Application Preparation Plan shall include verification and validation activities to ensure that the application tools and the generic software are compatible with each other and with the specific application, and to provide evidence that their application conditions are met.

8.4.1.8 A risk analysis shall be carried out covering the application development process, including the application tools and procedures, in order to validate the Application Preparation Plan and to meet the required software safety integrity level. The Application Preparation Plan shall include the risk analysis.

8.4.1.9 The Application Preparation Plan shall specify the requirements for the independence between staff carrying out verification, validation and preparation tasks according to 5.1.

NOTE Data preparation activities are carried out by application designers.

8.4.1.10 The Application Preparation Plan shall define a tool class for any hardware or software tools used in the application preparation lifecycle.

8.4.1.11 Where possible, the Application Preparation Plan shall call for notations for specifying requirements and design which are familiar to applications engineers. Where new notations are introduced, the necessary user documentation shall be provided, as well as training where appropriate.

8.4.1.12 An Application Data/Algorithms Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 8.2.

The requirement in 8.4.1.13 refers to the Application Data/Algorithms Verification Report.

8.4.1.13 Once the Application Preparation Plan has been established, verification shall address

a) that the Application Preparation Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 8.4.1.2 to 8.4.1.11,

b) the internal consistency of the Application Preparation Plan.

The results shall be recorded in an Application Data/Algorithms Verification Report.

8.4.1.14 The implementation of the Application Preparation Plan shall be verified and validated for each specific application.

8.4.2 Application Requirements Specification

8.4.2.1 An Application Requirements Specification shall be written, under the responsibility of the Requirements Manager, on the basis of the input documents from 8.2.

The requirements from 8.4.2.2 to 8.4.2.3 refer to the Application Requirements Specification.

8.4.2.2 The requirements for the specific application shall include the requirements which are specific to the installation under consideration (e.g. track layout, signal locations, speed limits for a signalling system), as well as a recap or reference to the application conditions of the generic software and the application tools, and the standards with which the application shall comply (e.g. signalling principles for a signalling system).

8.4.2.3 The requirements related to the application data and algorithms processed by the generic software of the system shall be specified at this stage.

8.4.2.4 An Application Data/Algorithms Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 8.2.

The requirement in 8.4.2.5 refers to the Application Data/Algorithms Verification Report.

8.4.2.5 Once the Application Requirements Specification has been established, verification shall address

a) that the Application Requirements Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 8.4.2.2 to 8.4.2.3,
b) the internal consistency of the Application Requirements Specification.

The results shall be recorded in an Application Data/Algorithms Verification Report.

8.4.3 Architecture and Design

The quantity and type of the generic hardware and software components to be used in the specific application shall be specified. The location of components, application data and algorithms in the specific application architecture shall be defined. The application data and algorithms processed by the generic software shall be designed at this stage.

8.4.4 Application Data/Algorithms Production

8.4.4.1 The application development process shall include the production and compilation of the source code of the generic and specific data/algorithms, as well as verification and testing activities related to this production. The use of diagrammatic languages is recommended for producing the source code of application algorithms. Refer to the Table A 16.

8.4.4.2 An Application Test Report shall be written, under the responsibility of the Tester, on the basis of the input documents from 8.2.

The requirement in 8.4.4.3 refers to the Application Test Report.

8.4.4.3 The Application Test Report shall document the correct and complete execution of the tests defined in Application Test Specification.

8.4.4.4 The Application Preparation Verification Report shall

a) document every activity performed to ensure correctness and completeness of data/algorithm and their coherency with application principles and specific application architecture,

b) evaluate compatibility of data/algorithms with generic application.

8.4.4.5 An Application Test Specification shall be written, under the responsibility of the Tester, on the basis of the input documents from 8.2.

The requirement in 8.4.4.6 refers to the Application Test Specification.

8.4.4.6 The Application Test Specification shall specify tests to be carried out at intermediate or final stage of data/algorithms preparation, in order to ensure

a) coherency and completeness of data/algorithms with respect to application principles,

b) coherency and completeness of data/algorithms with respect to specific application architecture.

8.4.4.7 An Application Data/Algorithms Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 8.2.

The requirement in 8.4.4.8 refers to the Application Data/Algorithms Verification Report.

8.4.4.8 Once the Application Test Specification has been established, verification shall address

a) that the Application Test Specification meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 8.4.4.6,

b) the internal consistency of the Application Test Specification.

The results shall be recorded in an Application Data/Algorithms Verification Report.

8.4.5 Application Integration and Testing Acceptance

8.4.5.1 For some systems the application data/algorithms can be integrated with the generic hardware and software for a factory test before installation on the target system. This may not be necessary where a sufficient degree of confidence can be obtained by other means. The application shall then be installed on the target system, and integration tests within the complete installation shall be carried out. Finally the target system shall be commissioned as a fully operational system, and a final acceptance process of the target system in the complete installation shall be carried out. The Application Test Report shall document the correct and complete execution of tests defined in the Application Test Specification. The Application Preparation Verification Report shall check the completeness and correctness of tests performed on the complete installation.

8.4.5.2 An Application Test Specification shall be written, under the responsibility of the Tester, on the basis of the input documents from 8.2.

The requirement in 8.4.5.3 refers to the Application Test Specification.

8.4.5.3 The Application Test Specification shall specify tests to be carried out to ensure

a) correct integration of data/algorithms on generic hardware and software, if needed,

b) correct integration of data/algorithms with complete installation.

8.4.5.4 An Application Data/Algorithms Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 8.2.

The requirement in 8.4.5.5 refers to the Application Data/Algorithms Verification Report.

8.4.5.5 Once the Application Test Specification has been established, verification shall address that the Application Test Specification meets the specific requirements in 8.4.5.3.

8.4.6 Application Validation and Assessment

Validation and assessment activities shall audit the performance of each stage of the life-cycle.

8.4.7 Application preparation procedures and tools

8.4.7.1 For each new type of system configured by application data/algorithms, specific procedures and tools shall be developed to allow the application development process specified in 8.4.1 to be applied to installations of the new system. Development of these tools shall be carried out in accordance with this Standard in parallel with the generic software and hardware for the system. The verification, validation and assessment activities shall ensure that the data preparation tools and the generic software are compatible.

8.4.7.2 Any compilation process shall be validated and assessed. It shall be noted that specialised compilers are usually necessary for the data and algorithm conversion.

8.4.7.3 All application data/algorithms and associated documentation for each specific application shall be subject to the software deployment requirements as specified in 9.1.

8.4.7.4 All application data/algorithms and associated documentation shall be subject to the software maintenance requirements specified in 9.2.

8.4.7.5 All application data/algorithms and associated documentation shall be placed under configuration management according to the requirements specified in 6.5 and 6.7. The configuration management of application data/algorithms can be separate from the generic software part.

8.4.7.6 The Application Verification Report demonstrates the coverage and enforcement of the application conditions of the generic software and application tools.

8.4.8 Development of Generic Software

8.4.8.1 Development of the generic software, which supports the execution of application data/algorithms, shall comply with the requirements in 7.1 to 7.7 of this Standard. The following additional requirements shall also be observed.

8.4.8.2 The types or classes of function which can be configured by application data/algorithms in each system and subsystem shall be identified in the Software Requirements Specification documents of the generic software. The safety integrity level allocated to functions will determine the standards to be applied to the subsequent development of the application data/ algorithms for all installations of the system.

8.4.8.3 During the design of the generic software the detailed interfaces between the generic software and the application data/algorithms shall be specified, unless this has already been specified at an earlier phase of the lifecycle, for example as a result of a requirement to use an existing application-specific language.

8.4.8.4 A rigid separation between the generic software and the application data/algorithms shall be enforced, i.e. it shall be possible to recompile and update either the generic software or the application data/algorithms without needing to update the other, unless there has been a change to the defined interface between the generic software and the application data/algorithms. Likewise, the applications specific data/algorithms shall be separated from the application-generic data/algorithms.

8.4.8.5 The change control procedures shall ensure that any amendment to the generic software may only be installed after it has been established that either the revised software is compatible with the original application data/algorithms or the application data/algorithms have been revised.

8.4.8.6 Care shall be taken in the verification process and validation test phase of the generic software in order to ensure that all relevant combinations of data and algorithms are considered.

If all relevant combinations of data and algorithms have not been considered in the verification, testing and validation process of the generic software, it shall be clearly identified as a limit of use of the generic software. A complement to verification, testing and validation process of the generic software shall be performed when some data or algorithms are defined beyond this limit.

8.4.8.7 The generic software shall be designed to detect corrupted application data/algorithms where this is feasible.

8.4.8.8 The designers shall publish the Release Note of the generic software and application tools by the Overall Software Testing/Final Validation phase of the generic software and application tools. The contents of these documents shall be subject to verification and validation activities.

The following topics shall be addressed in the document “Application conditions of the generic software and application tools”:

a) references to the user manuals of the generic software and application tools;

b) any constraints on the application data/algorithms e.g. imposed architecture or coding rules to meet the safety integrity levels.

9 Software deployment and maintenance

9.1 Software deployment

9.1.1 Objective

To ensure that the software performs as required, preserving the required software safety integrity level and dependability when it is deployed in the final environment of application.

9.1.2 Input documents

All design, development and analysis documents relevant to the deployment.

9.1.3 Output documents

a) Software Release and Deployment Plan

b) Software Deployment Manual

c) Release Notes

d) Deployment Records

e) Deployment Verification Report

9.1.4 Requirements

9.1.4.1 The deployment shall be carried out under the responsibility of the project manager.

9.1.4.2 Before delivering a software release, the software baseline shall be recorded and kept traceable under configuration management control. Pre-existing software and software developed according to a previous version of this Standard shall also be included.

9.1.4.3 The software release shall be reproducible throughout the baseline lifecycle.

9.1.4.4 A Release Note shall be written, under the responsibility of the Designer, on the basis of the input documents from 9.1.2.

The requirement in 9.1.4.5 refers to the Release Note.

9.1.4.5 A Release Note shall be provided that defines

a) the application conditions which shall be adhered to,

b) information of compatibility among software components and between software and hardware,

c) all restrictions in using the software (see 7.7.4.13).

9.1.4.6 A Software Deployment Manual shall be written on the basis of the input documents from 9.1.2.

The requirement in 9.1.4.7 refers to the Software Deployment Manual.

9.1.4.7 The Software Deployment Manual shall define procedures in order to correctly identify and install a software release.

9.1.4.8 In case of incremental deployment (i.e., deployment of single components), it is highly recommended for SIL 3 and SIL 4, and recommended for SIL 1 and SIL 2, that the software is designed to include facilities which ensure that activation of incompatible versions of software components is excluded.

9.1.4.9 Configuration management shall ensure that no harm results from the co-presence of different versions of the same software components where it cannot be avoided.

9.1.4.10 A rollback procedure (i.e., capability to return to the previous release) shall be available when installing a new software release.

9.1.4.11 The software shall have embedded self-identification mechanisms, allowing its identification at the loading process and after loading into the target. The self-identification mechanism should indicate version information for the software and any configuration data as well as the product identity.

NOTE The data within the code, containing the information about the software release, could be protected through coding (see Table A.3 “Error Detecting Codes”).

9.1.4.12 A Deployment Record shall be written on the basis of the input documents from 9.1.2.

The requirement in 9.1.4.13 refers to the Software Deployment Record.

9.1.4.13 A Deployment Record shall give evidence that intended software has been loaded, by inspection of the embedded self-identification mechanisms (see 9.1.4.11). This record shall be stored among the delivered system related documents like other verifications and is part of the commissioning and acceptance.

9.1.4.14 The deployed software shall be traceable to delivered installations.

NOTE This is of special importance when critical faults are discovered and need to be corrected in more than one installation.

9.1.4.15 Diagnostic information shall be provided by the software, as part of fault monitoring.

9.1.4.16 A Deployment Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 9.1.2.

Requirements from 9.1.4.17 to 9.1.4.19 refer to the Deployment Verification Report.

9.1.4.17 Once the Software Deployment Manual has been established, verification shall address

a) that the Software Deployment Manual meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.1.4.7,

b) the internal consistency of the Software Deployment Manual.

9.1.4.18 Once the Deployment Record has been established, verification shall address

a) that the Deployment Record meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.1.4.13,

b) the internal consistency of the Deployment Record.

9.1.4.19 Once the Release Note has been established, verification shall address

a) that the Release Note meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.1.4.5,

b) the internal consistency of the Release Note.

The results shall be recorded in a Deployment Verification Report.

9.1.4.20 Measures shall be included in the software package to prevent or detect errors occurring during storage, transfer, transmission or duplication of executable code or data. The executable code is recommended to be coded (see Table A.3 “Error Detecting Codes”) as part of checking the integrity of the code in the load process.

9.2 Software maintenance

9.2.1 Objective

To ensure that the software performs as required, preserving the required software safety integrity level and dependability when making corrections, enhancements or adaptations to the software itself. See also 6.6 “Modification and change control” of this Standard and phase 13 “Modification and retrofit” in IEC 62278.

9.2.2 Input documents

All relevant design, development and analysis documents.

9.2.3 Output documents

a) Software Maintenance Plan

b) Software Change Records

c) Software Maintenance Records

d) Software Maintenance Verification Report

9.2.4 Requirements

9.2.4.1 Although this Standard is not intended to be retrospective, applying primarily to new developments and only applying in its entirety to existing software if these are subjected to major modifications, 9.2 concerning software maintenance applies to all changes, even those of a minor nature. However, application of the whole of this Standard during upgrades and maintenance of existing software is highly recommended.

9.2.4.2 For any software safety integrity level, the supplier shall, before starting work on any change, decide whether the maintenance actions are to be considered as major or minor or whether the existing maintenance methods for the system are adequate. The decision shall be justified and recorded by the supplier and shall be submitted to the Assessor’s evaluation.

9.2.4.3 Maintenance shall be carried out in accordance with the guidelines contained in ISO/IEC 90003.

9.2.4.4 Maintainability shall be designed as an inherent aspect of the software, in particular by following the requirements of 7.3, 7.4 and 7.5. ISO/IEC 25010 series shall also be employed in order to implement and verify a minimum level of maintainability.

9.2.4.5 A Software Maintenance Plan shall be written on the basis of the input documents from 9.2.2.

The requirement 9.2.4.6 refers to the Software Maintenance Plan.

9.2.4.6 Procedures for the maintenance of software shall be established and recorded in the Software Maintenance Plan. These procedures shall also address

a) control of error reporting, error logs, maintenance records, change authorisation and software/system configuration and the techniques and measures in Table A.10,

b) verification, validation and assessment of any modification,

c) definition of the Authority which approves the changed software, and

d) authorisation for the modification.

9.2.4.7 A Software Maintenance Record shall be written on the basis of the input documents from 9.2.2.

The requirement in 9.2.4.8 refers to the Software Maintenance Record.

9.2.4.8 A Software Maintenance Record shall be established for each Software Item before its first release, and it shall be maintained. In addition to the requirements of ISO/IEC 90003:2014 for "Maintenance Records and Reports" (see ISO/IEC 90003:2014, “Maintenance”), this Record shall also include

a) references to all the Software Change Records for that software item,

b) change impact assessment,

c) test cases for components, including revalidation and regression testing data, and

d) software configuration history.

9.2.4.9 A Software Change Record shall be written on the basis of the input documents from 9.2.2.

The requirement in 9.2.4.10 refers to the Software Change Record.

9.2.4.10 A Software Change Record shall be established for each maintenance activity. This record shall include

a) the modification or change request, version, nature of fault, required change and source for change,

b) an analysis of the impact of the maintenance activity on the overall system, including hardware, software, human interaction and the environment and possible interactions,

c) the detailed specification of the modification or change carried out, and

d) revalidation, regression testing and re-assessment of the modification or change to the extent required by the software safety integrity level. The responsibility for revalidation can vary from project to project, according to the software safety integrity level. Also the impact of the modification or change on the process of revalidation can be confined to different system levels (only changed components, all identified affected components, the complete system). Therefore the Software Validation Plan shall address both problems, according to the software safety integrity level. The degree of independence of revalidation shall be the same as that for validation.

9.2.4.11 A Software Maintenance Verification Report shall be written, under the responsibility of the Verifier, on the basis of the input documents from 9.2.2.

Requirements from 9.2.4.12 to 9.2.4.14 refer to the Software Maintenance Verification Report.

9.2.4.12 Once the Software Maintenance Plan has been established, verification shall address

a) that the Software Maintenance Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.2.4.6,

b) the internal consistency of the Software Maintenance Plan.

9.2.4.13 Once the Software Maintenance Record has been established, verification shall address

a) that the Software Maintenance Record meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.2.4.8,

b) the internal consistency of the Software Maintenance Record.

9.2.4.14 Once the Software Change Record has been established, verification shall address

a) that the Software Change Record meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 9.2.4.10,

b) the internal consistency of the Software Change Record.

9.2.4.15 The maintenance activities shall be carried out following the Software Maintenance Plan.

9.2.4.16 The techniques and measures from Table A.10 shall be chosen. The selected combination shall be justified as a set satisfying 4.8 and 4.9.

9.2.4.17 Maintenance shall be performed with at least the same level of expertise, tools, documentation, planning and management as for the initial development of the software. This shall apply also to configuration management, change control, document control, and independence of involved parties.

9.2.4.18 External supplier control, problem reporting and corrective actions shall be managed with the same criteria specified in the relevant paragraphs of the Software Quality Assurance (6.5) as for new software development.

9.2.4.19 For each reported problem or enhancement a safety impact analysis shall be made.

9.2.4.20 For software under maintenance, mitigation actions proportionate to the identified risk shall be taken in order to ensure the overall integrity of the system whilst the reported problems are investigated and corrected.

Annex A
(normative)

Criteria for the selection of techniques and measures

A.1 General

The clauses of this Standard are associated within this annex to the clause tables (see Clause A.2, Table A.1 to Table A.11) to illustrate the means of achieving conformance. There exist lower level tables as well, the detailed tables (see Clause A.3, Table A.12 to Table A.23), which expand upon certain entries in the clause tables. For example, “Modelling” in Table A.2 is expanded upon in Table A.17. There also exists an informative Annex D which is referred to from the clause tables.

With each technique or measure in the tables there is a requirement for each software safety integrity level (SIL). In this version of the document, the requirements for software safety integrity levels 1 and 2 are the same for each technique. Similarly, each technique has the same requirements at software safety integrity levels 3 and 4. These requirements can be

'M' this symbol means that the use of a technique is mandatory,

'HR' this symbol means that the technique or measure is Highly Recommended for this safety integrity level. If this technique or measure is not used then the rationale for using alternative techniques shall be detailed in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan,

'R' this symbol means that the technique or measure is Recommended for this safety integrity level. This is a lower level of recommendation than an 'HR' and such techniques can be combined to form part of a package,

'-' this symbol means that the technique or measure has no recommendation for or against being used,

'NR' this symbol means that the technique or measure is positively Not Recommended for this safety integrity level. If this technique or measure is used then the rationale behind using it shall be detailed in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan.

The combination of techniques or measures are to be stated in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan with one or more techniques or measures being selected unless the notes attached to the table makes other requirements. These notes can include reference to approved techniques or approved combinations of techniques. If such techniques or combinations of techniques, including all respective mandatory techniques, are used, then the Assessor shall accept them as valid and shall only be concerned that they have been correctly applied. If a different set of techniques is used and can be justified, then the Assessor may find this acceptable.

A.2 Clauses tables

Table A.1 – Life cycle Issues and Documentation (5.3) *(1 of 2)*

DOCUMENTATION	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
Planning					
1. Software Quality Assurance Plan	HR	HR	HR	HR	HR
2. Software Quality Assurance Verification Report	HR	HR	HR	HR	HR
3. Software Configuration Management Plan	HR	HR	HR	HR	HR
4. Software Verification Plan	HR	HR	HR	HR	HR
5. Software Validation Plan	HR	HR	HR	HR	HR
Software requirements					
6. Software Requirements Specification	HR	HR	HR	HR	HR
7. Overall Software Test Specification	HR	HR	HR	HR	HR
8. Software Requirements Verification Report	HR	HR	HR	HR	HR
Architecture and design					
9. Software Architecture Specification	HR	HR	HR	HR	HR
10. Software Design Specification	HR	HR	HR	HR	HR
11. Software Interface Specifications	HR	HR	HR	HR	HR
12. Software Integration Test Specification	HR	HR	HR	HR	HR
13. Software/Hardware Integration Test Specification	HR	HR	HR	HR	HR
14. Software Architecture and Design Verification Report	HR	HR	HR	HR	HR
Component Design					
15. Software Component Design Specification	R	HR	HR	HR	HR
16. Software Component Test Specification	R	HR	HR	HR	HR
17. Software Component Design Verification Report	R	HR	HR	HR	HR
Component Implementation and Testing					
18. Software Source Code and supporting documentation	HR	HR	HR	HR	HR
19. Software Component Test Report	R	HR	HR	HR	HR
20. Software Source Code Verification Report	HR	HR	HR	HR	HR
Integration					
21. Software Integration Test Report	HR	HR	HR	HR	HR
22. Software/Hardware Integration Test Report	HR	HR	HR	HR	HR
23. Software Integration Verification Report	HR	HR	HR	HR	HR
Overall Software Testing / Final Validation					
24. Overall Software Test Report	HR	HR	HR	HR	HR
25. Overall Software Test Verification Report	HR	HR	HR	HR	HR
26. Software Validation Report	HR	HR	HR	HR	HR
27. Tools Validation Report	R	HR	HR	HR	HR
28. Release Note	HR	HR	HR	HR	HR

Table A.1 *(2 of 2)*

DOCUMENTATION	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
Systems configured by application data/ algorithms					
29. Application Requirements Specification	HR	HR	HR	HR	HR
30. Application Preparation Plan (see NOTE 2)	HR	HR	HR	HR	HR
31. Application Test Specification (see NOTE 2)	HR	HR	HR	HR	HR
32. Application Architecture and Design (see NOTE 2)	HR	HR	HR	HR	HR
33. Application Preparation Verification Report	HR	HR	HR	HR	HR
34. Application Test Report	HR	HR	HR	HR	HR
35. Source Code of Application Data/Algorithms	HR	HR	HR	HR	HR
36. Application Data/Algorithms Verification Report	HR	HR	HR	HR	HR
Software deployment					
37. Software Release and Deployment Plan	R	HR	HR	HR	HR
38. Software Deployment Manual	R	HR	HR	HR	HR
39. Release Notes	HR	HR	HR	HR	HR
40. Deployment Records	R	HR	HR	HR	HR
41. Deployment Verification Report	R	HR	HR	HR	HR
Software maintenance					
42. Software Maintenance Plan	R	HR	HR	HR	HR
43. Software Change Records	HR	HR	HR	HR	HR
44. Software Maintenance Records	R	HR	HR	HR	HR
45. Software Maintenance Verification Report	R	HR	HR	HR	HR
Software assessment					
46. Software Assessment Plan	R	HR	HR	HR	HR
47. Software Assessment Report	R	HR	HR	HR	HR

NOTE 1 According to 5.3.2.11 and 5.3.2.12, documents can be combined differently.

NOTE 2 Documents 29, 30 and 31 being HR or R depends on the importance defined in the process and where the verification takes place. E.g. data may only be needed to be verified but tested in the system domain while more functional properties need both test and verification. In this case HR has been marked but can be optional R.

NOTE 3 For Software Assessment Plan and Software Assessment Report see 6.4.1.2.

NOTE 4 The Software Quality Assurance Report is included in the Quality Management Report defined in IEC 62425.

Table A.2 – Software Requirements Specification (7.2)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Methods (based on a mathematical approach)	D.28	–	R	R	HR	HR
2. Modelling	Table A.17	R	R	R	HR	HR
3. Structured methodology	D.52	R	R	R	HR	HR
4. Decision Tables	D.13	R	R	R	HR	HR
Requirements: a) The Software Requirements Specification shall include a description of the problem in natural language and any necessary formal or semiformal notation. b) The table reflects additional requirements for defining the specification clearly and precisely. One or more of these techniques shall be selected to satisfy the Software Safety Integrity Level being used.						

Table A.3 – Software Architecture (7.3)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Defensive Programming	D.14	–	HR	HR	HR	HR
2. Fault Detection & Diagnosis	D.26	–	R	R	HR	HR
3. Error Correcting Codes	D.19	–	–	–	–	–
4. Error Detecting Codes	D.19	–	R	R	HR	HR
5. Failure Assertion Programming	D.24	–	R	R	HR	HR
6. Safety Bag Techniques	D.47	–	R	R	R	R
7. Diverse Programming	D.16	–	R	R	HR	HR
8. Recovery Block	D.44	–	R	R	R	R
9. Backward Recovery	D.5	–	NR	NR	NR	NR
10. Forward Recovery	D.30	–	NR	NR	NR	NR
11. Retry Fault Recovery Mechanisms	D.46	–	R	R	R	R
12. Memorising Executed Cases	D.36	–	R	R	HR	HR
13. Artificial Intelligence – Fault Correction	D.1	–	NR	NR	NR	NR
14. Dynamic Reconfiguration of software	D.17	–	NR	NR	NR	NR
15. Software Error Effect Analysis	D.25	–	R	R	HR	HR
16. Graceful Degradation	D.31	–	R	R	HR	HR
17. Information Hiding	D.33	–	–	–	–	–
18. Information Encapsulation	D.33	R	HR	HR	HR	HR
19. Fully Defined Interface	D.38	HR	HR	HR	M	M
20. Formal Methods	D.28	-	R	R	HR	HR
21. Modelling	Table A.17	R	R	R	HR	HR
22. Structured Methodology	D.52	R	HR	HR	HR	HR
23. Modelling supported by computer aided design and specification tools	Table A.17	R	R	R	HR	HR

Requirements:

a) Approved combinations of techniques for Software Safety Integrity Levels 3 and 4 are as follows:

1) 1, 7, 19, 22 and one from 4, 5, 12 or 21;

2) 1, 4, 19, 22 and one from 2, 5, 12, 15 or 21.

b) Approved combinations of techniques for Software Safety Integrity Levels 1 and 2 are as follows: 1, 19, 22 and one from 2, 4, 5, 7, 12, 15 or 21.

c) Some of these issues may be defined at the system level.

d) Error detecting codes may be used in accordance with the requirements of EN 50159.

NOTE Technique/measure 19 is for External Interfaces.

Table A.4 – Software Design and Implementation (7.4)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Methods	D.28	–	R	R	HR	HR
2. Modelling	Table A.17	R	HR	HR	HR	HR
3. Structured methodology	D.52	R	HR	HR	HR	HR
4. Modular Approach	D.38	HR	M	M	M	M
5. Components	Table A.20	HR	HR	HR	HR	HR
6. Design and Coding Standards	Table A.12	HR	HR	HR	M	M
7. Analysable Programs	D.2	HR	HR	HR	HR	HR
8. Strongly Typed Programming Language	D.49	R	HR	HR	HR	HR
9. Structured Programming	D.53	R	HR	HR	HR	HR
10. Programming Language	Table A.15	R	HR	HR	HR	HR
11. Language Subset	D.35	–	–	–	HR	HR
12. Object Oriented Programming	Table A.22 D.57	R	R	R	R	R
13. Procedural programming	D.60	R	HR	HR	HR	HR
14. Metaprogramming	D.59	R	R	R	R	R

Requirements:

a) An approved combination of techniques for Software Safety Integrity Levels 3 and 4 is 4, 5, 6, 8 and one from 1 or 2.

b) An approved combination of techniques for Software Safety Integrity Levels 1 and 2 is 3, 4, 5, 6 and one from 8, 9 or 10.

c) Metaprogramming shall be restricted to the production of the code of the software source before compilation.

Table A.5 – Verification and Testing (6.2 and 7.3, 7.5)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Proof	D.29	–	R	R	HR	HR
2. Static Analysis	Table A.19	–	HR	HR	HR	HR
3. Dynamic Analysis and Testing	Table A.13	–	HR	HR	HR	HR
4. Metrics	D.37	–	R	R	R	R
5. Traceability	D.58	R	HR	HR	M	M
6. Software Error Effect Analysis	D.25	–	R	R	HR	HR
7. Test Coverage for code	Table A.21	R	HR	HR	HR	HR
8. Functional/ Black-box Testing	Table A.14	HR	HR	HR	M	M
9. Performance Testing	Table A.18	–	HR	HR	HR	HR
10. Interface Testing	D.34	HR	HR	HR	HR	HR

Requirements:

a) For software Safety Integrity Levels 3 and 4, the approved combination of techniques is 3, 5, 7, 8 and one from 1, 2 or 6.

b) For Software Safety Integrity Level 1 and 2, the approved combinations of techniques is 5 together with one from 2, 3 or 8.

NOTE 1 Techniques/measures 1, 2, 4, 5, 6 and 7 are for verification activities.

NOTE 2 Techniques/measures 3, 8, 9 and 10 are for testing activities.

Table A.6 – Integration (7.6)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Functional and Black-box Testing	Table A.14	HR	HR	HR	HR	HR
2. Performance Testing	Table A.18	-	R	R	HR	HR

Table A.7 – Overall Software Testing (6.2 and 7.7)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Performance Testing	Table A.18	–	HR	HR	M	M
2. Functional and Black-box Testing	Table A.14	HR	HR	HR	M	M
3. Modelling	Table A.17	–	R	R	R	R
Requirement: a) For Software Safety Integrity Level 1 and 2 an approved combination of techniques is 1 and 2.						

Table A.8 – Software Analysis Techniques (6.3)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Static Software Analysis	D.13 D.37 Table A.19	R	HR	HR	HR	HR
2. Dynamic Software Analysis	Table A.13 Table A.14	–	R	R	HR	HR
3. Cause Consequence Diagrams	D.6	R	R	R	R	R
4. Event Tree Analysis	D.22	–	R	R	R	R
5. Software Error Effect Analysis	D.25	–	R	R	HR	HR
Requirement: a) One or more of these techniques shall be selected to satisfy the Software Safety Integrity Level being used.						

Table A.9 – Software Quality Assurance (6.5)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Accredited to ISO 9001	7.1	R	HR	HR	HR	HR
2. Compliant with ISO 9001	7.1	M	M	M	M	M
3. Compliant with ISO/IEC 90003	7.1	R	R	R	R	R
4. Company Quality System	7.1	M	M	M	M	M
5. Software Configuration Management	D.48	M	M	M	M	M
6. Checklists	D.7	R	HR	HR	HR	HR
7. Traceability	D.58	R	HR	HR	M	M
8. Data Recording and Analysis	D.12	HR	HR	HR	M	M
Requirement: a) This table shall be applied to different roles and all phases.						

Table A.10 – Software Maintenance (9.2)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Impact Analysis	D.32	R	HR	HR	M	M
2. Data Recording and Analysis	D.12	HR	HR	HR	M	M

Table A.11 – Data Preparation Techniques (8.4)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Tabular Specification Methods	D.68	R	R	R	R	R
2. Application specific language	D.69	R	R	R	R	R
3. Simulation	D.42	R	HR	HR	HR	HR
4. Functional testing	D.42	M	M	M	M	M
5. Checklists	D.7	R	HR	HR	M	M
6. Fagan inspection	D.23	–	R	R	R	R
7. Formal design reviews	D.56	R	HR	HR	HR	HR
8. Formal proof of correctness (of data)	D.29	–	–	–	HR	HR
9. Walkthrough	D.56	R	R	R	HR	HR
Requirements: a) For Software Safety Integrity Level 1 and 2 an approved combination of techniques is 1 and 4. b) For Software Safety Integrity Level 3 and 4 the approved combinations of techniques are 1, 4, 5 and 7 or 2, 3 and 6. NOTE The description of the reference D.29 is on programs while technique 8 in this context applies to formal proof of the correctness of data.						

A.3 Detailed tables

Table A.12 – Coding Standards

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Coding Standard	D.15	HR	HR	HR	M	M
2. Coding Style Guide	D.15	HR	HR	HR	HR	HR
3. No Dynamic Objects	D.15	–	R	R	HR	HR
4. No Dynamic Variables	D.15	–	R	R	HR	HR
5. Limited Use of Pointers	D.15	–	R	R	R	R
6. Limited Use of Recursion	D.15	–	R	R	HR	HR
7. No Unconditional Jumps	D.15	–	HR	HR	HR	HR
8. Limited size and complexity of Functions, Subroutines and Methods	D.38	HR	HR	HR	HR	HR
9. Entry/Exit Point strategy for Functions, Subroutines and Methods	D.38	R	HR	HR	HR	HR
9. Limited number of subroutine parameters	D.38	R	R	R	R	R
10. Limited use of Global Variables	D.38	HR	HR	HR	M	M
Requirement: a) It is accepted that techniques 3, 4 and 5 may be present as part of a validated compiler or translator.						

Table A.13 – Dynamic Analysis and Testing

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Test Case Execution from Boundary Value Analysis	D.4	–	HR	HR	HR	HR
2. Test Case Execution from Error Guessing	D.20	R	R	R	R	R
3. Test Case Execution from Error Seeding	D.21	–	R	R	R	R
4. Performance Modelling	D.39	–	R	R	HR	HR
5. Equivalence Classes and Input Partition Testing	D.18	R	R	R	HR	HR
6. Structure-Based Testing	D.50	–	R	R	HR	HR
Requirement: a) The analysis for the test cases is at the sub-system level and is based on the specification and/or the specification and the code.						

Table A.14 – Functional/Black Box Test

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Test Case Execution from Cause Consequence Diagrams	D.6	–	–	–	R	R
2. Prototyping/Animation	D.43	–	–	–	R	R
3. Boundary Value Analysis	D.4	R	HR	HR	HR	HR
4. Equivalence Classes and Input Partition Testing	D.18	R	HR	HR	HR	HR
5. Process Simulation	D.42	R	R	R	R	R
Requirement: a) The completeness of the simulation will depend upon the extent of the software safety integrity level, complexity and application.						

Table A.15 – Textual Programming Languages

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. ADA	D.54	R	HR	HR	HR	HR
2. MODULA-2	D.54	R	HR	HR	HR	HR
3. PASCAL	D.54	R	HR	HR	HR	HR
4. C or C++	D.54 D.35	R	R	R	R	R
5. PL/M	D.54	R	R	R	NR	NR
6. BASIC	D.54	R	NR	NR	NR	NR
7. Assembler	D.54	R	R	R	R	R
8. C#	D.54 D.35	R	R	R	R	R
9. JAVA	D.54 D.35	R	R	R	R	R
10. Statement List	D.54	R	R	R	R	R

Requirements:

a) The selection of the languages shall be based on the requirements given in 6.7 and 7.3.

b) There is no requirement to justify decisions taken to exclude specific programming languages.

NOTE 1 For information on assessing the suitability of a programming language see entry in Clause D.54 'Suitable Programming Languages'.

NOTE 2 If a specific language is not in the table, it is not automatically excluded. It is expected to conform to Clause D.54.

NOTE 3 Run-time systems associated with selected languages which are necessary to run application programs should still be justified for usage according to the Software Safety Integrity Level.

Table A 16 – Diagrammatic Languages for Application Algorithms

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Functional Block Diagrams	D.63	R	R	R	R	R
2. Sequential Function Charts	D.61	–	HR	HR	HR	HR
3. Ladder Diagrams	D.62	R	R	R	R	R
4. State Charts	D.64	R	HR	HR	HR	HR

Table A.17 – Modelling

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Data Modelling	D.65	R	R	R	HR	HR
2. Data Flow Diagrams	D.11	–	R	R	HR	HR
3. Control Flow Diagrams	D.66	R	R	R	HR	HR
4. Finite State Machines or State Transition Diagrams	D.27	–	HR	HR	HR	HR
5. Time Petri Nets	D.55	–	R	R	HR	HR
6. Decision/Truth Tables	D.13	R	R	R	HR	HR
7. Formal Methods	D.28	–	R	R	HR	HR
8. Performance Modelling	D.39	–	R	R	HR	HR
9. Prototyping/Animation	D.43	–	R	R	R	R
10. Structure Diagrams	D.51	–	R	R	HR	HR
11. Sequence Diagrams	D.67	R	HR	HR	HR	HR
Requirements: a) A modelling guideline shall be defined and used. b) At least one of the HR techniques shall be chosen.						

Table A.18 – Performance Testing

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Avalanche/Stress Testing	D.3	–	R	R	HR	HR
2. Response Timing and Memory Constraints	D.45	–	HR	HR	HR	HR
3. Performance Requirements	D.40	–	HR	HR	HR	HR

Table A.19 – Static Analysis

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Boundary Value Analysis	D.4	–	R	R	HR	HR
2. Checklists	D.7	–	R	R	R	R
3. Control Flow Analysis	D.8	–	HR	HR	HR	HR
4. Data Flow Analysis	D.10	–	HR	HR	HR	HR
5. Error Guessing	D.20	–	R	R	R	R
6. Walkthroughs/Design Reviews	D.56	HR	HR	HR	HR	HR

Table A.20 – Components

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Information Hiding	D.33	–	–	–	–	–
2. Information Encapsulation	D.33	R	HR	HR	HR	HR
3. Parameter Number Limit	D.38	R	R	R	R	R
4. Fully Defined Interface	D.38	R	HR	HR	M	M

Requirement:

a) Information Hiding and encapsulation are only highly recommended if there is no general strategy for data access.

NOTE Technique/measure 4 is for Internal Interfaces.

Table A.21 – Test Coverage for Code

Test coverage criterion	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Statement	D.50	R	HR	HR	HR	HR
2. Branch	D.50	–	R	R	HR	HR
3. Compound Condition	D.50	–	R	R	HR	HR
4. Data flow	D.50	–	R	R	HR	HR
5. Path	D.50	–	R	R	HR	HR

Requirements:

a) For every SIL, a quantified measure of coverage shall be developed for the test undertaken. This can support the judgment on the confidence gained in testing and the necessity for additional techniques.

b) For SIL 3 or 4 test coverage at component level should be measured according to the following:

- 2 and 3; or
- 2 and 4; or
- 5,

or test coverage at integration level should be measured according to one or more of 2, 3, 4 or 5.

c) Other test coverage criteria can be used, given that this can be justified. These criteria depend on the software architecture (see Table A.3) and the programming language (see Table A.15 and Table A 16).

d) Any code which it is not practicable to test shall be demonstrated to be correct using a suitable technique, e.g. static analysis from Table A.19.

NOTE 1 Statement coverage is automatically achieved by items 2 to 5.

NOTE 2 The test coverage criteria in this table are used for structure-based (code-based, white box) testing. Techniques/measures for functional (specification-based, black box) testing are given in Table A.14.

NOTE 3 A high percentage of coverage is usually difficult to achieve. The use of test case execution from boundary values (Clause D.4) and equivalence classes and input partition testing (Clause D.18) can enable a sufficient coverage to be achieved with a smaller number of tests.

NOTE 4 The difference between 2 and 3 depends in practice on the level of the programming language and the use of compound conditions. When single conditions are used only, for example as a result of compilation, 2 and 3 are considered identical.

Table A.22 – Object Oriented Software Architecture

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Traceability of the concept of the application domain to the classes of the architecture	–	R	R	R	HR	HR
2. Use of suitable frames, commonly used combinations of classes and design patterns	–	R	R	R	HR	HR
3. Object Oriented Detailed Design	Table A.23	R	R	R	HR	HR

Requirement:

a) When using existing frames and design patterns, the requirements of pre-existing software apply to these frames and patterns.

NOTE 1 The object-oriented approach presents information differently from procedural approaches, the following list contains recommendations that need specific consideration:

– understanding class hierarchies, and identification of the software function(s) that will be executed upon the invocation of a given method (including when using an existing class library);

– structure-based testing (Table A.13).

Traceability from application domain to class architecture is less important.

NOTE 2 For a part of the intended software a frame might exist from pre-existing software that has successfully solved a similar task and that is well known to the development personnel. Then use of that frame is considered good practice.

Table A.23 – Object Oriented Detailed Design

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Classes should have only one objective	–	R	R	R	HR	HR
2. Inheritance used only if the derived class is a refinement of its basic class	–	R	HR	HR	HR	HR
3. Depth of inheritance limited by coding standards	–	R	R	R	HR	HR
4. Overriding of operations (methods) under strict control	–	R	R	R	HR	HR
5. Multiple inheritance used only for interface classes	–	R	HR	HR	HR	HR
6. Inheritance from unknown classes	–	–	–	–	NR	NR

Requirements:

a) One class is characterized by having one responsibility, i.e. taking care of closely connected data and the operations on these data.

b) Care is required to avoid circular dependencies between objects.

