BS EN 61508-3:2010

BSI Standards Publication

Functional safety of electrical/ electronic/programmable electronic safety-related systems

Part 3: Software requirements

raising standards worldwide™

National foreword

This British Standard is the UK implementation of EN 61508-3:2010. It is identical to IEC 61508-3:2010. It supersedes BS EN 61508-3:2002 which is withdrawn.

The UK participation in its preparation was entrusted by Technical Committee GEL/65, Measurement and control, to Subcommittee GEL/65/1, System considerations.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

ISBN 978 0 580 56235 8

ICS 13.260; 25.040.40; 29.020; 35.080

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2010.

Amendments issued since publication

Amd. No.	Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61508-3

May 2010

ICS 25.040.40

Supersedes EN 61508-3:2001

English version

Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements

(IEC 61508-3:2010)

Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité - Partie 3: Exigences concernant les logiciels
(CEI 61508-3:2010)

Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 3: Anforderungen an Software
(IEC 61508-3:2010)

This European Standard was approved by CENELEC on 2010-05-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Management Centre: Avenue Marnix 17, B - 1000 Brussels

Ref. No. EN 61508-3:2010 E

Foreword

The text of document 65A/550/FDIS, future edition 2 of IEC 61508-3, prepared by SC 65A, System aspects, of IEC TC 65, Industrial-process measurement, control and automation, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 61508-3 on 2010-05-01.

This European Standard supersedes EN 61508-3:2001.

It has the status of a basic safety publication according to IEC Guide 104.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN and CENELEC shall not be held responsible for identifying any or all such patent rights.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2011-02-01
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2013-05-01

Annex ZA has been added by CENELEC.

Endorsement notice

The text of the International Standard IEC 61508-3:2010 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

[1] IEC 61511 series	NOTE	Harmonized in EN 61511 series (not modified).
[2] IEC 62061	NOTE	Harmonized as EN 62061.
[3] IEC 61800-5-2	NOTE	Harmonized as EN 61800-5-2.
[4] IEC 61508-5:2010	NOTE	Harmonized as EN 61508-5:2010 (not modified).
[5] IEC 61508-6:2010	NOTE	Harmonized as EN 61508-6:2010 (not modified).
[6] IEC 61508-7:2010	NOTE	Harmonized as EN 61508-7:2010 (not modified).
[7] IEC 60601 series	NOTE	Harmonized in 60601 series (partially modified).
[8] IEC 61131-3	NOTE	Harmonized as EN 61131-3.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 61508-1	2010	Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements	EN 61508-1	2010
IEC 61508-2	2010	Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems	EN 61508-2	2010
IEC 61508-4	2010	Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 4: Definitions and abbreviations	EN 61508-4	2010
IEC Guide 104	1997	The preparation of safety publications and the use of basic safety publications and group safety publications	-	-
ISO/IEC Guide 51	1999	Safety aspects - Guidelines for their inclusion in standards	-	-

CONTENTS

INTRODUCTION

Systems comprised of electrical and/or electronic elements have been used for many years to perform safety functions in most application sectors. Computer-based systems (generically referred to as programmable electronic systems) are being used in all application sectors to perform non-safety functions and, increasingly, to perform safety functions. If computer system technology is to be effectively and safely exploited, it is essential that those responsible for making decisions have sufficient guidance on the safety aspects on which to make these decisions.

This International Standard sets out a generic approach for all safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic (E/E/PE) elements that are used to perform safety functions. This unified approach has been adopted in order that a rational and consistent technical policy be developed for all electrically-based safety-related systems. A major objective is to facilitate the development of product and application sector international standards based on the IEC 61508 series.

NOTE 1 Examples of product and application sector international standards based on the IEC 61508 series are given in the bibliography (see references [1], [2] and [3]).

In most situations, safety is achieved by a number of systems which rely on many technologies (for example mechanical, hydraulic, pneumatic, electrical, electronic, programmable electronic). Any safety strategy must therefore consider not only all the elements within an individual system (for example sensors, controlling devices and actuators) but also all the safety-related systems making up the total combination of safety-related systems. Therefore, while this International Standard is concerned with E/E/PE safety-related systems, it may also provide a framework within which safety-related systems based on other technologies may be considered.

It is recognized that there is a great variety of applications using E/E/PE safety-related systems in a variety of application sectors and covering a wide range of complexity, hazard and risk potentials. In any particular application, the required safety measures will be dependent on many factors specific to the application. This International Standard, by being generic, will enable such measures to be formulated in future product and application sector international standards and in revisions of those that already exist.

This International Standard

– considers all relevant overall, E/E/PE system and software safety lifecycle phases (for example, from initial concept, through design, implementation, operation and maintenance to decommissioning) when E/E/PE systems are used to perform safety functions;
– has been conceived with a rapidly developing technology in mind; the framework is sufficiently robust and comprehensive to cater for future developments;
– enables product and application sector international standards, dealing with E/E/PE safety-related systems, to be developed; the development of product and application sector international standards, within the framework of this standard, should lead to a high level of consistency (for example, of underlying principles, terminology etc.) both within application sectors and across application sectors; this will have both safety and economic benefits;
– provides a method for the development of the safety requirements specification necessary to achieve the required functional safety for E/E/PE safety-related systems;
– adopts a risk-based approach by which the safety integrity requirements can be determined;
– introduces safety integrity levels for specifying the target level of safety integrity for the safety functions to be implemented by the E/E/PE safety-related systems;

NOTE 2 The standard does not specify the safety integrity level requirements for any safety function, nor does it mandate how the safety integrity level is determined. Instead it provides a risk-based conceptual framework and example techniques.

– sets target failure measures for safety functions carried out by E/E/PE safety-related systems, which are linked to the safety integrity levels;

– sets a lower limit on the target failure measures for a safety function carried out by a single E/E/PE safety-related system. For E/E/PE safety-related systems operating in

  – a low demand mode of operation, the lower limit is set at an average probability of a dangerous failure on demand of 10^{-5};

  – a high demand or a continuous mode of operation, the lower limit is set at an average frequency of a dangerous failure of 10^{-9} [h^{-1}];

NOTE 3 A single E/E/PE safety-related system does not necessarily mean a single-channel architecture.

NOTE 4 It may be possible to achieve designs of safety-related systems with lower values for the target safety integrity for non-complex systems, but these limits are considered to represent what can be achieved for relatively complex systems (for example programmable electronic safety-related systems) at the present time.

– sets requirements for the avoidance and control of systematic faults, which are based on experience and judgement from practical experience gained in industry. Even though the probability of occurrence of systematic failures cannot in general be quantified the standard does, however, allow a claim to be made, for a specified safety function, that the target failure measure associated with the safety function can be considered to be achieved if all the requirements in the standard have been met;

– introduces systematic capability which applies to an element with respect to its confidence that the systematic safety integrity meets the requirements of the specified safety integrity level;

– adopts a broad range of principles, techniques and measures to achieve functional safety for E/E/PE safety-related systems, but does not explicitly use the concept of fail safe. However, the concepts of “fail safe” and “inherently safe” principles may be applicable and adoption of such concepts is acceptable providing the requirements of the relevant clauses in the standard are met.

FUNCTIONAL SAFETY OF ELECTRICAL/ELECTRONIC/ PROGRAMMABLE ELECTRONIC SAFETY-RELATED SYSTEMS –

Part 3: Software requirements

1 Scope

1.1 This part of the IEC 61508 series

a) is intended to be utilized only after a thorough understanding of IEC 61508-1 and IEC 61508-2;

b) applies to any software forming part of a safety-related system or used to develop a safety-related system within the scope of IEC 61508-1 and IEC 61508-2. Such software is termed safety-related software (including operating systems, system software, software in communication networks, human-computer interface functions, and firmware as well as application software);

c) provides specific requirements applicable to support tools used to develop and configure a safety-related system within the scope of IEC 61508-1 and IEC 61508-2;

d) requires that the software safety functions and software systematic capability are specified;

NOTE 1 If this has already been done as part of the specification of the E/E/PE safety-related systems (see 7.2 of IEC 61508-2), then it does not have to be repeated in this part.

NOTE 2 Specifying the software safety functions and software systematic capability is an iterative procedure; see Figures 3 and 6.

NOTE 3 See Clause 5 and Annex A of IEC 61508-1 for documentation structure. The documentation structure may take account of company procedures, and of the working practices of specific application sectors.

NOTE 4 Note: See 3.5.9 of IEC 61508-4 for definition of the term "systematic capability".

e) establishes requirements for safety lifecycle phases and activities which shall be applied during the design and development of the safety-related software (the software safety lifecycle model). These requirements include the application of measures and techniques, which are graded against the required systematic capability, for the avoidance of and control of faults and failures in the software;

f) provides requirements for information relating to the software aspects of system safety validation to be passed to the organisation carrying out the E/E/PE system integration;

g) provides requirements for the preparation of information and procedures concerning software needed by the user for the operation and maintenance of the E/E/PE safety-related system;

h) provides requirements to be met by the organisation carrying out modifications to safety-related software;

i) provides, in conjunction with IEC 61508-1 and IEC 61508-2, requirements for support tools such as development and design tools, language translators, testing and debugging tools, configuration management tools;

NOTE 4 Figure 5 shows the relationship between IEC 61508-2 and IEC 61508-3.

j) Does not apply for medical equipment in compliance with the IEC 60601 series.

1.2 IEC 61508-1, IEC 61598-2, IEC 61508-3 and IEC 61508-4 are basic safety publications, although this status does not apply in the context of low complexity E/E/PE safety-related systems (see 3.4.3 of IEC 61508-4). As basic safety publications, they are intended for use by technical committees in the preparation of standards in accordance with the principles contained in IEC Guide 104 and ISO/IEC Guide 51. IEC 61508-1, IEC 61508-2, IEC 61508-3 and IEC 61508-4 are also intended for use as stand-alone publications. The horizontal safety

function of this international standard does not apply to medical equipment in compliance with the IEC 60601 series.