Annex B
(normative)

Key software roles and responsibilities

Table B.1 – Requirements Manager Role Specification

Role: Requirements Manager
Responsibilities: 1) shall be responsible for specifying the software requirements 2) shall own the Software Requirements Specification 3) shall establish and maintain traceability to and from system level requirements 4) shall ensure the specifications and software requirements are under change and configuration management including state, version and authorisation status 5) shall ensure consistency and completeness in the Software Requirements Specification (with reference to user requirements and final environment of application) 6) shall develop and maintain the software requirement documents
Key competences: 1) shall be competent in requirements engineering 2) shall be experienced in application's domain 3) shall be experienced in safety attributes of application's domain 4) shall understand the overall role of the system and the environment of application 5) shall understand analytical techniques and outcomes 6) shall understand applicable regulations 7) shall understand the requirements of IEC 62279

Table B.2 – Designer Role Specification

Role: Designer
Responsibilities: 1) shall transform specified software requirements into acceptable solutions 2) shall own the architecture and downstream solutions 3) shall define or select the design methods and supporting tools 4) shall apply appropriate design principles and standards 5) shall develop component specifications where appropriate 6) shall maintain traceability to and from the specified software requirements 7) shall develop and maintain the design documentation 8) shall ensure design documents are under change and configuration control

Key competences:

1) shall be competent in engineering appropriate to the application area
2) shall be competent in safety design principles
3) shall be competent in design analysis and design test methodologies
4) shall be able to work within design constraints in a given environment
5) shall be competent in understanding the problem domain
6) shall understand all the constraints imposed by the hardware platform, the operating system and the interfacing systems
7) shall understand the relevant clauses/subclauses of IEC 62279

Table B.3 – Implementer Role Specification

Role: Implementer
Responsibilities: 1) shall transform the design solutions into data/source code/other design representations 2) shall transform source code into executable code/other design representation 3) shall apply safety design principles 4) shall apply specified data preparation/coding standards 5) shall carry out analysis to verify the intermediate outcome 6) shall integrate software on the target machine 7) shall develop and maintain the implementation documents comprising the applied methods, data types, and listings 8) shall maintain traceability to and from design 9) shall maintain the generated or modified data/code under change and configuration control
Key competences: 1) shall be competent in engineering appropriate to the application area 2) shall be competent in the implementation language and supporting tools 3) shall be capable of applying the specified coding standards and programming styles 4) shall understand all the constraints imposed by the hardware platform, the operating system and the interfacing systems 5) shall understand the relevant clauses/subclauses of IEC 62279

Table B.4 – Tester Role Specification

Role: Tester
Responsibilities: 1) shall ensure that test activities are planned 2) shall develop the test specification (objectives and cases) 3) shall ensure traceability of test objectives against the specified software requirements and of test cases against the specified test objectives 4) shall ensure that the planned tests are implemented and specified tests are carried out 5) shall identify deviations from expected results and record them in test reports 6) shall communicate deviations with relevant change management body for evaluation and decision 7) shall capture outcomes in reports 8) shall select the software test equipment
Key competences: 1) shall be competent in the domain where testing is carried out e.g. software requirements, data, code, etc. 2) shall be competent in various test and verification approaches/methodologies and be able to identify the most suitable method in a given context 3) shall be capable of deriving test cases from given specifications 4) shall have analytical thinking ability and good observation skills 5) shall understand the relevant clauses/subclauses of IEC 62279

Table B.5 – Verifier Role Specification

Role: Verifier
Responsibilities: 1) shall develop a Software Verification Plan (which may include quality issues) stating what needs verification and what type of process (e.g. review, analysis, etc.) and test is required as evidence 2) shall check the adequacy (completeness, consistency, correctness, relevance and traceability) of the documented evidence from review, integration and testing with the specified verification objectives 3) shall identify anomalies, evaluate these in risk (impact) terms, record and communicate these to relevant change management body for evaluation and decision 4) shall manage the verification process (review, integration and testing) and ensure independence of activities as required 5) shall develop and maintain records on the verification activities 6) shall develop a Verification Report stating the outcome of the verification activities
Key competences: 1) shall be competent in the domain where verification is carried out e.g. software requirements, data, code, etc. 2) shall be competent in various verification approaches/methodologies and be able to identify the most suitable method or combination of methods in a given context 3) shall be capable of deriving the types of verification from given specifications 4) shall have analytical thinking ability and good observation skills 5) shall understand the relevant clauses/subclauses of IEC 62279

Table B.6 – Integrator Role Specification

Role: Integrator
Responsibilities: 1) shall manage the integration process using the software baselines 2) shall develop the Software and Software/Hardware Integration Test Specification for software components based on the Designer's component specifications and architecture stating what the necessary input components, the sequence of integration activities and the resultant integrated components are 3) shall develop and maintain records on the integration activities 4) shall identify integration anomalies, record and communicate these to relevant change management body for evaluation and decision 5) shall develop a component and overall system integration report stating the outcome of the integration
Key competences: 1) shall be competent in the domain where component integration is carried out, e.g. relevant programming languages, software interfaces, operating systems, data, platforms, code, etc. 2) shall be competent in various integration approaches/methodologies and be able to identify the most suitable method or combination of methods in a given context 3) shall be competent in understanding the design and functionality required at various intermediate levels 4) shall be capable of deriving the types of integration test from a set of integrated functions 5) shall have analytical thinking ability and good observation skills tending towards the system level perspective 6) shall understand the relevant clauses/subclauses of IEC 62279

Table B.7 – Validator Role Specification

Role: Validator
Responsibilities: 1) shall develop a system understanding of the software within the intended environment of application 2) shall develop a validation plan and specify the essential tasks and activities for software validation and agree this plan with the assessor 3) shall review the software requirements against the intended environment/use 4) shall review the software against the software requirements to ensure all of these are fulfilled 5) shall evaluate the conformity of the software process and the developed software against the requirements of this Standard including the assigned SIL 6) shall review the correctness, consistency and adequacy of the verification and testing 7) shall check the correctness, consistency and adequacy of test cases and executed tests 8) shall ensure all validation plan activities are carried out 9) shall review and classify all deviations in terms of risk (impact), record and submit to the body responsible for Change Management and decision making 10) shall give a recommendation on the suitability of the software for intended use and indicate any application constraints as appropriate 11) shall capture deviations from the validation plan 12) shall carry out audits, inspections or reviews on the overall project (as instantiations of the generic development process) as appropriate in various phases of development 13) shall review and analyse the validation reports relating to previous applications as appropriate 14) shall review that developed solutions are traceable to the software requirements 15) shall ensure the related hazard logs and remaining non-conformities are reviewed and all hazards closed out in an appropriate manner through elimination or risks control/transfer measures 16) shall develop a validation report 17) shall give agreement/disagreement for the release of the software
Key competences: 1) shall be competent in the domain where validation is carried out 2) shall be experienced in safety attributes of application's domain 3) shall be competent in various validation approaches/methodologies and be able to identify the most suitable method or combination of methods in a given context 4) shall be capable of deriving the types of validation evidence required from given specifications bearing in mind the intended application 5) shall be capable of combining different sources and types of evidence and synthesise an overall view about fitness for purpose or constraints and limitations of the application 6) shall have analytical thinking ability and good observation skills 7) shall have overall software understanding and perspective including understanding the application environment 8) shall understand the requirements of IEC 62279

Table B.8 – Assessor Role Specification

Role: Assessor
Responsibilities: 1) shall develop a system understanding of the software within the intended environment of application 2) shall develop an assessment plan and communicate this with the safety authority and the client organisation (contracting body of the assessor) 3) shall evaluate the conformity of the software process and the developed software against the requirements of this Standard including the assigned SIL 4) shall evaluate the competency of project staff and organisation for the software development 5) shall evaluate the verification and validation activities and the supporting evidence 6) shall evaluate the quality management systems adopted for the software development 7) shall evaluate the configuration and change management system and the evidence of its use and application 8) shall identify and evaluate in terms of risk (impact) any deviations from the software requirements in the assessment report 9) shall ensure that the assessment plan is implemented 10) shall carry out safety audits and inspections on the overall development process as appropriate at various phases of development 11) shall give a professional view on the fitness of the developed software for its intended use detailing any constraints, application conditions and observations for risk control as appropriate 12) shall develop an assessment report and maintain records on the assessment process
Key competences: 1) shall be competent in the domain/technologies where assessment is carried out 2) shall have acceptance/licence from a recognised safety authority 3) shall have / strive to continually gain sufficient levels of experience in the safety principles and the application of the principles within the application domain 4) shall be competent to check that a suitable method or combination of methods in a given context have been applied 5) shall be competent in understanding the relevant safety, human resource, technical and quality management processes in fulfilling the requirements of IEC 62279 6) shall be competent in assessment approaches/methodologies 7) shall have analytical thinking ability and good observation skills 8) shall be capable of combining different sources and types of evidence and synthesise an overall view about fitness for purpose or constraints and limitations on application 9) shall have overall software understanding and perspective including understanding the application environment 10) shall be able to judge the adequacy of all development processes (like quality management, configuration management, validation and verification processes) 11) shall understand the requirements of IEC 62279

Table B.9 – Project Manager Role Specification

Role: Project Manager
Responsibilities: 1) shall ensure that the quality management system and independency of roles according to 5.1 are in place for the project and progress is checked against the plans 2) shall allocate sufficient number of competent resources in the project to carry out the essential tasks including safety activities, bearing in mind the requisite independence of roles 3) shall ensure that a suitable validator has been appointed for the project as defined in IEC 62279 4) shall be responsible for the delivery and deployment of the software and ensure that safety requirements from the stakeholders are also fulfilled and delivered 5) shall allow sufficient time for the proper implementation and fulfilment of safety tasks 6) shall endorse partial and complete safety deliverables from the development process 7) shall ensure that sufficient records and traceability is maintained in safety related decision making
Key competences: 1) shall understand quality, competencies, organisational and management requirements of IEC 62279 2) shall understand the requirements of the safety process 3) shall be able to weigh different options and understand the impact on safety performance of a decision or selected options 4) shall understand the requirements of the software development process 5) shall understand the requirements of other relevant standards

Table B.10 – Configuration Manager Role Specification

Role: Configuration Manager
Responsibilities: 1) shall be responsible for the software configuration management plan 2) shall own the configuration management system 3) shall establish that all software components are clearly identified and independently versioned inside the configuration management system 4) shall prepare Release Notes which include incompatible versions of software components
Key competences: 1) shall be competent in software configuration management 2) shall understand the requirements of IEC 62279

Table B.11 – Quality Assurance Manager Role Specification

Role: Quality Assurance Manager
Responsibilities: 1) shall be responsible for the Software Quality Assurance Plan 2) shall apply the Quality Management System in given project 3) shall ensure that an audit trail can be established to allow verification and validation activities to be undertaken effectively 4) shall define quality goals in cooperation with the Project Manager 5) shall manage the software quality assurance process as defined in the Software Quality Assurance Plan 6) shall develop a Software Quality Assurance Report stating the outcome of the quality assurance activities as planned in the Software Quality Assurance Plan NOTE The Software Quality Assurance Report is included in the Quality Management Report defined in IEC 62425.
Key competences: 1) shall be competent in software Quality Management 2) shall understand the Quality Management in a given project 3) shall be competent in tailoring the Quality Management in a given project 4) shall be competent in software quality assurance activities as required in ISO 9001, ISO/IEC 90003, ISO/IEC 25010 and IEC 62279 5) shall be competent in the company quality system 6) shall be competent in software process improvement methods 7) shall have overall understanding of software configuration management, document control and traceability, data recording and analysis 8) shall be competent in performance of audits 9) shall have overall understanding of software engineering 10) shall have overall understanding of project management 11) shall have analytical thinking ability and good observation skills 12) shall have good communication abilities 13) shall understand the requirements of IEC 62279

Table B.12 – Reviewer Role Specification

Role: Reviewer
Responsibilities: 1) shall review an output document according to the defined criteria under the supervision of the verifier 2) shall provide a record of the review results such as those relating to 7.2.4.22 3) shall not be the same person as the producer of a document
Key competences: 1) shall be competent in the domain where the review is carried out, e.g. software requirements, design and testing. 2) shall be competent in fulfilling the objectives which is chosen for the review

Annex C
(informative)

Documents Control Summary

Table C.1 – Documents Control Summary *(1 of 2)*

PHASE	DOCUMENTATION	Written by	1st check	2nd check
Planning	1. Software Quality Assurance Plan	QAM	VER	VAL
	2. Software Quality Assurance Verification Report	VER		VAL
	3. Software Configuration Management Plan	CGM	VER	VAL
	4. Software Verification Plan	VER		VAL
	5. Software Validation Plan	VAL	VER	
Software requirements	6. Software Requirements Specification	RQM	VER	VAL
	7. Overall Software Test Specification	TST	VER	VAL
	8. Software Requirements Verification Report	VER		VAL
Architecture and design	9. Software Architecture Specification	DES	VER	VAL
	10. Software Design Specification	DES	VER	VAL
	11. Software Interface Specifications	DES	VER	VAL
	12. Software Integration Test Specification	INT	VER	VAL
	13. Software/Hardware Integration Test Specification	INT	VER	VAL
	14. Software Architecture and Design Verification Report	VER		VAL
Component design	15. Software Component Design Specification	DES	VER	VAL
	16. Software Component Test Specification	TST	VER	VAL
	17. Software Component Design Verification Report	VER		
Component implementation and testing	18. Software Source Code and Supporting Documentation	IMP	VER	VAL
	19. Software Component Test Report	TST	VER	VAL
	20. Software Source Code Verification Report	VER		VAL
Integration	21. Software Integration Test Report	INT	VER	VAL
	22. Software/Hardware Integration Test Report	INT	VER	VAL
	23. Software Integration Verification Report	VER		
Overall software testing / Final validation	24. Overall Software Test Report	TST	VER	VAL
	25. Overall Software Test Verification Report	VER		VAL
	26. Software Validation Report	VAL	VER	
	27. Tools Validation Report	[a]	VER	
	28. Software Quality Assurance Report	QAM	VER	
	29. Release Note	CGM	VER	VAL

Table C.1 *(2 of 2)*

PHASE	DOCUMENTATION	Written by	1st check	2nd check
Systems configured by application data/algorithms	30. Application Requirements Specification	RQM	VER	VAL
	31. Application Preparation Plan	RQM or DES	VER	VAL
	32. Application Test Specification	TST	VER	VAL
	33. Application Architecture and Design	DES	VER	VAL
	34. Application Preparation Verification Report	VER		
	35. Application Test Report	TST	VER	VAL
	36. Source Code of Application Data/Algorithms	DES	VER	VAL
	37. Application Data/Algorithms Verification Report	VER		VAL
Software deployment	38. Software Release and Deployment Plan	[a]	VER	VAL
	39. Software Deployment Manual	[a]	VER	VAL
	40. Release Notes	DES	VER	VAL
	41. Deployment Records	[a]	VER	VAL
	42. Deployment Verification Report	VER		
Software maintenance	43. Software Maintenance Plan	[a]	VER	VAL
	44. Software Change Records	[a]	VER	VAL
	45. Software Maintenance Records	[a]	VER	VAL
	46. Software Maintenance Verification Report	VER		VAL
Software assessment	47. Software Assessment Plan	ASR		
	48. Software Assessment Report	ASR		

a No specific role defined.

NOTE The Software Quality Assurance Report is included in the Quality Management Report defined in IEC 62425.

Annex D
(informative)

Aim and description of techniques

D.1 Artificial Intelligence Fault Correction

Aim

To be able to react to possible hazards in a very flexible way by introducing a mix (combination) of methods and process models and some kind of on-line safety and reliability analysis.

Description

In particular fault forecasting (calculating trends), fault correction, maintenance and supervisory actions may be supported by Artificial Intelligence-based systems in a very efficient way in diverse channels of a system, since the rules might be derived directly from the specifications and checked against these. Certain common faults which are introduced into specifications by implicitly already having some design and implementation rules in mind may be avoided effectively by this approach, especially when applying a combination of models and methods in a functional or descriptive manner.

The methods are selected such that faults may be corrected and the effects of failures be minimised, in order to meet the desired safety and reliability.

D.2 Analysable Programs

Aim

To design a program in a way that program analysis is easily feasible. The program behaviour shall be testable completely on the basis of the analysis.

Description

The intention is to produce programs which are easy to analyse using static analysis methods. In order to achieve this, the rules of structured programming should be followed, for instance:

– the component control flow should be composed of structured constructs, that is sequences, iterations and selection;
– the components should be small;
– the number of possible paths through a component is small;
– the individual program parts have to be designed so that they are decoupled as far as possible;
– the relation between the input and output parameters should be as simple as possible;
– complex calculations should not be used as the basis of branching and loop decisions;
– branch and loop decisions should be simply related to the component input parameters;
– boundaries between different types of mappings shall be simple.

D.3 Avalanche/Stress Testing

Aim

To burden the test object with an exceptionally high workload in order to show that the test object would stand normal workloads easily.

Description

There are a variety of test conditions which can be applied for avalanche/stress testing. Some of these test conditions are listed below:

– if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;
– if working on demands then the number of demands per time unit to the test object is increased beyond normal conditions;
– if the size of a database plays an important role then it is increased beyond normal conditions;
– influential devices are tuned to their maximum speed or lowest speed respectively;
– for the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.

Under these test conditions the time behaviour of the test object can be evaluated. The influence of load changes can be observed. The correct dimension of internal buffers or dynamic variables, stacks, etc., can be checked.

D.4 Boundary Value Analysis

Aim

To remove software errors occurring at parameter limits or boundaries.

Description

The input domain of the program is divided into a number of input classes. The tests should cover the boundaries and extremes of the classes. The tests check that the boundaries in the input domain of the specification coincide with those in the program. The use of the value zero, in a direct as well as in an indirect translation, is often error-prone and demands special attention:

– zero divisor;
– non-printing control characters;
– empty stack or list element;
– null matrix;
– zero table entry.

Normally the boundaries for input have a direct correspondence to the boundaries for the output range. Test cases should be written to force the output to its limited values. Consider also, if it is possible to specify a test case which causes output to exceed the specification boundary values.

If output is a sequence of data, for example a printed table, special attention should be paid to the first and the last elements and to lists containing none, 1 and 2 elements.

D.5 Backward Recovery

Aim

To provide correct functional operation in the presence of one or more faults.

Description

If a fault has been detected, the system is reset to an earlier internal state, the consistency of which has been proven before. This method implies saving of the internal state frequently at so-called well defined checkpoints. This may be done globally (for the complete database) or incremental (changes only between checkpoints). Then the system has to compensate for the changes which have taken place in the meantime by using journaling (audit trail of actions), compensation (all effects of these changes are nullified) or external (manual) interaction.

D.6 Cause Consequence Diagrams

Aim

To model, in a diagrammatic form, the sequence of events that can develop in a system as a consequence of combinations of basic events.

Description

It can be regarded as a combination of fault-tree and event-tree analysis. Starting from a critical event, a cause-consequence graph is traced backwards and forwards. In the backwards direction it is equivalent to a fault tree with the critical event as the given top event. In the forward direction the possible consequences arising from an event are identified. The graph can contain vertex symbols which describe the conditions for propagation along different branches from the vertex. Time delays can also be included. These conditions can also be described with fault trees. The lines of propagation can be combined with logical symbols, to make the diagram more compact. A set of standard symbols are defined for use in cause consequence diagrams. The diagrams can be used to compute the probability of occurrence of certain critical consequences.

D.7 Checklists

Aim

To provide a stimulus to critical appraisal of all aspects of the system rather than to lay down specific requirements.

Description

A set of questions to be completed by the person performing the checklist. Many of the questions are of a general nature and the Assessor shall interpret them as seems most appropriate to the particular system being assessed.

To accommodate wide variations in software and systems being validated, most checklists contain questions which are applicable to many types of system. As a result there may well be questions in the checklist being used which are not relevant to the system being dealt with and which should be ignored. Equally there may be a need, for a particular system, to supplement the standard checklist with questions specifically directed at the system being dealt with.

In any case it should be clear that the use of checklists depends critically on the expertise and judgement of the engineer selecting and applying the checklist. As a result the decisions taken by the engineer, with regard to the checklist(s) selected, and any additional or

superfluous questions, should be fully documented and justified. The objective is to ensure that the application of the checklists can be reviewed and that the same results will be achieved unless different criteria are used.

The object in completing a checklist is to be as concise as possible. When extensive justification is necessary this should be done by reference to additional documentation. Pass, Fail and Inconclusive, or some similar restricted set of tokens should be used to record the results for each question. This conciseness greatly simplifies the process of reaching an overall conclusion as to the results of the checklist assessment.

D.8 Control Flow Analysis

Aim

To detect poor and potentially incorrect program structures.

Description

Control Flow Analysis identifies suspect areas of code which do not follow good programming practice. The program is analysed to form a directed graph which can be analysed for

– inaccessible code, for instance, unconditional jumps which leaves blocks of code unreachable,
– knotted code, which is well structured code whose control graph is reducible by successive graph reductions to a single node. Poorly structured code can only be reduced to a knot composed of several nodes.

D.9 Common Cause Failure Analysis

Aim

To identify potential failures in redundant systems or redundant sub systems which would undermine the benefits of redundancy because of the appearance of the same failures in the redundant parts at the same time.

Description

Computer systems intended to take care of the safety of a plant often use redundancy in hardware and majority voting. This technique is used to avoid random component failures, which would tend to prevent the correct processing of data in a computer system.

However, some failures can be common to more than one component. For example, if a computer system is installed in one single room, shortcomings in the air-conditioning might reduce the benefits of redundancy. The same is true for other external effects on the computer system such as: fire, flooding, electromagnetic interference, plane crashes, and earthquakes. The computer system may also be affected by incidents related to operation and maintenance. It is essential, therefore, that adequate and well documented procedures are provided for operation and maintenance. Extensive training of operating and maintenance personnel is also essential.

Internal effects are also major contributors to Common-Cause Failures (CCF). They can stem from design errors in common or identical components and their interfaces, as well as ageing of components. CCF-Analysis has to search the system for such potential common failures. Methods of CCF-Analysis are general quality control, design reviews, verification and testing by an independent team, and analysis of real incidents with feedback of experience from similar systems. The scope of the analysis, however, goes beyond hardware. Even if 'diverse software' is used in difficult chains of a redundant computer system, there might be some

commonality in the software approaches which could give rise to CCF. Errors in the common specification, for example.

When CCFs do not occur exactly at the same time, precautions can be taken by means of comparison methods between the redundant chains which should lead to detection of a failure before this failure is common to all chains. CCF analysis should take this technique into account.

D.10 Data Flow Analysis

Aim

To detect poor and potentially incorrect program structures.

Description

Data Flow Analysis combines the information obtained from the control flow analysis with information about which variables are read or written in different portions of code. The analysis can check for

– variables that are read before they are written. This is very likely to be an error, and is certainly bad programming practice,
– variables that are written more than once without being read. This could indicate omitted code,
– variables that are written but never read. This could indicate redundant code.

There is an extension of data flow analysis known as information flow analysis, where the actual data flows (both within and between procedures) are compared with the design intent. This is normally implemented with a computerised tool where the intended data flows are defined using a structured comment that can be read by the tool.

D.11 Data Flow Diagrams

Aim

To describe the data flow through a program in a diagrammatic form.

Description

Data Flow Diagrams document how data input is transformed to output, with each stage in the diagram representing a distinct transformation.

The basic components of a data flow diagram include

– functions, represented by bubbles,
– data flows, represented by arrows,
– data stores, represented by open boxes,
– input/output, represented by special kinds of boxes.

Data flow diagrams describe how an input is transformed to an output. They do not, and should not, include control information or sequencing information. Each bubble can be considered as a stand alone black box which, as soon as its inputs are available, transforms them to its outputs.

One of the principal advantages of data flow diagrams is that they show transformations without making any assumptions about how these transformations are implemented.

The preparation of data flow diagrams is best approached by considering system inputs and working towards system outputs. Each bubble shall represent a distinct transformation – its output should, in some way, be different from its input. There are no rules for determining the overall structure of the diagram and constructing a data flow diagram is one of the creative aspects of system design. Like all design, it is an iterative process with early attempts refined in stages to produce the final diagram.

D.12 Data Recording and Analysis

Aim

To facilitate software process improvement by recording, validating and analysing relevant data from individual projects and persons. The relevance of the data is determined by the strategic goals of the organisation. The goals may be directed towards the evaluation of a particular software development method relative to the claims for it, e.g. with respect to defect prevention effectiveness.

Description

Data recording and analysis constitutes an essential part of software process improvement. The recording of valid data represents an important part of learning more about the software development process and to evaluate alternative software development methods.

Detailed records are maintained during a project, both on a project and individual basis. For instance, an engineer would be required to keep records which could include

– effort expanded on individual components,
– testing performed on each component,
– decisions and their rationale,
– achievement of project milestones,
– problems and their solutions.

During and at the conclusion of the project these records can be analysed to establish a wide variety of information. In particular data recording is very important for the maintenance of computer systems as the rationale for certain decisions made during the development project is not always known by the maintenance engineers.

Due to poor planning, data recording often tends to be over-volumed and out of focus. This can be avoided by following the principle that data recording should be driven by goals, questions, and metrics of relevance to what is strategically important to the organisation.

In order to achieve the desired accuracy, the data recording and validation process should proceed concurrently with the development, e.g. as part of the configuration control process.

D.13 Decision Tables (Truth Tables)

Aim

To provide a clear and coherent specification and analysis of complex logical combinations and relationships.

Description

These related methods use two dimensional tables to concisely describe logical relationships between Boolean program variables.

The conciseness and tabular nature of both methods make them appropriate as a means of analysing complex logical combinations expressed in code.

Both methods are potentially executable if used as specifications.

D.14 Defensive Programming

Aim

To produce programs which detect anomalous control flow, data flow or data values during their execution and react to these in a predetermined and acceptable manner.

Description

Many techniques can be used during programming to check for control or data anomalies. These can be applied systematically throughout the programming of a system to decrease the likelihood of erroneous data processing.

Two overlapping areas of defensive techniques can be identified. Intrinsic error-safe software is designed to accommodate its own design shortcomings. These shortcomings may be due to plain error of design or coding, or to erroneous requirements. The following lists some of the defensive techniques:

– variables should be range checked;
– where possible, values should be checked for plausibility;
– parameters to procedures should be type, dimension and range checked at procedure entry.

These first three recommendations help to ensure that the numbers manipulated by the program are reasonable, both in terms of the program function and physical significance of the variables.

Read-only and read-write parameters should be separated and their access checked. Functions should treat all parameters as read-only. Literal constants should not be write-accessible. This helps detect accidental overwriting or mistaken use of variables.

Error tolerant software is designed to 'expect' failures in its own environment or use outside nominal or expected conditions, and behave in a predefined manner. Techniques include the following:

– input variables and intermediate variables with physical significance should be checked for plausibility;
– the effect of output variables should be checked, preferably by direct observation of associated system state changes;
– the software should check its configuration. This could include both the existence and accessibility of expected hardware and also that the software itself is complete. This is particularly important for maintaining integrity after maintenance procedures.

Some of the defensive programming techniques such as control flow sequence checking, also cope with external failures.

D.15 Coding Standards and Style Guide

Aim

To ensure a uniform layout of the design documents and the produced code, enforce consistent programming and to enforce a standard design method which avoids errors.

Description

Coding Standards are rules and restrictions on a given programming language to avoid potential faults which can be made when using that language.

Coding standard content could include:

– language justification,
– scope and base standard when available,

NOTE For domain specific languages base standards may not be available.

– procedure for changing the coding standard,
– analysis of the potential faults and recommended treatment,
– restrictions to avoid the faults,
– portability.

Style guidelines deal with issues such as formatting and naming conventions, and although it can be highly subjective, more than anything style affects the readability of your code. The establishment of a common and consistent style for a project will facilitate understanding and maintenance of code developed by more than one programmer, and will make it easier for several people to cooperate in the development of the same program.

D.16 Diverse Programming

Aim

Detect and mask residual software design faults during execution of a program, in order to prevent safety critical failures of the system, and to continue operation for high reliability.

Description

In diverse programming a given program specification is implemented N times in different ways. The same input values are given to the N versions, and the results produced by the N versions are compared. If the result is considered to be valid, the result is transmitted to the computer outputs.

The N versions can run in parallel on separate computers, alternatively all versions can be run on the same computer and the results subjected to an internal vote. Different voting strategies can be used on the N versions depending on the application requirements.

– If the system has a safe state, then it is feasible to demand complete agreement (all N agree) otherwise a fail-safe output value is used. For simple trip systems the vote can be biased in the safe direction. In this case the safe action would be to trip if either version demanded a trip. This approach typically uses only two versions (N = 2).
– For systems with no safe state, majority voting strategies can be employed. For cases where there is no collective agreement, probabilistic approaches can be used in order to maximise the chance of selecting the correct value, for example, taking the middle value, temporary freezing of outputs until agreement returns etc.

This technique does not eliminate residual software design faults, but it provides a measure to detect and mask before they can affect safety.

Unfortunately, experiments and analytical studies show that N-version programming is not always as effective as desired. Even if different algorithms are used, diverse software versions too often fail on the same inputs.

Two alternatives to N-version programming are design diversity and functional diversity. Design diversity involves the use of multiple components, each designed in a different way

but implementing the same function. Functional diversity involves solving the same problem in functionally different ways. Irrespective of the approach, no effective method to assess the level of diversity is currently available.

D.17 Dynamic Reconfiguration

Aim

To maintain system functionality despite an internal fault.

Description

The logical architecture of the system has to be such that it can be mapped onto a subset of the available resources of the system. The architecture needs to be capable of detecting a failure in a physical resource and then remapping the logical architecture back onto the restricted resources left functioning. Although the concept is more traditionally restricted to recovery from failed hardware units, it is also applicable to failed software units if there is sufficient 'run-time redundancy' to allow a software re-try or if there is sufficient redundant data to isolate the failure.

Although traditionally applied to hardware, this technique is being developed for application to software and, thus, the total system. It shall be considered at the first system design stage.

D.18 Equivalence Classes and Input Partition Testing

Aim

To test the software adequately using a minimum of test data. The test data is obtained by selecting the partitions of the input domain required to exercise the software.

Description

This testing strategy is based on the equivalence relation of the inputs, which determines a partition of the input domain.

Test cases are selected with the aim of covering all subsets of this partition. At least one test case is taken from each equivalence class.

There are two basic possibilities for input partitioning which are

– equivalence classes may be defined on the specification. The interpretation of the specification may be either input oriented, for example the values selected are treated in the same way or output oriented, for example the set of values leading to the same functional result, and

– equivalence classes may be defined on the internal structure of the program. In this case the equivalence class results are determined from static analysis of the program, for example the set of values leading to the same path being executed.

D.19 Error Detecting and Correcting Codes

Aim

To detect and correct errors in sensitive information.

Description

For an information of n bits, a coded block of k bits is generated which enables errors to be detected and corrected. Different types of code include:

– hamming codes;
– cyclic codes;
– polynomial codes;
– hash codes;
– cryptographic codes.

D.20 Error Guessing

Aim

To remove common programming errors.

Description

Testing experience and intuition combined with knowledge and curiosity about the system under test may add some uncategorised test cases to the designed test case set. Special values or combinations of values may be error-prone. Some interesting test cases may be derived from inspection checklists. It may also be considered whether the system is robust enough. Can the buttons be pushed on the front-panel too fast or too often? What happens if two buttons are pushed simultaneously?

D.21 Error Seeding

Aim

To ascertain whether a set of test cases is adequate.

Description

Some known error types are inserted in the program, and the program is executed with the test cases under test conditions. If only some of the seeded errors are found, the test case set is not adequate. The ratio of found seeded errors to the total number of seeded errors is an estimate of the ratio of found real errors to total number errors. This gives a possibility of estimating the number of remaining errors and thereby the remaining test effort.

$$\frac{\text{Found seeded errors}}{\text{Total number of seeded errors}} = \frac{\text{Found real errors}}{\text{Total number of real errors}}$$

The detection of all the seeded errors may indicate either that the test case set is adequate, or that the seeded errors were too easy to find. The limitations of the method are that, in order,to obtain any usable results, the error types as well as the seeding positions shall reflect the statistical distribution of real errors.

If error seeding is used, the location of all errors shall be recorded, and the validator shall ensure that all seeded errors have been removed before the software release.

D.22 Event Tree Analysis

Aim

To model, in a diagrammatic form, the sequence of events that can develop in a system after an initiating event, and thereby indicate how serious consequences can occur.

Description

On the top of the diagram is written the sequence conditions that are relevant in the development following the initiating event which is the target of the analysis. Starting under the initiating event, one draws a line to the first condition in the sequence. There the diagram branches off into a 'yes' and a 'no' branch, describing how the future developments depend on the condition. For each of these branches, one continues to the next condition in a similar way. Not all conditions are, however, relevant for all branches. One continues to the end of the sequence, and each branch of the tree constructed in this way represents a possible consequence. The event tree can be used to compute the probability of the various consequences based on the probability and number of conditions in the sequence.

D.23 Fagan Inspections

Aim

To reveal errors in all phases of the program development.

Description

A 'formal' audit on quality assurance documents aimed at finding errors and omissions. The inspection process consists of five phases: Planning, Preparation, Inspection, Rework and Follow up. Each of these phases has its own separate objective. The complete system development (specification, design, coding and testing) shall be inspected.

D.24 Failure Assertion Programming

Aim

To detect residual faults during execution of a software program.

Description

The assertion programming method follows the idea of checking a pre-condition (before a sequence of statements is executed, the initial conditions are checked for validity) and a post-condition (results are checked after the execution of a sequence of statements). If either the pre-condition or the post-condition is not fulfilled, the processing stops with an error.

For example,

```
assert <pre-condition>;
    action 1;
        :
        :
    action x;
assert <post-condition>;
```

D.25 SEEA – Software Error Effect Analysis

Aim

To identify software components, their criticality; to propose means for detecting software errors and enhancing software robustness; to evaluate the amount of validation needed on the various software components.

Description

The analysis is done in three phases.

- Vital software components identification

  Determination of the depth of the analysis (at the level of a single instruction line, a group of instructions, a component, etc.) needed for each software component, from its specification.
- Software error analysis

  The result of this phase is a table listing the following information:
  - component name;
  - error considered;
  - consequences of the error at the module level;
  - consequences at the system level;
  - violated safety criterion;
  - error criticality;
  - proposed error detection means;
  - violated criterion if the detection means is implemented;
  - residual criticality if the detection means is implemented.
- Synthesis

  The synthesis identifies the remaining unsafe scenarios and the validation effort needed given the criticality of each module.

SEEA, being an in-depth analysis carried out by an independent team, is a powerful bug-finding method.

D.26 Fault Detection and Diagnosis

Aim

To detect faults in a system, which might lead to a failure, thus providing the basis for countermeasures in order to minimise the consequences of failures.

Description

Fault detection is the process of checking a system for erroneous states (which are caused, as explained before, by a fault within the (sub)system to be checked). The primary goal of fault detection is to inhibit the effect of wrong results. A system which delivers either correct results, or no results at all, is called "self-checking".

Fault detection is based on the principles of redundancy (mainly to detect hardware faults) and diversity (software faults). Some sort of voting is needed to decide on the correctness of results. Special methods applicable are assertion programming, N-version programming and the safety bag technique and on hardware level by introducing sensors, control loops, error checking codes, etc.

Fault detection may be achieved by checks in the value domain or in the time domain on different levels, especially on the physical (temperature, voltage, etc.), logical (error detecting codes), functional (assertions) or external level (plausibility checks). The results of these checks may be stored and associated with the data affected to allow failure tracking.

Complex systems are composed of subsystems. The efficiency of fault detection, diagnosis and fault compensation depends on the complexity of the interactions among the subsystems, which influences the propagation of faults.

Fault diagnosis isolates the smallest subsystem that may be identified. Smaller subsystems allow a more detailed diagnosis of faults (identification of erroneous states).

D.27 Finite State Machines/State Transition Diagrams

Aim

To define or implement the control structure of a system.

Description

Many systems can be defined in terms of their states, their inputs, and their actions. Thus when in state S1, on receiving input I a system might carry out action A and move to state S2. By defining a system's actions for every input in every state we can define a system completely. The resulting model of the system is called a Finite State Machine (FSM). It is often drawn as a so-called state transition diagram showing how the system moves from one state to another, or as a matrix in which the dimensions are state and input and the matrix cells contain the action and new state resulting from receipt of the input in the given state.

Where a system is complicated or has a natural structure this can be reflected in a layered Finite State Machine.

A specification or design expressed as an Finite State Machine can be checked for completeness (the system shall have an action and new state for every input in every state), for consistency (only one state change is defined for each state/input pair) and reachability (whether or not it is possible to get from one state to another by any sequence of inputs). These are important properties for critical systems and they can be checked. Tools to support these checks are easily written. Algorithms also exist that allow the automatic generation of test cases for verifying a Finite State Machine implementation or for animating a Finite State Machine model.

Several extensions of basic FSMs have been devised to improve the description of complex system behaviour. So called statecharts add hierarchy, composition (parallelism), inter-level transitions, history states, etc. A particularly useful feature is the nesting of internal states and transitions, giving the possibility to reveal or conceal the internal states at need. Statecharts are part of UML (Unified Modeling Language), and as a result supported by many commercial tools.

D.28 Formal Methods

D.28.1 General

Aim

"Formal Methods" refer to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems.

Description

"Mathematically rigorous" means that the specifications used in formal methods are well-formed statements in a mathematical logic and that the formal verifications are rigorous deductions in that logic (i.e. each step follows from a rule of inference and hence can be checked by a mechanical process.) The value of formal methods is that they provide a means to symbolically examine the entire state space of a digital design (whether hardware or software) and establish a correctness or safety property that is true for all possible inputs. However, this is rarely done in practice today (except for the critical components of safety critical systems) because of the enormous complexity of real systems.

Several approaches are used to overcome the astronomically-sized state spaces associated with real systems:

– apply formal methods to requirements and high-level designs where most of the details are abstracted away;
– apply formal methods to only the most critical components;
– analyse models of software and hardware where variables are made discrete and ranges drastically reduced;
– analyse system models in a hierarchical manner that enables "divide and conquer";
– automate as much of the verification as possible.

Although the use of mathematical logic is a unifying theme across the discipline of formal methods, there is no single best "formal method". Each application domain requires different modelling methods and different proof approaches. Furthermore, even within a particular application domain, different phases of the life-cycle may be best served by different tools and techniques. For example a theorem prover might be best used to analyse the correctness of a register transfer level description of a Fast Fourier Transform circuit, whereas algebraic derivational methods might best be used to analyse the correctness of the design refinements into a gate-level design. Therefore there are a large number of formal methods under development throughout the world.

Several examples of Formal Methods are described in the following subclauses of this bibliography. The list of examples here is not exhaustive. The Formal Methods described are CSP, CCS, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z Method, B Method and Model Checking.

D.28.2 CSP – Communicating Sequential Processes

Aim

CSP is a technique for the specification of concurrent software systems, i.e. systems of communicating processes operating concurrently.

Description

CSP provides a language for the specification of systems of processes and proof for verifying that the implementation of processes satisfies their specifications (described as a trace – permissible sequences of events).

A system is modelled as a network of independent processes. Each process is described in terms of all of its possible behaviours. A system is modelled by composing processes sequentially or in parallel. Processes can communicate (synchronise or exchange data) via channels, the communication only taking place when both processes are ready. The relative timing of events can be modelled.

The theory behind CSP was directly incorporated into the architecture of the Inmos transputer[1)], and the occam language [2)] allows a CSP-specified system to be directly implemented on a network of transputers.

D.28.3 CCS – Calculus of Communicating Systems

Aim

CCS is a means for describing and reasoning about the behaviour of systems of concurrent, communicating processes.

Description

Similar to CSP, CCS is a mathematical calculus concerned with the behaviour of systems. The system design is modelled as a network of independent processes operating sequentially or in parallel. Processes can communicate via ports (similar to CSP's channels), the communication is only taking place when both processes are ready. Non-determinism can be modelled. Starting from a high-level abstract description of the entire system (known as a trace), it is possible to carry out a step-wise refinement of the system into a composition of communicating processes whose total behaviour is that required of the whole system. Equally, it is possible to work in a bottom up fashion, combining processes and deducing the properties of the resulting system using inference rules related to the composition rules.

D.28.4 HOL – Higher Order Logic

Aim

This is a formal language intended as a basis for hardware specification and verification.

Description

HOL (Higher Order Logic) refers to a particular logic notation and its machine support system, both of which were developed at the University of Cambridge Computer Laboratory. The logic notation is mostly taken from Church's Simple Theory of Types and the machine support system is based upon the LCF (Logic of Computable Functions) system.

D.28.5 LOTOS

Aim

LOTOS is a means for describing and reasoning about the behaviour of systems of concurrent, communicating processes.

Description

LOTOS (Language for Temporal Ordering Specification) is based on CCS with additional features from the related algebras CSP and CIRCAL (Circuit Calculus). It overcomes the weakness of CCS in the handling of data structures and value expressions by combining it with a second component based on the abstract data type language ACT ONE. The process definition component of LOTOS could, however, be used with other formalisms for the description of abstract data types.

---

1) Inmos was a british semiconductor company which produced in the 80's an innovative microprocessor architecture intended for parallel processing, called the transputer. Later Inmos became part of SGS-Thomson then STMicroelectronics.

2) occam is a concurrent programming language that is named after William of Ockham of Occam's Razor fame. It is the native programming language of the Inmos transputer.

D.28.6 OBJ

Aim

To provide a precise system specification with user feed-back and system validation prior to implementation.

Description

OBJ is an algebraic specification language. Users specify requirements in terms of algebraic equations. The behavioural, or constructive, aspects of the system are specified in terms of operations acting on abstract data types (ADT). An ADT is like an Ada [3)] package where the operator behaviour is visible whilst the implementation details are 'hidden'.

An OBJ specification, and subsequent step-wise implementation, is amenable to the same formal proof techniques as other formal approaches. Moreover, since the constructive aspects of the OBJ specification are machine-executable, it is straightforward to achieve system validation from the specification itself. Execution is essentially the evaluation of a function by equation substitution (re-writing) which continues until specific output value is obtained. This executability allows end-users of the envisaged system to gain a 'view' of the eventual system at the system specification stage without the need to be familiar with the underlying formal specification techniques.

As with all other ADT techniques, OBJ is only applicable to sequential systems, or to sequential aspects of concurrent systems. OBJ has been widely used for the specification of both small and large-scale industrial applications.

D.28.7 Temporal logic

Aim

Direct expression of safety and operational requirements and formal demonstration that these properties are preserved in the subsequent development steps.

Description

Standard First Order Predicate Logic contains no concept of time. Temporal logic extends First Order logic by adding modal operators (e.g. 'Henceforth' and 'Eventually'). These operators can be used to qualify assertions about the system. For example, safety properties might be required to hold 'henceforth', whilst other desired system states might be required to be attained 'eventually' from some other initiating state. Temporal formulas are interpreted on sequences of states (behaviours). What constitutes a 'state' depends on the chosen level of description. It can refer to the whole system, a system component or the computer program. Quantified time intervals and constraints are not handled explicitly in temporal logic. Absolute timing has to be handled by creating additional time states as part of the state definition.

D.28.8 VDM – Vienna Development Method

Aim

The systematic specification and implementation of sequential programs.

3) Ada is a structured, statically typed, imperative, wide-spectrum, and object-oriented high-level computer programming language, extended from Pascal and other languages.

Description

VDM is a mathematically based specification technique and a technique for refining implementations in a way that allows proof of their correctness with respect to the specification.

The specification technique is model-based in that the system state is modelled in terms of set-theoretic structures on which are defined invariants (predicates), and operations on that state are modelled by specifying their pre and post-conditions in terms of the system state. Operations can be proved to preserve the system invariants.

The implementation of the specification is done by the reification of the system state in terms of data structures in the target language and by refinement of the operations in terms of a program in the target language. Reification and refinement steps give rise to proof obligations that establish their correctness. Whether or not these obligations are carried out is a choice made by the designer.

VDM is principally used in the specification stage but can be used in the design and implementation stages leading to source code. It can only be applied to sequential programs or the sequential processes in concurrent systems.

D.28.9 Z method

Aim

Z is a specification language notation for sequential systems and a design technique that allows the designer to proceed from a Z specification to executable algorithms in a way that allows proof of their correctness with respect to the specification.

Z is principally used in the specification stage but a method has been devised to go from specification into a design and an implementation. It is best suited to the development of data oriented, sequential systems.

Description

Like VDM, the specification technique is model-based in that the system state is modelled in terms of set-theoretic structures on which are defined invariants (predicates), and operations on that state are modelled by specifying their pre and post-conditions in terms of the system state. Operations can be proved to preserve the system invariants thereby demonstrating their consistency. The formal part of a specification is divided into schemas which allow the structuring of specifications through refinement.

Typically, a Z specification is a mixture of formal Z and informal explanatory text in natural language. (Formal text on its own can be too terse for easy reading and often its purpose needs to be explained, while the informal natural language can easily become vague and imprecise).

Unlike VDM, Z is a notation rather than a complete method. However an associated method (called B) has been developed which can be used in conjunction with Z.

D.28.10 B method

Aim

Like VDM, the purpose of B method is to model formally a system or software and to prove that the behaviour of the system or software respects the properties that were made explicit during modelling.

Description

The B modelling calls on mathematical items from the Set theory. On one hand, invariants (i.e. predicates) define the static properties of the model. On the other hand, operations establish post-conditions, thus defining its dynamic behaviour. The specification of a complex system or software is made possible by decomposing the model into “machines” tied together by links of different semantics.

Two main categories of modelling with B formalism can be distinguished.

– The former (historically the first), aims at developing software: in this case, the goal is to produce a program that respects its specification. The model consists of abstract machines (not necessarily deterministic) and step-by-step refinements of these machines, leading to deterministic implementations written in a pseudo-code called “B0”. This pseudo-code can then be automatically translated into a target programming language.
– The latter, aims at modelling systems and in this case we talk about “Event B”: the purpose is to specify, without ambiguity and coherently, a system that fulfils explicit properties. The model takes into account the system itself and its environment. The dynamics of the system is modelled by “events”, and the refinement technique is used in order to precise interactions between the system and its environment.

A set of Proof Obligations (logical assertions that are to be formally proved from the hypothesis that were extracted from the B formal model) is generated automatically. These Proof Obligations guarantee

– the existence of data that fulfil the static and dynamic properties of the model,
– that the operations (dynamic behaviour of the model) respect the invariant,
– that the refinement of data and operations (and the B0 pseudo-code if necessary) does not contradict the specification written in abstract machines,
– that each operation is called within the context of its pre-condition,
– in the case of software modelling, that the program does terminate (in particular, that each loop terminates).

Other Proof Obligations, for example verifying integer overflow or underflow, are also generated.

D.28.11 Model Checking

Aim

Given a model of a system, test automatically whether this model meets a given specification.

Description

Model checking is the process of checking whether a given structure is a model of a given logical formula. The concept is general and applies to all kinds of logics and suitable structures. A simple model-checking problem is testing whether a given formula in the propositional logic is satisfied by a given structure.

An important class of model checking methods have been developed to algorithmically verify formal systems. This is achieved by verifying if the structure, often derived from a hardware or software design, satisfies a formal specification, typically a temporal logic formula.

Model checking is most often applied to hardware designs. For software, because of undecidability (see Computability theory) the approach cannot be fully algorithmic; typically it may fail to prove or disprove a given property.

The structure is usually given as a source code description in an industrial hardware description language or a special-purpose language. Such a program corresponds to a finite

state machine, i.e., a directed graph consisting of nodes (or vertices) and edges. A set of atomic propositions is associated with each node, typically stating which memory elements are one. The nodes represent states of a system, the edges represent possible transitions which may alter the state, while the atomic propositions represent the basic properties that hold at a point of execution.

Formally, the problem can be stated as follows: given a desired property, expressed as a temporal logic formula p, and a structure M with initial state s, decide if. If M is finite, as it is in hardware, model checking reduces to a graph search.

D.29 Formal Proof

Aim

Using theoretical and mathematical models and rules it is possible to prove the correctness of a program or model without executing it.

Description

A number of assertions are stated at various locations in the program, and they are used as pre and post conditions to various paths in the program. The proof consists of showing that the program transfers the preconditions into the post-conditions according to a set of logical rules, and that the program terminates.

Several Formal Methods are described in this annex, for instance, CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z. The descriptions of these can be found under Clause D.28.

D.30 Forward Recovery

Aim

To provide correct functional operation in the presence of one or more faults.

Description

If a fault has been detected, the current state of the system is manipulated to obtain a state, which will be consistent some time later. This concept is especially suited for real-time systems with a small database and fast rate of change of the internal state. It is assumed that at least part of the system state may be imposed onto the environment, and only part of the system states are influenced (forced) by the environment.

D.31 Graceful Degradation

Aim

To maintain the more critical system functions available despite failures by dropping the less critical functions.

Description

This technique gives priorities to the various functions to be carried out by the system. The design then ensures that should there be insufficient resources to carry out all the system functions, then the higher priority functions are carried out in preference to the lower ones. For example, error and event logging functions may have lower priority than system control functions, in which case system control would continue if the hardware associated with error logging were to fail.

Another example would be a signalling system where in the event of loss of communication with the control centre the local lineside equipment automatically sets the available routes for the direction taken by the highest priority traffic. This would be a graceful degradation because trains on the priority routes would be able to pass through the area affected by the loss of communication with the control centre, but other movements, such as shunting movements, would not be possible.

D.32 Impact Analysis

Aim

To identify the effect that a change or an enhancement to a software will have to other components in that software as well as to other systems.

Description

Prior to a modification or enhancement being performed on the software an analysis shall be undertaken to identify the impact of the modification or enhancement on the software and to also identify the affected software systems and components.

After the analysis has been completed a decision is required concerning the reverification of the software system. This depends on the number of components affected, the criticality of the affected components and the nature of the change. The possible decisions are:

a) only the changed components to be reverified;

b) all identified affected components are reverified; and

c) the complete system is reverified.

D.33 Information Hiding / Encapsulation

Aim

To increase the robustness and maintainability of software.

Description

Data that is globally accessible to all software components can be accidentally or incorrectly modified by any of these components. Any changes to these data structures may require detailed examination of the code and extensive modifications.

Information hiding is a general approach for minimising these difficulties. The key data structures are 'hidden' and can only be manipulated through a defined set of access procedures. This allows the internal structures to be modified or further procedures to be added without affecting the functional behaviour of the remaining software. For example, a named directory might have access procedures Insert, Delete and Find. The access procedures and internal data structures could be re-written (e.g. to use a different look-up method or to store the names on a hard disk) without affecting the logical behaviour of the remaining software using these procedures.

This concept of an abstract data type is directly supported in a number of programming languages, but the basic principle can be applied whatever programming language is used.

D.34 Interface Testing

Aim

To demonstrate that interfaces of subprograms do not contain any errors or any errors that lead to failures in a particular application of the software or to detect all errors that may be relevant.

Description

Several levels of detail or completeness of testing are feasible. The most important levels are testing

- all interface variables at their extreme positions,
- all interface variable individually at their extreme values with other interface variables at normal values,
- all values of the domain of each interface variable with other interface variables at normal values,
- all values of all variables in combination (this may only be feasible for small interfaces),
- the specified test conditions relevant to each call of each subroutine.

These tests are particularly important if their interfaces do not contain assertions that detect incorrect parameter values. They are also important after new configurations of pre-existing subroutines have been generated.

D.35 Language Subset

Aim

To reduce the probability of introducing programming faults and increase the probability of detecting any remaining faults.

Description

The language is examined to identify programming constructs which are either error-prone or difficult to analyse, for example, using static analysis methods. A language subset is then defined which excludes these constructs.

D.36 Memorising Executed Cases

Aim

To force the software to fail safe if it executes an unlicensed path.

Description

During licensing a record is made of all relevant details of each program execution. During normal operation each program execution is compared with the set of the licensed executions. If it differs, a safety action is taken.

The execution record can be the sequence of the individual decision-to-decision paths (DDpaths) or the sequence of the individual accesses to arrays, records or volumes, or both.

Different methods of storing execution paths are possible. Hash-coding methods can be used to map the execution sequence onto a single large number or sequence of numbers. During normal operation the execution path value shall be checked against the stored cases before any output operation occurs.

Since the possible combinations of decision-to-decision paths during one program is very large, it may not be feasible to treat programs as a whole. In this case, the technique can be applied at component level.

D.37 Metrics

Aim

To predict the attributes of programs from properties of the software itself rather than from its development or test history.

Description

These models evaluate some structural properties of the software and relate this to a desired attribute such as complexity. Software tools are required to evaluate most of the measures. Some of the metrics which can be applied are given below:

– Graph Theoretic Complexity: this measure can be applied early in the lifecycle to assess trade-offs, and is based on the complexity of the program control graph, represented by its cyclomatic number;

– number of ways to activate a certain component (accessibility): the more a component can be accessed, the more likely it is to be debugged;

– Halstead complexity measures: this measure computes the program length by counting the number of operators and operands. It provides a measure of complexity and estimates development resources;

– number of entries and exits per component: minimising the number of entry/exit points is a key feature of structured design and programming techniques.

D.38 Modular Approach

Aim

Decomposition of a software into small comprehensible parts in order to limit the complexity of the software.

Description

A Modular Approach or modularisation contains several rules for the design, coding and maintenance phases of a software project. These rules vary according to the design method employed during design. Most methods contain the following rules:

– a module/component shall have a single well defined task or function to fulfil;

– connections between modules/components shall be limited and strictly defined, coherence in one module/component shall be strong;

– collections of subprograms shall be built providing several levels of modules/components;

– subprograms shall have a single entry and a single exit only;

– modules/components shall communicate with other modules/components via their interfaces. Where global or common variables are used they shall be well structured, access shall be controlled and their use shall be justified in each instance;

– all module/component interfaces shall be fully documented;

– any modules/components interface shall contain the minimum number of parameters necessary for the modules/components function; and

– a suitable restriction of parameter number shall be specified, typically 5.

D.39 Performance Modelling

Aim

To ensure that the working capacity of the system is sufficient to meet the specified requirements.

Description

The requirements specification includes throughput and response requirements for specific functions, perhaps combined with constraints on the use of total system resources. The proposed system design is compared against the stated requirements by

– defining a model of the system processes, and their interactions,
– identifying the use of resources by each process, for example, processor time, communications bandwidth, storage devices, etc.),
– identifying the distribution of demands placed upon the system under average and worst-case conditions,
– computing the mean and worst-case throughput and response times for the individual system functions.

For simple systems, an analytic solution may be possible whilst for more complex systems, some form of simulation is required to obtain accurate results.

Before detailed modelling, a simpler 'resource budget' check can be used which sums the resources requirements of all the processes. If the requirements exceed designed system capacity, the design is infeasible. Even if the design passes this check, performance modelling may show that excessive delays and response times occur due to resource starvation. To avoid this situation engineers often design systems to use some fraction (e.g. 50 %) of the total resources so that the probability of resource starvation is reduced.

D.40 Performance Requirements

Aim

To establish that the performance requirements of a software have been satisfied.

Description

An analysis is performed of both the system and the Software Requirements Specifications to identify all general and specific, explicit and implicit performance requirements.

Each performance requirement is examined in turn to determine

– the success criteria to be obtained,
– whether a measure against the success criteria can be obtained,
– the potential accuracy of such measurements,
– the project stages at which the measurements can be estimated, and
– the project stages at which the measurements can be made.

The practicability of each performance requirement is then analysed in order to obtain a list of performance requirements, success criteria and potential measurements. The main objectives are:

a) each performance requirement is associated with at least one measurement;
b) where possible, accurate and efficient measurements are selected which can be used as early in the development process as possible;

c) essential and optional performance requirements and success criteria are identified, and

d) where possible, advantage shall be taken of the possibility of using a single measurement for more than one performance requirement.

D.41 Probabilistic Testing

Aim

To gain a quantitative figure about the reliability properties of the investigated software. This figure may address the related levels of confidence and significance and

a) a failure probability per demand,

b) a failure probability during a certain period of time, and

c) a probability of error containment.

From these figures other parameters may be derived such as

– probability of failure free execution,

– probability of survival,

– availability,

– MTBF or failure rate, and

– probability of safe execution.

Description

Probabilistic considerations are either based on a probabilistic test or on operating experience. Usually the number of tests cases of observed operating cases is very large.

In order to facilitate testing, usually automatic aids are taken. They concern the details of test data provision and test output supervision. Large tests are run on large host computers with the appropriate process simulation periphery. Test data is selected both according to systematic and random view points. The first concerns the overall test control, for example, guarantee a test data profile. The random selection takes the individual test cases in detail.

Individual test harnesses, test executions and test supervisions are determined by the detailed test aims as described above. Other important conditions are given through the mathematical prerequisites to be fulfilled in order to enable the test evaluation in view of the intended test aim.

Probabilistic figures about the behaviour of any test object may also be derived from operating experience. Provided the same conditions are met, the same mathematics can be applied as for the evaluation of test results.

D.42 Process Simulation

Aim

To test the function of a software, together with its interface to the outside world, without allowing it to modify the real world in any way.

Description

The creation of a system, for testing purposes only, which mimics the behaviour of the system to be controlled by the system under test.

The simulation may be software only or a combination of software and hardware. It shall

– provide all the inputs of the system under test which will exist when the system is installed,
– respond to outputs from the system in a way which faithfully represents the controlled equipment,
– have provision for operator inputs to provide any perturbations with which the system under test is required to cope.

When software is being tested, the simulation may be a simulation of the target hardware with its inputs and outputs.

D.43 Prototyping / Animation

Aim

To check the feasibility of implementing the system against the given constraints. To communicate the specifier's interpretation of the system to the customer, in order to locate misunderstandings.

Description

A sub-set of system functions, constraints, and performance requirements are selected. A prototype is built using high level tools. At this stage, constraints such as the target computer, implementation language, program size, maintainability and robustness need not be considered. The prototype is evaluated against the customer's criteria and the system requirements may be modified in the light of this evaluation.

D.44 Recovery Block

Aim

To increase the likelihood of the program performing its intended function.

Description

Several different program sections are written, often independently, each of which is intended to perform the same desired function. The final program is constructed from these sections. The first section, called the primary, is executed first. This is followed by an acceptance test of the result it calculates. If the test is passed then the result is accepted and passed on to subsequent parts of the system. If it fails, any side effects of the first are reset and the second section, called the first alternative, is executed. This too is followed by an acceptance test and is treated as in the first case. A second, third or even more alternatives can be provided if desired.

D.45 Response Timing and Memory Constraints

Aim

To ensure that the system will meet its temporal and memory requirements.

Description

The requirements specification for the system and the software includes memory and response requirements for specific functions, perhaps combined with constraints on the use of total system resources. An analysis is performed which will identify the distribution demands under average and worst case conditions. This analysis requires estimates of the resource usage and elapsed time of each system function. These estimates can be obtained in several

ways, for example, comparison with an existing system or the prototyping and bench-marking of time critical systems.

D.46 Re-Try Fault Recovery Mechanisms

Aim

To attempt functional recovery from a detected fault condition by re-try mechanisms.

Description

In the event of a detected fault or error condition, attempts are made to recover the situation by re-executing the same code. Recovery by re-try can be as complete as a re-boot and a re-start procedure or a small re-scheduling and re-starting task, after a software time-out or a task watchdog action. Re-try techniques are commonly used in communication fault or error recovery and re-try conditions could be flagged from a communication protocol error (check sum, etc.) or from a communication acknowledgement response time-out.

D.47 Safety Bag

Aim

To protect against residual specification and implementation faults in software which adversely affect safety.

Description

A safety bag is an external monitor, implemented on an independent computer to a different specification. This safety bag is solely concerned with ensuring the main computer performs safe, not necessarily correct, actions. The safety bag continuously monitors the main computer. The safety bag prevents the system from entering an unsafe state. In addition if it detects that the main computer is entering a potentially hazardous state, the system has to be brought back to a safe state either by the safety bag or the main computer.

D.48 Software Configuration Management

Aim

Software Configuration management aims to ensure the consistency of groups of development deliverables as those deliverables change. Configuration Management, in general, applies to both hardware and software development.

Description

Software Configuration Management is a technique used throughout development. In essence, it requires the recording of the production of every version of every "significant" deliverable and of every relationship between different versions of the different deliverables. The resulting records allow the designer to determine the effect on other deliverables of a change to one deliverable. In particular, systems or subsystems can be reliably re-built from consistent sets of component versions.

D.49 Strongly Typed Programming Languages

Aim

Reduce the probability of faults by using a language which permits a high level of checking by the compiler.

Description

Such languages usually allow user-defined data types to be defined from the basic language data types (such as INTEGER, REAL). These types can then be used in exactly the same way as the basic types, but strict checks are imposed to ensure the correct type is used. These checks are imposed over the whole program, even if this is built from separately compiled units. The checks also ensure that the number and the type of procedure arguments match even when referenced from separately compiled components.

Strongly typed languages also support other aspects of good software engineering practice such as easily analysable control structures (e.g. IF ... THEN ... ELSE, DO ... WHILE, etc.) which lead to well-structured programs.

Typical examples of strongly typed languages are Pascal, Ada and Modula-2.

D.50 Structure Based Testing

Aim

To apply tests which exercise certain subsets of the program structure.

Description

Based on an analysis of the program a set of input data is chosen such that a large fraction of selected program elements are exercised. The program elements exercised can vary depending on the level of rigour required:

– Statements: this is the least rigorous test since it is possible to execute all code statements without exercising both branches of a conditional statement.

– Branches: both sides of every branch should be checked. This may be impractical for some types of defensive code.

– Compound conditions: every condition in a compound conditional branch (i.e. linked by AND/OR) is exercised.

– LCSAJ (Linear Code Sequence And Jump): a linear code sequence and jump is any linear sequence of code statements including conditional jumps terminated by a jump. Many potential sub-paths will be infeasible due to constraints on the input data imposed by the execution of earlier code.

– Data flow: the execution paths are selected on the basis of data usage for example a path where the same variable is both written and read.

– Call graph: a program is composed of subroutines which may be invoked from other subroutines. The call graph is the tree of subroutine invocations in the program. Tests are designed to cover all invocations in the tree.

– Entire path: execute all possible paths through the code. Complete testing is normally infeasible due to the very large number of potential paths.

D.51 Structure Diagrams

Aim

To show the structure of a program diagrammatically.

Description

Structure Diagrams are a notation which complements Data Flow Diagrams. They describe the programming system and a hierarchy of parts and display this graphically, as a tree. They document how elements of a data flow diagram can be implemented as a hierarchy of program units.

A structure chart shows relationships between program units without including any information about the order of activation of these units. It is drawn using the following three symbols:

a) a rectangle annotated with the name of the unit;

b) an arrow connecting these rectangles;

c) a circled arrow, annotated with the name of data passed to and from elements in the structure chart. Normally, the circled arrow is drawn parallel to the arrow connecting the rectangles in the chart.

From any non trivial data flow diagram, it is possible to derive a number of different structure charts.

Structure charts derived from data flow diagrams represent a first level structure of the system, where each box on the structure chart represents a bubble in the data flow diagram. Naturally, deeper levels can be described using the same technique.

D.52 Structured Methodology

Aim

The main aim of Structured Methodologies is to promote the quality of software development by focusing attention on the early parts of the life-cycle. The methods aim to achieve this through both precise and intuitive procedures and notations (assisted by computers) to identify the existence of requirements and implementation features in a logical order and a structured manner.

Description

A range of Structured Methodologies exist. Some such as SSADM, LBMS are designed for traditional data-processing and transaction processing functions, while others (MASCOT, JSD, real-time Yourdon) are more oriented to process-control and real-time applications (which tend to be more safety-critical).

Structured Methods are essentially "thought tools" for systematically perceiving and partitioning a problem or system. Their main features are

– a logical order of thought, breaking a large problem into manageable stages,

– identification of total system, including the environment as well as the required system,

– decomposition of data and function in the required system,

– checklists, i.e. lists of the sort of things that need definition,

– low intellectual overhead – simple, intuitive, pragmatic.

The supporting notations tend to be precise for identifying problem and system entities (e.g. processes and data flows), but the processing functions performed by these entities tend to be expressed using informal notations. However some methods do make partial use of (mathematically) formal notations (for example JSD makes use of regular expressions: Yourdon, SOM and SDL utilise finite state machines). This precision not only reduces the scope for misunderstanding, it provides scope for automatic processing.

Another benefit of structured notation is their visibility, enabling a specification or design to be checked intuitively by a user, against his powerful but unstated knowledge.

D.53 Structured Programming

Aim

To design and implement the software component in a way which makes practical the analysis of the software component. This analysis should be capable of discovering all significant component behaviour.

Description

The software component should contain the minimum of structural complexity. Complicated branching should be avoided. Loop constraints and branching should (where possible) be simply related to input parameters. The software component should be divided into appropriately small modules, and the interaction of these modules should be explicit. Features of the programming language which encourage the above approach should be used in preference to other features which are (allegedly) more efficient, except where efficiency takes absolute priority (e.g. some safety-critical systems).

D.54 Suitable Programming languages

Aim

To support the requirements of this Standard as much as possible, in particular, defensive programming, strong typing, structured programming and possibly assertions. The programming language chosen should lead to easily verifiable code with a minimum of effort and facilitate program development, verification and maintenance.

Description

The language should be fully and unambiguously defined. The language should be user or problem oriented rather than machine oriented. Widely used languages or their subsets are preferred to special purpose languages.

In addition to the already referenced features the language should provide for

– block structure,
– translation time checking,
– run time type and array bound checking, and
– parameter checking.

The language should encourage

– the use of small and manageable components,
– restriction of access to data in defined components,
– definition of variable sub-ranges, and
– any other type of error limiting constructs.

It is desirable that the language is supported by a suitable translator, appropriate libraries of pre-existing components, a debugger and tools for both version control and development.

Features which make verification difficult and therefore should be avoided are:

– unconditional jumps excluding subroutine calls;
– recursion;
– pointers, heaps or any type of dynamic variables or objects;
– interrupt handling at source code level;
– multiple entries or exits of loops, blocks or subprograms;

– implicit variable initialisation or declaration;
– variant records and equivalence; and
– procedural parameters.

Low level languages, in particular assembly languages, present problems due to their machine oriented nature.

D.55 Time Petri Nets

Aim

To model relevant aspects of the system behaviour and to assess and possibly improve safety and operational requirements through analysis and re-design.

Description

Petri nets belong to a class of graph theoretic models which are suitable for representing information and control flow in systems exhibiting concurrency and asynchronous behaviour.

A Petri net is a network of places and transitions. The places may be 'marked' or 'unmarked'. A transition is 'enabled' when all the input places to it are marked. When enabled, it is permitted (but not obliged) to 'fire'. If it fires, the input marks are removed, and each output place from the transition is marked instead.

The potential hazards are represented as particular states (markings) in the model. Extended Petri nets allow timing features of the system to be modelled. Although 'classical' Petri nets concentrate on control flow aspects, several extensions have been proposed to incorporate dataflow into the model.

D.56 Walkthroughs / Design Reviews

Aim

To detect errors in some product of the development process as soon and as economically as possible.

Description

IEC have published a Guide on Formal Design Reviews, IEC 61160, which includes a general description of formal design reviews, their objectives, details of the various design review types, the composition of a design review team and their associated duties and responsibilities. IEC 61160 also provides general guidelines for planning and conducting formal design reviews, as well as specific details concerning the role of independent specialists within a design review team.

IEC 61160 recommends that a "formal design review shall be conducted for all new products/processes, new applications, and revisions to existing products and manufacturing processes which affect the function, performance, safety, reliability, ability to inspect maintainability, availability, ability to cost, and other characteristics affecting the end product/process, users or bystanders".

A code walkthrough consists of a walkthrough team selecting a small set of paper test cases, representative sets of inputs and corresponding expected outputs for the program. The test data is then manually traced through the logic of the program.

D.57 Object Oriented Programming

Aim

To enable rapid prototyping, to more easily reuse existing software components, to achieve information hiding, to reduce the likelihood of errors during the whole lifecycle, to reduce the necessary effort during the maintenance phase, to break down complex problems into more easily manageable small problems, to reduce the dependencies between software components, to create more easily extendible applications.

Description

Object oriented programming is a fundamentally new way of thinking about software based on abstractions that exist in the real world rather than based on computational abstractions. Object oriented programming organises software as a collection of objects that incorporate both data structure and behaviour. This is in contrast to conventional programming where data structure and behaviour are only loosely connected.

Object: an object consists of a private data area and set of operations – so called methods – on that object. Methods may be public or private. No other software component is allowed to read or change the private data of an object directly. Every other software component has to use the public methods on that object to read or write data in the private data area of an object.

Object Class: by specifying an object class (often in the form of a type definition) you enable the instantiation of numerous objects of the same class, i.e., all instantiations have the private data area and the methods defined in the object class.

(Multiple) Inheritance: an object class can inherit the private data area and the methods of one (or more) superclasses (object classes above it in the class hierarchy) with being allowed to add some private data, to add some methods or to modify the implementations of the inherited methods. Using Inheritance multiple object class trees can be built.

Polymorphism: the same operation may behave differently on different object classes, e.g. the write operation for a terminal object writes characters to that terminal and a write operation to a file object writes characters to that file.

Drawback: Object oriented programming languages may lead to an additional need for resources with a negative impact on system performance.

D.58 Traceability

Aim

The objective of traceability is to ensure that all requirements can be shown to have been properly met and that no untraceable material has been introduced.

Description

Traceability to requirements shall be an important consideration in the validation of a system and means shall be provided to allow this to be demonstrated throughout all phases of the lifecycle.

Traceability shall be considered applicable to both functional and non-functional requirements and shall particularly address

a) traceability of requirements to the design or other objects which fulfil them,

b) traceability of design objects to the implementation objects which instantiate them,

c) traceability of requirements and design objects to the operational and maintenance objects required to be applied in the safe and proper use of the system,

d) traceability of requirements, design, implementation, operation and maintenance objects, to the verification and test plans and specifications which will determine their acceptability,

e) traceability of verification and test plans and specifications to the test or other reports which record the results of their application.

Where requirements, design or other objects are instantiated as a number of separate documents, traceability shall be maintained within the document structures and in a hierarchical manner.

The output of the traceability process shall be the subject of formal Configuration Management.

D.59 Metaprogramming

Aim

Metaprogramming allows programmers to get more done in the same amount of time as they would take to write all the code manually.

Description

Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data or that do part of the work during compilation time that is otherwise done at run time.

The language in which the metaprogram is written is called the metalanguage. The language of the programs that are manipulated is called the object language. The ability of a programming language to be its own metalanguage is called reflection or reflexivity.

Reflection is a valuable language feature to facilitate metaprogramming. Having the programming language itself as a first-class data type (as in Lisp) is also very useful. Generic programming invokes a metaprogramming facility within a language, in those languages supporting it.

Metaprogramming usually works through one of two ways. The first way is to expose the internals of the run-time engine to the programming code through application programming interfaces (APIs). The second approach is dynamic execution of string expressions that contain programming commands. Thus, "programs can write programs". Although both approaches can be used, most languages tend to lean toward one or the other.

D.60 Procedural programming

Aim

Specifying the steps the program shall take to reach the desired state.

Description

Procedural programming based upon the concept of the procedure call. Procedures, also known as routines, subroutines, methods, or functions (not to be confused with mathematical functions, but similar to those used in functional programming) simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

D.61 Sequential Function Charts

Aim

Describing program algorithms in a diagrammatic way.

Description

The SFC elements allow partitioning a unit of application algorithms into a set of steps and transitions interconnected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition. Since SFC elements require storage of state information, the only units of application algorithms which can be structured using these elements are function blocks.

See IEC 61131-3:2013, 6.7.

D.62 Ladder Diagram

Aim

Describing a program in a diagrammatic way.

Description

See IEC 61131-3:2013, 8.2.

D.63 Functional Block Diagram

Aim

Describing a function between input variables and output variables in a diagrammatic way.

Description

See IEC 61131-3:2013, 8.3.

D.64 State Chart or State Diagram

Aim

Describing the behaviour of a system in a diagrammatic way.

Description

State Chart or State diagrams are used to describe the behaviour of a system. State diagrams can describe the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

State diagram can be used to graphically represent finite state machines. This was introduced by Taylor Booth in his 1967 book "Sequential Machines and Automata Theory". Another possible representation is the State transition table.

A classic form of a state diagram for a finite state machine is a directed graph.

D.65 Data modelling

Aim

Creating a data model.

Description

Data modelling in computer science is the process of creating a data model by applying formal data model descriptions using data modelling techniques.

A data model in software engineering is an abstract model that describes how data is represented and accessed. Data models formally define data objects and relationships among data objects for a domain of interest. Some typical applications of database models include supporting the development of databases and enabling the exchange of data for a particular area of interest. Data models are specified in a data modelling language.

D.66 Control Flow Diagram/Control Flow Graph

Aim

Describing the behaviour of a system in a diagrammatic way.

Description

In computer science, a control flow diagram or a control flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. Each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow. There are, in most presentations, two specially designated blocks: the entry block, through which control enters into the flow graph, and the exit block, through which all control flow leaves.

The CFG is essential to many compiler optimisations and static analysis tools.

Reachability is another graph property useful in optimisation. If a block/subgraph is not connected from the subgraph containing the entry block, that block is unreachable during any execution, and so is unreachable code; it can be safely removed. If the exit block is unreachable from the entry block, it indicates an infinite loop. Again, dead code and some infinite loops are possible even if the programmer did not explicitly code that way: optimisations like constant propagation and constant folding followed by jump threading could collapse multiple basic blocks into one, cause edges to be removed from a CFG, etc., thus possibly disconnecting parts of the graph.

terminology
these terms are commonly used when discussing control flow graphs

entry block
block through which all control flow enters the graph

exit block
block through which all control flow leaves the graph

back edge
edge that points to an ancestor in a depth-first (DFS) traversal of the graph

critical edge
edge which is neither the only edge leaving its source block, nor the only edge entering its destination block. These edges should be split (a new block should be created in the middle of the edge) in order to insert computations on the edge

abnormal edge
edge whose destination is unknown. These edges tend to inhibit optimisation. Exception handling constructs can produce them

impossible edge
(also known as a fake edge) edge which has been added to the graph solely to preserve the property that the exit block postdominates all blocks. It cannot ever be traversed

dominator
block M dominates block N if every path from the entry that reaches block N has to pass through block M. The entry block dominates all blocks

postdominator
block M postdominates block N if every path from N to the exit has to pass through block M. The exit block postdominates all blocks

immediate dominator
block M immediately dominates block N if M dominates N, and there is no intervening block P such that M dominates P and P dominates N. In other words, M is the last dominator on any path from entry to N. Each block has a unique immediate dominator, if it has any at all

immediate postdominator
analogous to immediate dominator

dominator tree
ancillary data structure depicting the dominator relationships. There is an arc from Block M to Block N if M is an immediate dominator of N. This graph is a tree, since each block has a unique immediate dominator. This tree is rooted at the entry block

postdominator tree
analogous to dominator tree. This tree is rooted at the exit block

loop header
sometimes called the entry point of the loop, a dominator that is the target of a loop-forming back edge. Dominates all blocks in the loop body

loop pre-header
suppose block M is a dominator with several incoming edges, some of them being back edges (so M is a loop header). It is advantageous to several optimisation passes to break M up into two blocks Mpre and Mloop. The contents of M and back edges are moved to Mloop, the rest of the edges are moved to point into Mpre, and a new edge from Mpre to Mloop is inserted (so that Mpre is the immediate dominator of Mloop). In the beginning, Mpre would be empty, but passes like loop-invariant code motion could populate it. Mpre is called the loop pre-header, and Mloop would be the loop header

D.67 Sequence diagram

Aim

Describing the interaction between processes or components in a diagrammatic way.

Description

A sequence diagram is a kind of interaction diagram, that shows how processes or components operate one with another and in what order.

D.68 Tabular Specification Methods

Aim

The aim is to provide a standardised and well-structured means of defining the data driven functions of a system.

Description

Tabular notations such as signalling control tables are a well established method of documenting the installation specific requirements for a railway signalling system.

The technique is suitable where the types of relationships between elements of the system are standardised.

Advantage: The format of the table and the possible entries in each field can serve as a checklist during verification.

D.69 Application specific language

Aim

The aim is to provide a means of specifying the functionality of a data-driven system using concepts and terminology which are easily assimilated by applications engineers who may not be familiar with conventional programming languages.

Description

An application specific language typically combines control constructs which are similar to conventional high-level programming languages with operators which are specific to the type of system.

The technique is suitable where Boolean decisions need to be specified, but may also be applicable elsewhere.

Advantage: Flexibility, allowing data to be produced for unusual circumstances which may not have been foreseen when the system was originally designed.

D.70 UML (Unified Modeling Language)

Aim

To represent software programs and related artefacts in a manner that allows complexity reduction by means of abstraction. By allowing modelling of an existing or planned design in terms of a variety of diagram types, UML facilitates assessment of the key characteristics of the design on basis of representations at appropriate levels of detail. UML is frequently used in so-called model-driven development, supported by commercial products. This development style aims at improving the quality of the software and the productivity of the developers by the use of high-level modelling languages.

Description

UML is a standardized general-purpose modelling language, originating from the use of graphically oriented software specification languages and object-oriented programming languages. Building on this tradition, UML reuses many of the concepts and methods of its predecessors. The models are written in terms of one or more diagram types, classified as structure diagrams and behaviour diagrams, the latter also comprising four diagram types classified as interaction diagrams.

Structure diagrams

- Package diagrams: Show the contents of and relationships between different packages, each containing related model elements.
- Class diagrams: Specify object types with their different features and their relationships with other object types, based on an adaption of traditional entity-relationship diagrams.
- Object diagrams: Show how different objects (class instances) are related to each other.
- Composite structure diagrams: Show the internal structure of a classifier (such as a class or component) and its interaction points to other parts of the system.
- Component diagrams: Show the components that compose the system, their interrelationships, interactions and external interfaces.
- Deployment diagrams: Specify how software is distributed across an execution platform.

Behaviour diagrams

- Activity diagrams: Describe algorithmic behaviours, using an adaption of traditional flowcharts that allows modelling of data transfer and concurrent execution.
- State machine diagrams: Describe event-driven behaviour by means of finite state machines (statecharts).
- Use case diagrams: Model actors interacting with the system to achieve specific use cases.
- Interaction diagrams (communication diagrams, interaction overview diagrams, sequence diagrams, timing diagrams): Describe scenarios comprising activities performed by communicating objects.

While UML is a generic modelling language, domain-specific interpretations are made possible by means of profiles. By refining standard UML concepts, profiles make it possible to make such interpretations by using the extensions defined in the profile. In this way, UML is used as a basis for defining domain-specific languages.

D.71 Domain specific languages

Aim

To represent software programs and related artefacts in a language tailored to a particular domain.

Description

A domain specific language (DSL) is a programming, specification or modelling language created specifically to solve problems in a particular application domain or problem domain, or with a particular technique. The language is based on concepts and features relevant to this domain. Domain specific languages are also known as special-purpose languages, in contrast to general-purpose programming languages or modelling languages like Java and UML.

One of the important benefits of domain specific languages is the possibility to represent and solve problems within a particular domain without the need for knowledge about general programming, specification or modelling. As a consequence, programs, specifications or models can be produced at a higher level, possibly by the end-user. By providing constructs tailored to this domain, and possibly means for automated code generation, a DSL generally also increases the productivity of the programmer and the quality of the resulting product. The code generation is typically implemented as an application generator using the DSL as input.

Bibliography

IEC 61131-3:2013, *Programmable controllers – Part 3: Programming languages*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 62280, *Railway applications – Communication, signalling and processing systems – Safety-related communication in transmission systems*

IEC 62425:2007, *Railway applications – Communication, signalling and processing systems – Safety related electronic systems for signalling*

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

APPLICATIONS FERROVIAIRES – SYSTÈMES DE SIGNALISATION, DE TÉLÉCOMMUNICATION ET DE TRAITEMENT – LOGICIELS POUR SYSTÈMES DE COMMANDE ET DE PROTECTION FERROVIAIRE

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62279 a été établie par le comité d'études 9 de l'IEC: Matériels et systèmes électriques ferroviaires.

La présente norme est basée sur l'EN 50128:2011.

Cette deuxième édition annule et remplace la première édition, parue en 2002, dont elle constitue une révision technique.

Les modifications techniques majeures par rapport à l'édition précédente sont les suivantes:

- des exigences relatives à la gestion et à l'organisation, à la définition des rôles et des compétences, au déploiement et à la maintenance des logiciels ont été ajoutées;

- un nouveau paragraphe concernant les outils a été ajouté en 6.7, fondé sur l'IEC 61508-2:2010;
- les tableaux en Annexe A ont été mis à jour;
- une nouvelle Annexe B concernant les principaux rôles et responsabilités relatifs au logiciel a été introduite;
- une nouvelle Annexe C concernant le résumé du contrôle des documents a été introduite;
- l'Annexe B concernant la Bibliographie des techniques a été révisée et mise à jour comme nouvelle Annexe D.

Les modifications majeures par rapport à l'EN 50128:2011 sont énumérées ci-après:

- l'article concernant les outils en 6.7 a été mis à jour;
- l'Annexe B concernant les principaux rôles et responsabilités relatifs au logiciel a été modifiée.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
9/2023/FDIS	9/2046/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Cette norme doit être utilisée conjointement avec l'IEC 62278:2002, *Applications ferroviaires – Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (FDMS)*.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

La présente Norme fait partie intégrante d'un groupe de normes connexes. Les autres sont l'IEC 62278:2002, *Applications ferroviaires – Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (FDMS)* et l'IEC 62425:2007, *Applications ferroviaires – Systèmes de signalisation, de télécommunications et de traitement – Systèmes électroniques de sécurité pour la signalisation*.

L'IEC 62278:2002 traite des systèmes au niveau le plus général, tandis que l'IEC 62425:2007 traite des processus d'approbation des systèmes individuels qui peuvent exister dans le cadre du système ferroviaire global de contrôle-commande et de protection. La présente Norme traite en particulier des méthodes à utiliser pour fournir des logiciels répondant aux exigences d'intégrité de la sécurité imposées par ces considérations plus larges.

La présente Norme fournit un ensemble d'exigences qu'il convient qu'il soit respecté par le développement, le déploiement et la maintenance de tout logiciel de sécurité destiné aux applications ferroviaires de contrôle-commande et de protection. Elle définit les exigences concernant la structure organisationnelle, la relation entre organisations et la répartition des responsabilités impliquées dans les activités de développement, de déploiement et de maintenance. Des critères de qualification et d'expertise du personnel sont également fournis dans la présente Norme.

Le concept-clé de la présente Norme est celui des niveaux d'intégrité de logiciel. La présente Norme traite de cinq niveaux d'intégrité de la sécurité logicielle dans lesquels SIL 0 correspond au niveau d'intégrité de sécurité le plus bas et SIL 4 au niveau le plus élevé. Plus le risque résultant d'une défaillance logicielle est élevé, plus le niveau d'intégrité de la sécurité logicielle est élevé.

La présente Norme a identifié des techniques et mesures applicables aux cinq niveaux d'intégrité de la sécurité logicielle. Les techniques et mesures requises pour les Niveaux d'intégrité de la sécurité logicielle 0 à 4 sont indiquées dans les tableaux normatifs de l'Annexe A. Dans la présente norme, les techniques requises pour le niveau 1 sont identiques à celles du niveau 2, et les techniques requises pour le niveau 3 sont identiques à celles du niveau 4. La présente Norme ne fournit aucune ligne directrice sur le niveau d'intégrité logicielle approprié pour un risque donné. Cette décision sera tributaire de nombreux facteurs, notamment de la nature de l'application, de la limite dans laquelle les autres systèmes assurent des fonctions de sécurité ainsi que de facteurs socio-économiques.

Le processus de spécification des fonctions de sécurité allouées au logiciel est défini dans le domaine d'application des normes IEC 62278 et IEC 62425.

La présente Norme spécifie les mesures nécessaires au respect de ces exigences.

L'IEC 62278 et l'IEC 62425 exigent qu'une approche systématique soit adoptée pour ce qui concerne:

a) l'identification des dangers, l'évaluation des risques et la prise de décisions en fonction de critères de risque,

b) l'identification de la réduction des risques nécessaire au respect des critères d'acceptation de risque;

c) la définition d'une Spécification des Exigences de Sécurité du Système, globale, qui décrit les protections indispensables en vue d'atteindre la réduction des risques requise,

d) le choix d'une architecture système adaptée,

e) la planification, le contrôle et la maîtrise des activités techniques et de management nécessaires pour transformer la Spécification des exigences de sécurité en un Système de sécurité dont l'intégrité de la sécurité est validée.

Au fur et à mesure que la spécification se décompose en une conception comprenant des composants et des systèmes relatifs à la sécurité, l'affectation des niveaux d'intégrité de la sécurité est effectuée. Finalement, cela conduit aux niveaux d'intégrité de la sécurité logicielle requis.

L'état actuel de la technique est tel que ni l'application des méthodes d'assurance qualité (mesures d'évitement des défauts et mesures de détection des défauts), ni l'application d'approches logicielles à tolérance aux pannes ne peuvent garantir la sécurité absolue du logiciel. Il n'existe aucun moyen connu de prouver l'absence de défauts dans un logiciel relatif à la sécurité raisonnablement complexe, en particulier l'absence de défauts de spécification et de conception.