1.3 One of the responsibilities of a technical committee is, wherever applicable, to make use of basic safety publications in the preparation of its publications. In this context, the requirements, test methods or test conditions of this basic safety publication will not apply unless specifically referred to or included in the publications prepared by those technical committees.

1.4 Figure 1 shows the overall framework of the IEC 61508 series and indicates the role that IEC 61508-3 plays in the achievement of functional safety for E/E/PE safety-related systems.

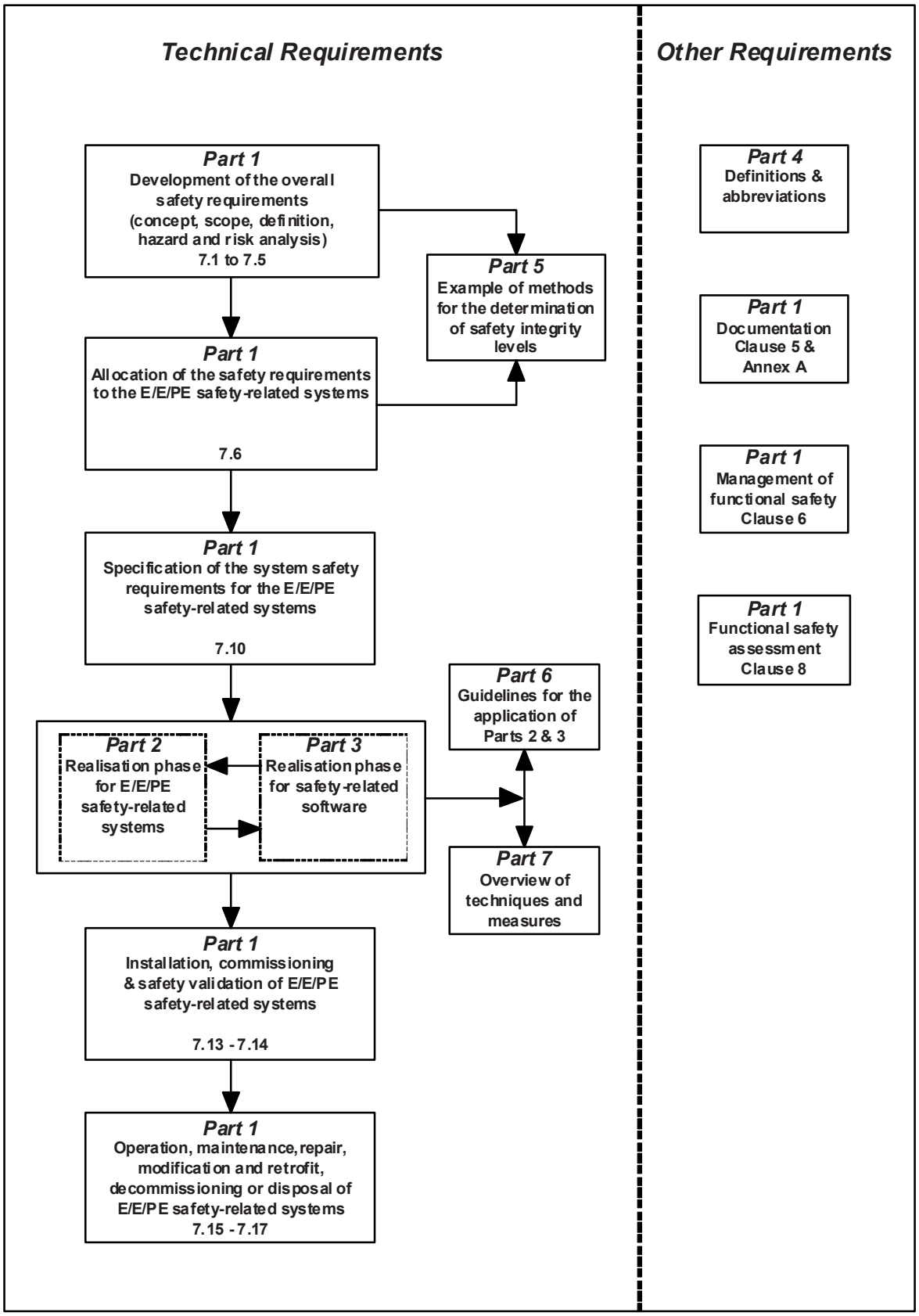

Figure 1 – Overall framework of the IEC 61508 series

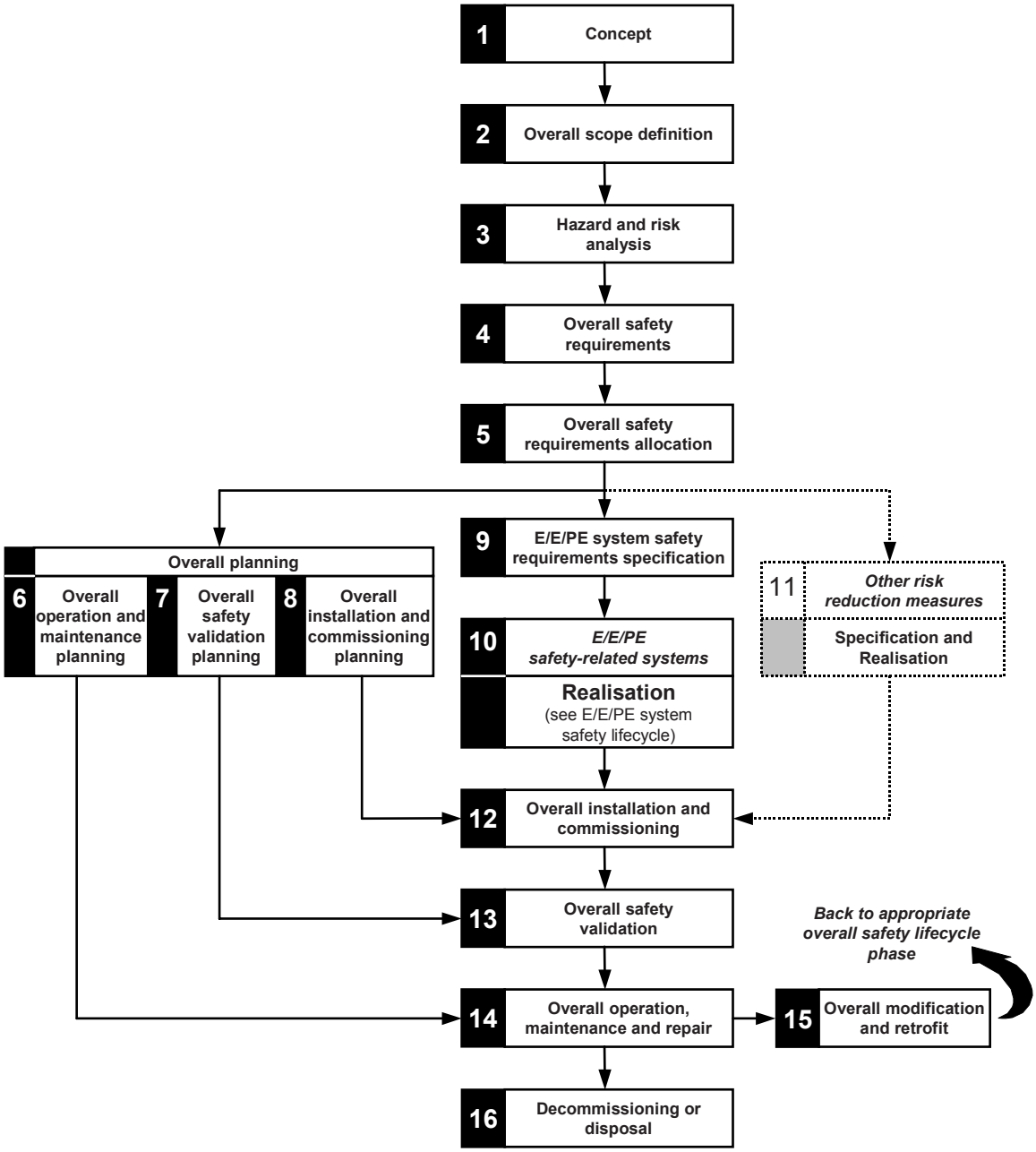

Figure 2 – Overall safety lifecycle

2 Normative references

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

IEC 61508-1: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements*

IEC 61508-2: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems*

IEC 61508-4: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations*

IEC Guide 104:1997, *The preparation of safety publications and the use of basic safety publications and group safety publications*

IEC/ISO Guide 51:1999, *Safety aspects – Guidelines for their inclusion in standards*

3 Definitions and abbreviations

For the purposes of this document, the definitions and abbreviations given in IEC 61508-4 apply.

4 Conformance to this standard

The requirements for conformance to this standard are given in Clause 4 of IEC 61508-1.

5 Documentation

The objectives and requirements for documentation are given in Clause 5 of IEC 61508-1.

6 Additional requirements for management of safety-related software

6.1 Objectives

The objectives are as detailed in 6.1 of IEC 61508-1.

6.2 Requirements

6.2.1 The requirements are as detailed in 6.2 of IEC 61508-1, with the following additional requirements.

6.2.2 The functional safety planning shall define the strategy for software procurement, development, integration, verification, validation and modification to the extent required by the safety integrity level of the safety functions implemented by the E/E/PE safety-related system.

NOTE The philosophy of this approach is to use the functional safety planning as an opportunity to customize this standard to take account of the required safety integrity for each safety function implemented by the E/E/PE safety-related system.