Les principes appliqués dans le développement de logiciels à haute intégrité incluent, sans s'y limiter:

– des méthodes de conception descendante,
– la modularité,
– la vérification de chaque phase du cycle de vie du développement,
– des composants vérifiés et des bibliothèques de composants,
– une documentation claire et la traçabilité,
– des documents aptes à être audités,
– la validation,
– l'évaluation,
– la gestion de configuration et le contrôle des modifications, et
– l'étude appropriée des questions de compétence de l'organisation et du personnel.

La Spécification des exigences de sécurité du système identifie toutes les fonctions de sécurité allouées au logiciel et détermine leur niveau d'intégrité de la sécurité. Les étapes fonctionnelles successives de l'application de la présente Norme sont montrées à la Figure 1 et consistent à:

a) définir la Spécification des Exigences du Logiciel et, en parallèle, considérer l'architecture du logiciel. La stratégie de sécurité pour le logiciel et le niveau d'intégrité de la sécurité logicielle (7.2 et 7.3) sont développés dans l'architecture du logiciel;
b) concevoir, développer et soumettre à essai le logiciel selon le Plan d'Assurance Qualité du Logiciel, le niveau d'intégrité de la sécurité logicielle et le cycle de vie du logiciel (7.4 et 7.5);
c) mener à bien l'intégration logiciel/logiciel et l'intégration logiciel/matériel sur le matériel cible et vérifier la fonctionnalité (7.6);
d) accepter et déployer le logiciel (7.7 et 9.1);
e) si la maintenance du logiciel est requise pendant la vie opérationnelle, réactiver, le cas échéant, la présente Norme (9.2).

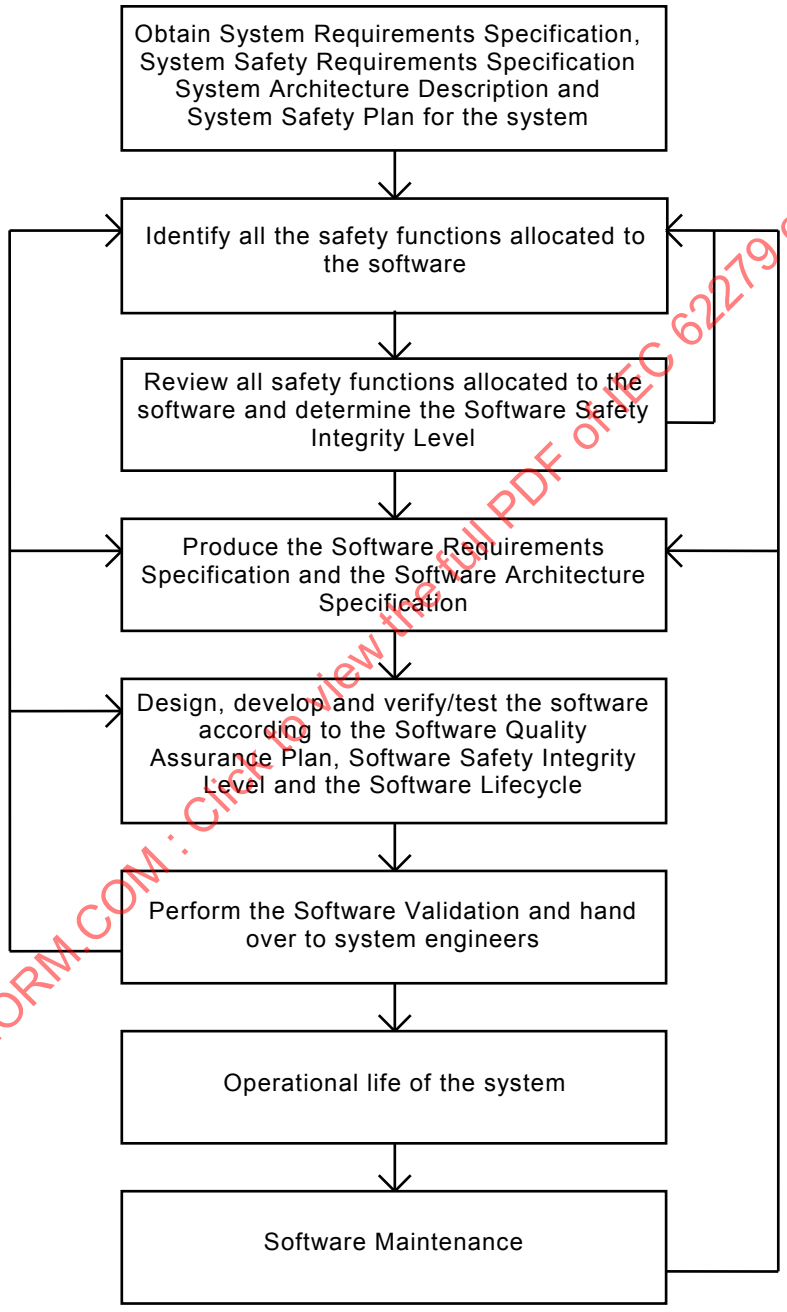
Un certain nombre d'activités se déroulent au cours du développement du logiciel, parmi lesquelles les essais (6.1), la vérification 6.2), la validation (6.3), l'évaluation (6.4), l'assurance qualité (6.5) et le contrôle des modifications et des évolutions (6.6).

Des exigences sont données en ce qui concerne les outils (6.7) et les systèmes qui sont configurés par des données d'application ou par des algorithmes d'application (Article 8).

Des exigences sont également fournies en ce qui concerne l'indépendance des rôles et la compétence du personnel impliqué dans le développement du logiciel (5.1, 5.2 et Annexe B).

La présente Norme n'impose pas l'utilisation d'un cycle de vie spécifique de développement du logiciel. Cependant des ensembles illustratifs de cycle de vie et de documentation sont fournis en 5.3 (Figure 3 et Figure 4) et en 7.1.

Des tableaux ont été établis pour classer diverses techniques/mesures par rapport aux niveaux d'intégrité de la sécurité logicielle. Les tableaux sont donnés en Annexe A. En référence croisée avec les tableaux, la bibliographie fournit une brève description de chaque technique/mesure avec des références à des sources complémentaires d'informations. La Bibliographie de techniques est donnée en Annexe D.



Anglais	Français
Obtain System Requirements Specification, System Safety Requirements Specification, System Architecture Description and System Safety Plan for the system	Obtenir pour le système la Spécification des Exigences du Système, la Spécification des Exigences de Sécurité du Système, la Description de l'Architecture du Système et le Plan de Sécurité du Système
Identify all the safety functions allocated to the software	Identifier toutes les fonctions de sécurité allouées au

Anglais	Français
	logiciel
Review all safety functions allocated to the software and determine the Software Safety Integrity Level	Examiner toutes les fonctions de sécurité allouées au logiciel et déterminer le niveau d'intégrité de la sécurité logicielle
Produce the Software Requirements Specification and the Software Architecture Specification	Produire la Spécification des Exigences du Logiciel et la Spécification de l'Architecture du Logiciel
Design, develop and verify/test the software according to the Software Quality Assurance Plan, Software Safety Integrity Level and the Software Lifecycle	Concevoir, développer et vérifier/soumettre à essai le logiciel conformément au Plan d'Assurance Qualité du Logiciel, au niveau d'intégrité de la sécurité logicielle et au cycle de vie du logiciel
Perform the Software Validation and hand over to system engineers	Effectuer la validation du logiciel et la transmettre aux ingénieurs système
Operational life of the system	Période de fonctionnement opérationnel du système
Software Maintenance	Maintenance du logiciel

Figure 1 – Démarche illustrative relative au logiciel

APPLICATIONS FERROVIAIRES – SYSTÈMES DE SIGNALISATION, DE TÉLÉCOMMUNICATION ET DE TRAITEMENT – LOGICIELS POUR SYSTÈMES DE COMMANDE ET DE PROTECTION FERROVIAIRE

1 Domaine d'application

1.1 La présente Norme internationale spécifie les exigences relatives aux processus et aux techniques applicables au développement de logiciel pour des systèmes électroniques programmables utilisés dans les applications ferroviaires de contrôle-commande et de protection. Elle est destinée à être utilisée dans tout domaine comportant des implications de sécurité. Ces systèmes peuvent être mis en œuvre à l'aide de microprocesseurs dédiés, de contrôleurs à logique programmable (automates programmables), de systèmes multiprocesseurs distribués, de grands systèmes dotés d'un calculateur central ou à l'aide d'autres architectures.

1.2 La présente Norme est exclusivement applicable au logiciel et à l'interaction entre le logiciel et le système auquel il appartient.

1.3 La présente Norme n'est pas pertinente pour les logiciels qui ont été identifiés comme n'ayant aucun impact sur la sécurité, c'est-à-dire pour les logiciels dont les défaillances ne peuvent pas affecter des fonctions de sécurité identifiées. Le concept de SIL 0 est introduit en raison de la présence d'une incertitude dans l'évaluation du risque, voire dans l'identification des dangers. Au moins les exigences associées au SIL 0 de la présente Norme sont satisfaites pour la partie logicielle des fonctions qui ont un impact sur la sécurité en dessous de SIL 1.

1.4 La présente Norme s'applique à tous les logiciels de sécurité utilisés dans des systèmes ferroviaires de contrôle-commande et de protection, y compris:

– la programmation d'applications,
– les systèmes d'exploitation,
– les outils,
– les microprogrammes.

La programmation d'applications inclut la programmation de haut niveau, la programmation de bas niveau et la programmation spécifique personnalisée (par exemple: la logique à contacts d'un contrôleur logique programmable).

1.5 La présente Norme traite également de l'utilisation de logiciels et d'outils préexistants. Ces logiciels peuvent être utilisés si les exigences spécifiques en 7.3.4.7 et 6.5.4.16 relatives aux logiciels préexistants et aux outils en 6.7 sont satisfaites.

1.6 Un logiciel développé selon une version quelconque de la présente Norme sera considéré comme étant conforme et non soumis aux exigences relatives aux logiciels préexistants.

1.7 La présente Norme considère que la conception moderne d'applications utilise fréquemment des logiciels génériques qui sont adaptés à servir de base pour diverses applications. Ces logiciels génériques sont ensuite configurés par des données et/ou des algorithmes, afin de produire le logiciel exécutable pour l'application. Les Articles généraux 1 à 6 et 9 de la présente Norme s'appliquent aux logiciels génériques ainsi qu'aux données d'application et algorithmes d'application. L'Article spécifique 7 s'applique uniquement pour

les logiciels génériques alors que l'Article 8 fournit les exigences spécifiques pour les données d'application et algorithmes d'application.

1.8 La présente Norme ne vise pas à traiter des problèmes commerciaux. Il convient de traiter ceux-ci comme une partie essentielle de tout accord contractuel. Tous les articles de la présente Norme nécessitent une étude soignée dans toute situation commerciale.

1.9 La présente Norme n'est pas destinée à être rétroactive. Elle s'applique donc principalement aux nouveaux développements et n'est applicable intégralement aux systèmes existants que s'ils font l'objet de modifications importantes. Pour les modifications mineures, seul 9.2 s'applique. Le chargé d'évaluation analyse les preuves fournies dans la documentation du logiciel pour confirmer si, oui ou non, la détermination de la nature et de l'étendue des modifications du logiciel est adéquate. Cependant, il est hautement recommandé d'appliquer la présente Norme pendant les mises à niveau et la maintenance des logiciels existants.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 62278:2002, *Applications ferroviaires – Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (FDMS)*

ISO/IEC 90003:2014, *Ingénierie du logiciel – Lignes directrices pour l'application de l'ISO 9001:2008 aux logiciels informatiques*

ISO/IEC 25010 (série), *Ingénierie des systèmes et du logiciel – Exigences de qualité et évaluation des systèmes et du logiciel (SQuaRE) – Modèles de qualité du système et du logiciel*

ISO 9000, *Systèmes de management de la qualité – Principes essentiels et vocabulaire*

ISO 9001:2008, *Systèmes de management de la qualité – Exigences*

3 Termes, définitions et abréviations

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.1.1
évaluation
processus d'analyse afin de déterminer si un logiciel, qui peut inclure des processus, de la documentation, des composants logiciels et/ou matériels de systèmes et sous-systèmes, satisfait aux exigences spécifiées et afin de formuler un jugement sur le fait que le logiciel répond à l'objectif attendu

Note 1 à l'article: L'évaluation de la sécurité est une évaluation axée sur les propriétés de sécurité d'un système, mais sans s'y limiter.

3.1.2
chargé d'évaluation
entité qui mène à bien une évaluation

3.1.3
logiciel standard disponible dans le commerce (COTS)
logiciel défini par les besoins du marché, disponible dans le commerce et dont l'adéquation aux besoins a été démontrée par un large éventail d'utilisateurs

Note 1 à l'article: L'abréviation «COTS» est dérivée du terme anglais développé correspondant «commercial off-the-shelf».

3.1.4
composant
partie constitutive de logiciel qui a des interfaces et un comportement bien définis par rapport à la conception et à l'architecture du logiciel

Note 1 à l'article: Un composant logiciel satisfait aux critères suivants:

– il est conçu conformément aux "Composants" (voir Tableau A.20);
– il couvre un sous-ensemble spécifique des exigences relatives au logiciel;
– il est clairement identifié et a une version indépendante au sein du système de gestion de configuration ou est une partie intégrante d'un ensemble de composants (par exemple: sous-systèmes) qui ont une version indépendante.

3.1.5
gestionnaire de la configuration
entité qui est chargée de mettre en œuvre et d'exécuter les processus pour la gestion de configuration des documents, logiciels et outils connexes, y compris la gestion des modifications et des évolutions

3.1.6
client
entité qui achète un système de contrôle-commande et de protection du ferroviaire comprenant le logiciel

3.1.7
concepteur
entité qui analyse et transforme des exigences spécifiées en solutions de conception acceptables qui ont le niveau requis d'intégrité de la sécurité

3.1.8
entité
personne, groupe ou organisation qui remplit un rôle tel que défini dans la présente Norme

3.1.9
erreur
faute
défaut
état anormal pouvant conduire à une erreur dans un système

Note 1 à l'article: Un défaut peut être aléatoire ou systématique.

3.1.10
erreur
écart par rapport à la conception prévue pouvant conduire à un comportement ou à une défaillance inattendu(e) du système

3.1.11
défaillance
différence inacceptable entre la performance requise et la performance observée

3.1.12
tolérance aux fautes
capacité intégrée d'un système à délivrer, d'une manière correcte et continue, un service tel qu'il est spécifié, en présence d'un nombre limité de défauts matériels ou logiciels

3.1.13
micrologiciel
firmware
logiciel enregistré dans une mémoire morte ou dans une mémoire semi-permanente telle qu'une mémoire flash, d'une manière qui est fonctionnellement indépendante du logiciel applicatif

3.1.14
logiciel générique
logiciel pouvant être utilisé pour une grande variété d'installations simplement en fournissant des données et/ou algorithmes propres à l'application

3.1.15
réalisateur
entité qui transforme des choix de conception en leur réalisation physique

3.1.16
intégration
processus d'assemblage d'éléments de logiciel et/ou de matériel, selon la spécification de conception et d'architecture, et d'essais de l'ensemble intégré

3.1.17
chargé d'intégration
entité qui mène à bien l'intégration du logiciel

3.1.18
logiciel préexistant
logiciel développé avant l'application dont il est ici question, incluant les logiciels COTS (standards disponibles dans le commerce) et libres

3.1.19
logiciel libre
code source à la disposition du grand public avec des restrictions de droits d'auteur assouplies ou inexistantes

3.1.20
contrôleur logique programmable
automate programmable
système de commande informatisé, doté d'une mémoire programmable par l'utilisateur pour le stockage des instructions, pour mettre en œuvre des fonctions spécifiques

3.1.21
gestion de projet
conduite administrative et/ou technique d'un projet, y compris les aspects de sécurité

3.1.22
chef de projet
entité qui mène à bien la gestion de projet

3.1.23
fiabilité
capacité d'une entité à remplir une fonction requise, dans des conditions données pendant une durée donnée

3.1.24
robustesse
aptitude d'un élément à détecter et gérer les situations anormales

3.1.25
gestionnaire des exigences
entité qui mène à bien la gestion des exigences

3.1.26
gestion des exigences
processus consistant à identifier, documenter, analyser, classer par ordre de priorité et accepter par accord les exigences et ensuite contrôler les modifications et évolutions et communiquer avec les parties prenantes

Note 1 à l'article: La gestion des exigences est un processus continu tout au long d'un projet.

3.1.27
risque
combinaison du taux d'occurrence d'accidents et d'incidents occasionnant un dommage (causé par un danger) et de la gravité de ce dommage

3.1.28
sécurité
absence de niveaux inacceptables de risque de dommage aux personnes

3.1.29
autorité de tutelle
organisme chargé de certifier que les logiciels ou services relatifs à la sécurité sont conformes aux exigences de sécurité statutaires applicables

3.1.30
fonction de sécurité
fonction qui met en œuvre tout ou partie d'une exigence relative à la sécurité

3.1.31
logiciel de sécurité
logiciel qui exécute des fonctions de sécurité

3.1.32
logiciel
création intellectuelle comprenant les programmes, les procédures, les règles, les données et toute documentation associée en rapport avec le fonctionnement d'un système

3.1.33
référentiel logiciel
ensemble complet et cohérent du code source, des fichiers exécutables, des fichiers de configuration, des scripts d'installation et de la documentation qui sont nécessaires à une version diffusée d'un logiciel

Note 1 à l'article: Les informations concernant les compilateurs, les systèmes d'exploitation, les logiciels préexistants et les outils associés font partie intégrante du référentiel. Elles permettront à l'organisation de reproduire des versions bien définies et seront les données d'entrée pour de futures versions diffusées pour améliorations ou de la mise à niveau dans la phase de maintenance.

3.1.34
déploiement du logiciel
transfert, installation et activation d'un référentiel logiciel livrable correspondant à une version déjà diffusée et évaluée

3.1.35
cycle de vie du logiciel
ensemble des activités survenant pendant une période qui commence à la spécification du logiciel et se termine quand le logiciel ne peut plus être utilisé

Note 1 à l'article: En principe, le cycle de vie du logiciel inclut une phase de spécification des exigences, une phase de conception, une phase d'essai, une phase d'intégration, une phase de déploiement et une phase de maintenance.

3.1.36
maintenabilité du logiciel
capacité d'un logiciel à être modifié pour corriger les défauts, pour améliorer la performance ou d'autres attributs, ou pour l'adapter à un environnement différent

3.1.37
maintenance du logiciel
action, ou ensemble d'actions, effectuée sur le logiciel après son déploiement afin de renforcer ou corriger sa fonctionnalité

3.1.38
niveau d'intégrité de la sécurité logicielle
numéro de classification déterminant les techniques et mesures qui sont à appliquer au logiciel

Note 1 à l'article: Les logiciels relatifs à la sécurité ont été classés en cinq niveaux d'intégrité de sécurité, dans lesquels 0 est le plus bas et 4 le plus élevé.

3.1.39
fournisseur
entité qui conçoit et bâtit un système de contrôle-commande et de protection du ferroviaire comprenant le logiciel ou des parties de celui-ci

3.1.40
niveau d'intégrité de la sécurité du système
numéro indiquant le degré de confiance requis sur le fait qu'un système intégré, comprenant du matériel et du logiciel, respectera ses exigences de sécurité spécifiées

3.1.41
chargé des essais
entité qui mène à bien les essais

3.1.42
essais
processus d'exécution d'un logiciel dans des conditions maîtrisées de manière à s'assurer de son comportement et de sa performance par rapport à la Spécification des Exigences correspondante

3.1.43
classe d'outils T1
classe sans génération de sorties susceptibles de contribuer, directement ou indirectement, au code exécutable (y compris les données) du logiciel

Note 1 à l'article: Les exemples de T1 comprennent: un éditeur de texte ou un outil d'aide à la conception ou aux exigences démuni de toute capacité de génération automatique de code; les outils de contrôle de configuration.

3.1.44
classe d'outils T2
classe permettant la réalisation de l'essai ou de la vérification de la conception ou du code exécutable, tandis que des erreurs internes à l'outil peuvent ne pas arriver à révéler des défauts, mais sans créer directement des erreurs dans le logiciel exécutable

Note 1 à l'article: Des exemples de T2 comprennent: les générateurs de simulateur d'essai, les outils de mesure de la couverture des essais, les outils d'analyse statique.

3.1.45
classe d'outils T3
classe avec génération des sorties susceptibles de contribuer, directement ou indirectement, au code exécutable (y compris les données) du système de sécurité

Note 1 à l'article: Les exemples de T3 comprennent: les compilateurs de code source, les compilateurs de données/algorithmes, les outils pour modifier les points de consigne au cours du fonctionnement du système; les compilateurs d'optimisation lorsque la relation entre le programme de code source et le code objet généré n'est pas évidente; les compilateurs qui incorporent un ensemble d'instructions exécutables dans le code exécutable.

3.1.46
traçabilité
facilité avec laquelle un lien peut être établi entre deux ou plusieurs produits d'un processus de développement, et plus particulièrement, ceux possédant entre eux une relation de prédécesseur à successeur ou de maître à esclave

3.1.47
validation
processus d'analyse suivie d'un jugement fondé sur des preuves afin de déterminer si un élément (par exemple: processus, documentation, logiciel ou application) répond aux besoins d'un utilisateur, en particulier en termes de sécurité et de qualité et en mettant l'accent sur l'adéquation de son fonctionnement selon son objectif dans son environnement prévu

3.1.48
chargé de validation
entité qui est responsable de la validation

3.1.49
vérification
processus d'examen suivi d'un jugement fondé sur des preuves que des éléments de sortie (processus, documentation, logiciel ou application) d'une phase de développement spécifique satisfont aux exigences de la phase en question en termes de complétude, de correction et de cohérence

Note 1 à l'article: La vérification est surtout fondée sur une revue de documents en sortie (documents relatifs à la conception, à la mise en œuvre, aux essais, etc.).

3.1.50
chargé de vérification
entité qui est responsable d'une ou plusieurs activités de vérification

3.2 Abréviations

ASR	Assessor (Chargé d'évaluation)
COTS	Commercial off-the-shelf (Standard, disponibles dans le commerce)
CGM	Configuration Manager (Gestionnaire de la Configuration)
DES	Designer (Concepteur)
HR	Hautement recommandé
IMP	Implementer (Réalisateur)
INT	Integrator (Chargé d'intégration)
JSD	Méthode Jackson System Development

M	Mandatory (Obligatoire)
MASCOT	Modular Approach to Software Construction, Operation and Test (Approche modulaire à la construction, à l'exploitation et à l'essai de logiciel)
NR	Non recommandé
CdP	Chef de projet
QAM	Quality Assurance Manager (Gestionnaire de l'assurance qualité)
R	Recommandé
FDMS	Fiabilité, disponibilité, maintenabilité et sécurité
RQM	Requirements Manager (Gestionnaire des exigences)
SDL	Specification and Description Language (Langage de spécification et de description)
SFC	Sequential Function Charts (Graphes séquentiels de fonction)
SIL	Safety Integrity Level (Niveau d'intégrité de la sécurité)
SOM	Service Oriented Modeling (Modélisation orientée service)
SSADM	Structured Systems Analysis & Design Methodology (Analyse structurée des systèmes et méthodologie de conception)
TST	Tester (Chargé des essais)
V&V	Vérification et Validation
VAL	Validator (Chargé de validation)
VER	Verifier (Chargé de vérification)

4 Objectifs, conformité et niveaux d'intégrité de la sécurité logicielle

4.1 L'allocation au logiciel de fonctions de sécurité du système ainsi que d'interfaces de logiciel doit être identifiée dans la documentation du système. Le système dans lequel le logiciel est intégré doit être complètement défini quant aux éléments suivants:

– les fonctions et interfaces;
– les conditions d'application;
– la configuration ou l'architecture du système;
– les dangers à maîtriser;
– les exigences d'intégrité de la sécurité;
– l'allocation des exigences et du niveau de SIL au logiciel et au matériel;
– les contraintes de temps.

NOTE L'allocation des exigences d'intégrité de la sécurité peut conduire à un SIL différent pour des parties bien distinctes de logiciel et de matériel d'un sous-système. Cette allocation dépend de la contribution des parties de logiciel et de matériel aux fonctions relatives à la sécurité et dépend aussi des mécanismes de limitation des défaillances y compris la séparation de fonction avec un SIL différent.

4.2 L'intégrité du logiciel doit être spécifiée comme étant l'un des cinq niveaux, de SIL 0 (le plus bas) à SIL 1 à 4 (le plus élevé) pour l'intégrité de la sécurité.

4.3 Le niveau d'intégrité de la sécurité logicielle requis doit être décidé et évalué au niveau du système sur la base du niveau d'intégrité de la sécurité du système et du niveau de risque associé à l'utilisation du logiciel dans le système.

4.4 Les exigences associées à SIL 0 de la présente Norme doivent être satisfaites pour la partie logicielle des fonctions qui ont un degré d'incertitude relatif à l'impact sur la sécurité généralement en dessous de SIL 1.

4.5 Pour se conformer à la présente Norme, on doit montrer que les exigences définies dans chaque paragraphe ont été satisfaites vis-à-vis du niveau d'intégrité de la sécurité logicielle et vis-à-vis des techniques et méthodes définies à l'Annexe A.

4.6 Lorsqu'une exigence est qualifiée par les termes "dans la limite requise par le niveau d'intégrité de la sécurité logicielle", cela indique qu'une panoplie de techniques et de mesures doit être utilisée pour satisfaire à l'exigence en question.

4.7 Lorsque 4.6 est appliqué, les tableaux de l'Annexe A normative doivent être utilisés pour faciliter la sélection des techniques et des mesures appropriées au niveau d'intégrité de la sécurité logicielle. La sélection doit être documentée dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document mentionné par celui-ci. Des lignes directrices sur ces techniques sont données dans l'Annexe D informative.

4.8 Si une technique ou une mesure comptant parmi celles qui sont hautement recommandées (HR) dans les tableaux n'est pas utilisée, la justification de l'utilisation de techniques alternatives doit être détaillée et enregistrée soit dans le Plan d'Assurance Qualité du Logiciel, soit dans un autre document mentionné par ce dernier. Cette formalité n'est pas nécessaire si l'on a recours à une combinaison approuvée des techniques fournies dans le tableau correspondant. Il doit être démontré que les techniques choisies ont été correctement appliquées.

4.9 Si l'utilisation d'une technique ou d'une mesure non contenue dans les tableaux est proposée, son efficacité et sa pertinence par rapport au respect de l'exigence particulière et de l'objectif global du paragraphe doivent être justifiées et enregistrées soit dans le Plan d'Assurance Qualité du Logiciel, soit dans un autre document mentionné par le Plan d'Assurance Qualité du Logiciel.

4.10 La conformité aux exigences d'un paragraphe particulier et à leurs techniques et mesures respectives détaillées dans les tableaux doit être vérifiée par l'examen des documents requis par la présente Norme. Le cas échéant, d'autres preuves tangibles, l'audit et la présence lors des essais doivent aussi être pris en compte.

5 Organisation et gestion du développement logiciel

5.1 Organisation, rôles et responsabilités

5.1.1 Objet

5.1.1.1 Assurer que l'ensemble du personnel responsable du logiciel est organisé, habilité et capable d'assumer ses responsabilités.

NOTE L'indépendance entre les rôles peut être requise afin de réduire la possibilité que des personnes dans des rôles différents souffrent des mêmes idées fausses ou commettent les mêmes erreurs. Cette forme d'indépendance peut être obtenue en employant des personnes différentes dans des rôles différents, mais elle n'exige habituellement pas que les rôles soient joués dans des parties différentes de l'organisation ou dans des entreprises différentes (sauf exigence spécifiquement contraire). Il convient également que les personnes jouant des rôles qui impliquent de formuler des jugements au sujet de l'acceptabilité d'un produit ou d'un processus du point de vue de la sécurité ne soient influencées ni par une pression exercée par leurs homologues ou leurs superviseurs ni par des considérations de profits commerciaux. Cette forme d'indépendance est plus susceptible d'exiger que des rôles différents soient joués dans des parties différentes de l'organisation ou dans une entreprise différente. En général, un degré plus grand de sécurité exige un degré plus grand d'indépendance pour les divers rôles et organisations.

5.1.2 Exigences

5.1.2.1 Au minimum, le fournisseur doit mettre en œuvre les parties de l'ISO 9001:2008 qui traitent de l'organisation et de la gestion du personnel et des responsabilités.

5.1.2.2 Les responsabilités doivent être conformes aux exigences définies dans l'Annexe B.

5.1.2.3 Le personnel affecté à ces rôles impliqués dans le développement ou la maintenance du logiciel doit être nommé et enregistré.

5.1.2.4 Un Chargé d'évaluation doit être désigné par le fournisseur, le client ou l'autorité de tutelle.

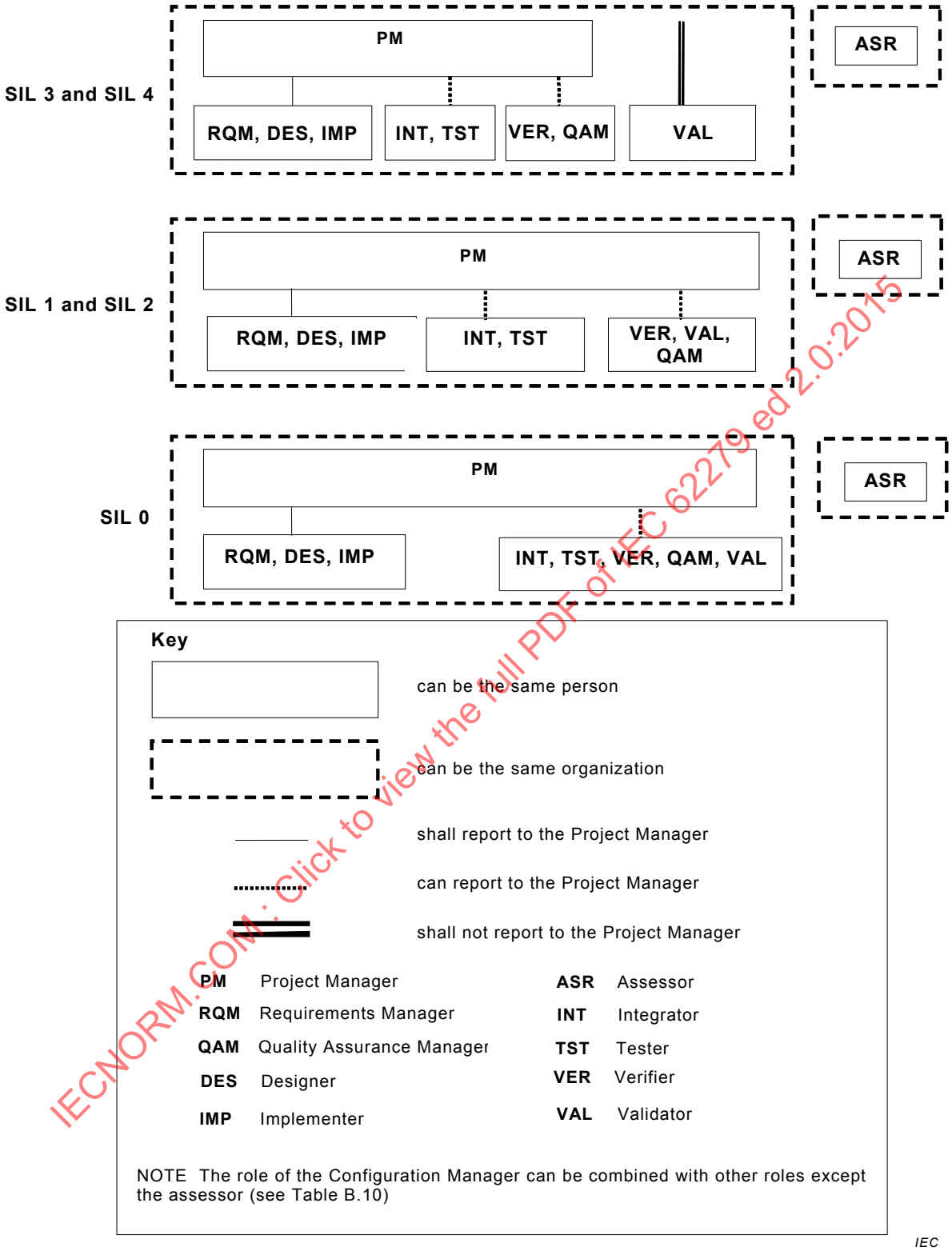
5.1.2.5 Le Chargé d'évaluation doit être indépendant du fournisseur. Cependant, à la discrétion de l'autorité de tutelle, le chargé d'évaluation peut appartenir à l'organisation du fournisseur ou à celle du client, mais être indépendant du projet de développement.

5.1.2.6 Le Chargé d'évaluation doit être indépendant du projet.

5.1.2.7 Autorité doit être donnée au Chargé d'évaluation pour accomplir l'évaluation du logiciel.

5.1.2.8 Le Chargé de validation doit signifier son accord/désaccord pour toute version diffusée du logiciel.

5.1.2.9 Tout au long du Cycle de vie du logiciel, l'attribution des rôles aux personnes doit être conforme aux spécifications données de 5.1.2.10 à 5.1.2.14, dans toute la mesure requise par le SIL du logiciel.



Anglais	Français
Key	Légende
can be the same person	peuvent être la même personne
can be the same organization	peuvent être dans la même organisation
shall report to the Project Manager	doit rendre compte au Chef de Projet
can report to the Project Manager	peut avoir à rendre compte au Chef de Projet
shall not report to the Project Manager	ne doit pas avoir à rendre compte au Chef de Projet

Anglais	Français
PM Project Manager	CdP Chef de Projet
ASR Assessor	ASR Chargé d'évaluation
RQM Requirements Manager	RQM Gestionnaire des exigences
INT Integrator	INT Chargé d'intégration
QAM Quality Assurance Manager	QAM Gestionnaire de l'assurance qualité
TST Tester	TST Chargé des essais
DES Designer	DES Concepteur
VER Verifier	VER Chargé de vérification
IMP Implementer	IMP Réalisateur
VAL Validator	VAL Chargé de validation
PM	CdP
NOTE The role of the Configuration Manager can be combined with other roles except the assessor (see Table B.10)	NOTE Le rôle du Gestionnaire de la Configuration peut être combiné à d'autres rôles, excepté le rôle de Chargé d'évaluation (voir Tableau B.10)

Figure 2 – Illustration de la structure organisationnelle préférentielle

NOTE La Figure 2 est uniquement une illustration de la structure organisationnelle préférentielle.

5.1.2.10 La structure organisationnelle préférentielle pour le SIL 3 et le SIL 4 correspond à ce que:

a) Le Gestionnaire des exigences, le Concepteur et le Réalisateur pour un composant logiciel peuvent être la même personne.

b) Le Gestionnaire des exigences, le Concepteur et le Réalisateur pour un composant logiciel doivent rendre compte au Chef de projet.

c) Le Chargé d'intégration et le Chargé des essais pour un composant logiciel peuvent être la même personne.

d) Le Chargé d'intégration et le Chargé des essais pour un composant logiciel peuvent rendre compte au Chef de projet ou au Chargé de validation.

e) Le Chargé de vérification ou le Gestionnaire de l'assurance qualité peut rendre compte au Chef de projet ou au Chargé de validation.

f) Le Chargé de validation ne doit pas rendre compte au Chef de projet. Autrement dit, le Chef de projet ne doit avoir aucune influence sur les décisions du Chargé de validation, mais le Chargé de validation fournit au Chef de projet des informations sur ses décisions.

g) Une personne qui est un Gestionnaire des exigences, un Concepteur ou un Réalisateur pour un composant logiciel ne doit être ni un Chargé des essais ni un Chargé d'intégration pour le même composant logiciel.

h) Une personne qui est un Chargé d'intégration ou un Chargé des essais pour un composant logiciel ne doit être ni un Gestionnaire des exigences ni un Concepteur ni un Réalisateur pour le même composant logiciel.

i) Une personne qui est un Chargé de vérification ou un Gestionnaire de l'assurance qualité ne doit être ni un Gestionnaire des exigences, ni un Concepteur, ni un Réalisateur, ni un Chargé d'intégration, ni un Chargé des essais, ni un Chargé de validation.

j) Une personne qui est un Chargé de validation ne doit être ni un Gestionnaire des exigences, ni un Gestionnaire de l'assurance qualité, ni un Concepteur, ni un Réalisateur, ni un Chargé d'intégration, ni un Chargé des essais, ni un Chargé de vérification.

k) Une personne qui est Chef de projet peut en plus remplir les rôles de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification à la condition que les exigences relatives à l'indépendance entre ces rôles supplémentaires soient respectées.

l) Le Chef de projet, le Gestionnaire des exigences, le Gestionnaire de l'assurance qualité, le Concepteur, le Réalisateur, le Chargé d'intégration, le Chargé des essais, le Chargé de vérification et le Chargé de validation peuvent appartenir à la même organisation.

m) Le Chargé d'évaluation doit être indépendant et, du point de vue organisationnel, indépendant par rapport aux rôles de Chef de projet, de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification et de Chargé de validation.

Cependant, les options suivantes peuvent s'appliquer:

n) Une personne qui est Chargé de validation peut aussi remplir le rôle de Chargé de vérification, mais en maintenant toujours l'indépendance par rapport au Chef de projet. Dans ce cas, les documents produits par le Chargé de vérification doivent être revus par une autre personne compétente ayant le même niveau d'indépendance que le Chargé de validation. Cette option organisationnelle doit être soumise à l'approbation du Chargé d'évaluation.

o) Une personne qui est un Chargé de vérification ou un Gestionnaire de l'assurance qualité peut également remplir le rôle de Chargé d'intégration et de Chargé des essais. Dans ce cas, le rôle du Chargé de validation doit vérifier l'adéquation des preuves documentées obtenues par intégration et essais avec les objectifs de vérification spécifiés.

5.1.2.11 La structure organisationnelle préférentielle pour le SIL 1 et le SIL 2 correspond à ce que:

a) Le Gestionnaire des exigences, le Concepteur et le Réalisateur pour un composant logiciel peuvent être la même personne et doivent rendre compte au Chef de projet.

b) Le Chargé d'intégration et le Chargé des essais pour un composant logiciel peuvent être la même personne.

c) Le Chargé d'intégration et le Chargé des essais pour un composant logiciel peuvent rendre compte au Chef de projet ou au Chargé de validation.

d) Le Chargé de vérification, le Gestionnaire de l'assurance qualité et le Chargé de validation peuvent être la même personne.

e) Le Chargé de vérification, le Gestionnaire de l'assurance qualité et le Chargé de validation peuvent rendre compte au Chef de projet.

f) Une personne qui est un Gestionnaire d'exigences, un Gestionnaire de l'assurance qualité, un Concepteur ou un Réalisateur pour un composant logiciel ne doit être ni un Chargé des essais, ni un Chargé d'intégration pour le même composant logiciel.

g) Une personne qui est un Chargé d'intégration ou un Chargé des essais pour un composant logiciel ne doit être ni un Gestionnaire des exigences ni un Concepteur ni un Réalisateur pour le même composant logiciel.

h) Une personne qui est un Chargé de vérification, un Gestionnaire de l'assurance qualité ou un Chargé de validation ne doit être ni un Gestionnaire des exigences, ni un Concepteur, ni un Réalisateur, ni un Chargé d'intégration, ni un Chargé des essais.

i) Une personne qui est un Chef de projet peut en plus remplir les rôles de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification ou de Chargé de validation à la condition que les exigences relatives à l'indépendance entre ces rôles supplémentaires soient respectées.

j) Le Chef de projet, le Gestionnaire des exigences, le Gestionnaire de l'assurance qualité, le Concepteur, le Réalisateur, le Chargé d'intégration, le Chargé des essais, le Chargé de vérification et le Chargé de validation peuvent appartenir à la même organisation.

k) Le Chargé d'évaluation doit être indépendant et, du point de vue organisationnel, indépendant par rapport aux rôles de Chef de projet, de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification et de Chargé de validation.

Cependant, les options suivantes peuvent s'appliquer:

l) Une personne qui est un Chargé de vérification ou un Gestionnaire de l'assurance qualité peut aussi remplir le rôle de Chargé d'intégration et de Chargé des essais. Dans ce cas, le rôle du Chargé de validation doit inclure la revue des documents produits par le Chargé de vérification ou le Gestionnaire de l'assurance qualité, conservant donc deux niveaux de contrôle au sein de l'organisation du projet.

m) Une personne qui est un Chargé de validation peut aussi remplir le rôle de Chargé de vérification, de Gestionnaire de l'assurance qualité, de Chargé d'intégration et de Chargé des essais. Dans ce cas, les documents produits par le Chargé de vérification ou le Gestionnaire de l'assurance qualité doivent être revus par une autre personne compétente ayant le même niveau d'indépendance que le Chargé de validation. Cette option organisationnelle doit être soumise à l'approbation du Chargé d'évaluation.

5.1.2.12 La structure organisationnelle préférentielle pour le SIL 0 correspond à ce que:

a) Le Gestionnaire des exigences, le Concepteur et le Réalisateur pour un composant logiciel peuvent être la même personne et doivent être gérés par le Chef de projet.

b) Le Chargé d'intégration, le Chargé des essais, le Chargé de vérification, le Gestionnaire de l'assurance qualité et le Chargé de validation pour un composant logiciel peuvent être la même personne.

c) Le Chargé d'intégration, le Chargé des essais, le Chargé de vérification, le Gestionnaire de l'assurance qualité et le Chargé de validation peuvent être gérés par le Chef de projet.

d) Une personne qui est un Gestionnaire des exigences, un Concepteur ou un Réalisateur pour un composant logiciel ne doit être ni un Chargé des essais ni un Chargé d'intégration pour le même composant logiciel.

e) Une personne qui est un Chargé de vérification, un Gestionnaire de l'assurance qualité ou un Chargé de validation ne doit être ni un Gestionnaire des exigences, ni un Concepteur, ni un Réalisateur.

f) Une personne qui est un Chef de projet peut en plus remplir les rôles de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification ou de Chargé de validation à la condition que les exigences relatives à l'indépendance entre ces rôles supplémentaires soient respectées.

g) Le Chef de projet, le Gestionnaire des exigences, le Gestionnaire de l'assurance qualité, le Concepteur, le Réalisateur, le Chargé d'intégration, le Chargé des essais, le Chargé de vérification et le Chargé de validation peuvent appartenir à la même organisation.

h) Le Chargé d'évaluation doit être indépendant et, du point de vue organisationnel, indépendant par rapport aux rôles de Chef de projet, de Gestionnaire des exigences, de Gestionnaire de l'assurance qualité, de Concepteur, de Réalisateur, de Chargé d'intégration, de Chargé des essais, de Chargé de vérification et de Chargé de validation.

Cependant, les variantes suivantes peuvent s'appliquer:

i) Le Gestionnaire des exigences, le Concepteur, le Réalisateur, le Chargé d'intégration et le Chargé des essais peuvent être la même personne.

j) Le Chargé de validation, le Chargé de vérification et le Gestionnaire de l'assurance qualité peuvent aussi être la même personne.

k) Une personne qui est un Chargé de vérification, un Gestionnaire de l'assurance qualité ou un Chargé de validation ne doit être ni un Gestionnaire des exigences, ni un Concepteur, ni un Réalisateur.

5.1.2.13 Les rôles de Gestionnaire des exigences, de Concepteur et de Réalisateur pour un certain composant peuvent exercer les rôles de Chargé des essais et de Chargé d'intégration pour un composant différent.

5.1.2.14 Les rôles du Chargé de vérification, du Gestionnaire de l'assurance qualité et du Chargé de validation doivent être définis au niveau du projet et doivent rester inchangés tout au long du projet de développement.

5.2 Compétence du personnel

5.2.1 Objets

Assurer que l'ensemble du personnel responsable du logiciel a les compétences pour s'acquitter de ces responsabilités en démontrant la capacité d'exécuter les tâches correspondantes de manière correcte, efficace et cohérente à un niveau de qualité élevé et dans des conditions variables.

5.2.2 Exigences

5.2.2.1 Les principales compétences requises pour chaque rôle dans le développement du logiciel sont définies à l'Annexe B. Si une expérience, des capacités ou des qualifications supplémentaires sont requises pour un rôle dans le cycle de vie du logiciel, celles-ci doivent être définies dans le Plan d'Assurance Qualité du Logiciel.

5.2.2.2 La preuve documentée de la compétence du personnel, y compris les connaissances techniques, les qualifications, l'expérience adéquate et la formation appropriée, doit être conservée par l'organisation du fournisseur afin de démontrer une organisation de sécurité appropriée.

5.2.2.3 Une fois apportée, à la satisfaction d'un chargé d'évaluation ou par certification, la preuve de la démonstration des compétences de l'ensemble du personnel affecté aux divers rôles, chaque individu doit démontrer un maintien et un développement continus des compétences. Cela peut se démontrer par la tenue d'un journal de bord montrant que l'activité est effectuée correctement sur une base régulière et qu'une formation complémentaire est dispensée conformément à l'ISO 9001 et à 6.2.2 «Compétence, sensibilisation et formation» de l'ISO/IEC 90003:2014.

5.2.2.4 L'organisation doit maintenir les procédures de gestion des compétences du personnel à être apte aux rôles appropriés en conformité avec des normes de qualité existantes.

5.3 Questions relatives au cycle de vie et à la documentation

5.3.1 Objets

5.3.1.1 Structurer le développement du logiciel en phases et activités définies.

5.3.1.2 Enregistrer toutes les informations en rapport avec le logiciel tout au long de son cycle de vie.

5.3.2 Exigences

5.3.2.1 Un modèle de cycle de vie pour le développement du logiciel doit être choisi. Ce modèle doit être détaillé dans le Plan d'Assurance Qualité du Logiciel conformément à 6.5.

Deux exemples de modèle de cycle de vie sont présentés à la Figure 3 et à la Figure 4.

5.3.2.2 Le modèle de cycle de vie doit prendre en compte la possibilité d'itérations à l'intérieur des phases et entre celles-ci.

5.3.2.3 Les procédures d'Assurance Qualité doivent être menées à bien parallèlement aux activités du cycle de vie et utiliser les mêmes termes.

5.3.2.4 Le Plan d'Assurance Qualité du Logiciel, le Plan de Vérification du Logiciel, le Plan de Validation du Logiciel et le Plan de Gestion de Configuration du Logiciel doivent être rédigés au début du projet et maintenus tout au long du cycle de vie de développement du logiciel.

5.3.2.5 Toutes les activités à exécuter pendant une phase doivent être définies et planifiées avant le commencement de la phase.

5.3.2.6 Tous les documents doivent être structurés de manière à permettre un enrichissement continu parallèlement au processus de développement.

5.3.2.7 La traçabilité de chaque document doit être assurée en termes d'un numéro de référence unique et d'une relation définie et documentée à d'autres documents.

5.3.2.8 Chaque terme, acronyme ou abréviation, doit avoir le même sens dans tous les documents. Si cela n'est pas possible pour des raisons historiques, les différents sens doivent être énumérés avec leurs références.

5.3.2.9 À l'exception de la documentation liée aux logiciels préexistants (voir 7.3.4.7), chaque document doit être rédigé conformément aux règles suivantes:

– il doit contenir ou mettre en œuvre toutes les conditions et exigences applicables du document précédent avec lequel il a une relation hiérarchique;
– il ne doit pas être en contradiction avec le document précédent.

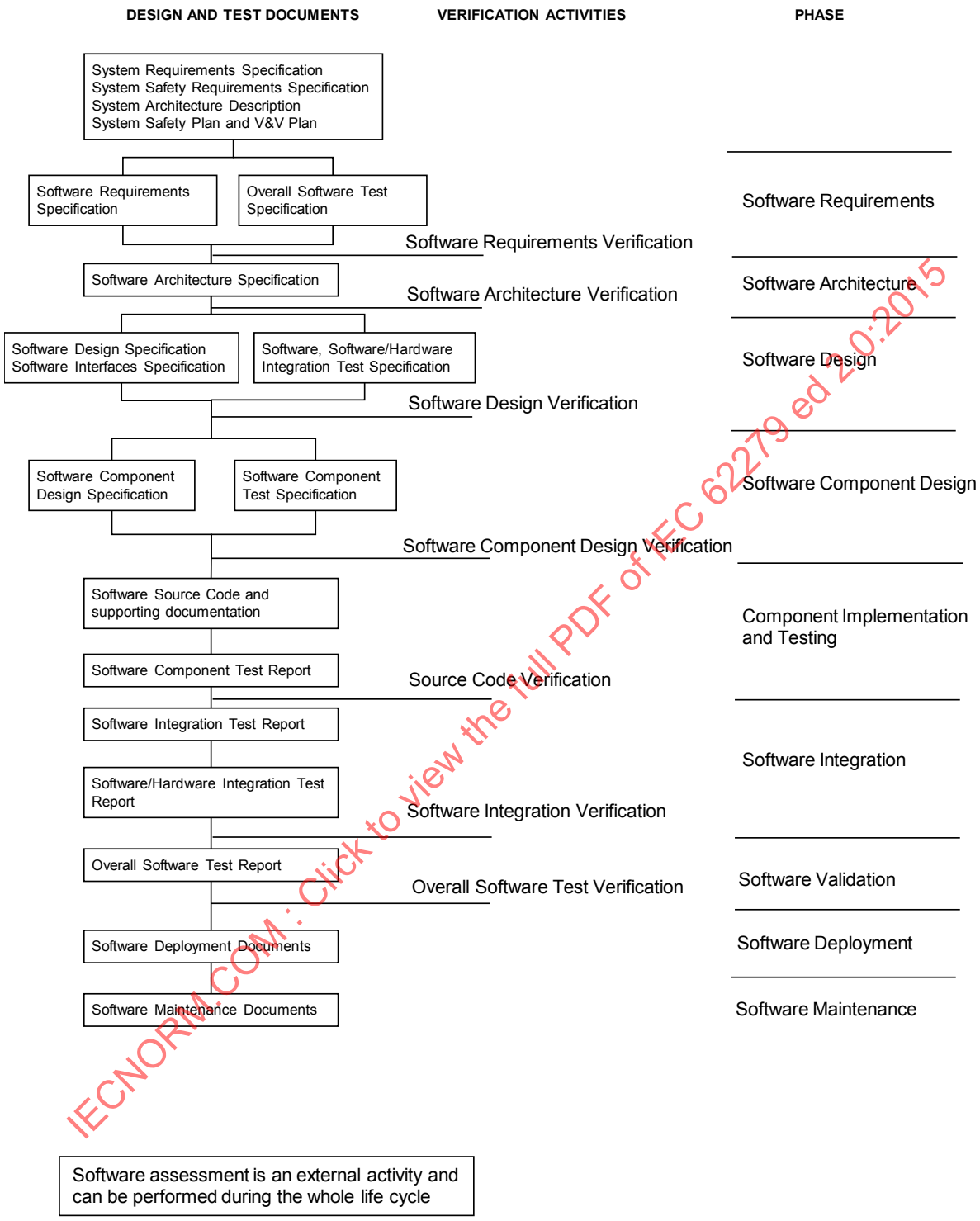
5.3.2.10 Il doit être fait référence à chaque entité ou concept en utilisant le même nom ou la même description dans tous les documents.

5.3.2.11 Le contenu de tous les documents doit être enregistré dans un format approprié à la manipulation, au traitement et au stockage.

5.3.2.12 Lorsque des documents produits par des rôles indépendants sont combinés en un seul document, la relation aux parties produites par un rôle indépendant quelconque doit être tracée au sein du document.

5.3.2.13 Les documents peuvent être regroupés ou divisés conformément à 5.3.2.12. Certaines étapes du développement peuvent être combinées, divisées ou, lorsque cela est justifié, éliminées, à la discrétion du Chef de projet et avec l'accord du Chargé de validation.

5.3.2.14 Tout cycle de vie et toute structure de documentation adoptés doivent satisfaire à tous les objets et exigences de la présente Norme.

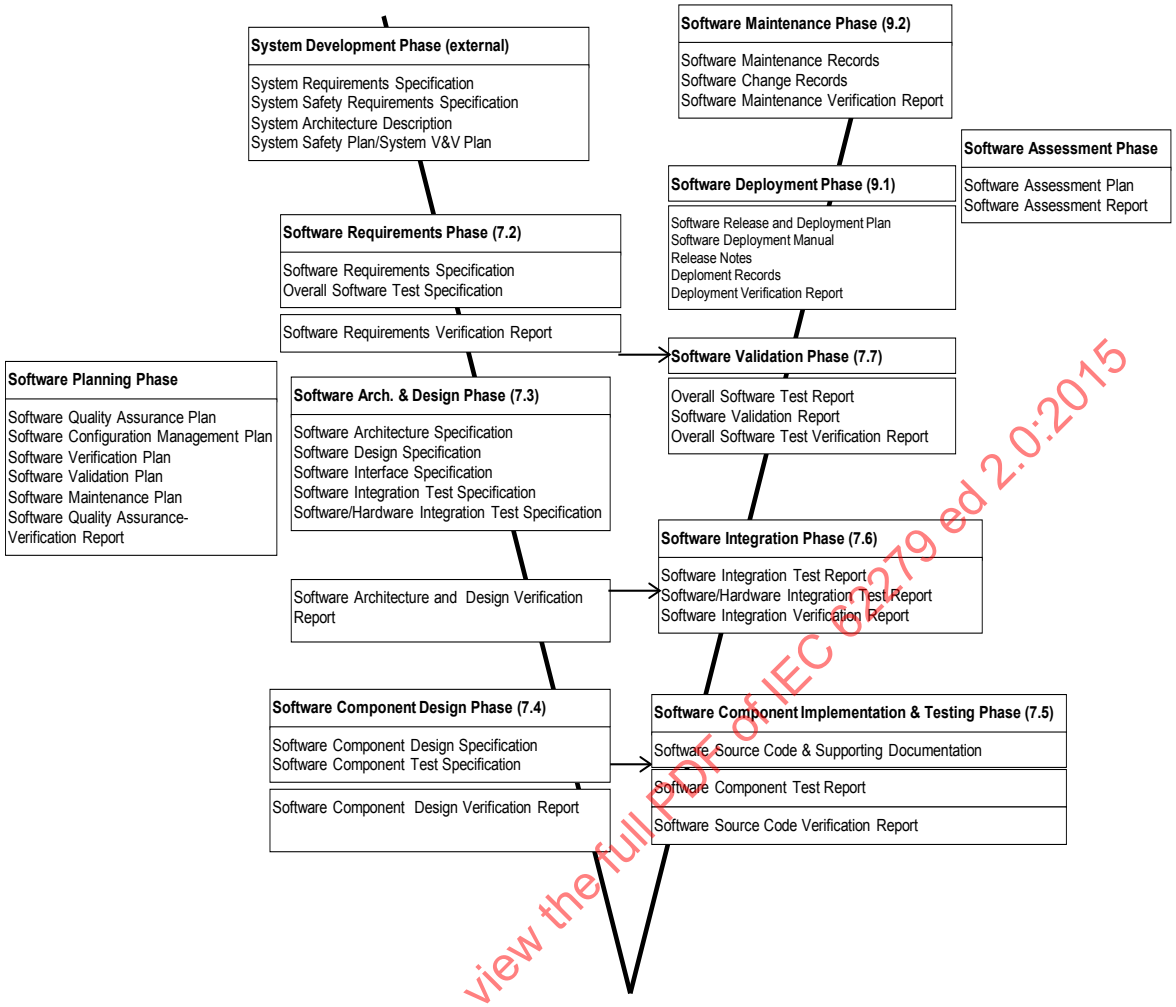


IEC

Anglais	Français
DESIGN AND TEST DOCUMENTS	DOCUMENTS DE CONCEPTION ET D'ESSAI
VERIFICATION ACTIVITIES	ACTIVITES DE VERIFICATION
PHASE	PHASE
System Requirements Specification	Spécification des Exigences du Système
System Safety Requirements Specification	Spécification des Exigences de Sécurité du Système
System Architecture Description	Description de l'Architecture du Système

Anglais	Français
System Safety Plan and V&V Plan	Plan de Sécurité du Système et Plan V&V
Software Requirements Specification	Spécification des Exigences du Logiciel
Overall Software Test Specification	Spécification des Essais d'Ensemble du Logiciel
Software Requirements Verification	Vérification des Exigences du Logiciel
Software Requirements	Exigences relatives au logiciel
Software Architecture Specification	Spécification de l'Architecture du Logiciel
Software Architecture Verification	Vérification de l'Architecture du Logiciel
Software Architecture	Architecture du logiciel
Software Design Specification	Spécification de la Conception du Logiciel
Software Interfaces Specification	Spécification des Interfaces Logiciel
Software, Software/Hardware Integration Test Specification	Spécification des essais d'Intégration de Logiciel, Logiciel/Matériel
Software Design	Conception du Logiciel
Software Design Verification	Vérification de la Conception du Logiciel
Software Component Design Specification	Spécification de la Conception des Composants logiciels
Software Component Test Specification	Spécification des essais des Composants Logiciels
Software Component Design	Conception des Composants Logiciels
Software Component Design Verification	Vérification de la Conception des Composants Logiciels
Software Source Code and supporting documentation	Code Source du Logiciel et Documentation de Support
Software Component Test Report	Rapport des Essais des Composants Logiciels
Component Implementation and Testing	Mise en œuvre et Essais des composants
Source Code Verification	Vérification du Code Source
Software Integration Test Report	Rapport des Essais d'Intégration du Logiciel
Software/Hardware Integration Test Report	Rapport des Essais d'Intégration Logiciel/Matériel
Software Integration	Intégration du Logiciel
Software Integration Verification	Vérification de l'Intégration du Logiciel
Overall Software Test Report	Rapport des Essais d'Ensemble du Logiciel
Software Validation	Validation du logiciel
Overall Software Test Verification	Vérification des Essais d'Ensemble du Logiciel
Software Deployment Documents	Documents de Déploiement du Logiciel
Software Deployment	Déploiement du logiciel
Software Maintenance Documents	Documents de Maintenance du Logiciel
Software Maintenance	Maintenance du Logiciel
Software assessment is an external activity and can be performed during the whole life cycle	L'évaluation du logiciel est une activité externe et peut être accomplie pendant tout le cycle de vie

Figure 3 – Illustration du Cycle de vie de développement 1



IEC

Anglais	Français
Software Planning Phase	**Phase de Planification du Logiciel**
Software Quality Assurance Plan	Plan d'Assurance Qualité du Logiciel
Software Configuration Management Plan	Plan de Gestion de la Configuration du Logiciel
Software Verification Plan	Plan de Vérification du Logiciel
Software Validation Plan	Plan de Validation du Logiciel
Software Maintenance Plan	Plan de Maintenance du Logiciel
Software Quality Assurance Verification Report	Rapport de Vérification d'Assurance Qualité du Logiciel
System Development Phase (external)	**Phase de Développement du Système (externe)**
System Requirements Specification	Spécification des Exigences du Système
System Safety Requirements Specification	Spécification des Exigences de Sécurité du Système
System Architecture Description	Description de l'Architecture du Système
System Safety Plan/System V&V Plan	Plan de Sécurité du Système / Plan V&V du Système
Software Requirements Phase (7.2)	**Phase d'Exigences du Logiciel (7.2)**
Software Requirements Specification	Spécification des Exigences du Logiciel
Overall Software Test Specification	Spécification des Essais d'Ensemble du Logiciel
Software Requirements Verification Report	Rapport de Vérification des Exigences du Logiciel
Software Arch. & Design Phase (7.3)	**Phase d'Arch. et Conception du Logiciel (7.3)**
Software Architecture Specification	Spécification de l'Architecture du Logiciel

Anglais	Français
Software Design Specification	Spécification de la Conception du Logiciel
Software Interface Specification	Spécification des Interfaces Logiciel
Software Integration Test Specification	Spécification des essais d'Intégration du Logiciel
Software/Hardware Integration Test Specification	Spécification des essais d'Intégration Logiciel/Matériel
Software Architecture and Design Verification Report	Rapport de Vérification de l'Architecture et de la Conception du Logiciel
Software Component Design Phase (7.4)	**Phase de Conception des Composants logiciels (7.4)**
Software Component Design Specification	Spécification de la Conception des Composants logiciels
Software Component Test Specification	Spécification des essais des Composants Logiciels
Software Component Design Verification Report	Rapport de Vérification de la Conception des Composants Logiciels
Software Maintenance Phase (9.2)	**Phase de Maintenance du Logiciel (9.2)**
Software Maintenance Records	Registres de Maintenance du Logiciel
Software Change Records	Registres des Modifications du Logiciel
Software Maintenance Verification Report	Rapport de Vérification de la Maintenance du Logiciel
Software Deployment Phase (9.1)	**Phase de Déploiement du Logiciel (9.1)**
Software Release and Deployment Plan	Plan de livraison et de déploiement du Logiciel
Software Deployment Manual	Manuel de Déploiement du Logiciel
Release Notes	Fiche de livraison
Deploiment Records	Registres de Déploiement
Deployment Verification Report	Rapport de Vérification du Déploiement
Software Validation Phase (7.7)	**Phase de Validation du Logiciel (7.7)**
Overall Software Test Report	Rapport des Essais d'Ensemble du Logiciel
Software Validation Report	Rapport de Validation du Logiciel
Overall Software Test Verification Report	Rapport de Vérification des Essais d'Ensemble du Logiciel
Software Integration Phase (7.6)	**Phase d'Intégration du Logiciel (7.6)**
Software Integration Test Report	Rapport des Essais d'Intégration du Logiciel
Software/Hardware Integration Test Report	Rapport des Essais d'Intégration Logiciel/Matériel
Software Integration Verification Report	Rapport de Vérification de l'Intégration du Logiciel
Software Component Implementation & Testing Phase (7.5)	**Phase de Mise en œuvre et des Essais des Composants logiciels (7.5)**
Software Source Code & Supporting Documentation	Code Source du Logiciel et Documentation de Support
Software Component Test Report	Rapport des Essais des Composants Logiciels
Software Source Code Verification Report	Rapport de Vérification du Code Source du Logiciel
Software Assessment Phase	**Phase d'Évaluation du Logiciel**
Software Assessment Plan	Plan d'Évaluation du Logiciel
Software Assessment Report	Rapport d'Évaluation du Logiciel

Figure 4 – Illustration du Cycle de vie de développement 2

6 Assurance du logiciel

6.1 Essais du logiciel

6.1.1 Objet

Le but des essais du logiciel, tels qu'effectués par le Chargé des essais et/ou le Chargé d'intégration, est de s'assurer du comportement ou de la performance du logiciel par rapport à la spécification d'essai correspondante jusqu'au périmètre atteignable par la couverture des essais choisie.

6.1.2 Documents en entrée

L'ensemble de la documentation nécessaire du système, du matériel et du logiciel telle que spécifiée dans le Plan de vérification du logiciel.

6.1.3 Documents en sortie

a) Spécification des Essais d'Ensemble du Logiciel

b) Rapport des Essais d'Ensemble du Logiciel

c) Spécification des essais d'Intégration du Logiciel

d) Rapport des Essais d'Intégration du Logiciel

e) Spécification des essais d'Intégration Logiciel/Matériel

f) Rapport des Essais d'Intégration Logiciel/Matériel

g) Spécification des essais des Composants Logiciels

h) Rapport des Essais des Composants Logiciels

6.1.4 Exigences

6.1.4.1 Les essais effectués par d'autres entités telles que le Gestionnaire des exigences, le Concepteur ou le Réalisateur, s'ils sont pleinement documentés et totalement conformes aux exigences suivantes, peuvent être acceptés par le Chargé de vérification.

6.1.4.2 Le matériel de mesure utilisé pour les essais doit être étalonné de manière appropriée. Tous les outils, matériels ou logiciels utilisés pour les essais doivent être démontrés adaptés aux objectifs.

6.1.4.3 Les essais du logiciel doivent être documentés par une Spécification d'essai et un Rapport d'essai, tels que définis dans ce qui suit.

6.1.4.4 Chaque Spécification d'essai doit documenter ce qui suit:

a) les objectifs d'essai;

b) les scénarios d'essai, les données d'essai et les résultats attendus;

c) les types d'essai à effectuer;

d) l'environnement d'essai, les outils, la configuration et les programmes;

e) les critères d'essai sur lesquels l'achèvement de l'essai sera jugé;

f) les critères à satisfaire et les degrés de couverture des essais à atteindre;

g) les rôles et responsabilités du personnel impliqué dans le processus d'essai;

h) les exigences couvertes par la Spécification d'essai;

i) la sélection et l'utilisation du matériel d'essai du logiciel.

6.1.4.5 Un Rapport d'essai doit être émis comme suit:

a) le Rapport des Essais doit mentionner les noms des Chargés des essais, énoncer les résultats des essais et indiquer si, oui ou non, les objectifs des essais et les critères des

essais selon la Spécification d'essai ont été satisfaits. Les échecs doivent être documentés et récapitulés;

b) les scénarios d'essai et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine en vue d'une analyse ultérieure;

c) les essais doivent être reproductibles et, dans la mesure du possible, exécutés par des moyens automatiques;

d) les scripts d'essai pour l'exécution automatique des essais doivent être vérifiés;

e) l'identité et la configuration de tous les éléments concernés (matériel utilisé, logiciel utilisé, équipements utilisés, étalonnage des équipements, informations de version du logiciel soumis à essai ainsi que les informations de version de la spécification d'essai) doivent être documentées;

f) une évaluation de la couverture des essais et de l'achèvement des essais doit être fournie et tout écart consigné.

6.2 Vérification du logiciel

6.2.1 Objet

L'objet de la vérification du logiciel est d'examiner et parvenir à un jugement fondé sur la preuve que les éléments de sortie (processus, documentation, logiciel ou application) d'une phase de développement spécifique satisfont aux exigences et aux plans en ce qui concerne la complétude, l'exactitude et la cohérence. Ces activités sont gérées par le Chargé de vérification.

6.2.2 Documents en entrée

Toute la documentation nécessaire concernant le système, le matériel et le logiciel.

6.2.3 Documents en sortie

a) Plan de Vérification du Logiciel

b) Rapport(s) de Vérification du Logiciel

c) Rapport de vérification concernant l'Assurance Qualité du logiciel

6.2.4 Exigences

6.2.4.1 La vérification doit être documentée par au moins un Plan de Vérification du Logiciel et un ou plusieurs Rapports de vérification (relatifs au processus).

6.2.4.2 Un Plan de Vérification du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base de la documentation nécessaire.

Les exigences de 6.2.4.3 à 6.2.4.9 se rapportent au Plan de Vérification du Logiciel.

6.2.4.3 Le Plan de Vérification du Logiciel doit décrire les activités à exécuter pour assurer une vérification correcte et la prise en charge correcte des besoins particuliers en matière de conception ou autre vérification.

6.2.4.4 Pendant le développement (et selon la taille du système), il est admis que le plan soit divisé en un certain nombre de sous-documents qui seront complétés à mesure que les besoins détaillés de vérification deviennent plus clairs.

6.2.4.5 Le Plan de Vérification du Logiciel doit documenter tous les critères, techniques et outils à utiliser dans le processus de vérification. Le Plan de Vérification du Logiciel doit inclure les techniques et mesures choisies dans le Tableau A.5, le Tableau A.6, le Tableau A.7 et le Tableau A.8. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8, 4.9 et 4.10.

6.2.4.6 Le Plan de Vérification du Logiciel doit décrire les activités à exécuter pour assurer l'exactitude et la cohérence vis-à-vis des entrées de chaque phase. Ces activités comprennent la revue, les essais et l'intégration.

6.2.4.7 Dans chaque phase de développement, il doit être montré que les exigences en matière de fonctionnement, de performance et de sécurité sont respectées.

6.2.4.8 Les résultats de chaque vérification doivent être conservés dans un format défini ou référencé dans le Plan de Vérification du Logiciel.

6.2.4.9 Le Plan de Vérification du Logiciel doit inclure les éléments suivants:

a) la sélection des stratégies et des techniques de vérification (pour éviter de compliquer inutilement l'évaluation de la vérification et des essais, la préférence doit être donnée à la sélection de techniques qui sont elles-mêmes facilement analysables);

b) sélection de techniques issues du Tableau A.5, du Tableau A.6, du Tableau A.7 et du Tableau A.8;

c) la sélection et la documentation des activités de vérification;

d) l'évaluation des résultats de vérification obtenus;

e) l'évaluation des exigences de sécurité et de robustesse;

f) les rôles et responsabilités du personnel impliqué dans le processus de vérification;

g) le degré de couverture des essais fonctionnels qu'il est requis d'atteindre;

h) la structure et le contenu de chaque étape de vérification, notamment pour la Vérification des Exigences du Logiciel (7.2.4.22), la Vérification de l'Architecture et de la Conception du Logiciel (7.3.4.41, 7.3.4.42), la Vérification des Composants logiciels (7.4.4.13), la Vérification du Code Source du Logiciel (7.5.4.10), la Vérification de l'Intégration (7.6.4.13) d'une manière qui facilite la revue par rapport au Plan de Vérification du Logiciel.

6.2.4.10 Un Rapport de Vérification d'Assurance Qualité du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 6.2.2.

L'exigence en 6.2.4.12 se rapporte au Rapport de Vérification d'Assurance Qualité du Logiciel.

6.2.4.11 Le Chargé de vérification doit s'assurer que la vérification du Plan de Vérification du logiciel est accomplie par une personne compétente en la matière. Pour être conforme aux règles de bonnes pratiques et d'indépendance selon la présente norme, le Chargé de vérification ne doit pas vérifier lui-même le Plan de Vérification du Logiciel, s'il en est l'auteur.

6.2.4.12 Une fois que le Plan de Vérification du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Plan de Vérification du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 6.2.4.3 à 6.2.4.9,

b) la cohérence interne du Plan de Vérification du Logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification d'Assurance Qualité du Logiciel.

6.2.4.13 Tous les éventuels Rapports de Vérification du Logiciel doivent être rédigés, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée. Ces rapports peuvent être partitionnés pour la clarté et la commodité et doivent suivre le Plan de Vérification du Logiciel. L'exigence en 6.2.4.14 se rapporte aux Rapports de Vérification du Logiciel.

6.2.4.14 Chaque Rapport de Vérification du Logiciel doit documenter ce qui suit:

a) l'identité et la configuration des éléments vérifiés ainsi que les noms des Chargés de vérification;

b) les éléments qui ne sont pas conformes aux spécifications;

c) les composants, les données, les structures et les algorithmes mal adaptés au problème;

d) les erreurs ou déficiences détectées;

e) le respect du Plan de Vérification du Logiciel ou l'écart par rapport à celui-ci (en cas d'écart le Rapport de Vérification doit expliquer s'il est critique ou non);

f) les hypothèses éventuelles;

g) un récapitulatif des résultats de vérification.

6.3 Validation du logiciel

6.3.1 Objet

6.3.1.1 L'objet de la validation du logiciel est de démontrer que les processus et leurs sorties sont tels que le logiciel a le niveau défini d'intégrité de la sécurité logicielle, qu'il satisfait aux exigences relatives au logiciel et qu'il est adapté à l'application prévue. Cette activité est exécutée par le Chargé de validation.

6.3.1.2 Les principales activités de validation sont de démontrer par l'analyse et/ou des essais que toutes les exigences relatives au logiciel sont spécifiées, mises en œuvre, soumises à essai et satisfaites comme requis par le niveau de SIL applicable et d'évaluer la criticité pour la sécurité de toutes les anomalies et non-conformités en se basant sur les résultats des revues, des analyses et des essais.

6.3.2 Documents en entrée

L'ensemble de la documentation du système, du matériel et du logiciel telle que spécifiée dans la présente Norme.

6.3.3 Documents en sortie

a) Plan de Validation du Logiciel

b) Rapport de Validation du Logiciel

6.3.4 Exigences

6.3.4.1 Les activités de Validation du Logiciel doivent être développées et exécutées, avec leurs résultats évalués, par un Chargé de validation ayant un niveau approprié d'indépendance tel que défini en 5.1.

6.3.4.2 La validation doit être documentée avec, au moins, un Plan de Validation du Logiciel et un Rapport de Validation du Logiciel, tels que définis ci-après.

6.3.4.3 Un Plan de Validation du Logiciel doit être rédigé, sous la responsabilité du Chargé de validation, sur la base des documents en entrée.

Les exigences de 6.3.4.4 à 6.3.4.6 se rapportent au Plan de Validation du Logiciel.

6.3.4.4 Le Plan de Validation du Logiciel doit inclure un résumé justifiant le choix de la stratégie de validation. La justification doit inclure, selon le niveau requis d'intégrité de la sécurité logicielle, la prise en considération:

a) de techniques manuelles et/ou automatisées;

b) de techniques statiques et/ou dynamiques;

c) de techniques analytiques et/ou statistiques;

d) des essais dans un environnement réel et/ou simulé.

6.3.4.5 Le Plan de Validation du Logiciel doit identifier les étapes nécessaires pour démontrer l'adéquation de toute Spécification de Logiciel pour satisfaire aux exigences de sécurité exposées dans la Spécification des exigences de sécurité du système.

6.3.4.6 Le Plan de Validation du Logiciel doit identifier les étapes nécessaires pour démontrer l'adéquation de la Spécification des Essais d'Ensemble du Logiciel en tant qu'essai par rapport à la Spécification des Exigences du logiciel.

6.3.4.7 Un Rapport de Validation du Logiciel doit être rédigé, sous la responsabilité du Chargé de validation, sur la base des documents en entrée.

Les exigences de 6.3.4.8 à 6.3.4.11 se rapportent au Rapport de Validation du Logiciel.

6.3.4.8 Les résultats de la validation doivent être documentés dans le Rapport de Validation du Logiciel.

6.3.4.9 Le chargé de validation doit contrôler que le processus de vérification est complet.

6.3.4.10 Le Rapport de Validation du Logiciel doit énoncer de manière exhaustive la configuration du logiciel qui a été validée.

6.3.4.11 Le Rapport de Validation doit clairement identifier toutes les éventuelles déficiences connues du logiciel et l'impact qu'elles peuvent avoir sur l'utilisation du logiciel.

6.3.4.12 L'entité chargée de la Validation doit fournir une revue des documents en sortie issus de 6.3.3.

6.3.4.13 Une fois que le Plan de Validation du Logiciel a été défini, la revue de ce document doit porter sur:

a) le fait que le Plan de Validation du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 6.3.4.4 à 6.3.4.6,

b) la cohérence interne du Plan de Validation du Logiciel.

6.3.4.14 Une fois que le Rapport de Validation du Logiciel a été établi, la revue de ce document doit porter sur:

a) le fait que le Rapport de Validation du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 6.3.4.8 à 6.3.4.11 et de 7.7.4.8 à 7.7.4.12,

b) la cohérence interne du Rapport de Validation du Logiciel.

6.3.4.15 Le Chargé de validation doit être habilité à exiger ou exécuter des revues, analyses et essais supplémentaires.

6.3.4.16 Le logiciel ne doit être diffusé pour l'exploitation qu'après autorisation par le Chargé de validation.

6.3.4.17 Il est permis d'utiliser la simulation et la modélisation pour compléter le processus de validation.

6.4 Évaluation du logiciel

6.4.1 Objet

6.4.1.1 Évaluer que les processus du cycle de vie et leurs sorties sont tels que le logiciel est conforme aux niveaux définis d'intégrité de la sécurité logicielle 1 à 4 et qu'il est adapté à son application prévue.

6.4.1.2 Pour les logiciels SIL 0, les exigences de la présente norme doivent être satisfaites, mais lorsqu'un certificat énonçant la conformité à l'ISO 9001 est disponible, aucune évaluation n'est requise.

6.4.2 Documents en entrée

a) Spécification des Exigences de Sécurité du Système

b) Spécification des Exigences du Logiciel

c) Tous les autres documents nécessaires pour mener à bien le processus d'évaluation.

6.4.3 Documents en sortie

a) Plan d'Évaluation du Logiciel

b) Rapport d'Évaluation du Logiciel

6.4.4 Exigences

6.4.4.1 L'évaluation du logiciel doit être menée à bien par un Chargé d'évaluation qui est indépendant comme décrit en 5.1.2.6 et en 5.1.2.7.

6.4.4.2 Un logiciel muni d'un Rapport d'Évaluation de Logiciel délivré par un autre Chargé d'évaluation n'a pas à faire l'objet d'une nouvelle évaluation. Le Chargé d'évaluation doit s'assurer que le logiciel est adapté à son utilisation prévue dans l'environnement prévu et que l'évaluation précédente énonçait que le logiciel a atteint un niveau d'intégrité de sécurité au moins égal au niveau requis.

6.4.4.3 Le Chargé d'évaluation doit avoir accès à l'ensemble de la documentation associée au projet tout au long du processus de développement.

6.4.4.4 Un Plan d'Évaluation du Logiciel doit être rédigé, sous la responsabilité du Chargé d'évaluation, sur la base des documents en entrée issus de 6.4.2. Lorsque cela est approprié, il est permis d'utiliser une procédure ou Plan d'évaluation de logiciel générique documenté(e) existant(e). L'exigence en 6.4.4.5 se rapporte au Plan d'Évaluation du Logiciel.

6.4.4.5 Le Plan d'Évaluation du Logiciel doit comporter le périmètre suivant:

a) les aspects dont traite l'évaluation;

b) les activités tout au long du processus d'évaluation et leur lien séquentiel aux activités d'ingénierie;

c) les documents à prendre en considération;

d) les énoncés des critères d'acceptation et de rejet et la manière de traiter des cas de non-conformité;

e) les exigences relatives au contenu et à la forme du Rapport d'Évaluation du Logiciel.

6.4.4.6 L'entité chargée de l'évaluation doit fournir une revue de l'évaluation sur la base des documents en entrée issus de 6.4.2.

L'exigence en 6.4.4.7 se rapporte à la vérification interne ou à la revue par des pairs de l'évaluation.

6.4.4.7 Une fois que le Plan d'Évaluation du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Plan d'Évaluation du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 6.4.4.5,

b) la cohérence interne du Plan d'Évaluation du Logiciel.

6.4.4.8 Le Chargé d'évaluation doit évaluer si le logiciel du système est adapté à l'objectif prévu et s'il répond correctement aux problèmes de sécurité dérivés de la Spécification des Exigences de Sécurité du Système.

6.4.4.9 Le Chargé d'évaluation doit évaluer si un ensemble approprié de techniques issues de l'Annexe A, qui sont adaptées pour le développement prévu, a été sélectionné et appliqué en fonction du niveau requis d'intégrité de sécurité.

En outre, le Chargé d'évaluation doit tenir compte de la mesure dans laquelle chaque technique issue de l'Annexe A est appliquée, c'est-à-dire si elle s'applique à la totalité ou seulement à une partie du logiciel, et également rechercher des preuves indiquant qu'elle est appliquée correctement.

6.4.4.10 Le Chargé d'évaluation doit évaluer le système de gestion de la configuration et des modifications ainsi que les preuves de son utilisation et son application.

6.4.4.11 Le Chargé d'évaluation doit examiner la preuve de la compétence de l'équipe de projet selon l'Annexe B et doit évaluer l'organisation quant au développement du logiciel selon 5.1.

6.4.4.12 Pour tout logiciel contenant des conditions d'application relatives à la sécurité, le Chargé d'évaluation doit rechercher les écarts consignés, les non-conformités aux exigences et les non-conformités enregistrées si celles-ci ont un impact sur la sécurité et juger si la justification fournie par le projet est acceptable. Le résultat doit être énoncé dans le rapport d'évaluation.

6.4.4.13 Le Chargé d'évaluation doit évaluer les activités de vérification et de validation et les pièces justificatives.

6.4.4.14 Le Chargé d'évaluation doit donner son accord sur le périmètre et le contenu du Plan de Validation du Logiciel. Cet accord doit également énoncer la présence du chargé d'évaluation pendant les essais.

6.4.4.15 Le Chargé d'évaluation peut conduire des audits et inspections (par exemple: présence lors des essais) tout au long du processus de développement entier. Le Chargé d'évaluation peut demander un travail supplémentaire de vérification et de validation.

NOTE Il est avantageux d'impliquer le Chargé d'évaluation à un stade précoce du projet.

6.4.4.16 Un Rapport d'Évaluation du Logiciel doit être rédigé sous la responsabilité du Chargé d'évaluation. Les exigences de 6.4.4.17 à 6.4.4.19 se rapportent au Rapport d'Évaluation du Logiciel.

6.4.4.17 Le Rapport d'Évaluation du Logiciel doit satisfaire aux exigences du Plan d'Évaluation du Logiciel et fournir une conclusion et des recommandations.

6.4.4.18 Le Chargé d'évaluation doit enregistrer ses activités comme une base cohérente pour le Rapport d'Évaluation du Logiciel. Celles-ci doivent être résumées dans le Rapport d'Évaluation du Logiciel.

6.4.4.19 Le Chargé d'évaluation doit identifier et évaluer toute non-conformité aux exigences de la présente Norme et juger l'impact sur le résultat final. Ces non-conformités et le jugement à leur sujet doivent être énumérés dans le Rapport d'Évaluation du Logiciel.

6.5 Assurance qualité du logiciel

6.5.1 Objets

6.5.1.1 Identifier, superviser et contrôler toutes les activités, à la fois techniques et de gestion, qui sont nécessaires pour assurer que le logiciel possède la qualité requise. Cela est nécessaire pour fournir la démarche qualitative requise contre les défauts systématiques et s'assurer qu'une trace peut être laissée afin de permettre un déroulement efficace des activités de vérification et de validation.

6.5.1.2 Apporter la preuve que toutes les activités ci-dessus ont été menées à bien.

6.5.2 Documents en entrée

Tous les documents disponibles à chaque étape du cycle de vie.

6.5.3 Documents en sortie

a) Plan d'Assurance Qualité du Logiciel

b) Plan de Gestion de la Configuration du Logiciel, s'il n'est pas disponible au niveau du système

c) Rapport de vérification concernant l'Assurance Qualité du logiciel

NOTE Le Rapport d'Assurance Qualité du Logiciel est inclus dans le Rapport de Gestion de la Qualité défini dans l'IEC 62425.

6.5.4 Exigences

6.5.4.1 Tous les plans doivent être communiqués au commencement du projet et mis à jour pendant le cycle de vie.

6.5.4.2 Les organisations prenant part au développement du logiciel doivent mettre en œuvre et utiliser un Système d'Assurance Qualité conforme à l'ISO 9000, pour prendre en charge les exigences de la présente Norme. La certification ISO 9001 est hautement recommandée.

6.5.4.3 Un Plan de Vérification d'Assurance Qualité du Logiciel doit être rédigé, sous la responsabilité du Gestionnaire de l'assurance qualité, sur la base des documents en entrée de 6.5.2.

Les exigences de 6.5.4.4 à 6.5.4.6 se rapportent au Plan d'Assurance Qualité du Logiciel.

6.5.4.4 Un Plan d'Assurance Qualité du Logiciel doit être rédigé et doit être spécifique au projet. Il doit mettre en œuvre les exigences en 6.5.4.5.

6.5.4.5 Pour le moins, les éléments suivants doivent être spécifiés ou référencés dans le Plan d'Assurance Qualité du Logiciel.

a) Définition du modèle de cycle de vie

   1) activités et tâches élémentaires compatibles avec les plans (par exemple: Plan de Sécurité) qui ont été établis au niveau Système;
   2) critères d'entrée et de sortie de chaque activité;
   3) entrées et sorties de chaque activité;
   4) principales activités qualité;

5) l'entité responsable de chaque activité.

b) Structure de la documentation.

c) Contrôle de la documentation:

1) rôles impliqués dans la rédaction, le contrôle et l'approbation;

2) étendue de la distribution;

3) archivage.

d) Suivi et traçabilité des écarts.

e) Méthodes, mesures et outils pour l'assurance qualité en fonction des niveaux alloués d'intégrité de la sécurité (voir Annexe A).

f) Justifications, telles que définies de 4.7 à 4.9, que chaque combinaison de techniques ou de mesures sélectionnées conformément à l'Annexe A est appropriée au niveau défini d'intégrité de la sécurité logicielle.