6.2.3 Software configuration management shall:

a) apply administrative and technical controls throughout the software safety lifecycle, in order to manage software changes and thus ensure that the specified requirements for safety-related software continue to be satisfied;

b) guarantee that all necessary operations have been carried out to demonstrate that the required software systematic capability has been achieved;

c) maintain accurately and with unique identification all configuration items which are necessary to meet the safety integrity requirements of the E/E/PE safety-related system. Configuration items include at least the following: safety analysis and requirements; software specification and design documents; software source code modules; test plans

and results; verification documents; pre-existing software elements and packages which are to be incorporated into the E/E/PE safety-related system; all tools and development environments which are used to create or test, or carry out any action on, the software of the E/E/PE safety-related system;

d) apply change-control procedures:

- to prevent unauthorized modifications; to document modification requests;
- to analyse the impact of a proposed modification, and to approve or reject the request;
- to document the details of, and the authorisation for, all approved modifications;
- to establish configuration baseline at appropriate points in the software development, and to document the (partial) integration testing of the baseline;
- to guarantee the composition of, and the building of, all software baselines (including the rebuilding of earlier baselines).

NOTE 1 Management decision and authority is needed to guide and enforce the use of administrative and technical controls.

NOTE 2 At one extreme, an impact analysis may include an informal assessment. At the other extreme, an impact analysis may include a rigorous formal analysis of the potential adverse impact of all proposed changes which may be inadequately understood or implemented. See IEC 61508-7 for guidance on impact analysis.

e) ensure that appropriate methods are implemented to load valid software elements and data correctly into the run-time system;

NOTE 3 This may include consideration of specific target location systems as well as general systems. Software other than application might need a safe loading method, e.g. firmware.

f) document the following information to permit a subsequent functional safety audit: configuration status, release status, the justification (taking account of the impact analysis) for and approval of all modifications, and the details of the modification;

g) formally document the release of safety-related software. Master copies of the software and all associated documentation and version of data in service shall be kept to permit maintenance and modification throughout the operational lifetime of the released software.

NOTE 4 For further information on configuration management, see IEC 61508-7

7 Software safety lifecycle requirements

7.1 General

7.1.1 Objective

The objective of the requirements of this subclause is to structure the development of the software into defined phases and activities (see Table 1 and Figures 3 to 6).

7.1.2 Requirements

7.1.2.1 A safety lifecycle for the development of software shall be selected and specified during safety planning in accordance with Clause 6 of IEC 61508-1.

7.1.2.2 Any software lifecycle model may be used provided all the objectives and requirements of this clause are met.

7.1.2.3 Each phase of the software safety lifecycle shall be divided into elementary activities with the scope, inputs and outputs specified for each phase.

NOTE See Figures 3, 4 and Table 1.

7.1.2.4 Provided that the software safety lifecycle satisfies the requirements of Table 1, it is acceptable to tailor the V-model (see Figure 6) to take account of the safety integrity and the complexity of the project.

NOTE 1 A software safety lifecycle model which satisfies the requirements of this clause may be suitably customized for the particular needs of the project or organisation. The full list of lifecycle phases in Table 1 is suitable for large newly developed systems. In small systems, it might be appropriate, for example, to merge the phases of software system design and architectural design.

NOTE 2 See Annex G for the characteristics of data-driven systems (e.g. full variability / limited variability programming languages, extent of data configuration) that may be relevant when customising the software safety lifecycle.

7.1.2.5 Any customisation of the software safety lifecycle shall be justified on the basis of functional safety.

7.1.2.6 Quality and safety assurance procedures shall be integrated into safety lifecycle activities.

7.1.2.7 For each lifecycle phase, appropriate techniques and measures shall be used. Annexes A and B provide a guide to the selection of techniques and measures, and references to IEC 61508-6 and IEC 61508-7. IEC 61508-6 and IEC 61508-7 give recommendations on specific techniques to achieve the properties required for systematic safety integrity. Selecting techniques from these recommendations does not guarantee by itself that the required safety integrity will be achieved.

NOTE Success in achieving systematic safety integrity depends on selecting techniques with attention to the following factors:

– the consistency and the complementary nature of the chosen methods, languages and tools for the whole development cycle;

– whether the developers use methods, languages and tools they fully understand;

– whether the methods, languages and tools are well-adapted to the specific problems encountered during development.

7.1.2.8 The results of the activities in the software safety lifecycle shall be documented (see Clause 5).

NOTE Clause 5 of IEC 61508-1 considers the documented outputs from the safety lifecycle phases. In the development of some E/E/PE safety-related systems, the output from some safety lifecycle phases may be a distinct document, while the documented outputs from several phases may be merged. The essential requirement is that the output of the safety lifecycle phase be fit for its intended purpose.

7.1.2.9 If at any phase of the software safety lifecycle, a modification is required pertaining to an earlier lifecycle phase, then an impact analysis shall determine (1) which software modules are impacted, and (2) which earlier safety lifecycle activities shall be repeated.

NOTE At one extreme, an impact analysis may include an informal assessment. At the other extreme, an impact analysis may include a rigorous formal analysis of the potential adverse impact of all proposed changes which may be inadequately understood or implemented. See IEC 61508-7 for guidance on impact analysis.

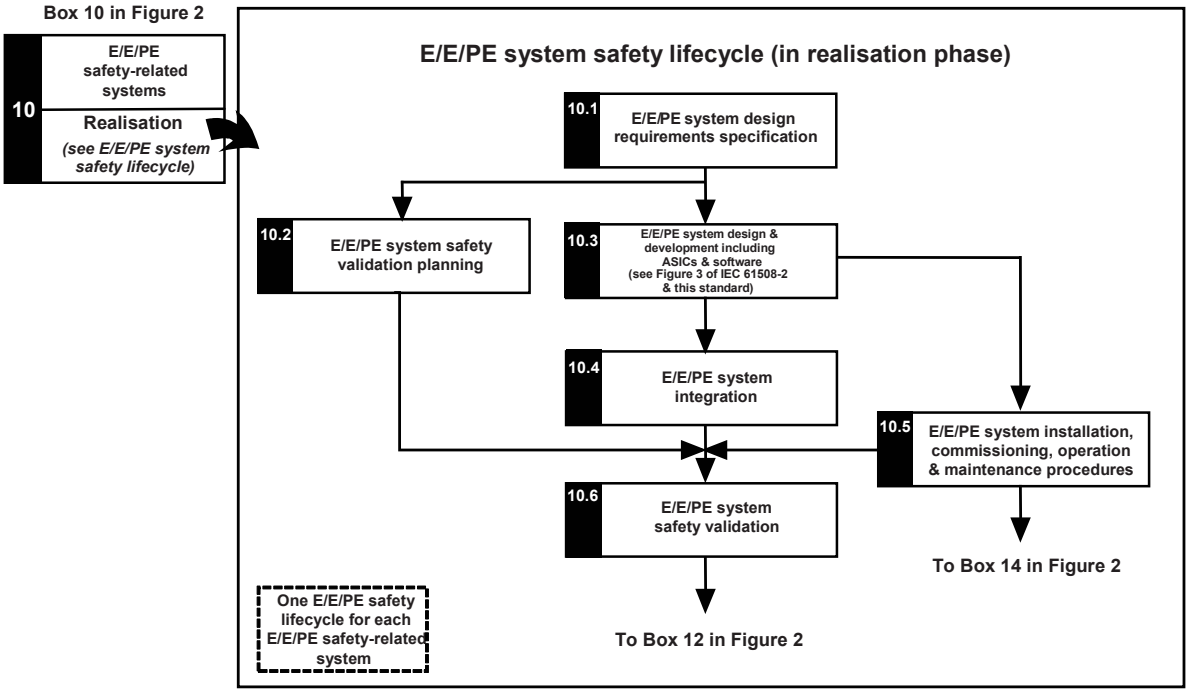

Figure 3 – E/E/PE system safety lifecycle (in realisation phase)