Certaines des informations requises du Plan d'Assurance Qualité du Logiciel peuvent être contenues dans d'autres documents, tels qu'un Plan de Gestion de la Configuration de Logiciel séparé, un Plan de Maintenance, un plan de Vérification du Logiciel et un Plan de Validation du Logiciel. Les paragraphes du Plan d'Assurance Qualité du Logiciel doivent référencer les documents qui contiennent les informations. Dans tous les cas, le contenu de chaque paragraphe du Plan d'Assurance Qualité du Logiciel doit être spécifié, directement ou par référence à un autre document.

Les documents référencés doivent être revus afin de s'assurer qu'ils fournissent toutes les informations requises et qu'ils répondent complètement aux exigences de la présente Norme.

6.5.4.6 Les activités, actions, documents, etc. de l'assurance qualité requis par tous les paragraphes normatifs de la présente Norme doivent être spécifiés ou référencés dans le Plan d'Assurance Qualité du Logiciel et adaptés en fonction du projet spécifique.

6.5.4.7 Un Rapport de Vérification d'Assurance Qualité du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 6.5.2.

L'exigence en 6.5.4.8 se rapporte au Rapport de Vérification d'Assurance Qualité du Logiciel.

6.5.4.8 Une fois que le Plan d'Assurance Qualité du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Plan d'Assurance Qualité du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 6.5.4.4 à 6.5.4.6,

b) la cohérence interne du Plan d'Assurance Qualité du Logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification d'Assurance Qualité du Logiciel.

6.5.4.9 Chaque document de planification doit comporter un alinéa spécifiant les détails relatifs à sa propre mise à jour tout au long du projet: fréquence, responsabilité, méthode.

6.5.4.10 Chaque document du logiciel et chaque produit à fournir doivent être soumis au contrôle de configuration à partir du moment de sa première diffusion.

6.5.4.11 Les modifications apportées à tous les éléments sous le contrôle de la Gestion de Configuration doivent être autorisées et enregistrées.

6.5.4.12 En plus du développement de logiciel, le Système de Gestion de Configuration doit aussi couvrir l'environnement de développement du logiciel utilisé pendant le cycle de vie complet

Cette extension, nécessaire pour la reproductibilité du développement et pour les activités de maintenance, doit concerner tous les outils, traducteurs, fichiers de données et des essais, fichiers de paramétrage et plateformes matérielles supportant ces derniers.

6.5.4.13 Le fournisseur doit établir, documenter et maintenir des procédures pour le contrôle des fournisseurs externes, notamment

– des méthodes et des enregistrements significatifs visant à assurer que les logiciels fournis par des fournisseurs externes respectent les exigences définies. Les logiciels développés préalablement doivent être garantis conformes au niveau requis d'intégrité de la sécurité logicielle et de sûreté de fonctionnement. Les nouveaux logiciels doivent être développés et maintenus en conformité avec le Plan d'Assurance Qualité du Logiciel du fournisseur ou avec un Plan d'Assurance Qualité du Logiciel spécifique, préparé par le fournisseur externe en accord avec le Plan d'Assurance Qualité Logiciel du fournisseur,
– des méthodes et enregistrements correspondants visant à assurer que les exigences transmises au fournisseur externe sont adéquates et complètes.

6.5.4.14 La traçabilité des exigences doit être largement prise en considération dans la validation d'un système de sécurité et des moyens doivent être fournis pour permettre de le démontrer d'un bout à l'autre de toutes les phases du cycle de vie.

6.5.4.15 Dans le contexte de la présente Norme et dans une limite appropriée au niveau d'intégrité de la sécurité logicielle spécifié, la traçabilité doit concerner particulièrement

a) la traçabilité des exigences par rapport à la conception ou aux autres objets qui y répondent,
b) la traçabilité des objets de la conception jusqu'aux objets de la mise en œuvre qui les instancient,
c) la traçabilité des exigences et des objets de conception vers les essais (composant, intégration, essai d'ensemble) et les analyses qui les vérifient.

La traçabilité doit être soumise à la Gestion de Configuration.

6.5.4.16 Dans des cas particuliers (par exemple celui des logiciels préexistants ou logiciels prototypés), la traçabilité peut être établie après la mise en œuvre et/ou la documentation du code mais avant la vérification/validation. Dans ces cas, il doit être démontré que la vérification/validation est aussi efficace qu'elle l'aurait été avec la traçabilité sur toutes les phases.

6.5.4.17 Il doit être démontré que les objets de spécification des exigences, de conception ou de mise en œuvre qui ne peuvent pas être correctement tracés n'ont aucune influence sur la sécurité ou l'intégrité du système.

6.6 Contrôle des modifications et des évolutions

6.6.1 Objets

6.6.1.1 Assurer que le logiciel fonctionne comme requis, en préservant l'intégrité de la sécurité logicielle et la sûreté de fonctionnement lorsqu'on procède à des modifications du logiciel.

6.6.1.2 Ces objectifs sont gérés par le Gestionnaire de la Configuration.

6.6.2 Documents en entrée

a) Plan d'Assurance Qualité du Logiciel

b) Plan de Gestion de la Configuration du Logiciel

c) Tout document approprié de conception, de réalisation et d'analyse

d) Demandes de modification

e) Analyse de l'impact des modifications et autorisation des modifications

6.6.3 Documents en sortie

a) Tous les documents en entrée modifiés

b) Registres des modifications du logiciel (voir 9.2.4.11)

c) Enregistrements de la nouvelle Configuration

6.6.4 Exigences

6.6.4.1 Le Processus de Gestion des Modifications doit définir au moins les aspects suivants:

a) la documentation nécessaire pour le compte-rendu des problèmes et/ou les actions correctives, dans le but de permettre le retour d'informations vers la direction responsable;

b) l'analyse des informations recueillies dans les comptes-rendus de problèmes pour en identifier les causes;

c) les pratiques à suivre pour le compte rendu, le suivi et la résolution des problèmes identifiés à la fois au cours de la phase de développement et au cours de la phase de maintenance du logiciel;

d) les responsabilités organisationnelles spécifiques par rapport au développement et à la maintenance du logiciel;

e) la manière d'appliquer des contrôles pour s'assurer que des actions correctives sont prises et qu'elles sont efficaces;

f) l'analyse d'impact de l'effet des modifications sur le composant de logiciel en développement ou déjà livré;

g) l'analyse d'impact doit énoncer la revérification, la revalidation et la réévaluation nécessaires pour le changement;

h) lorsque plusieurs modifications sont apportées, l'analyse d'impact doit envisager l'impact cumulé;

NOTE Plusieurs modifications peuvent nécessiter cumulativement un nouvel essai complet.

i) l'autorisation avant la mise en œuvre.

6.6.4.2 Toutes les modifications doivent déclencher un retour à une phase appropriée du cycle de vie. Toutes les phases ultérieures doivent être menées à bien selon les procédures spécifiées pour les phases spécifiques en conformité avec les exigences de la présente Norme.

6.7 Outils et langages

6.7.1 Objets

L'objet est de fournir la preuve que les défaillances potentielles des outils ne nuisent pas à la sécurité des données produites par l'ensemble des outils sans que cela ne soit détecté par des mesures techniques et/ou organisationnelles extérieures à l'outil. Pour ce faire, les outils logiciels sont catégorisés en trois classes, à savoir T1, T2 et T3 (voir les définitions en 3.1).

Lorsque des outils sont utilisés en remplacement d'opérations manuelles, la preuve de l'intégrité de la sortie du jeu d'outils peut être présentée par les mêmes étapes de processus que si la sortie était produite dans une opération manuelle. Ces étapes de processus

pourraient être remplacées par des méthodes en variante si une argumentation sur l'intégrité de la sortie des outils est donnée et le niveau d'intégrité du logiciel n'est pas diminué par le remplacement.

6.7.2 Documents en entrée

Spécification des outils ou manuel.

6.7.3 Documents en sortie

Rapport de validation des outils (si cela s'avère nécessaire, voir 6.7.4.4 ou 6.7.4.6).

6.7.4 Exigences

6.7.4.1 Les outils logiciels doivent être choisis comme partie cohérente des activités de développement du logiciel.

NOTE Des outils appropriés d'aide au développement du logiciel sont utilisés afin d'augmenter l'intégrité du logiciel en réduisant la probabilité d'introduction ou de non-détection de défauts au cours du développement. Les exemples d'outils pertinents pour les phases de cycle de vie de développement du logiciel sont notamment

a) les outils de transformation ou de traduction qui convertissent une représentation de logiciel ou de conception (par exemple: un texte ou un schéma) d'un niveau d'abstraction en un autre les outils d'affinement de la conception, les compilateurs, les assembleurs, les éditeurs de liens, les chargeurs et les outils de génération de code,

b) les outils de vérification et de validation tels que les logiciels d'analyse statique, les contrôleurs de couverture des essais, les aides de démonstration de théorèmes, les simulateurs et les vérificateurs de modèle,

c) les outils de diagnostic utilisés pour maintenir et surveiller le logiciel dans des conditions de fonctionnement,

d) les outils d'infrastructure tels que les systèmes d'aide au développement,

e) les outils de contrôle de configuration tels que les outils de contrôle des versions,

f) les outils de données d'application qui produisent ou maintiennent les données qui sont requises pour définir les paramètres et instancier les fonctions du système, par exemple, paramètres de fonction, plages d'instruments, niveaux d'alarme et de déclenchement, états de sortie à adopter à la défaillance, disposition géographique.

Les outils sélectionnés sont capables de coopérer. Dans ce contexte, des outils coopèrent si les grandeurs de sortie d'un outil ont le contenu et le format adéquats pour être injectées automatiquement en entrée dans un outil suivant, réduisant ainsi au maximum la possibilité d'introduire une erreur humaine dans la manipulation des résultats intermédiaires.

Les outils sont généralement sélectionnés et démontrés en fonction de leur compatibilité avec les besoins de l'application.

La disponibilité des outils adéquats pour fournir les services nécessaires tout au long de la durée de vie du logiciel est également prise en considération.

6.7.4.2 Le choix des outils des classes T2 et T3 doit être justifié (voir 7.3.4.12). La justification doit inclure l'identification de défaillances potentielles qui peuvent être injectées dans la sortie des outils et les mesures pour éviter ou gérer ces défaillances.

6.7.4.3 Tous les outils des classes T2 et T3 doivent avoir une spécification ou un manuel qui définit le comportement de l'outil et les éventuelles instructions ou contraintes concernant son utilisation.

6.7.4.4 Pour chaque outil de la classe T3, la preuve que sa sortie est conforme à la spécification de la sortie ou que ses défaillances sont détectées doit être disponible. La preuve peut être fondée sur les mêmes justifications qui seraient fournies pour un processus manuel mis en œuvre à la place de l'outil ou sur un argumentaire si des types de preuve différents sont possibles (par exemple: validation de l'outil). La preuve peut aussi être basée sur

a) une combinaison appropriée d'utilisations antérieures réussies dans des environnements similaires et pour des applications similaires (au sein de l'organisation ou d'autres organisations),

b) la validation d'outil telle que spécifiée en 6.7.4.5,

c) divers codes redondants qui permettent la détection et la maîtrise des défaillances provoquant l'introduction de défauts par un outil,

d) la conformité aux niveaux d'intégrité de sécurité dérivés de l'analyse de risque appliquée au processus et aux procédures, y compris les outils,

e) d'autres méthodes appropriées pour éviter ou détecter et maîtriser les défaillances introduites par des outils.

NOTE 1 Un historique des versions peut apporter l'assurance de la maturité de l'outil et un enregistrement des erreurs/ambiguïtés associées à son utilisation dans l'environnement.

NOTE 2 La preuve énumérée pour T3 peut aussi servir pour les outils T2 pour juger du caractère correct des résultats.

6.7.4.5 Les résultats de la validation d'outils doivent être documentés et couvrir les résultats suivants:

a) un enregistrement des activités de validation;

b) la version du manuel d'outils utilisé;

c) les fonctions de l'outil validées;

d) les outils et le matériel utilisés;

e) les résultats des activités de validation; les résultats documentés de la validation doivent énoncer soit que le logiciel a passé avec succès la validation, soit les raisons de son rejet;

f) les scénarios d'essai et leurs résultats, en vue d'une analyse ultérieure;

g) les divergences entre les résultats attendus et les résultats réels.

6.7.4.6 Lorsque la preuve de la conformité en 6.7.4.4 n'est pas disponible, des mesures effectives doivent exister pour maîtriser les défaillances du logiciel exécutable de sécurité qui résultent de défauts imputables à l'outil.

NOTE 1 Un exemple en est la génération de codes redondants divers qui permettent la détection et la maîtrise de défaillances provoquant l'introduction de défauts par un traducteur.

NOTE 2 À titre d'exemple, l'adéquation aux besoins d'un compilateur non certifié peut être justifiée comme suit.

Le code objet produit par le compilateur a été soumis à une combinaison d'essais, de contrôles et d'analyses qui sont capables d'assurer le caractère correct du code dans la mesure où il est compatible avec le niveau visé d'intégrité de sécurité. En particulier, les points ci-après s'appliquent à tous les essais, contrôles et analyses.

– Il a été montré que les essais avaient une couverture suffisamment élevée du code mis en œuvre. S'il existe un code inaccessible par des essais, il a été montré par des contrôles ou des analyses que la fonction concernée s'exécutait correctement lorsque le code était atteint sur la cible.

– Des contrôles et analyses ont été appliqués au code objet et se sont avérés capables de détecter les types d'erreurs susceptibles de résulter d'un défaut du compilateur.

– Plus aucune traduction avec le compilateur n'a eu lieu après les essais, le contrôle et l'analyse.

– Si une compilation ou traduction supplémentaire est effectuée, tous les essais, contrôles et analyses seront répétés.

6.7.4.7 La représentation du logiciel ou de la conception (y compris le langage de programmation) choisie doit

a) avoir un traducteur qui a été évalué quant à son adéquation aux besoins, y compris, le cas échéant, l'évaluation par rapport aux étalons internationaux ou nationaux,

b) être compatible avec les caractéristiques de l'application,

c) contenir des caractéristiques qui facilitent la détection des erreurs de conception ou de programmation,

d) prendre en charge des caractéristiques qui sont compatibles avec la méthode de conception.

Un langage de programmation est l'une des classes de représentations de logiciel ou de conception. Un Traducteur convertit une représentation de logiciel ou de conception (par exemple: un texte ou un schéma) d'un niveau d'abstraction en un autre. Les exemples de Traducteurs sont notamment les outils d'affinement de la conception, les compilateurs, les assembleurs, les éditeurs de liens, les chargeurs et les outils de génération de code.

L'évaluation d'un Traducteur peut être effectuée pour un projet d'application spécifique ou pour une classe d'applications. Dans le second cas, toutes les informations nécessaires sur l'outil concernant son utilisation prévue et appropriée doivent être mises à la disposition de l'utilisateur de l'outil. L'évaluation de l'outil pour un projet spécifique peut alors se réduire à une vérification de l'adéquation générale de l'outil au projet et de la conformité à la «spécification ou manuel» (c'est-à-dire l'utilisation correcte de l'outil). L'utilisation correcte peut inclure des activités de vérification supplémentaires dans les limites du projet spécifique.

Un équipement de validation peut être utilisé pour évaluer l'adéquation aux besoins d'un traducteur en fonction de critères définis, qui doivent inclure des exigences fonctionnelles et non fonctionnelles. Pour les exigences fonctionnelles du traducteur, les essais dynamiques peuvent constituer la technique de validation principale. Si cela est possible, une série d'essais automatiques doit être utilisée.

6.7.4.8 Lorsque 6.7.4.7 ne peut pas être complètement satisfait, l'adéquation aux besoins du langage et toute mesure complémentaire traitant des éventuelles insuffisances identifiées du langage doivent être justifiées et évaluées.

NOTE Voir NOTE 2 en 6.7.4.6.

6.7.4.9 Lorsqu'une génération automatique de code ou une traduction automatique similaire a lieu, l'adéquation du Traducteur automatique au développement de logiciel de sécurité doit être évaluée à l'endroit du cycle de vie de développement auquel les outils au développement sont choisis.

6.7.4.10 La gestion de configuration doit assurer que pour les outils des classes T2 et T3, seules des versions justifiées sont utilisées.

6.7.4.11 Chaque nouvelle version d'un outil utilisé doit être justifiée (voir Tableau 1). Cette justification peut reposer sur une preuve fournie pour une version antérieure s'il est fourni des preuves suffisantes que

a) les différences fonctionnelles (éventuelles) n'affecteront pas la compatibilité de l'outil avec le reste du jeu d'outils,

b) la nouvelle version n'est pas susceptible de contenir de nouveaux défauts inconnus significatifs.

NOTE La preuve que la nouvelle version n'est pas susceptible de contenir de nouveaux défauts inconnus significatifs peut reposer sur une identification crédible des changements apportés et sur une analyse des actions de vérification et de validation exécutées.

6.7.4.12 La relation entre les classes d'outils et les paragraphes applicables est définie dans le Tableau 1.

Tableau 1 – Relation entre les classes d'outils et les paragraphes applicables

Classe d'outils	Paragraphes applicables
T1	6.7.4.1
T2	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.10, 6.7.4.11
T3	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.4, 6.7.4.5 (ou 6.7.4.6), 6.7.4.7, 6.7.4.8, 6.7.4.9, 6.7.4.10, 6.7.4.11

NOTE La sélection et l'application des outils sont généralement alignées sur l'intégrité de la sécurité des produits en développement. La relation entre les classes d'outils et les méthodologies d'assurance appropriées à des SIL spécifiques pour les produits en développement où les outils sont utilisés est illustrée dans le Tableau 2.

Tableau 2 – Illustration de la relation entre classes d'outils et SIL de produit

Classe d'outils	Méthodologie d'Assurance d'Outil	SIL de produit en développement
T1	Pas exigée	Pas nécessaire
T2 et T3	1 – Preuve d'une utilisation réussie dans des environnements similaires ou 2 – Exécuter une bibliothèque de scénarios d'essai avec des résultats déterministes pour constater l'intégrité fonctionnelle	1-2
	1 – Suivre les exigences complètes de la présente norme qui sont appropriées au SIL d'application spécifique au développement ou à l'acquisition de l'outil ou 2 – Exécuter une bibliothèque de scénarios d'essai/séries d'essais largement reconnu(e)s avec des résultats déterministes pour constater l'intégrité fonctionnelle ou 3 – Appliquer divers outils au système et comparer la qualité de fonctionnement du produit de travail, en recherchant des différences	3-4

7 Développement de logiciel générique

7.1 Cycle de vie et documentation pour logiciel générique

7.1.1 Objets

Fournir une description du logiciel lui-même, depuis les niveaux supérieurs d'abstraction jusqu'à l'affinement des détails, afin de créer un cadre pour la démonstration de la sécurité obtenue ainsi que pour les actions de maintenance futures.

7.1.2 Exigences

7.1.2.1 Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, les documents énumérés dans le Tableau A.1 doivent être produits pour un logiciel générique.

7.1.2.2 La séquence de documents à fournir tels qu'ils sont décrits dans le Tableau A.1 reflète un modèle de cascade linéaire idéal. Ce modèle n'est toutefois pas destiné à servir de référence dans le sens de programmation et de liaison d'activités, car il serait difficile d'obtenir une stricte conformité dans la pratique. Des phases peuvent se chevaucher, mais les activités de vérification et de validation doivent démontrer la cohérence des entrées et des sorties (documents et logiciels) au sein des phases et entre les phases.

Cependant, le but principal de la documentation prévue est de fournir une description du logiciel lui-même, depuis les niveaux supérieurs d'abstraction jusqu'à l'affinement des détails, afin de créer un cadre pour la démonstration de la sécurité obtenue ainsi que pour les actions de maintenance futures.

7.2 Exigences relatives au logiciel

7.2.1 Objets

7.2.1.1 Décrire un jeu complet d'exigences pour les logiciels respectant l'ensemble des Exigences de Système et de Sécurité relatives aux logiciels et fournir un jeu complet de documents pour chaque phase ultérieure.

7.2.1.2 Décrire la Spécification des Essais d'Ensemble du Logiciel.

7.2.2 Documents en entrée

a) Spécification des Exigences du Système

b) Spécification des Exigences de Sécurité du Système

c) Description de l'Architecture du Système

d) Spécifications d'interface externe (par exemple: Spécification des interfaces Logiciel/Logiciel, Spécification des interfaces Logiciel/Matériel)

e) Plan d'Assurance Qualité du Logiciel

f) Plan de Validation du Logiciel

7.2.3 Documents en sortie

a) Spécification des Exigences du Logiciel

b) Spécification des Essais d'Ensemble du Logiciel

c) Rapport de Vérification des Exigences du Logiciel

7.2.4 Exigences

7.2.4.1 Une Spécification des Exigences du Logiciel doit être rédigée, sous la responsabilité du Gestionnaire des Exigences, sur la base des documents en entrée issus de 7.2.2.

Les exigences de 7.2.4.2 à 7.2.4.15 se rapportent à la Spécification des Exigences du Logiciel.

7.2.4.2 La Spécification des Exigences du Logiciel doit exprimer les propriétés requises du logiciel développé. Ces propriétés, qui sont toutes (à l'exception de la sécurité) définies dans la série ISO/IEC 25010, doivent inclure

a) la fonctionnalité (y compris le dimensionnement et la performance en temps de réponse),

b) la robustesse et la maintenabilité,

c) la sécurité (y compris les fonctions de sécurité et les niveaux d'intégrité de la sécurité logicielle qui leur sont associés),

d) l'efficacité,

e) l'utilisabilité,

f) la portabilité.

7.2.4.3 Le niveau d'intégrité de la sécurité logicielle doit être dérivé comme indiqué à l'Article 4 et enregistré dans la Spécification des Exigences du Logiciel.

7.2.4.4 Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, la Spécification des Exigences du Logiciel doit être exprimée et structurée de manière à être

a) complète, claire, précise, sans équivoque, susceptible d'être vérifiée, soumise à essai, maintenue et réalisée,

b) traçable en remontant à tous les documents en entrée.

7.2.4.5 La Spécification des Exigences du Logiciel doit inclure des modes d'expression et des descriptions qui sont compréhensibles par le personnel responsable impliqué dans le cycle de vie du logiciel.

7.2.4.6 La Spécification des Exigences du Logiciel doit identifier et documenter toutes les interfaces avec tout autre système, qu'il se trouve à l'intérieur ou à l'extérieur de l'équipement sous contrôle, y compris les opérateurs, chaque fois qu'une connexion directe existe ou qu'elle est prévue.

7.2.4.7 Tous les modes de fonctionnement applicables doivent être détaillés dans la Spécification des Exigences du Logiciel.

7.2.4.8 Tous les modes de comportement applicables des systèmes électroniques programmables, notamment le comportement de défaillance, doivent être documentés ou référencés (par exemple: documentation au niveau système) dans la Spécification des Exigences du Logiciel.

7.2.4.9 Toute contrainte entre le matériel et le logiciel doit être documentée ou référencée (par exemple: documentation au niveau système) dans la Spécification des Exigences du Logiciel.

7.2.4.10 Dans la limite requise par la description de la documentation du système, la Spécification des Exigences du Logiciel doit envisager l'autocontrôle logiciel et le contrôle du matériel par le logiciel. L'autocontrôle logiciel consiste en la détection et le compte rendu par le logiciel de ses propres défaillances et erreurs.

7.2.4.11 La Spécification des Exigences du Logiciel doit inclure des exigences concernant les essais périodiques des fonctions, dans la limite requise par la Spécification des Exigences de Sécurité du Système.

7.2.4.12 La Spécification des Exigences du Logiciel doit inclure des exigences permettant de soumettre à essai toutes les fonctions de sécurité pendant le fonctionnement global du système, dans la limite requise par la Spécification des Exigences de Sécurité du Système.

7.2.4.13 Toutes les fonctions que le logiciel est tenu d'exécuter, notamment celles liées à la réalisation du niveau requis d'intégrité de la sécurité du système, doivent être clairement identifiées dans la Spécification des Exigences du Logiciel.

7.2.4.14 Les éventuelles fonctions non liées à la sécurité que le logiciel est tenu d'exécuter doivent être clairement identifiées dans la Spécification des Exigences du Logiciel.

7.2.4.15 La Spécification des Exigences du Logiciel doit être appuyée par des techniques et des mesures issues du Tableau A.2. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.2.4.16 Une Spécification des Essais d'Ensemble du Logiciel doit être rédigée, sous la responsabilité du Chargé des essais, sur la base de la Spécification des Exigences du Logiciel.

Les exigences de 7.2.4.17 à 7.2.4.19 se rapportent à la Spécification des Essais d'Ensemble du Logiciel.

7.2.4.17 La Spécification des Essais d'Ensemble du Logiciel doit être une description des essais qui sont à effectuer sur le logiciel complet.

7.2.4.18 La Spécification des Essais d'Ensemble du Logiciel doit choisir des techniques et mesures issues du Tableau A.7. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.2.4.19 La Spécification des Essais d'Ensemble du Logiciel doit identifier les scénarios d'essai pour chacune des fonctions requises, y compris

a) les signaux d'entrée requis avec leurs séquences et leurs valeurs,

b) les signaux de sortie attendus avec leurs séquences et leurs valeurs,

c) les critères de succès aux essais, en incluant les aspects performance et qualité.

7.2.4.20 Un Rapport de Vérification des Exigences du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base de la Spécification des Exigences de Sécurité du Système, de la Spécification des Exigences du Logiciel, de la Spécification des Essais d'Ensemble du Logiciel et du Plan d'Assurance Qualité du Logiciel.

Les exigences de 7.2.4.21 à 7.2.4.22 se rapportent au Rapport de Vérification des Exigences du Logiciel.

7.2.4.21 Le Rapport de Vérification des Exigences du Logiciel doit être rédigé conformément aux exigences génériques établies pour tous les Rapports de Vérification (voir 6.2.4.14).

7.2.4.22 Une fois que la Spécification des Exigences du Logiciel a été définie, la vérification doit porter sur

a) l'adéquation de la Spécification des Exigences du Logiciel pour satisfaire aux exigences exposées dans la Spécification des Exigences du Système, la Spécification des Exigences de Sécurité du Système et le Plan d'Assurance Qualité du Logiciel,

b) le fait que la Spécification des Exigences du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.2.4.2 à 7.2.4.15,

c) l'adéquation de la Spécification des Essais d'Ensemble du Logiciel pour l'essai par rapport à la Spécification des Exigences du Logiciel,

d) la définition de toute activité supplémentaire afin de démontrer la couverture correcte des exigences qui ne se prêtent pas à des essais,

e) la cohérence interne de la Spécification des Exigences du Logiciel,

f) l'adéquation de la Spécification des Exigences du Logiciel pour respecter ou prendre en compte les contraintes entre matériel et logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification des Exigences du Logiciel.

7.3 Architecture et Conception

7.3.1 Objets

7.3.1.1 Développer une architecture de logiciel qui satisfait aux exigences du logiciel.

7.3.1.2 Identifier et évaluer l'importance des interactions matériel/logiciel pour la sécurité.

7.3.1.3 Choisir une méthode de conception si aucune n'a été définie au préalable.

7.3.1.4 Concevoir un logiciel d'un niveau défini d'intégrité de la sécurité logicielle à partir des documents en entrée.

7.3.1.5 Assurer que le système résultant et son logiciel se prêteront facilement à des essais depuis le commencement. Puisque la vérification et les essais seront un élément crucial de la validation, un intérêt tout particulier doit être porté aux besoins en matière de vérification et d'essai tout au long de la mise en œuvre.

7.3.2 Documents en entrée

Spécification des Exigences du Logiciel

7.3.3 Documents en sortie

a) Spécification de l'Architecture du Logiciel

b) Spécification de la Conception du Logiciel

c) Spécifications des Interfaces Logiciel

d) Spécification des essais d'Intégration du Logiciel

e) Spécification des essais d'Intégration Logiciel/Matériel

f) Rapport de Vérification de l'Architecture et de la Conception du Logiciel

7.3.4 Exigences

7.3.4.1 Une Spécification d'Architecture de Logiciel doit être rédigée, sous la responsabilité du Concepteur, sur la base de la Spécification des Exigences du Logiciel.

Les exigences de 7.3.4.2 à 7.3.4.14 se rapportent à la Spécification de l'Architecture du Logiciel.

7.3.4.2 L'architecture de logiciel proposée doit être établie et détaillée dans la Spécification d'Architecture du Logiciel.

7.3.4.3 La Spécification de l'Architecture du Logiciel doit prendre en considération la faisabilité de la réalisation de la Spécification des Exigences du Logiciel au niveau requis d'intégrité de la sécurité logicielle.

NOTE L'Architecture du Logiciel vise à réduise au maximum la taille et la complexité de la partie sécurité de l'application.

7.3.4.4 La Spécification de l'Architecture du Logiciel doit identifier, analyser et détailler l'importance de toutes les interactions matériel/logiciel.

7.3.4.5 La Spécification de l'Architecture du Logiciel doit identifier tous les composants logiciels et déterminer pour ces composants:

a) si ces composants sont nouveaux ou existants;

b) si ces composants ont été précédemment validés et, si tel est le cas, leurs conditions de validation;

c) le niveau d'intégrité de la sécurité logicielle du composant.

7.3.4.6 Les composants logiciels doivent

a) couvrir un sous-ensemble défini d'exigences relatives au logiciel,

b) être clairement identifiés et constituer des versions indépendantes dans le système de gestion de la configuration.

7.3.4.7 L'utilisation de logiciels préexistants doit être soumise aux restrictions suivantes.

a) Pour tous les niveaux d'intégrité de la sécurité logicielle, les informations suivantes doivent être clairement identifiées et documentées:

- les exigences auxquelles les logiciels préexistants sont censés satisfaire;
- les hypothèses relatives à l'environnement des logiciels préexistants;
- les interfaces à d'autres parties du logiciel.

b) Pour tous les niveaux d'intégrité de la sécurité logicielle, les logiciels préexistants doivent être inclus dans le processus de validation du logiciel complet.

c) Pour les niveaux d'intégrité de la sécurité logicielle SIL 3 et SIL 4, les précautions suivantes doivent être prises:

- une analyse des défaillances possibles des logiciels préexistants et de leurs conséquences sur le logiciel complet doit être menée à bien;
- une stratégie doit être définie en vue de détecter les défaillances des logiciels préexistants et de protéger le système contre ces défaillances;
- le processus de vérification et de validation doit assurer

  1) que les logiciels préexistants satisfont aux exigences allouées,

  2) que les défaillances des logiciels préexistants sont détectées et le système dans lequel les logiciels préexistants sont intégrés est protégé de ces défaillances,

  3) que les hypothèses relatives à l'environnement des logiciels préexistants sont respectées.

d) Les logiciels préexistants doivent être accompagnés d'une description (c'est-à-dire fonctions, contraintes et preuves) suffisamment précise (par exemple, limitée aux fonctions utilisées) et complète. La description doit contenir les contraintes relatives au matériel et/ou logiciel que le Chargé d'intégration doit connaître et prendre en considération au cours de l'application. En particulier, elle constitue le moyen d'informer le Chargé d'intégration des raisons de la conception du logiciel, de ses propriétés, de son comportement et de ses caractéristiques.

NOTE Des preuves statistiques peuvent être utilisées dans la stratégie de validation des logiciels préexistants.

7.3.4.8 L'utilisation des composants logiciels vérifiés existants développés conformément à la présente Norme dans la conception doit être préférentielle chaque fois que possible.

7.3.4.9 Lorsque le logiciel est constitué de composants de niveaux d'intégrité de la sécurité logicielle différents, tous les composants logiciels doivent alors être traités comme s'ils appartenaient au plus élevé de ces niveaux, à moins qu'il n'existe une preuve d'indépendance entre les composants de plus haut niveau d'intégrité de la sécurité logicielle et les composants du plus bas niveau d'intégrité de la sécurité logicielle. Cette preuve doit être enregistrée dans la Spécification de l'Architecture du Logiciel.

7.3.4.10 La Spécification de l'Architecture du Logiciel doit décrire la stratégie de développement du logiciel dans la limite requise par le niveau d'intégrité de la sécurité logicielle. La Spécification de l'Architecture du Logiciel doit être exprimée et structurée de manière à être

a) complète, cohérente, claire, précise, sans équivoque, susceptible d'être vérifiée, soumise à essai, maintenue et réalisée,

b) traçable depuis la Spécification des Exigences du Logiciel.

7.3.4.11 Les mesures pour gérer les défauts doivent être incluses dans la Spécification de l'Architecture du Logiciel afin d'assurer l'équilibre entre les stratégies d'évitement de fautes et de traitement des fautes.

7.3.4.12 La Spécification de l'Architecture du Logiciel doit démontrer que les techniques, les mesures et les outils choisis forment un ensemble qui satisfait à la Spécification des Exigences du Logiciel au niveau requis d'intégrité de la sécurité logicielle.

7.3.4.13 La Spécification de l'Architecture du Logiciel doit prendre en compte les exigences de 8.4.8 lorsque le logiciel est configuré par des données ou algorithmes d'application.

7.3.4.14 La Spécification de l'Architecture du Logiciel doit choisir des techniques et mesures issues du Tableau A.3. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.3.4.15 La taille et la complexité de l'architecture de logiciel développée doivent être équilibrées.

7.3.4.16 Le prototypage peut être utilisé dans n'importe quelle phase pour identifier les exigences ou obtenir une vue plus détaillée des exigences et de leurs conséquences.

7.3.4.17 Un code issu d'un prototype ne peut être utilisé dans le système cible que s'il est démontré que le code et son développement ainsi que sa documentation satisfont à la présente Norme.

7.3.4.18 Une Spécification des Interfaces Logiciel pour toutes les interfaces entre les composants du logiciel et la frontière du logiciel global doit être rédigée, sous la responsabilité du Concepteur, sur la base de la Spécification des Exigences du Logiciel et de la Spécification de l'Architecture du Logiciel.

L'exigence en 7.3.4.19 se rapporte à la Spécification des Interfaces Logiciel.

7.3.4.19 La description des interfaces doit porter sur

a) les préconditions/postconditions,

b) la définition et la description de toutes les valeurs aux limites pour toutes les données spécifiées,

c) le comportement lorsque la valeur à la limite est dépassée,

d) le comportement lorsque la valeur est à la limite,

e) pour les données d'entrée et de sortie à temps critique

   1) contraintes de temps et exigences pour un fonctionnement correct,

   2) gestion des exceptions,

f) la mémoire allouée pour les tampons d'interface et les mécanismes pour détecter que la mémoire ne peut pas être allouée ou tous les tampons sont pleins, selon le cas,

g) l'existence de mécanismes de synchronisation entre les fonctions (voir le point e)).

Toutes les données en provenance et à destination des interfaces doivent être définies pour la plage complète de valeurs définies par le type des données, y compris les plages qui ne sont pas utilisées lorsqu'elles sont traitées par les fonctions:

h) définition et description de toutes les classes d'équivalence pour toutes les données spécifiées et de chaque fonction logicielle les utilisant,

i) définition des classes d'équivalence inutilisées ou interdites.

NOTE Le type de données comprend les éléments suivants:

a) les paramètres d'entrée et les résultats de sortie des fonctions et/ou procédures;

b) les données spécifiées dans les télégrammes ou paquets de communication;

c) les données du matériel.

7.3.4.20 Une Spécification de la Conception du Logiciel doit être rédigée, sous la responsabilité du Concepteur, sur la base de la Spécification des Exigences du Logiciel, de la Spécification de l'Architecture du Logiciel et de la Spécification des Interfaces Logiciel.

Les exigences de 7.3.4.21 à 7.3.4.24 se rapportent à la Spécification de la Conception du Logiciel.

7.3.4.21 Les documents en entrée doivent être disponibles, bien que pas nécessairement finalisés, avant le début du processus de conception.

7.3.4.22 La Spécification de la Conception du Logiciel doit décrire la conception du logiciel basée sur une décomposition en composants chaque composant comportant une Spécification de la Conception des Composants Logiciels et une Spécification des essais des Composants Logiciels.

7.3.4.23 La Spécification de la Conception du Logiciel doit décrire

a) les composants logiciels tracés par rapport à l'architecture du logiciel et leur niveau d'intégrité de la sécurité,
b) les interfaces des composants logiciels avec l'environnement,
c) les interfaces entre les composants logiciels,
d) les structures de données,
e) l'allocation et la traçabilité des exigences sur les composants,
f) les algorithmes principaux et le séquencement,
g) les mécanismes de compte-rendu des erreurs.

7.3.4.24 La Spécification de la Conception du Logiciel doit choisir des techniques et mesures issues du Tableau A.4. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.3.4.25 Des normes de codage doivent être développées et spécifier

a) les bonnes pratiques de programmation, telles que définies par le Tableau A.12,
b) les mesures pour éviter ou détecter les erreurs qui peuvent être commises pendant l'application du langage et qui ne sont pas détectables pendant la vérification (voir 7.5 et 7.6). De telles défaillances sont dérivées par analyse sur toutes les caractéristiques du langage,
c) les procédures pour la documentation du code source.

7.3.4.26 Le choix d'une norme de codage doit être justifié dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

7.3.4.27 Les normes de codage doivent être utilisées pour le développement de tous les logiciels et être référencées dans le Plan d'Assurance Qualité du Logiciel.

7.3.4.28 Conformément au niveau d'intégrité de la sécurité logicielle, la méthode de conception choisie doit offrir des caractéristiques qui facilitent

a) l'abstraction, la modularité et d'autres caractéristiques pour maîtriser la complexité,
b) l'expression claire et précise des éléments suivants:
    1) la fonctionnalité;
    2) le flux d'informations entre les composants;
    3) les séquencements et les informations temporelles;
    4) le parallélisme;
    5) la structure et les propriétés des données;
c) la compréhension humaine,
d) la vérification et la validation,
e) la maintenance du logiciel.

7.3.4.29 Une Spécification des essais d'Intégration du Logiciel doit être rédigée, sous la responsabilité du Chargé d'intégration, sur la base de la Spécification des Exigences du Logiciel, de la Spécification de l'Architecture du Logiciel, de la Spécification de la Conception du Logiciel et de la Spécification des Interfaces Logiciel.

Les exigences de 7.3.4.30 à 7.3.4.32 se rapportent à la Spécification des essais d'Intégration du Logiciel.

7.3.4.30 La Spécification des essais d'Intégration du Logiciel doit être rédigée conformément aux exigences génériques établies pour une Spécification d'essai (voir 6.1.4.4).

7.3.4.31 La Spécification des essais d'Intégration du Logiciel doit traiter de ce qui suit:

a) il doit être montré que chaque composant logiciel fournit les interfaces spécifiées pour d'autres composants en exécutant les composants ensemble;

b) il doit être montré que le logiciel se comporte d'une façon appropriée lorsque l'interface est soumise à des entrées qui sont hors spécification;

c) les données en entrée requises avec leurs séquences et leurs valeurs doivent être la base des scénarios d'essai;

d) les données de sortie prévues avec leurs séquences et leurs valeurs doivent être la base des scénarios d'essai;

e) il doit être indiqué quels résultats de l'essai des composants (voir 7.5.4.5 et 7.5.4.7) sont censés être réutilisés pour l'essai d'intégration du logiciel.

7.3.4.32 La Spécification des essais d'Intégration du Logiciel doit choisir des techniques et des mesures issues du Tableau A.5. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.3.4.33 Une Spécification des essais d'Intégration Logiciel/Matériel doit être rédigée, sous la responsabilité du Chargé d'intégration, sur la base de la Description de la Conception du Système, de la Spécification des Exigences du Logiciel, de la Spécification de l'Architecture du Logiciel et de la Spécification de la Conception du Logiciel.

Les exigences de 7.3.4.34 à 7.3.4.39 se rapportent à la Spécification des essais d'Intégration Logiciel/Matériel.

7.3.4.34 Il convient qu'une Spécification des essais d'Intégration Logiciel/Matériel soit créée tôt dans le cycle de vie de développement, afin que les essais d'intégration puissent être convenablement menés et que les besoins particuliers en matière de conception ou autre intégration puissent être correctement assurés. Selon la taille du système, il est permis de scinder pendant le développement la Spécification des essais d'Intégration Logiciel/Matériel en un certain nombre de sous-documents qui seront naturellement complétés à mesure que les conceptions de type matériel et logiciel évoluent et que les besoins détaillés d'intégration deviennent plus clairs.

7.3.4.35 La Spécification des essais d'Intégration Logiciel/Matériel doit faire une distinction entre les activités qui peuvent être réalisées par le fournisseur dans ses locaux et celles exigeant l'accès au site de l'utilisateur.

7.3.4.36 La Spécification des essais d'Intégration Logiciel/Matériel doit traiter de ce qui suit:

a) il doit être montré que le logiciel fonctionne correctement sur le matériel en utilisant le matériel via les interfaces matériel spécifiées;

b) il doit être montré que le logiciel peut gérer les défauts matériels comme demandé;

c) la temporisation et la performance requises doivent être démontrées;

d) les données en entrée requises avec leurs séquences et leurs valeurs doivent être la base des scénarios d'essai;

e) les données en sortie prévues avec leurs séquences et leurs valeurs doivent être la base des scénarios d'essai;

f) il doit être indiqué quels résultats de l'essai des composants (voir 7.5.4.5) et de l'essai d'intégration du logiciel (voir 7.6.4.3) sont censés être réutilisés pour les essais d'intégration logiciel/matériel.

7.3.4.37 La Spécification des essais d'Intégration Logiciel/Matériel doit documenter les éléments suivants:

a) les scénarios d'essai et les données liées à l'essai;

b) les types d'essai qui doivent être effectués;

c) l'environnement d'essai y compris les outils, le logiciel de support et la description de la configuration;

d) les critères d'essai sur lesquels l'achèvement de l'essai sera jugé.

7.3.4.38 La Spécification des essais d'Intégration Logiciel/Matériel doit être rédigée conformément aux exigences génériques établies pour une Spécification d'essai (voir 6.1.4.4).