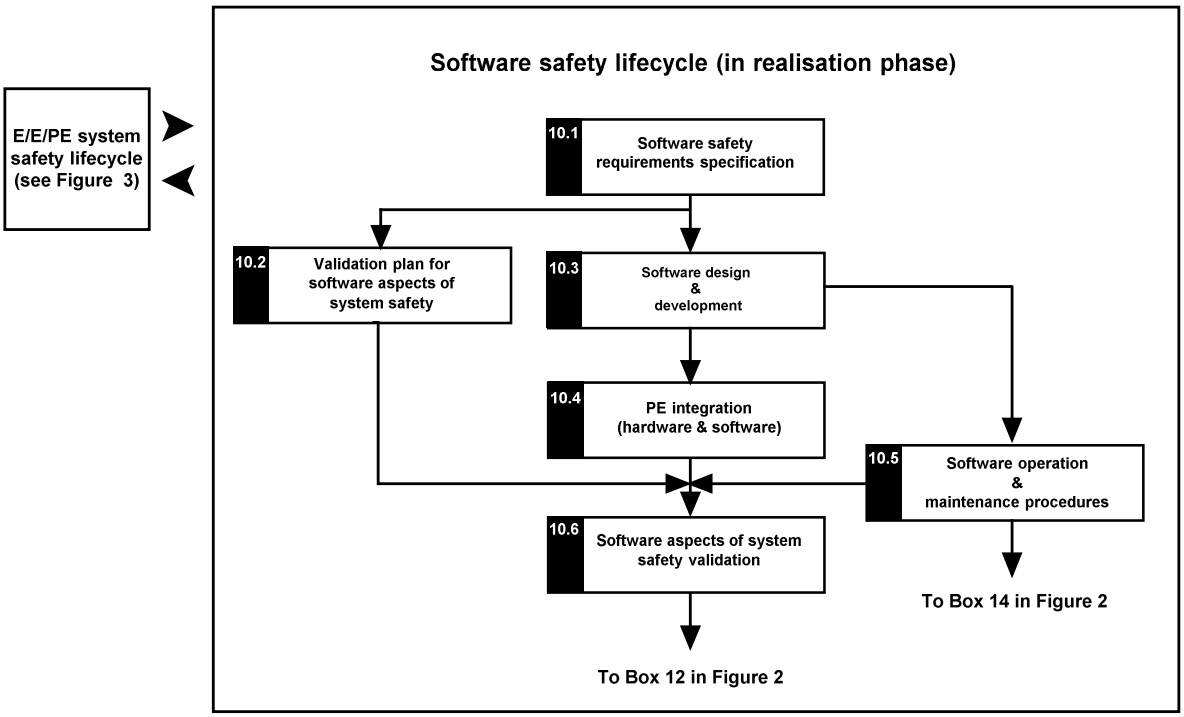

Figure 4 – Software safety lifecycle (in realisation phase)

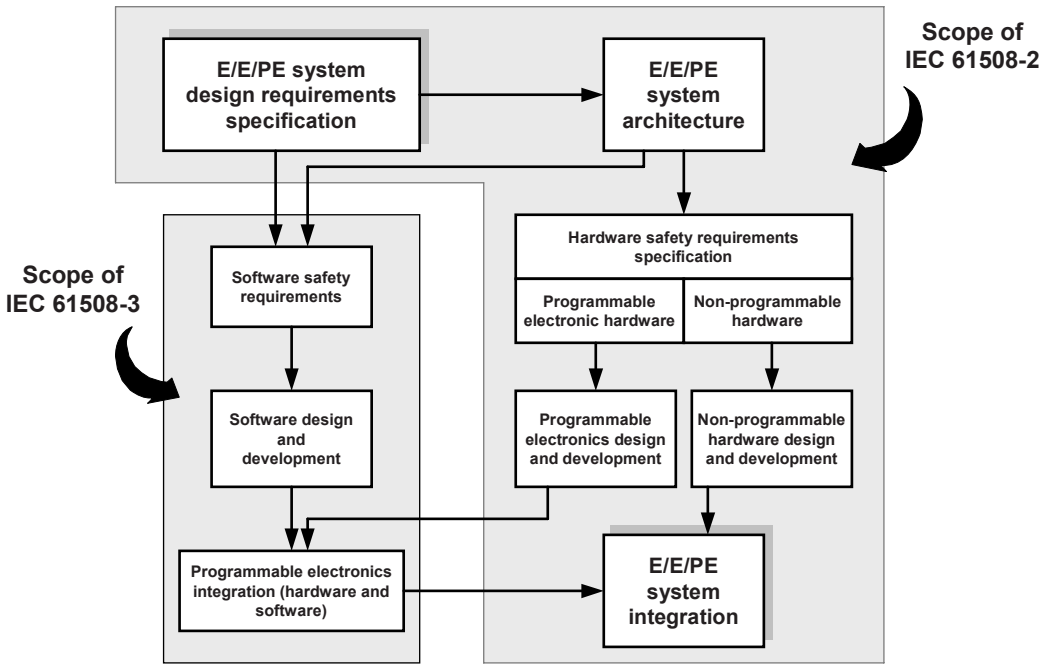

Figure 5 – Relationship and scope for IEC 61508-2 and IEC 61508-3

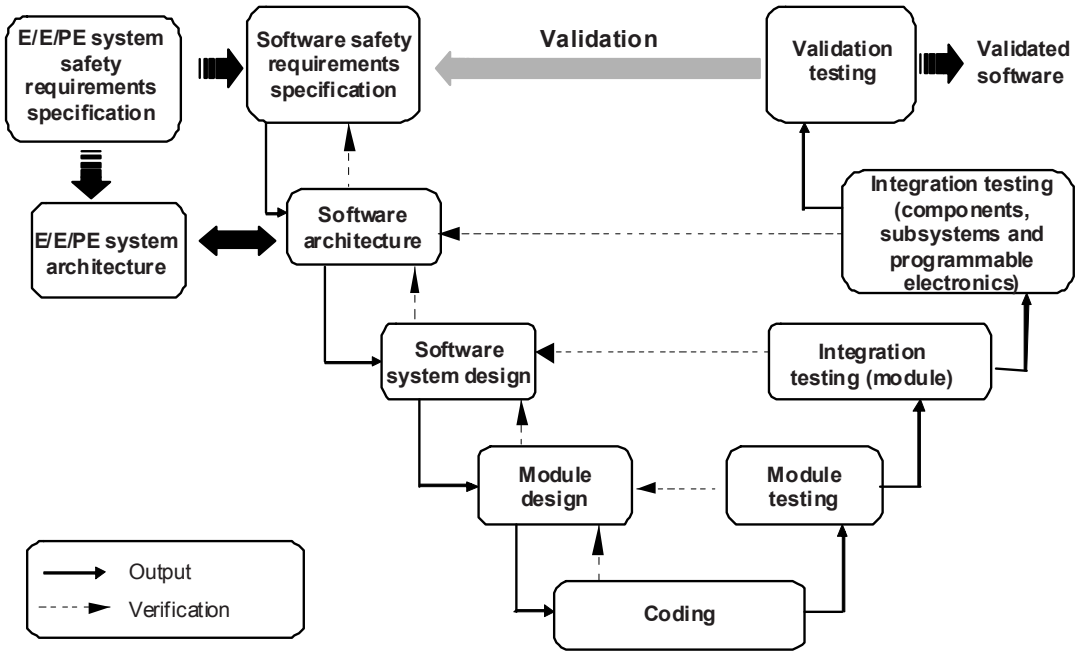

Figure 6 – Software systematic capability and the development lifecycle (the V-model)