7.3.4.39 La Spécification des essais d'Intégration Logiciel/Matériel doit choisir des techniques et des mesures issues du Tableau A.5. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.3.4.40 Un Rapport de Vérification de l'Architecture et de la Conception du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base de la Spécification des Exigences du Logiciel, de la Spécification de l'Architecture du Logiciel, de la Spécification de la Conception du Logiciel, de la Spécification des essais d'Intégration du Logiciel et de la Spécification des essais d'Intégration Logiciel/Matériel.

Les exigences de 7.3.4.41 à 7.3.4.43 se rapportent au Rapport de Vérification de l'Architecture et de la Conception du Logiciel.

7.3.4.41 Le Rapport de Vérification de l'Architecture et de la Conception du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport de Vérification (voir 6.2.4.14).

7.3.4.42 Une fois que les Spécifications de l'Architecture, de l'Interface et de la Conception du Logiciel ont été définies, la vérification doit porter sur

a) la cohérence interne des Spécifications de l'Architecture, de l'Interface et de la Conception du Logiciel,

b) l'adéquation des Spécifications de l'Architecture, de l'Interface et de la Conception du Logiciel à satisfaire à la Spécification des Exigences du Logiciel en ce qui concerne la cohérence et la complétude,

c) le fait que la Spécification de l'Architecture du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.16 ainsi qu'aux exigences spécifiques de 7.3.4.1 à 7.3.4.14,

d) le fait que la Spécification des Interfaces Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.16 ainsi qu'aux exigences spécifiques de 7.3.4.18 à 7.3.4.19,

e) le fait que la Spécification de la Conception du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.16 ainsi qu'aux exigences spécifiques de 7.3.4.20 à 7.3.4.24,

f) l'adéquation de la Spécification de l'Architecture du Logiciel et de la Spécification de la Conception du Logiciel pour tenir compte des contraintes relatives au matériel et au logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification de l'Architecture et de la Conception du Logiciel.

7.3.4.43 Après que les Spécifications des Essais d'Intégration du Logiciel et d'Intégration Logiciel/Matériel ont été définies, la vérification doit porter sur

a) le fait que la Spécification des essais d'Intégration du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.16 ainsi qu'aux exigences spécifiques de 7.3.4.29 à 7.3.4.32,

b) le fait que la Spécification des essais d'Intégration Logiciel/Matériel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.16 ainsi qu'aux exigences spécifiques de 7.3.4.33 à 7.3.4.39.

Les résultats doivent être enregistrés dans un Rapport de Vérification de l'Architecture et de la Conception du Logiciel.

7.4 Conception du composant

7.4.1 Objets

7.4.1.1 Développer une Conception des Composants Logiciels qui respecte les exigences de la Spécification de la Conception du Logiciel dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

7.4.1.2 Développer une Spécification des essais des Composants Logiciels qui respecte les exigences de la Spécification de la Conception des Composants Logiciels dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

7.4.2 Documents en entrée

Spécification de la Conception du Logiciel

7.4.3 Documents en sortie

a) Spécification de la Conception des Composants logiciels

b) Spécification des essais des Composants Logiciels

c) Rapport de Vérification de la Conception des Composants Logiciels

7.4.4 Exigences

7.4.4.1 Une Spécification de la Conception des Composants Logiciels doit être rédigée pour chaque composant, sous la responsabilité du Concepteur, sur la base de la Spécification de la Conception du Logiciel.

Les exigences de 7.4.4.2 à 7.4.4.6 se rapportent à la Spécification de la Conception des Composants logiciels.

7.4.4.2 Pour chaque composant logiciel, les informations suivantes doivent être disponibles:

– l'auteur;

– l'historique de la configuration; et

– courte description.

L'historique de la configuration doit inclure une identification précise de la version courante et de toutes les versions antérieures, en spécifiant la version, la date et l'auteur, et une description des changements apportés par rapport à la version précédente.

7.4.4.3 La Spécification de la Conception des Composants Logiciels doit traiter des éléments suivants:

a) l'identification de toutes les unités de plus bas niveau (par exemple: sous-programmes, méthodes, procédures) tracées par rapport au niveau supérieur,

b) leurs interfaces détaillées avec l'environnement et les autres composants avec le détail de leurs entrées/sorties,

c) leur niveau d'intégrité de sécurité sans autre répartition dans le composant lui-même,

d) les algorithmes et les structures de données détaillés.

Chaque Spécification de la Conception des Composants Logiciels doit être autocohérente et permettre la transformation en code des composants correspondants.

7.4.4.4 Chaque Spécification de la Conception des Composants Logiciels doit être lisible, compréhensible et susceptible d'être soumise à essai.

7.4.4.5 La taille et la complexité de chaque Composant Logiciel développé doivent être équilibrées.

7.4.4.6 La Spécification de la Conception des Composants Logiciels doit choisir des techniques et mesures issues du Tableau A.4. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.4.4.7 Pour chaque composant, une Spécification des essais des Composants logiciels doit être rédigée, sous la responsabilité du Chargé des essais, sur la base de la Spécification de la Conception des Composants Logiciels.

Les exigences de 7.4.4.8 à 7.4.4.10 se rapportent à la Spécification des essais des Composants logiciels.

7.4.4.8 La Spécification des essais des Composants logiciels doit être rédigée conformément aux exigences génériques établies pour une Spécification d'essai (voir 6.1.4.4).

7.4.4.9 Une Spécification des essais des Composants Logiciels doit être produite et chaque composant doit être soumis à essai par rapport à elle. Ces essais doivent montrer que chaque composant exécute la fonction prévue. La Spécification des essais des Composants Logiciels doit définir et justifier les critères requis et le degré exigé de couverture des essais dans la limite requise par le niveau d'intégrité de la sécurité logicielle. Les essais doivent être conçus de manière à atteindre trois objectifs:

a) confirmer que le composant exécute les fonctions prévues (essais boîte noire);

b) vérifier comment les parties internes du composant interagissent pour mener à bien les fonctions prévues (essais boîte noire/boîte blanche);

c) confirmer que toutes les parties du composant sont soumises à essai (essais boîte blanche).

7.4.4.10 La Spécification des essais des Composants logiciels doit choisir des techniques et mesures issues du Tableau A.5). La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

7.4.4.11 Un Rapport de Vérification de la Conception des Composants logiciels doit être rédigé, sous la responsabilité du chargé de Vérification, sur la base de la Spécification de la

Conception du Logiciel, de la Spécification de la Conception des Composants logiciels et de la Spécification des essais des Composants Logiciels.

Les exigences de 7.4.4.12 à 7.4.4.13 se rapportent au Rapport de Vérification de la Conception des Composants logiciels.

7.4.4.12 Le Rapport de Vérification de la Conception des Composants logiciels doit être rédigé conformément aux exigences génériques établies pour un Rapport de Vérification (voir 6.2.4.14).

7.4.4.13 Une fois que chaque Spécification de la Conception des Composants Logiciels a été définie, la vérification doit porter sur:

a) l'adéquation de la Spécification de la Conception des Composants Logiciels pour satisfaire à la Spécification de la Conception du Logiciel,

b) le fait que la Spécification de la Conception des Composants Logiciels satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.4.4.1 à 7.4.4.6,

c) l'adéquation de la Spécification des essais des Composants Logiciels comme ensemble de scénarios d'essai pour la Spécification de la Conception des Composants Logiciels,

d) le fait que la Spécification des essais des Composants logiciels satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.4.4.7 à 7.4.4.10,

e) la décomposition de la Spécification de la Conception du Logiciel en composants logiciels et la Spécification de la Conception des Composants Logiciels en faisant référence à

   1) la faisabilité de la performance requise,

   2) la testabilité pour une vérification ultérieure, et

   3) la maintenabilité pour permettre une modification ultérieure.

Les résultats doivent être enregistrés dans un Rapport de Vérification de la Conception des Composants Logiciels.

7.5 Mise en œuvre et essais des composants

7.5.1 Objets

Réaliser un logiciel pouvant être analysé, soumis à essai, vérifié et maintenu. Les essais des composants font aussi partie de cette phase.

7.5.2 Documents en entrée

a) Spécification de la Conception des Composants logiciels

b) Spécification des essais des Composants Logiciels

7.5.3 Documents en sortie

a) Code Source du Logiciel et Documentation de Support

b) Rapport des Essais des Composants Logiciels

c) Rapport de Vérification du Code Source du Logiciel

7.5.4 Exigences

7.5.4.1 Le code source du logiciel doit être rédigé sous la responsabilité du Réalisateur sur la base de la Spécification de la Conception des Composants Logiciels. Les exigences de 7.5.4.2 à 7.5.4.4 se rapportent au code source du logiciel.

7.5.4.2 La taille et la complexité du code source développé doivent être équilibrées.

7.5.4.3 Le code source du logiciel doit être lisible, compréhensible et susceptible d'être soumis à essai.

7.5.4.4 Le code source du logiciel doit être soumis au contrôle de configuration avant le commencement des essais documentés des composants.

7.5.4.5 Un Rapport des Essais des Composants Logiciels doit être rédigé, sous la responsabilité du Chargé des essais, sur la base de la Spécification des essais des Composants Logiciels et du Code Source du Logiciel.

Les exigences de 7.5.4.6 à 7.5.4.7 se rapportent au Rapport des Essais des Composants logiciels.

7.5.4.6 Le Rapport des Essais des Composants logiciels doit être rédigé conformément aux exigences génériques établies pour un Rapport d'essai (voir 6.1.4.5).

7.5.4.7 Le Rapport des Essais des Composants Logiciels doit inclure les caractéristiques suivantes.

a) un énoncé des résultats des essais et si chaque composant remplit les exigences de sa Spécification de la Conception des Composants Logiciels,

b) un énoncé de la couverture des essais doit être fourni pour chaque composant, montrant que le degré requis de couverture des essais a été obtenu pour tous les critères requis. Tout code exécutable inaccessible doit être justifié en fonction du SIL associé (voir Tableau A.21).

7.5.4.8 Un Rapport de Vérification du Code Source du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base de la Spécification de la Conception des Composants Logiciels, de la Spécification des essais des Composants Logiciels et du Code Source du Logiciel.

Les exigences de 7.5.4.9 à 7.5.4.10 se rapportent au Rapport de Vérification du Code Source du Logiciel.

7.5.4.9 Le Rapport de Vérification du Code Source du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport de Vérification (voir 6.2.4.14).

7.5.4.10 Après que le Code Source du Logiciel et le Rapport des Essais des Composants Logiciels ont été définis, la vérification doit porter sur

a) l'adéquation du Code Source du Logiciel comme une mise en œuvre de la Spécification de la Conception des Composants logiciels,

b) l'utilisation correcte des techniques et mesures choisies issues du Tableau A.4 comme un ensemble satisfaisant à 4.8 et 4.9,

c) la détermination de l'application correcte des normes de codage,

d) le fait que le Code Source du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.5.4.1 à 7.5.4.4,

e) l'adéquation du Rapport des Essais des Composants Logiciels comme enregistrement des essais effectués conformément à la Spécification des essais des Composants Logiciels.

Les résultats doivent être enregistrés dans un Rapport de Vérification du Code Source du Logiciel.

7.6 Intégration

7.6.1 Objets

7.6.1.1 Mener à bien l'intégration du logiciel et l'intégration logiciel/matériel.

7.6.1.2 Démontrer que le logiciel et le matériel interagissent correctement pour exécuter les fonctions attendues.

7.6.2 Documents en entrée

a) Spécification des essais d'Intégration Logiciel/Matériel

b) Spécification des essais d'Intégration du Logiciel

7.6.3 Documents en sortie

a) Rapport des Essais d'Intégration du Logiciel

b) Rapport des Essais d'Intégration Logiciel/Matériel

c) Rapport de Vérification de l'Intégration du Logiciel

7.6.4 Exigences

7.6.4.1 L'intégration des composants logiciels doit être le processus de regroupement progressif de chacun des composants logiciels soumis à essai au préalable au sein d'une entité de manière à ce que les interfaces des composants et le logiciel assemblé puissent être convenablement soumis à essai avant l'intégration et les essais du système.

7.6.4.2 Au cours de l'intégration logiciel/matériel, toute modification ou tout changement apporté(e) au système intégré doit faire l'objet d'une étude d'impact qui doit identifier tous les composants affectés et les activités indispensables à une nouvelle vérification.

7.6.4.3 Un Rapport des Essais d'Intégration du Logiciel doit être rédigé, sous la responsabilité du Chargé d'Intégration, sur la base de la Spécification des essais d'Intégration du Logiciel.

Les exigences de 7.6.4.4 à 7.6.4.6 se rapportent au Rapport des Essais d'Intégration du Logiciel.

7.6.4.4 Le Rapport des Essais d'Intégration du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport d'essai (voir 6.1.4.5).

7.6.4.5 Un Rapport des Essais d'Intégration du Logiciel doit être présenté comme suit:

a) un Rapport des Essais d'Intégration du Logiciel doit être présenté; il doit énoncer les résultats des essais et préciser si les objectifs et les critères de la Spécification des essais d'Intégration du Logiciel ont été respectés. En cas d'échec les circonstances doivent être consignées dans le rapport;

b) les scénarios d'essai et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine en vue d'une analyse ultérieure;

c) les essais doivent être reproductibles et, dans la mesure du possible, exécutés par des moyens automatiques;

d) le Rapport des Essais d'Intégration du Logiciel doit détailler l'identité et la configuration de tous les éléments concernés.

7.6.4.6 Le Rapport des Essais d'Intégration du Logiciel doit démontrer l'utilisation correcte des techniques et mesures choisies issues du Tableau A.6 comme un ensemble satisfaisant à 4.8 et 4.9.

7.6.4.7 Un Rapport des Essais d'Intégration Logiciel/Matériel doit être rédigé, sous la responsabilité du Chargé d'intégration, sur la base de la Spécification des essais d'Intégration Logiciel/Matériel.

Les exigences de 7.6.4.8 à 7.6.4.10 se rapportent au Rapport des Essais d'Intégration Logiciel/Matériel.

7.6.4.8 Le Rapport des Essais d'Intégration Logiciel/Matériel doit être rédigé conformément aux exigences génériques établies pour un Rapport d'essai (voir 6.1.4.5).

7.6.4.9 Un Rapport des Essais d'Intégration Logiciel/Matériel doit être rédigé comme suit:

a) le Rapport des Essais d'Intégration Logiciel/Matériel doit contenir les résultats des essais et préciser si les objectifs et les critères de la Spécification des essais d'Intégration Logiciel/Matériel ont été respectés. En cas d'échec les circonstances doivent être enregistrées;

b) les scénarios d'essai et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine en vue d'une analyse ultérieure;

c) le Rapport des Essais d'Intégration Logiciel/Matériel doit détailler l'identité et la configuration de tous les éléments concernés.

7.6.4.10 Le Rapport des Essais d'Intégration Logiciel/Matériel doit démontrer l'utilisation correcte des techniques et mesures choisies issues du Tableau A.6 comme un ensemble satisfaisant à 4.8 et 4.9.

7.6.4.11 Un Rapport de Vérification de l'Intégration du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des Spécifications des Essais d'Intégration du Logiciel et Logiciel/Matériel et des rapports d'essai correspondants.

Les exigences de 7.6.4.12 à 7.6.4.13 se rapportent au Rapport de Vérification de l'Intégration du Logiciel.

7.6.4.12 Le Rapport de Vérification de l'Intégration du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport de Vérification (voir 6.2.4.14).

7.6.4.13 Après que le Rapport des Essais d'Intégration du Logiciel et le Rapport des Essais d'Intégration Logiciel/Matériel ont été définis, la vérification doit porter sur

a) l'adéquation du Rapport des Essais d'Intégration du Logiciel comme enregistrement des essais effectués conformément à la Spécification des essais d'Intégration du Logiciel,

b) si, oui ou non, le Rapport des Essais d'Intégration du Logiciel satisfait aux exigences de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.6.4.3 à 7.6.4.6,

c) l'adéquation du Rapport des Essais d'Intégration du Logiciel/Matériel comme enregistrement des essais effectués conformément à la Spécification des essais d'Intégration Logiciel/Matériel,

d) si, oui ou non, le Rapport des Essais d'Intégration Logiciel/Matériel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 7.6.4.7 à 7.6.4.10.

7.7 Essais d'ensemble du logiciel / Validation finale

7.7.1 Objets

Analyser et soumettre à essai l'ensemble intégré logiciel/matériel pour assurer la conformité avec la Spécification des Exigences du Logiciel, en mettant tout particulièrement l'accent sur

les aspects fonctionnels et ceux de sécurité selon le niveau d'intégrité de la sécurité logicielle et vérifier qu'il est adapté à son application prévue.

7.7.2 Documents en entrée

a) Spécification des Exigences du Logiciel

b) Spécification des Essais d'Ensemble du Logiciel

c) Plan de Vérification du Logiciel

d) Plan de Validation du Logiciel

e) L'ensemble de la documentation du matériel et logiciel, y compris les résultats intermédiaires de la vérification

f) Spécification des Exigences de Sécurité du Système

7.7.3 Documents en sortie

a) Rapport des Essais d'Ensemble du Logiciel

b) Rapport de Vérification des Essais d'Ensemble du Logiciel

c) Rapport de Validation du Logiciel

d) Fiche de livraison

7.7.4 Exigences

7.7.4.1 Un Rapport des Essais d'Ensemble du Logiciel doit être rédigé, sous la responsabilité du Chargé des essais, sur la base de la Spécification des Essais d'Ensemble du Logiciel.

Les exigences de 7.7.4.2 à 7.7.4.4 se rapportent au Rapport des Essais d'Ensemble du Logiciel.

7.7.4.2 Le Rapport des Essais d'Ensemble du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport d'essai (voir 6.1.4.5).

7.7.4.3 Le Chargé de validation doit spécifier et exécuter des essais supplémentaires à sa discrétion ou les faire exécuter par le Chargé des essais. Alors que les Essais d'Ensemble du Logiciel sont principalement fondés sur la structure de la Spécification des Exigences du Logiciel, la valeur ajoutée que doit apporter le Chargé de validation est constituée des essais qui sollicitent le système par des scénarios complexes reflétant les besoins de l'utilisateur.

7.7.4.4 Les résultats de l'ensemble des essais et analyses doivent être consignés dans le Rapport des Essais d'Ensemble du Logiciel.

7.7.4.5 Un Rapport de Vérification des Essais d'Ensemble de Logiciel, qui doit être rédigé sous la responsabilité du Chargé de vérification.

7.7.4.6 Le logiciel doit être mis en œuvre soit en le connectant à des éléments réels de matériel ou de systèmes réels avec lesquels il aurait une interface lors de son fonctionnement, soit par simulation de signaux d'entrée et de charges pilotés par des grandeurs de sortie. Il doit être soumis à essai dans les conditions présentes en fonctionnement normal ou présentes lors de situations prévues et de conditions non voulues exigeant une action du système. Lorsque des entrées ou charges simulées sont utilisées, il doit être montré qu'elles ne diffèrent pas sensiblement de celles rencontrées en fonctionnement.

NOTE Les entrées ou charges simulées peuvent remplacer les entrées ou les charges qui ne sont présentes qu'au niveau système ou dans des modes défaillants.

7.7.4.7 Un Rapport de Validation du Logiciel doit être rédigé, sous la responsabilité du Chargé de validation, sur la base du Plan de Validation du Logiciel.

Les exigences de 7.7.4.7 à 7.7.4.12 se rapportent au Rapport de Validation du Logiciel.

7.7.4.8 Le Rapport de Validation du Logiciel doit être rédigé conformément aux exigences génériques établies pour un Rapport de Validation (voir 6.3.4.7 à 6.3.4.11).

7.7.4.9 Une fois l'intégration terminée ainsi que les Essais d'Ensemble du Logiciel et les analyses achevés, un Rapport de Validation du Logiciel doit être produit comme suit:

a) il doit énoncer si les objectifs et les critères du Plan de Validation du Logiciel ont été respectés. Les écarts par rapport au plan doivent être consignés et justifiés;

b) il doit contenir un énoncé récapitulatif des résultats des essais et préciser si le logiciel complet sur sa machine cible remplit les exigences exposées dans la Spécification des Exigences du Logiciel;

c) il doit contenir une évaluation de la couverture des essais concernant les exigences de la Spécification des Exigences du Logiciel;

d) une évaluation d'autres activités de vérification conformément au Plan et Rapport de Vérification du Logiciel doit être effectuée accompagnée d'une vérification que la traçabilité des exigences est complètement exécutée et couverte;

e) si le Chargé de validation produit ses propres scénarios d'essai qui ne sont pas remis au Chargé des essais, le Rapport de Validation du Logiciel doit les documenter selon 6.3.4.7 à 6.3.4.11.

7.7.4.10 Le Rapport de Validation du Logiciel doit contenir la confirmation que chaque combinaison de techniques ou de mesures choisies conformément à l'Annexe A est appropriée pour le niveau défini d'intégrité de la sécurité logicielle. Il doit contenir une évaluation de l'efficacité globale de la combinaison de techniques et de mesures adoptée en prenant en compte la taille et la complexité du logiciel produit et en tenant compte des résultats réels des activités d'essai, de vérification et de validation.

7.7.4.11 Les points suivants doivent être traités dans le Rapport de Validation du Logiciel:

a) documentation de l'identité et de la configuration du logiciel;

b) énoncé identifiant de façon appropriée les logiciels et les équipements de support technique;

c) énoncé identifiant de façon appropriée les modèles de simulation utilisés;

d) énoncé relatif à l'adéquation de la Spécification des Essais d'Ensemble du Logiciel;

e) recueil et conservation de la trace des éventuels écarts trouvés;

f) revue et évaluation de tous les écarts en termes de risque (impact);

g) énoncé que le projet a correctement géré les actions correctives conformément aux processus et procédures de gestion des modifications et avec une identification claire de toute divergence décelée;

h) énoncé de chaque restriction donnée par les écarts d'une manière traçable;

i) conclusion indiquant si, oui ou non, le logiciel est adéquat pour son application prévue, en tenant compte des conditions et des contraintes d'application.

7.7.4.12 Les éventuelles divergences constatées, y compris les erreurs et les non-conformités avec la présente Norme ou avec tout(e) exigence ou plan concernant le logiciel, ainsi que les contraintes et les limitations, doivent être clairement identifiées dans un paragraphe séparé dans le Rapport de Validation du Logiciel, évaluées quant au niveau d'intégrité de sécurité et incluses dans toute Fiche de livraison qui accompagne le logiciel livré.

7.7.4.13 Une fiche de livraison qui accompagne le logiciel livré doit contenir toutes les restrictions dans l'utilisation du logiciel. Ces restrictions sont dérivées des

a) erreurs détectées,

b) non-conformités avec la présente Norme,

c) degrés de respect des exigences,

d) degrés de respect d'un plan éventuel.

8 Développement de données d'application ou d'algorithmes d'application: systèmes configurés par des données d'application ou par des algorithmes d'application

8.1 Objets

8.1.1 Un trait caractéristique de nombreux systèmes ferroviaires est la nécessité de concevoir chaque installation en vue de répondre aux exigences individuelles d'une application spécifique. Un système configuré par des données d'application et/ou par des algorithmes d'application permet d'adapter spécialement un logiciel générique homologué avec des exigences individuelles pour chaque application spécifique.

L'objectif du développement de données d'application est l'obtention correcte des données à partir de l'installation concernée et la vérification du comportement prévu, suivie d'une évaluation du processus de développement utilisé pour les données d'application en question.

Les exigences pour le développement d'algorithmes d'application sont les mêmes que celles concernant le développement d'un logiciel générique comme décrit aux Articles 1 à 7 et 9.

Un exemple type est un système dont le logiciel générique est préconfiguré pour une application ferroviaire générique par un jeu d'algorithmes d'application et qui est en outre configuré ensuite selon chaque installation spécifique par instanciation et interconnexion des algorithmes d'application et par un jeu de données de configuration. Par exemple, les principes de signalisation d'un système d'enclenchement (par exemple: une gestion de signaux, une gestion d'aiguilles) peuvent être mis en œuvre par un jeu d'algorithmes d'application.

Les données d'application se présentent en général sous forme de valeurs de paramètres ou de descriptions (identité, type, emplacement, etc.) d'objets externes. Les algorithmes d'application peuvent se présenter sous forme de diagrammes fonctionnels, de diagrammes d'états et de diagrammes à contacts, qui déterminent la réponse souhaitée du système en fonction de ses entrées, de son état courant et de valeurs de paramètres spécifiques. Les algorithmes d'application comprennent des connexions et opérations logiques à effectuer.

Les données et algorithmes d'application sont en général produits à l'aide d'outils dédiés. Ils peuvent être exprimés aux formats de tableaux ou de diagrammes, qui peuvent être interprétés ou compilés en codes exécutables souvent après conversion en codes sources manipulés via des langages spécialisés (avec syntaxe et sémantique).

La personnalisation des systèmes par l'aptitude à la configuration donne au Concepteur différents degrés de maîtrise sur la fonctionnalité détaillée du logiciel.

8.1.2 Les procédures et les outils utilisés pour leur développement doivent être appropriés au niveau d'intégrité de sécurité du système tel que déterminé par la fonction pour laquelle ils sont développés.

8.1.3 Les paragraphes de 8.4 décrivent les exigences pour le développement initial d'un système configurable et pour le développement subséquent de chaque jeu de données/algorithmes spécifiques à l'application.

8.2 Documents en entrée

a) Spécification des Exigences de Logiciel pour le logiciel générique

b) Spécification de l'Architecture de Logiciel pour le logiciel générique

c) Conditions d'application du logiciel générique et des outils d'application

d) Manuels de l'utilisateur du logiciel générique et des outils d'application

8.3 Documents en sortie

a) Plan de préparation de l'application

b) Spécification des Exigences de l'Application

c) Architecture et Conception de l'Application

d) Spécification d'essai de l'Application

e) Rapport des Essais d'Application

f) Rapport de Vérification de la Préparation de l'Application

g) Code Source des données/algorithmes d'application

h) Rapport de vérification des données/algorithmes d'application

8.4 Exigences

8.4.1 Processus de développement de l'Application

8.4.1.1 Un Plan de Préparation d'Application doit être rédigé, sous la responsabilité du Gestionnaire des Exigences ou du Concepteur, sur la base des documents en entrée issus de 8.2.

Les exigences de 8.4.1.2 à 8.4.1.11 se rapportent au Plan de Préparation de l'Application.

8.4.1.2 Un Plan de Préparation de l'Application doit être produit afin de définir et détailler le processus de développement de l'application, y compris toutes les activités, tous les documents à fournir et tous les rôles qui en ont la charge. Il peut être produit soit pour chaque application spécifique, soit pour une classe d'applications spécifiques, c'est-à-dire pour une application générique.

8.4.1.3 Le Plan de Préparation de l'Application doit définir une structure de documentation pour le processus de préparation de l'application.

8.4.1.4 Le Plan de Préparation de l'Application doit choisir des techniques et mesures issues du Tableau A.11. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

8.4.1.5 Le Plan de Préparation de l'Application doit spécifier les procédures et les outils d'application (avec leur classification basée sur 6.7) qui sont à utiliser dans le processus de développement de l'application.

8.4.1.6 Le Plan de Préparation de l'Application doit inclure des activités de vérification et de validation pour assurer que les données et algorithmes d'application sont complets, corrects et compatibles les uns avec les autres et avec l'application générique, et fournir la preuve que les conditions d'application de l'application générique sont respectées. Ces activités de vérification et de validation peuvent être remplacées par la vérification et la validation exécutées sur les outils qui produisent les données/algorithmes d'application. Les résultats sont regroupés dans le Rapport de Vérification de la Préparation d'Application et le Rapport des Essais d'Application.

8.4.1.7 Le Plan de Préparation de l'Application doit inclure des activités de vérification et de validation pour assurer que les outils d'application et le logiciel générique sont compatibles les uns avec les autres et avec l'application spécifique, et fournir la preuve que leurs conditions d'application sont respectées.

8.4.1.8 Une analyse du risque doit être effectuée et couvrir le processus de développement de l'application, notamment les outils et procédures de l'application afin de valider le Plan de Préparation de l'Application et satisfaire au niveau requis d'intégrité de la sécurité logicielle. Le Plan de Préparation de l'Application doit inclure l'analyse du risque.

8.4.1.9 Le Plan de Préparation de l'Application doit spécifier les exigences d'indépendance entre les équipes réalisant les tâches de vérification, de validation et de préparation selon 5.1.

NOTE Les activités de préparation des données sont accomplies par les concepteurs d'application.

8.4.1.10 Le Plan de Préparation de l'Application doit définir une classe d'outils pour tout outil matériel ou logiciel utilisé dans le cycle de vie de la préparation de l'application.

8.4.1.11 Chaque fois que possible, le Plan de Préparation de l'Application doit utiliser, pour spécifier les exigences et la conception, des notations qui sont familières aux ingénieurs d'application. Lorsque de nouvelles notations sont introduites, la documentation utilisateur nécessaire doit être fournie ainsi que la formation le cas échéant.

8.4.1.12 Un Rapport de Vérification des Données/Algorithmes d'Application doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.1.13 se rapporte au Rapport de Vérification de Données/Algorithmes d'Application.

8.4.1.13 Une fois que le Plan de Préparation de l'Application a été défini, la vérification doit porter sur

a) le fait que le Plan de Préparation de l'Application satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 8.4.1.2 à 8.4.1.11,

b) la cohérence interne du Plan de Préparation de l'Application.

Les résultats doivent être consignés dans un Rapport de Vérification des Données/Algorithmes d'Application.

8.4.1.14 La mise en œuvre du Plan de Préparation de l'Application doit être vérifiée et validée pour chaque application spécifique.

8.4.2 Spécification des Exigences de l'Application

8.4.2.1 Une Spécification des Exigences de l'Application doit être rédigée, sous la responsabilité du Gestionnaire des Exigences, sur la base des documents en entrée issus de 8.2.

Les exigences de 8.4.2.2 à 8.4.2.3 se rapportent à la Spécification des Exigences de l'Application.

8.4.2.2 Les exigences pour l'application spécifique doivent comprendre les exigences qui sont spécifiques à l'installation à l'étude (par exemple: disposition des voies, emplacements des signaux, vitesses limites pour le système de signalisation) ainsi qu'une récapitulation ou une référence relative aux conditions d'application du logiciel générique et des outils d'application, et les normes auxquelles il est indispensable que l'application se conforme (par exemple: les principes de signalisation pour un système de signalisation).

8.4.2.3 Les exigences relatives aux données et algorithmes d'application traités par le logiciel générique du système doivent être spécifiées à ce stade.

8.4.2.4 Un Rapport de Vérification des Données/Algorithmes d'Application doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.2.5 se rapporte au Rapport de Vérification de Données/Algorithmes d'Application.

8.4.2.5 une fois que la Spécification des Exigences de l'Application a été définie, la vérification doit porter sur:

a) le fait que la Spécification des Exigences de l'Application satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques de 8.4.2.2 à 8.4.2.3,

b) la cohérence interne de la Spécification des Exigences de l'Application.

Les résultats doivent être consignés dans un Rapport de Vérification des Données/Algorithmes d'Application.

8.4.3 Architecture et Conception

La quantité et le type des composants matériels et logiciels génériques à utiliser dans l'application spécifique doivent être spécifiés. L'emplacement des composants des données d'application et des algorithmes d'application dans l'architecture de l'application spécifique doit être défini. Les données et algorithmes d'application traités par le logiciel générique doivent être conçus à ce stade.

8.4.4 Production des Données/Algorithmes d'Application

8.4.4.1 Le processus de développement de l'application doit inclure la production et la compilation du code source des données/algorithmes génériques et spécifiques ainsi que les activités de vérification et d'essais liées à cette production. L'utilisation de langages en diagramme est recommandée pour produire le code source des algorithmes d'application. Se référer au Tableau A.16.

8.4.4.2 Un Rapport des Essais de l'Application doit être rédigé, sous la responsabilité du Chargé des Essais, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.4.3 se rapporte au Rapport des Essais de l'Application.

8.4.4.3 Le Rapport des Essais de l'Application doit documenter l'exécution correcte et complète des essais définis dans la Spécification d'essai de l'Application.

8.4.4.4 Le Rapport de Vérification de la Préparation d'Application doit

a) documenter chaque activité exécutée afin d'assurer le caractère correct et la complétude des données/algorithme et leur cohérence avec les principes de l'application et l'architecture de l'application générique,

b) évaluer la compatibilité des données/algorithmes avec l'application générique.

8.4.4.5 Une Spécification d'essai de l'Application doit être rédigée, sous la responsabilité du Chargé des Essais, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.4.6 se rapporte à la Spécification d'essai de l'Application.

8.4.4.6 La Spécification d'essai de l'Application doit spécifier les essais à réaliser à un stade intermédiaire ou final de la préparation des données/algorithmes, afin d'assurer:

a) la cohérence et la complétude des données/algorithmes par rapport aux principes de l'application,

b) la cohérence et la complétude des données/algorithmes par rapport à l'architecture de l'application spécifique.

8.4.4.7 Un Rapport de Vérification des Données/Algorithmes d'Application doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.4.8 se rapporte au Rapport de Vérification de Données/Algorithmes d'Application.

8.4.4.8 Une fois que la Spécification d'essai de l'Application a été définie, la vérification doit porter sur:

a) le fait que la Spécification d'essai de l'Application satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 8.4.4.6,

b) la cohérence interne de la Spécification d'essai de l'Application.

Les résultats doivent être consignés dans un Rapport de Vérification des Données/Algorithmes d'Application.

8.4.5 Intégration de l'Application et Acceptation des Essais

8.4.5.1 Dans certains systèmes, les données/algorithmes d'application peuvent être intégré(e)s dans le matériel et le logiciel génériques pour un essai en usine avant l'installation sur le système cible. Dans les cas où le niveau de confiance peut être obtenu par d'autres moyens, il n'est pas exclu de supprimer cette démarche. L'application doit alors être installée sur le système cible et les essais d'intégration au sein de l'installation complète doivent être effectués. Enfin, le système cible doit être mis en service comme système pleinement opérationnel et un processus d'acceptation finale du système cible dans l'installation complète doit être réalisé. Le Rapport des Essais de l'Application doit documenter l'exécution correcte et complète des essais définis dans la Spécification d'essai de l'Application. Le Rapport de Vérification de la Préparation de l'Application doit vérifier la complétude et le caractère correct des essais réalisés sur l'installation complète.

8.4.5.2 Une Spécification d'essai de l'Application doit être rédigée, sous la responsabilité du Chargé des Essais, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.5.3 se rapporte à la Spécification d'essai de l'Application.

8.4.5.3 La Spécification d'essai de l'Application doit spécifier les essais à réaliser pour assurer:

a) l'intégration correcte des données/algorithmes sur le matériel et le logiciel génériques, si besoin est,

b) l'intégration correcte des données/algorithmes avec l'installation complète.

8.4.5.4 Un Rapport de Vérification des Données/Algorithmes d'Application doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 8.2.

L'exigence en 8.4.5.5 se rapporte au Rapport de Vérification de Données/Algorithmes d'Application.

8.4.5.5 Une fois que la Spécification d'essai de l'Application a été définie, la vérification doit porter sur le fait que la Spécification d'essai de l'Application satisfait aux exigences spécifiques en 8.4.5.3.

8.4.6 Validation et Évaluation de l'Application

Les activités de validation et d'évaluation doivent contrôler la performance de chaque étape du cycle de vie.

8.4.7 Procédures et outils de préparation de l'application

8.4.7.1 Pour chaque nouveau type de système configuré par des données/algorithmes d'application, des procédures et outils spécifiques doivent être développés pour permettre d'appliquer aux installations du nouveau système le processus de développement d'application spécifié en 8.4.1. Le développement de ces outils doit être réalisé selon la présente Norme en parallèle avec le matériel et le logiciel génériques du système. Les activités de vérification, de validation et d'évaluation doivent assurer que les outils de préparation des données et le logiciel générique sont compatibles.

8.4.7.2 Tout processus de compilation doit être validé et évalué. Il doit être noté que des compilateurs spécialisés sont en général nécessaires pour la conversion des données et des algorithmes.

8.4.7.3 La totalité des données/algorithmes d'application et de la documentation associée pour chaque application spécifique doit être soumise aux exigences de déploiement du logiciel telles que spécifiées en 9.1.

8.4.7.4 La totalité des données/algorithmes d'application et de la documentation associée doit être soumise aux exigences de maintenance du logiciel spécifiées en 9.2.

8.4.7.5 La totalité des données/algorithmes d'application et de la documentation associée doit être placée sous la gestion de configuration selon les exigences spécifiées en 6.5 et 6.7. La gestion de configuration des données/algorithmes d'application peut être séparée de la partie logiciel générique.

8.4.7.6 Le Rapport de Vérification d'Application démontre la couverture et la mise en application des conditions d'application du logiciel générique et des outils d'application.

8.4.8 Développement de logiciel générique

8.4.8.1 Le développement du logiciel générique, qui prend en charge l'exécution des données/algorithmes d'application, doit se conformer aux exigences de 7.1 à 7.7 de la présente Norme. Les exigences supplémentaires suivantes doivent également être respectées.

8.4.8.2 Les types ou classes de fonction qui peuvent être configuré(e)s par des données/algorithmes d'application dans chaque système et sous-système doivent être identifié(e)s dans les documents de la Spécification des Exigences du Logiciel du logiciel générique. Le niveau d'intégrité de la sécurité alloué aux fonctions déterminera les normes à appliquer au développement ultérieur des données/algorithmes d'application pour toutes les installations du système.

8.4.8.3 Au cours de la phase de conception du logiciel générique, les interfaces détaillées entre le logiciel générique et les données/algorithmes d'application doivent être spécifiées, à moins qu'elles n'aient déjà été spécifiées lors d'une phase précédente du cycle de vie, par exemple en raison d'une exigence d'utilisation d'un langage d'application spécifique existant.

8.4.8.4 Il doit être mis en pratique une séparation rigide entre le logiciel générique et les données/algorithmes d'application, autrement dit il doit être possible de recompiler et de mettre à jour, soit le logiciel générique, soit les données/algorithmes d'application, sans la nécessité de mettre l'autre à jour, à moins qu'une modification n'ait été apportée dans l'interface définie entre le logiciel générique et les données/algorithmes d'application. De

même, les données/algorithmes spécifiques des applications doivent être séparé(e)s des données/algorithmes d'application génériques.

8.4.8.5 Les procédures de contrôle des modifications doivent assurer que toute modification apportée au logiciel générique peut être installée uniquement quand il a été établi que le logiciel révisé est compatible avec les données/algorithmes d'application d'origine ou que les données/algorithmes d'application ont fait l'objet d'une révision.

8.4.8.6 Une attention particulière doit être accordée au processus de vérification et à la phase des essais de validation afin d'assurer que toutes les combinaisons pertinentes de données et d'algorithmes sont examinées.

Si toutes les combinaisons pertinentes de données et d'algorithmes n'ont pas été examinées dans le processus de vérification, des essais et de validation du logiciel générique, cela doit être clairement identifié comme une limite d'utilisation du logiciel générique. Un complément du processus de vérification, des essais et de validation du logiciel générique doit être exécuté en cas de définition d'un certain nombre des données ou d'algorithmes au-delà de cette limite.

8.4.8.7 Lorsque cela est réalisable, le logiciel générique doit être conçu pour détecter les données/algorithmes d'application altéré(e)s.

8.4.8.8 Les concepteurs doivent éditer la Fiche de Livraison du logiciel générique et des outils d'application au plus tard dans la Phase des Essais d'Ensemble du Logiciel/Validation Finale relative au logiciel générique et aux outils d'application. Le contenu de ces documents doit être soumis aux activités de vérification et de validation.

Les rubriques suivantes doivent être traitées dans le document “Conditions d'application du logiciel générique et des outils d'application”:

a) les références aux manuels d'utilisateur du logiciel générique et outils d'application;

b) les éventuelles contraintes sur les données/algorithmes d'application, par exemple: règles d'architecture ou de codage imposées pour respecter les niveaux d'intégrité de la sécurité.

9 Déploiement et maintenance du logiciel

9.1 Déploiement du logiciel

9.1.1 Objet

Assurer que le logiciel fonctionne comme requis, en préservant le niveau requis d'intégrité de la sécurité logicielle et la sûreté de fonctionnement requise, lorsqu'il est déployé dans l'environnement final d'application.

9.1.2 Documents en entrée

Tous les documents de conception, de développement et d'analyse pertinents pour le déploiement.

9.1.3 Documents en sortie

a) Plan de livraison et de déploiement du Logiciel

b) Manuel de Déploiement du Logiciel

c) Fiche de livraison

d) Enregistrements du Déploiement

e) Rapport de Vérification du Déploiement

9.1.4 Exigences

9.1.4.1 Le déploiement doit être effectué sous la responsabilité du chef de projet.

9.1.4.2 Avant de livrer une diffusion de logiciel, le référentiel du logiciel doit être enregistré et maintenu traçable sous le contrôle de la gestion de configuration. Les logiciels préexistants et ceux qui sont développés conformément à une version antérieure de la présente Norme doivent également être inclus.

9.1.4.3 La diffusion du logiciel doit être reproductible tout au long du cycle de vie du référentiel du logiciel.

9.1.4.4 Une Fiche de livraison doit être rédigée, sous la responsabilité du Concepteur, sur la base des documents en entrée issus de 9.1.2.

L'exigence en 9.1.4.5 se rapporte à la Fiche de livraison.

9.1.4.5 Une fiche de livraison doit être fournie et définir:

a) les conditions d'application qui doivent être respectées,

b) les informations de compatibilité parmi les composants du logiciel et entre le logiciel et le matériel,

c) toutes les restrictions dans l'utilisation du logiciel (voir 7.7.4.13).

9.1.4.6 Un Manuel de Déploiement du Logiciel doit être rédigé sur la base des documents en entrée issus de 9.1.2.

L'exigence en 9.1.4.7 se rapporte au Manuel de Déploiement du Logiciel.

9.1.4.7 Le Manuel de Déploiement du Logiciel doit définir les procédures pour identifier et installer correctement une diffusion de logiciel.

9.1.4.8 En cas de déploiement incrémentiel (c'est-à-dire le déploiement de composants individuels), il est hautement recommandé pour les SIL 3 et SIL 4 et recommandé pour les SIL 1 et SIL 2 que le logiciel soit conçu de manière à contenir des moyens assurant que l'activation de versions incompatibles du logiciel est exclue.

9.1.4.9 La gestion de configuration doit assurer qu'aucun dommage ne résultera de la présence simultanée de différentes versions des mêmes composants logiciels lorsque cela ne peut être évité.

9.1.4.10 Une procédure de retour en arrière (c'est-à-dire une capacité de retourner à la diffusion précédente) doit être disponible lors de l'installation d'une nouvelle diffusion du logiciel.

9.1.4.11 Le logiciel doit avoir des mécanismes d'auto-identification intégrés, permettant son identification au moment du processus de chargement et après chargement dans la cible. Il convient que le mécanisme d'auto-identification indique les informations de version pour le logiciel et toutes les éventuelles données de configuration ainsi que l'identité du produit.

NOTE Les données du code contenant les informations sur la diffusion du logiciel peuvent être protégées par codage (voir Tableau A.3 “Codes de détection d'erreur”).

9.1.4.12 Un Registre de Déploiement doit être rédigé sur la base des documents en entrée issus de 9.1.2.

L'exigence en 9.1.4.13 se rapporte au Registre de Déploiement du Logiciel.

9.1.4.13 Un Registre de Déploiement doit donner la preuve que le logiciel prévu a été chargé, par examen des mécanismes d'auto-identification intégrés (voir 9.1.4.11). Ce registre doit être conservé parmi les documents liés au système livré comme les autres vérifications et fait partie de la mise en service et de l'acceptation.

9.1.4.14 Le logiciel déployé doit être traçable aux installations livrées.

NOTE Ceci revêt une importance spéciale lorsque des défauts critiques sont découverts et nécessitent d'être corrigés dans plus d'une installation.

9.1.4.15 Des informations relatives au diagnostic doivent être fournies par le logiciel, comme partie de la surveillance des défauts.

9.1.4.16 Un Rapport de Vérification de Déploiement doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 9.1.2.

Les exigences de 9.1.4.17 à 9.1.4.19 se rapportent au Rapport de Vérification du Déploiement.

9.1.4.17 Une fois que le Manuel de Déploiement du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Manuel de Déploiement du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.1.4.7,

b) la cohérence interne du Manuel de Déploiement du Logiciel.

9.1.4.18 Une fois que le Registre de Déploiement a été défini, la vérification doit porter sur:

a) le fait que le Registre de Déploiement satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.1.4.13,

b) la cohérence interne du Registre de Déploiement.

9.1.4.19 Une fois que la Fiche de livraison a été définie, la vérification doit porter sur:

a) le fait que la Fiche de livraison satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.1.4.5,

b) la cohérence interne de la Fiche de livraison.

Les résultats doivent être enregistrés dans un Rapport de Vérification du Déploiement.

9.1.4.20 Des mesures doivent être incluses dans le conditionnement du logiciel pour empêcher ou détecter les erreurs survenant pendant le stockage, le transfert, la transmission ou la duplication du code exécutable ou des données exécutables. Il est recommandé de coder le code exécutable (voir Tableau A.3 “Codes de détection d'erreur”) comme faisant partie de la vérification de l'intégrité du code dans le processus de chargement du logiciel.

9.2 Maintenance du logiciel

9.2.1 Objet

Assurer que le logiciel fonctionne comme requis, en préservant le niveau d'intégrité de la sécurité logicielle et la sûreté de fonctionnement lorsqu'on procède à des corrections, des améliorations ou des adaptations sur le logiciel lui-même. Voir aussi 6.6 “Contrôle des modifications et des évolutions” de la présente Norme et la phase 13 “Modification et remise à niveau” dans l'IEC 62278.

9.2.2 Documents en entrée

L'ensemble approprié des documents de conception, de développement et d'analyse.

9.2.3 Documents en sortie

a) Plan de Maintenance du Logiciel

b) Registres des Modifications du Logiciel

c) Enregistrements de la Maintenance du Logiciel

d) Rapport de Vérification de la Maintenance du Logiciel

9.2.4 Exigences

9.2.4.1 Bien que la présente Norme ne soit pas censée être rétroactive, car elle s'applique principalement aux nouveaux développements et ne s'applique dans sa totalité qu'à des logiciels existants si ceux-ci sont soumis à des modifications majeures, le paragraphe 9.2 concernant la maintenance du logiciel s'applique à tous les changements, même mineurs. Par contre, il est hautement recommandé d'appliquer la totalité de la présente Norme pendant les mises à niveau et la maintenance des logiciels existants.

9.2.4.2 Quel que soit le niveau d'intégrité de la sécurité logicielle, le fournisseur doit, avant de commencer tout travail de modification, décider si les actions de maintenance sont à considérer comme majeures ou mineures ou si les méthodes existantes de maintenance du système sont appropriées. La décision doit être justifiée et enregistrée par le fournisseur et doit être soumise à l'évaluation du Chargé d'évaluation.

9.2.4.3 La maintenance doit être réalisée conformément aux directives exposées dans l'ISO/IEC 90003.

9.2.4.4 La maintenabilité doit être conçue comme un aspect intrinsèque du logiciel, en particulier en respectant les exigences en 7.3, 7.4 et 7.5. La série ISO/IEC 25010 doit également être utilisée pour mettre en œuvre et vérifier le niveau minimum de maintenabilité.

9.2.4.5 Un Plan de Maintenance du Logiciel doit être rédigé sur la base des documents en entrée issus de 9.2.2.

L'exigence en 9.2.4.6 se rapporte au Plan de Maintenance du Logiciel.

9.2.4.6 Les procédures de maintenance du logiciel doivent être définies et enregistrées dans le Plan de Maintenance du Logiciel. Ces procédures doivent également traiter:

a) le contrôle du compte rendu des erreurs, des journaux des erreurs, des enregistrements de maintenance, de l'autorisation de modification et de la configuration logiciel/système ainsi que les techniques et mesures dans le Tableau A.10,

b) la vérification, la validation et l'évaluation de toute modification,

c) la définition de l'Autorité qui approuve le logiciel modifié, et

d) l'autorisation de la modification.

9.2.4.7 Un Enregistrement de la Maintenance du Logiciel doit être rédigé sur la base des documents en entrée issus de 9.2.2.

L'exigence en 9.2.4.8 se rapporte à l'Enregistrement de Maintenance du Logiciel.

9.2.4.8 Un Enregistrement de la Maintenance du Logiciel doit être établi pour chaque logiciel avant sa première diffusion et doit être mis à jour. Outre les exigences de l'ISO/IEC 90003:2014 concernant les "registres et rapports de maintenance" (voir ISO/IEC 90003:2014, “Maintenance”), cet Enregistrement doit également inclure:

a) des références à tous les registres des modifications de logiciel applicables à ce logiciel,

b) l'évaluation de l'impact du changement,

c) des scénarios d'essai pour les composants, incluant les données d'essai pour la revalidation et la régression, et

d) l'historique de la configuration logicielle.

9.2.4.9 Un Registre des Modifications du Logiciel doit être rédigé sur la base des documents en entrée issus de 9.2.2.

L'exigence en 9.2.4.10 se rapporte au Registre des Modifications du Logiciel.

9.2.4.10 Un Registre des Modifications du Logiciel doit être établi pour chaque activité de maintenance. Cet enregistrement doit inclure

a) la demande de modification ou de changement, la version, la nature du défaut, le changement requis et la source du changement,

b) une analyse de l'impact de l'activité de maintenance sur le système global, y compris l'interaction matérielle, logicielle et humaine ainsi que les interactions avec l'environnement et interactions éventuelles,

c) la spécification particulière de la modification ou du changement effectué(e), et

d) la revalidation, les essais de régression et la réévaluation de la modification ou du changement dans la limite requise par le niveau d'intégrité de la sécurité logicielle. La responsabilité en matière de revalidation peut varier d'un projet à l'autre, selon le niveau d'intégrité de la sécurité logicielle. L'impact de la modification ou du changement sur le processus de revalidation peut aussi se limiter à différents niveaux du système (seulement les composants modifiés, tous les composants affectés identifiés, le système complet). Le Plan de Validation du Logiciel doit donc aborder les deux problèmes, selon le niveau d'intégrité de la sécurité logicielle. Le degré d'indépendance de la revalidation doit être le même que celui de la validation.

9.2.4.11 Un Rapport de Vérification de la Maintenance du Logiciel doit être rédigé, sous la responsabilité du Chargé de vérification, sur la base des documents en entrée issus de 9.2.2.

Les exigences de 9.2.4.12 à 9.2.4.14 se rapportent au Rapport de Vérification de la Maintenance du Logiciel.

9.2.4.12 Une fois que le Plan de Maintenance du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Plan de Maintenance du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.2.4.6,

b) la cohérence interne du Plan de Maintenance du Logiciel.

9.2.4.13 Une fois que l'Enregistrement de Maintenance du Logiciel a été défini, la vérification doit porter sur:

a) le fait que L'Enregistrement de Maintenance du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.2.4.8,

b) la cohérence interne de l'Enregistrement de Maintenance du Logiciel.

9.2.4.14 Une fois que le Registre des Modifications du Logiciel a été défini, la vérification doit porter sur:

a) le fait que le Registre des Modifications du Logiciel satisfait aux exigences générales de lisibilité et de traçabilité de 5.3.2.7 à 5.3.2.10 et de 6.5.4.14 à 6.5.4.17 ainsi qu'aux exigences spécifiques en 9.2.4.10,

b) la cohérence interne du Registre des Modifications du Logiciel.

9.2.4.15 Les activités de maintenance doivent être effectuées selon le Plan de Maintenance du Logiciel.

9.2.4.16 Les techniques et mesures issues du Tableau A.10 doivent être choisies. La combinaison choisie doit être justifiée comme un ensemble satisfaisant à 4.8 et à 4.9.

9.2.4.17 La maintenance doit être réalisée avec un niveau de compétence, d'outils, de documentation, de planification et de gestion au moins égal à celui utilisé dans le développement initial du logiciel. Ceci doit s'appliquer également à la gestion de la configuration, au contrôle des évolutions, au contrôle des documents et à l'indépendance des parties impliquées.

9.2.4.18 Le contrôle des fournisseurs externes, le compte rendu des problèmes et les actions correctives doivent être gérés en fonction des mêmes critères spécifiés dans les alinéas applicables de l'Assurance Qualité du Logiciel (voir 6.5) que ceux appliqués pour le développement de nouveau logiciel.

9.2.4.19 Pour chaque problème consigné ou chaque mise en application communiquée, une analyse d'impact sur la sécurité doit être effectuée.

9.2.4.20 Pour les logiciels en maintenance, les actions d'atténuation à la hauteur du risque identifié doivent être prises afin d'assurer l'intégrité globale du système pendant que les problèmes communiqués font l'objet d'une investigation et sont corrigés.

Annexe A
(normative)

Critères de sélection des techniques et mesures

A.1 Généralités

Les articles de la présente Norme sont associés dans cette annexe aux tableaux d'articles (voir Article A.2, Tableau A.1 à Tableau A.11) pour illustrer les moyens d'assurer la conformité. Il existe aussi des tableaux de niveau inférieur, les tableaux détaillés (voir Article A.3, Tableau A.12 à Tableau A.23), qui développent certaines entrées des tableaux d'articles. Par exemple, l'entrée “Modélisation” dans le Tableau A.2 est développée dans le Tableau A.17. Il existe aussi une Annexe D informative qui est référencée à partir des tableaux d'articles.

Avec chaque technique ou mesure figurant dans les tableaux, il existe une exigence pour chaque niveau d'intégrité de la sécurité logicielle (SIL). Dans cette version du document, les exigences pour les niveaux 1 et 2 d'intégrité de la sécurité logicielle sont les mêmes pour chaque technique. De la même façon, chaque technique comporte les mêmes exigences aux niveaux 3 et 4 d'intégrité de la sécurité logicielle. Ces exigences peuvent être les suivantes:

'M' ce symbole signifie que l'utilisation d'une technique est obligatoire ("Mandatory"),

'HR' ce symbole signifie que cette technique ou mesure est Hautement Recommandée pour ce niveau d'intégrité de la sécurité. Si cette technique ou mesure n'est pas utilisée, la justification de l'utilisation d'autres techniques doit être détaillée dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel;

'R' ce symbole signifie que la technique ou mesure est Recommandée pour ce niveau d'intégrité de la sécurité. Il s'agit d'un niveau de recommandation inférieur à un symbole 'HR' et il est possible de combiner ces techniques pour former une partie d'un ensemble conditionnel;

'-' ce symbole signifie que la technique ou mesure ne comporte aucune recommandation pour ou contre son utilisation;

'NR' ce symbole signifie que la technique ou mesure est catégoriquement Non Recommandée pour ce niveau d'intégrité de la sécurité. Si cette technique ou mesure est utilisée, la raison de cette décision doit être détaillée dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel.

La combinaison de techniques ou de mesures est à énoncer dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel en sélectionnant une ou plusieurs techniques ou mesures, à moins que les notes jointes au tableau n'imposent d'autres exigences. Ces notes peuvent inclure la référence à des techniques approuvées ou à des combinaisons de techniques approuvées. Si de telles techniques ou combinaisons, y compris toutes les techniques obligatoires respectives, sont utilisées, le chargé d'évaluation doit accepter leur validité et ne doit se préoccuper que de leur application correcte. Si un ensemble différent de techniques est utilisé et peut être justifié, il est admis que le chargé d'évaluation les trouve acceptables.

A.2 Tableaux d'articles

Tableau A.1 – Problèmes liés au cycle de vie et Documentation (5.3) *(1 de 2)*

DOCUMENTATION	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
Planification					
1. Plan d'Assurance Qualité du Logiciel	HR	HR	HR	HR	HR
2. Rapport de vérification concernant l'Assurance Qualité du logiciel	HR	HR	HR	HR	HR
3. Plan de Gestion de la Configuration du Logiciel	HR	HR	HR	HR	HR
4. Plan de Vérification du Logiciel	HR	HR	HR	HR	HR
5. Plan de Validation du Logiciel	HR	HR	HR	HR	HR
Exigences relatives au logiciel					
6. Spécification des Exigences du Logiciel	HR	HR	HR	HR	HR
7. Spécification des Essais d'Ensemble du Logiciel	HR	HR	HR	HR	HR
8. Rapport de Vérification des Exigences du Logiciel	HR	HR	HR	HR	HR
Architecture et Conception					
9. Spécification de l'Architecture du Logiciel	HR	HR	HR	HR	HR
10. Spécification de la Conception du Logiciel	HR	HR	HR	HR	HR
11. Spécifications des Interfaces Logiciel	HR	HR	HR	HR	HR
12. Spécification des essais d'Intégration du Logiciel	HR	HR	HR	HR	HR
13. Spécification des essais d'Intégration Logiciel/Matériel	HR	HR	HR	HR	HR
14. Rapport de Vérification de l'Architecture et de la Conception du Logiciel	HR	HR	HR	HR	HR
Conception du composant					
15. Spécification de la Conception des Composants logiciels	R	HR	HR	HR	HR
16. Spécification des essais des Composants Logiciels	R	HR	HR	HR	HR
17. Rapport de Vérification de la Conception des Composants Logiciels	R	HR	HR	HR	HR
Mise en œuvre et essais des composants					
18. Code Source du Logiciel et Documentation de Support	HR	HR	HR	HR	HR
19. Rapport des Essais des Composants Logiciels	R	HR	HR	HR	HR
20. Rapport de Vérification du Code Source du Logiciel	HR	HR	HR	HR	HR
intégration					
21. Rapport des Essais d'Intégration du Logiciel	HR	HR	HR	HR	HR
22. Rapport des Essais d'Intégration Logiciel/Matériel	HR	HR	HR	HR	HR
23. Rapport de Vérification de l'Intégration du Logiciel	HR	HR	HR	HR	HR
Essais d'ensemble du logiciel / Validation finale					
24. Rapport des Essais d'Ensemble du Logiciel	HR	HR	HR	HR	HR
25. Rapport de Vérification des Essais d'Ensemble du Logiciel	HR	HR	HR	HR	HR
26. Rapport de Validation du Logiciel	HR	HR	HR	HR	HR
27. Rapport de Validation des Outils	R	HR	HR	HR	HR
28. Fiche de livraison	HR	HR	HR	HR	HR

Tableau A.1 *(2 de 2)*

DOCUMENTATION	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
Systèmes configurés par des données/algorithmes d'application					
29. Spécification des Exigences de l'Application	HR	HR	HR	HR	HR
30. Plan de Préparation de l'Application (voir NOTE 2)	HR	HR	HR	HR	HR
31. Spécification d'essai de l'Application (voir NOTE 2)	HR	HR	HR	HR	HR
32. Architecture et Conception de l'Application (voir NOTE 2)	HR	HR	HR	HR	HR
33. Rapport de Vérification de la Préparation de l'Application	HR	HR	HR	HR	HR
34. Rapport des Essais d'Application	HR	HR	HR	HR	HR
35. Code Source des données/algorithmes d'application	HR	HR	HR	HR	HR
36. Rapport de vérification des données/algorithmes d'application	HR	HR	HR	HR	HR
Déploiement du logiciel					
37. Plan de livraison et de déploiement du Logiciel	R	HR	HR	HR	HR
38. Manuel de Déploiement du Logiciel	R	HR	HR	HR	HR
39. Fiche de livraison	HR	HR	HR	HR	HR
40. Enregistrements du Déploiement	R	HR	HR	HR	HR
41. Rapport de Vérification du Déploiement	R	HR	HR	HR	HR
Maintenance du logiciel					
42. Plan de Maintenance du Logiciel	R	HR	HR	HR	HR
43. Registres des Modifications du Logiciel	HR	HR	HR	HR	HR
44. Enregistrements de la Maintenance du Logiciel	R	HR	HR	HR	HR
45. Rapport de Vérification de la Maintenance du Logiciel	R	HR	HR	HR	HR
Évaluation du logiciel					
46. Plan d'Évaluation du Logiciel	R	HR	HR	HR	HR
47. Rapport d'Évaluation du Logiciel	R	HR	HR	HR	HR

NOTE 1 Selon 5.3.2.11 et 5.3.2.12, les documents peuvent être regroupés différemment.

NOTE 2 Le fait que les documents 29, 30 et 31 soient HR ou R dépend de l'importance définie dans le processus et de l'endroit où la vérification a lieu. Par exemple, il n'est pas exclu que les données soient juste à vérifier mais soient soumises à essai dans le domaine du système alors que des propriétés plus fonctionnelles sont tenues de faire l'objet à la fois d'essai et de vérification. Dans ce cas, HR a été marqué, mais peut être un R facultatif.

NOTE 3 Pour le Plan d'Évaluation du Logiciel et le Rapport d'évaluation du Logiciel, voir 6.4.1.2.

NOTE 4 Le Rapport d'Assurance Qualité du Logiciel est inclus dans le Rapport de Gestion de la Qualité défini dans l'IEC 62425.

Tableau A.2 – Spécification des Exigences du Logiciel (7.2)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Méthodes formelles (basées sur une approche mathématique)	D.28	-	R	R	HR	HR
2. Modélisation	Tableau A.17	R	R	R	HR	HR
3. Méthodologie structurée	D.52	R	R	R	HR	HR
4. Tables de Décision	D.13	R	R	R	HR	HR
Exigences: a) La Spécification des Exigences du Logiciel doit inclure une description du problème en langage naturel et toutes les notations formelles ou non formelles nécessaires. b) Le tableau reproduit des exigences supplémentaires permettant d'obtenir une spécification claire et précise. Une ou plusieurs de ces techniques doivent être sélectionnées pour satisfaire au niveau d'intégrité de la sécurité logicielle utilisé.						

Tableau A.3 – Architecture du Logiciel (7.3)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Programmation défensive	D.14	-	HR	HR	HR	HR
2. Détection des défauts & diagnostic	D.26	-	R	R	HR	HR
3. Codes correcteurs d'erreurs	D.19	-	-	-	-	-
4. Codes détecteurs d'erreurs	D.19	-	R	R	HR	HR
5. Programmation par assertion	D.24	-	R	R	HR	HR
6. Techniques de sécurité contrôlée (safety bag)	D.47	-	R	R	R	R
7. Programmation diversifiée	D.16	-	R	R	HR	HR
8. Bloc de rattrapage	D.44	-	R	R	R	R
9. Rattrapage par régression	D.5	-	NR	NR	NR	NR
10. Rattrapage par progression	D.30	-	NR	NR	NR	NR
11. Mécanismes de rattrapage par réexécution	D.46	-	R	R	R	R
12. Mémorisation des cas exécutés	D.36	-	R	R	HR	HR
13. Intelligence artificielle – Correction des défauts	D.1	-	NR	NR	NR	NR
14. Reconfiguration dynamique du logiciel	D.17	-	NR	NR	NR	NR
15. Analyse des effets des erreurs du logiciel	D.25	-	R	R	HR	HR
16. Dégradation contrôlée	D.31	-	R	R	HR	HR
17. Masquage d'informations	D.33	-	-	-	-	-
18. Encapsulation d'informations	D.33	R	HR	HR	HR	HR
19. Interface entièrement définie	D.38	HR	HR	HR	M	M
20. Méthodes formelles	D.28	-	R	R	HR	HR
21. Modélisation	Tableau A.17	R	R	R	HR	HR
22. Méthodologie structurée	D.52	R	HR	HR	HR	HR
23. Modélisation prise en charge par conception assistée par ordinateur et par les outils de spécification	Tableau A.17	R	R	R	HR	HR

Exigences:

a) Les combinaisons de techniques approuvées applicables aux niveaux 3 et 4 d'intégrité de la sécurité logicielle sont les suivantes:

1) 1, 7, 19, 22 et une qui est choisie entre 4, 5, 12 et 21;

2) 1, 4, 19, 22 et une qui est choisie entre 2, 5, 12, 15 et 21.

b) Les combinaisons de techniques approuvées applicables aux niveaux 1 et 2 d'intégrité de la sécurité logicielle sont les suivantes: 1, 19, 22 et une qui est choisie entre 2, 4, 5, 7, 12, 15 et 21.

c) Il est permis de définir certains de ces problèmes au niveau du système.

d) Des codes de détection d'erreurs peuvent être utilisés selon les exigences de la norme EN 50159.

NOTE La technique/mesure 19 concerne les Interfaces externes.

Tableau A.4 – Conception et mise en œuvre du logiciel (7.4)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Méthodes formelles	D.28	-	R	R	HR	HR
2. Modélisation	Tableau A.17	R	HR	HR	HR	HR
3. Méthodologie structurée	D.52	R	HR	HR	HR	HR
4. Approche modulaire	D.38	HR	M	M	M	M
5. Composants	Tableau A.20	HR	HR	HR	HR	HR
6. Normes de conception et de codage	Tableau A.12	HR	HR	HR	M	M
7. Programmes analysables	D.2	HR	HR	HR	HR	HR
8. Langage de programmation à fort typage	D.49	R	HR	HR	HR	HR
9. Programmation structurée	D.53	R	HR	HR	HR	HR
10. Langage de programmation	Tableau A.15	R	HR	HR	HR	HR
11. Sous-ensemble de langage	D.35	-	-	-	HR	HR
12. Programmation orientée objet	Tableau A.22 D.57	R	R	R	R	R
13. Programmation procédurale	D.60	R	HR	HR	HR	HR
14. Métaprogrammation	D.59	R	R	R	R	R

Exigences:

a) Une combinaison approuvée de techniques applicables aux niveaux 3 et 4 d'intégrité de la sécurité logicielle est 4, 5, 6, 8 et une qui est choisie entre 1 et 2.

b) Une combinaison approuvée de techniques applicables aux niveaux 1 et 2 d'intégrité de la sécurité logicielle est 3, 4, 5, 6 et une qui est choisie entre 8, 9 et 10.

c) La métaprogrammation doit être limitée à la production du code du logiciel source avant compilation.

Tableau A.5 – Vérification et Essais (6.2 et 7.3, 7.5)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Preuve formelle	D.29	-	R	R	HR	HR
2. Analyse statique	Tableau A.19	-	HR	HR	HR	HR
3. Analyse et essais dynamiques	Tableau A.13	-	HR	HR	HR	HR
4. Métriques	D.37	-	R	R	R	R
5. Traçabilité	D.58	R	HR	HR	M	M
6. Analyse des effets des erreurs du logiciel	D.25	-	R	R	HR	HR
7. Couverture des essais pour le code	Tableau A.21	R	HR	HR	HR	HR
8. Essais fonctionnels et boîte noire	Tableau A.14	HR	HR	HR	M	M
9. Essais de performance	Tableau A.18	-	HR	HR	HR	HR
10. Essais d'interface	D.34	HR	HR	HR	HR	HR

Exigences:

a) Pour les niveaux 3 et 4 d'intégrité de la sécurité logicielle, la combinaison approuvée de techniques est 3, 5, 7, 8 et une qui est choisie entre 1, 2 et 6.

b) Pour les niveaux 1 et 2 d'intégrité de la sécurité logicielle, la combinaison approuvée de techniques est 5 avec une qui est choisie entre 2, 3 et 8.

NOTE 1 Les techniques/mesures 1, 2, 4, 5, 6 et 7 concernent les activités de vérification.

NOTE 2 Les techniques/mesures 3, 8, 9 et 10 concernent les activités des essais.

Tableau A.6 – Intégration (7.6)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Essais fonctionnels et boîte noire	Tableau A.14	HR	HR	HR	HR	HR
2. Essais de performance	Tableau A.18	-	R	R	HR	HR

Tableau A.7 – Essais d'Ensemble du Logiciel (6.2 et 7.7)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Essais de performance	Tableau A.18	-	HR	HR	M	M
2. Essais fonctionnels et boîte noire	Tableau A.14	HR	HR	HR	M	M
3. Modélisation	Tableau A.17	-	R	R	R	R
Exigence: a) Pour les niveaux 1 et 2 d'intégrité de la sécurité logicielle, une combinaison approuvée de techniques est 1 et 2.						

Tableau A.8 – Techniques d'analyse logicielle (6.3)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Analyse logicielle statique	D.13 D.37 Tableau A.19	R	HR	HR	HR	HR
2. Analyse logicielle dynamique	Tableau A.13 Tableau A.14	-	R	R	HR	HR
3. Schémas de cause et de conséquence	D.6	R	R	R	R	R
4. Analyse par arbre des événements	D.22	-	R	R	R	R
5. Analyse des effets des erreurs du logiciel	D.25	-	R	R	HR	HR
Exigence: a) Une ou plusieurs de ces techniques doivent être sélectionnées pour satisfaire au niveau d'intégrité de la sécurité logicielle utilisé.						

Tableau A.9 – Assurance Qualité du logiciel (6.5)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Accréditée par l'ISO 9001	7.1	R	HR	HR	HR	HR
2. Conforme à l'ISO 9001	7.1	M	M	M	M	M
3. Conforme à la norme ISO/IEC 90003	7.1	R	R	R	R	R
4. Système qualité de l'entreprise	7.1	M	M	M	M	M
5. Gestion de la configuration du logiciel	D.48	M	M	M	M	M
6. Listes de contrôle	D.7	R	HR	HR	HR	HR
7. Traçabilité	D.58	R	HR	HR	M	M
8. Enregistrement et analyse des données	D.12	HR	HR	HR	M	M
Exigence: a) Ce tableau doit être appliqué aux différents rôles et à toutes les phases.						

Tableau A.10 – Maintenance du Logiciel (9.2)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Analyse d'impact	D.32	R	HR	HR	M	M
2. Enregistrement et analyse des données	D.12	HR	HR	HR	M	M

Tableau A.11 – Techniques de préparation des données (8.4)

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Méthodes de spécification en tableaux	D.68	R	R	R	R	R
2. Langage spécifique à l'application	D.69	R	R	R	R	R
3. Simulation	D.42	R	HR	HR	HR	HR
4. Essais fonctionnels	D.42	M	M	M	M	M
5. Listes de contrôle	D.7	R	HR	HR	M	M
6. Inspection de Fagan	D.23	-	R	R	R	R
7. Examens formels de la conception	D.56	R	HR	HR	HR	HR
8. Preuve formelle du caractère correct (des données)	D.29	-	-	-	HR	HR
9. Revue de projet structurée	D.56	R	R	R	HR	HR

Exigences:

1) Pour les niveaux 1 et 2 d'intégrité de la sécurité logicielle, une combinaison approuvée de techniques est 1 et 4.

2) Pour les niveaux 3 et 4 d'intégrité de la sécurité logicielle, les combinaisons approuvées de techniques sont 1, 4, 5 et 7 ou 2, 3 et 6.

NOTE La description de la référence D.29 concerne les programmes alors que la technique 8 dans ce contexte s'applique à la preuve formelle du caractère correct des données.

A.3 Tableaux détaillés

Tableau A.12 – Normes de codage

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Norme de codage	D.15	HR	HR	HR	M	M
2. Guide du style de codage	D.15	HR	HR	HR	HR	HR
3. Pas d'objets dynamiques	D.15	-	R	R	HR	HR
4. Pas de variables dynamiques	D.15	-	R	R	HR	HR
5. Usage limité des pointeurs	D.15	-	R	R	R	R
6. Usage limité de la récursivité	D.15	-	R	R	HR	HR
7. Pas de branchements inconditionnels	D.15	-	HR	HR	HR	HR
8. Taille et complexité limitées des fonctions, sous-programmes et méthodes	D.38	HR	HR	HR	HR	HR
9. Stratégie de Point d'entrée/sortie pour les fonctions, sous-programmes et méthodes	D.38	R	HR	HR	HR	HR
9. Nombre limité de paramètres de sous-programme	D.38	R	R	R	R	R
10. Utilisation limitée de variables globales	D.38	HR	HR	HR	M	M

Exigence:

a) Il est accepté que les techniques 3, 4 et 5 puissent être présentes comme partie intégrante d'un compilateur ou d'un traducteur validé.

Tableau A.13 – Analyse et Essais dynamiques

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Exécution d'un scénario d'essai à partir d'une analyse des valeurs aux limites	D.4	-	HR	HR	HR	HR
2. Exécution d'un scénario d'essai à partir de la supposition d'erreurs	D.20	R	R	R	R	R
3. Exécution d'un scénario d'essai à partir de l'insertion d'erreurs	D.21	-	R	R	R	R
4. Modélisation des performances	D.39	-	R	R	HR	HR
5. Essais de classes d'équivalence et de partition d'entrée	D.18	R	R	R	HR	HR
6. Essais structurels	D.50	-	R	R	HR	HR
Exigence: a) Pour les scénarios d'essai, l'analyse se fait au niveau du sous-système et repose sur la spécification et/ou sur la spécification et le code.						

Tableau A.14 – Essai fonctionnel/boîte noire

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Exécution d'un scénario d'essai à partir des schémas de cause et de conséquence	D.6	-	-	-	R	R
2. Prototypage/Animation	D.43	-	-	-	R	R
3. Analyse des valeurs aux limites	D.4	R	HR	HR	HR	HR
4. Essais de classes d'équivalence et de partition d'entrée	D.18	R	HR	HR	HR	HR
5. Simulation du processus	D.42	R	R	R	R	R
Exigence: a) L'exhaustivité de la simulation dépendra de la limite du niveau d'intégrité de la sécurité logicielle, de la complexité et de l'application.						

Tableau A.15 – Langages de programmation textuels

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. ADA	D.54	R	HR	HR	HR	HR
2. MODULA-2	D.54	R	HR	HR	HR	HR
3. PASCAL	D.54	R	HR	HR	HR	HR
4. C ou C++	D.54 et D.35	R	R	R	R	R
5. PL/M	D.54	R	R	R	NR	NR
6. BASIC	D.54	R	NR	NR	NR	NR
7. Assembleur	D.54	R	R	R	R	R
8. C#	D.54 et D.35	R	R	R	R	R
9. JAVA	D.54 et D.35	R	R	R	R	R
10. Liste d'instructions	D.54	R	R	R	R	R

Exigences:

a) Le choix des langages doit être fondé sur les exigences données en 6.7 et 7.3.

b) Il n'y a aucune exigence de justifier les décisions prises d'exclure des langages de programmation spécifiques.

NOTE 1 Pour des informations relatives à l'évaluation de la pertinence d'un langage de programmation, voir la rubrique à l'Article D.54 ‘Langages de programmation adaptés’.

NOTE 2 Si un langage spécifique ne figure pas dans le tableau, il n'est pas automatiquement exclu. Il est censé se conformer à l'Article D.54.

NOTE 3 Il convient que les systèmes d'exécution associés à des langages sélectionnés qui sont nécessaires pour exécuter des programmes d'application soient encore justifiés quant à leur utilisation selon le niveau d'intégrité de la sécurité logicielle.

Tableau A.16 – Langages diagrammatiques pour algorithmes d'application

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Diagrammes fonctionnels	D.63	R	R	R	R	R
2. Graphes séquentiels de fonction	D.61	-	HR	HR	HR	HR
3. Diagrammes à contacts	D.62	R	R	R	R	R
4. Graphiques d'états	D.64	R	HR	HR	HR	HR

Tableau A.17 – Modélisation

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Modélisation de données	D.65	R	R	R	HR	HR
2. Diagrammes des flux de données	D.11	-	R	R	HR	HR
3. Diagrammes de flux de commande	D.66	R	R	R	HR	HR
4. Diagrammes d'états finis/Schémas de transition d'état	D.27	-	HR	HR	HR	HR
5. Réseaux de Pétri temporels	D.55	-	R	R	HR	HR
6. Tables de décision/de vérité	D.13	R	R	R	HR	HR
7. Méthodes formelles	D.28	-	R	R	HR	HR
8. Modélisation des performances	D.39	-	R	R	HR	HR
9. Prototypage/Animation	D.43	-	R	R	R	R
10. Schémas de structure	D.51	-	R	R	HR	HR
11. Diagrammes de séquence	D.67	R	HR	HR	HR	HR
Exigences: a) Une ligne directrice de modélisation doit être définie et utilisée. b) Au moins l'une des techniques HR doit être choisie.						

Tableau A.18 – Essais de Performance

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Essais en avalanche/en surcharge	D.3	-	R	R	HR	HR
2. Temps de réponse et contraintes de place mémoire	D.45	-	HR	HR	HR	HR
3. Exigences en matière de performance	D.40	-	HR	HR	HR	HR

Tableau A.19 – Analyse statique

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Analyse des valeurs aux limites	D.4	-	R	R	HR	HR
2. Listes de contrôle	D.7	-	R	R	R	R
3. Analyse de Flux de Contrôle	D.8	-	HR	HR	HR	HR
4. Analyse du flux de données	D.10	-	HR	HR	HR	HR
5. Supposition d'erreurs	D.20	-	R	R	R	R
6. Revues/Examens de la conception	D.56	HR	HR	HR	HR	HR

Tableau A.20 – Composants

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Masquage d'informations	D.33	-	-	-	-	-
2. Encapsulation d'informations	D.33	R	HR	HR	HR	HR
3. Limite du nombre de paramètres	D.38	R	R	R	R	R
4. Interface entièrement définie	D.38	R	HR	HR	M	M
Exigence: a) Le masquage et l'encapsulation d'informations sont seulement hautement recommandés s'il n'existe pas de stratégie générale pour l'accès aux données. NOTE La technique/mesure 4 concerne les Interfaces internes.						

Tableau A.21 – Couverture des essais pour le code

Critère de couverture des essais	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Instruction	D.50	R	HR	HR	HR	HR
2. Branchement	D.50	-	R	R	HR	HR
3. Conditions composées	D.50	-	R	R	HR	HR
4. Flux de données	D.50	-	R	R	HR	HR
5. Chemin	D.50	-	R	R	HR	HR

Exigences:

a) Pour chaque SIL, une mesure quantifiée de couverture doit être développée pour les essais entrepris. Cela peut soutenir le jugement sur la confiance acquise dans les essais et la nécessité de techniques supplémentaires.

b) Pour le SIL 3 ou 4, il convient de mesurer la couverture des essais au niveau composant selon:

- 2 et 3; ou
- 2 et 4; ou
- 5,

ou il convient de mesurer la couverture des essais au niveau d'intégration selon l'un ou plusieurs choisis parmi 2, 3, 4 ou 5.

c) D'autres critères peuvent être utilisés pour la couverture des essais, étant donné que cela peut être justifié. Ces critères dépendent de l'architecture du logiciel (voir Tableau A.3) et du langage de programmation (voir Tableau A.15 et Tableau A.16).

d) Tout code qu'il n'est pas pratique de soumettre à essai doit être démontré correct en utilisant une technique appropriée, par exemple une analyse statique issue du Tableau A.19.

NOTE 1 La couverture des instructions est automatiquement obtenue par les éléments 2 à 5.

NOTE 2 Les critères de couverture des essais dans ce tableau sont utilisés pour des essais structurels (basés sur code, boîte blanche). Les techniques/mesures pour les essais fonctionnels (basés sur spécification, boîte noire) sont données dans le Tableau A.14.

NOTE 3 Il est en général difficile d'obtenir un haut pourcentage de couverture. L'utilisation de l'exécution de scénarios d'essai à partir de valeurs aux limites (Article D.4) et des essais de classes d'équivalence et de partition d'entrée (Article D.18) peut permettre d'obtenir une couverture suffisante avec un plus petit nombre d'essais.

NOTE 4 La différence entre 2 et 3 dépend dans la pratique du niveau du langage de programmation et de l'utilisation de conditions composées. Lorsque des conditions simples sont utilisées, par exemple, comme résultat d'une compilation, 2 et 3 sont considérées comme étant identiques.

Tableau A.22 – Architecture de logiciel orienté objet

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Traçabilité du concept du domaine d'application aux classes de l'architecture	-	R	R	R	HR	HR
2. Utilisation de cadres adaptés, de combinaisons de classes et de modèles de conception communément utilisées	-	R	R	R	HR	HR
3. Conception détaillée orientée objet	Tableau A.23	R	R	R	HR	HR

Exigence:

a) Lorsque des cadres et des modèles de conception existants sont utilisés, les exigences relatives aux logiciels préexistants s'appliquent à ces cadres et modèles

NOTE 1 L'approche orientée objet présente les informations différemment des approches procédurales; la liste ci-après contient les recommandations qui requièrent une considération spécifique:

– compréhension des hiérarchies de classes et identification des fonction(s) logicielle(s) dont l'exécution aura lieu sur invocation d'une méthode donnée (y compris lors de l'utilisation d'une bibliothèque de classes existante);

– essais structurels (Tableau A.13).

La traçabilité depuis le domaine d'application jusqu'à l'architecture de classes est moins importante.

NOTE 2 Pour une partie du logiciel visé, il peut exister un cadre issu de logiciels préexistants qui a résolu avec succès une tâche similaire et qui est bien connu du personnel de développement. L'utilisation de ce cadre est considérée comme une bonne pratique.

Tableau A.23 – Conception détaillée orientée objet

TECHNIQUE/MESURE	Réf	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Il convient que les classes n'aient qu'un seul objectif	-	R	R	R	HR	HR
2. Héritage utilisé uniquement si la classe dérivée est un affinement de sa classe de base	-	R	HR	HR	HR	HR
3. Profondeur de l'héritage limitée par des normes de codage	-	R	R	R	HR	HR
4. Neutralisation d'opérations (méthodes) sous contrôle strict	-	R	R	R	HR	HR
5. Héritage multiple utilisé uniquement pour les classes d'interface	-	R	HR	HR	HR	HR
6. Héritage à partir de classes inconnues	-	-	-	-	NR	NR

Exigences:

a) Une classe donnée est caractérisée par le fait d'avoir une responsabilité donnée, c'est-à-dire la prise en charge de données étroitement liées et des opérations sur ces données.

b) Il faut veiller à éviter les dépendances circulaires entre objets.

Annexe B
(normative)

Principaux rôles et responsabilités relatifs au logiciel

Tableau B.1 – Spécification du Rôle du Gestionnaire des Exigences

Rôle: Gestionnaire des Exigences
Responsabilités: 1. doit être chargé de spécifier les exigences relatives au logiciel 2. doit être le propriétaire de la Spécification des Exigences du Logiciel 3. doit établir et maintenir la traçabilité vers et depuis les exigences du niveau système 4. doit assurer que les exigences relatives aux spécifications et aux logiciels sont prises en compte dans la gestion des modifications et de la configuration, y compris l'état, la version et le statut d'autorisation 5. doit assurer la cohérence et la complétude dans la Spécification des Exigences du Logiciel (avec référence aux exigences de l'utilisateur et de l'environnement final d'application) 6. doit développer et maintenir les documents d'exigences relatives au logiciel
Principales compétences: 1. doit être compétent en ingénierie des exigences 2. doit avoir l'expérience du domaine d'application 3. doit avoir l'expérience des critères de sécurité dans le domaine d'application 4. doit comprendre le rôle global du système et l'environnement d'application 5. doit comprendre les techniques analytiques et leurs résultats 6. doit comprendre les réglementations applicables 7. doit comprendre les exigences de l'IEC 62279

Tableau B.2 – Spécification du Rôle du Concepteur

Rôle: (Designer) Concepteur
Responsabilités: 1. doit transformer les exigences spécifiées relatives au logiciel en solutions acceptables 2. doit être propriétaire des solutions d'architecture et de conception descendante 3. doit définir ou sélectionner les méthodes de conception et les outils 4. doit appliquer les principes et normes appropriés de conception de la sécurité 5. doit développer des spécifications de composants, le cas échéant 6. doit maintenir la traçabilité vers et depuis les exigences spécifiées relatives au logiciel 7. doit développer et maintenir la documentation de la conception 8. doit assurer que les documents de conception sont pris en compte par la gestion des modifications et de la configuration
Principales compétences: 1. doit être compétent dans l'ingénierie appropriée au domaine d'application 2. doit être compétent en matière de principes de conception de la sécurité 3. doit être compétent en analyse de conception et en méthodologies des essais de conception 4. doit être capable de travailler dans les limites des contraintes de conception dans un environnement donné 5. doit être compétent pour comprendre le domaine du problème 6. doit comprendre toutes les contraintes imposées par la plateforme matérielle, le système d'exploitation et les systèmes d'interfaçage 7. doit comprendre les article/paragraphes correspondants de l'IEC 62279