Table 1 – Software safety lifecycle – overview

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 4 box number	Title					
10.1	Software safety requirements specification	To specify the requirements for safety-related software in terms of the requirements for software safety functions and the requirements for software systematic capability; To specify the requirements for the software safety functions for each E/E/PE safety-related system necessary to implement the required safety functions; To specify the requirements for software systematic capability for each E/E/PE safety-related system necessary to achieve the safety integrity level specified for each safety function allocated to that E/E/PE safety-related system	PE system; software system	7.2.2	E/E/PE safety requirements specification as developed during allocation (see IEC 61508-1) E/E/PE system safety requirements specification (from IEC 61508-2)	software safety requirements specification
10.2	Validation plan for software aspects of system safety	To develop a plan for validating the software aspects of system safety	PE system; software system	7.3.2	software safety requirements specification	validation plan for software aspects of system safety
10.3	Software design and development	Architecture: To create a software architecture that fulfils the specified requirements for safety-related software with respect to the required safety integrity level; To evaluate the requirements placed on the software by the hardware architecture of the E/E/PE safety-related system, including the significance of E/E/PE hardware/software interactions for safety of the equipment under control	PE system; software system	7.4.3	software safety requirements specification; E/E/PE system hardware architecture design (from IEC 61508-2)	software architecture design; software architecture integration test specification; software/ PE integration test specification (also required by IEC 61508-2)
10.3	Software design and development	Support tools and programming languages: To select a suitable set of tools, including languages and compilers, run-time system interfaces, user interfaces, and data formats and representations for the required safety integrity level, over the whole safety lifecycle of the software which assists verification, validation, assessment and modification	PE system; software system; support tools; programming language	7.4.4	software safety requirements specification; software architecture design	support tools and coding standards; selection of development tools

Table 1 *(continued)*

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 4 box number	Title					
10.3	Software design and development	Detailed design and development (software system design): To design and implement software that fulfils the specified requirements for safety-related software with respect to the required safety integrity level, which is analysable and verifiable, and which is capable of being safely modified	major elements and subsystems of software architectural design.	7.4.5	software architecture design; support tools and coding standards.	Software system design specification; software system integration test specification.
10.3	Software design and development	Detailed design and development (individual software module design): To design and implement software that fulfils the specified requirements for safety-related software with respect to the required safety integrity level, which is analysable and verifiable, and which is capable of being safely modified	software system design	7.4.5	software system design specification; support tools and coding standards	software module design specification; software module test specification
10.3	Software design and development	Detailed code implementation: To design and implement software that fulfils the specified requirements for safety-related software with respect to the required safety integrity level, which is analysable and verifiable, and which is capable of being safely modified	individual software modules	7.4.6	software module design specification; support tools and coding standards	source code listing; code review report
10.3	Software design and development	Software module testing: To verify that the requirements for safety-related software (in terms of the required software safety functions and the software systematic capability) have been achieved To show that each software module performs its intended function and does not perform unintended functions To ensure, in so far as it is appropriate, that configuration of PE systems by data fulfils the specified requirements for the software systematic capability	software modules	7.4.7	software module test specification; source code listing; code review report	software module test results; verified and tested software modules

Table 1 *(continued)*

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 4 box number	Title					
10.3	Software design and development	Software integration testing: To verify that the requirements for safety-related software (in terms of the required software safety functions and the software systematic capability) have been achieved To show that all software modules, elements and subsystems interact correctly to perform their intended function and do not perform unintended functions To ensure, in so far as it is appropriate, that configuration of PE systems by data fulfils the specified requirements for the software systematic capability	software architecture; software system	7.4.8	software system integration test specification	software system integration test results; verified and tested software system
10.4	Programmable electronics integration (hardware and software)	To integrate the software onto the target programmable electronic hardware; To combine the software and hardware in the safety-related programmable electronics to ensure their compatibility and to meet the requirements of the intended safety integrity level	program-mable electronics hardware; integrated software	7.5.2	software architecture integration test specification; software/PE integration test specification (also required by IEC 61508-2). Integrated programmable electronics	software architecture integration test results; programmable electronics integration test results; verified and tested integrated programmable electronics
10.5	Software operation and modification procedures	To provide information and procedures concerning software necessary to ensure that the functional safety of the E/E/PE safety-related system is maintained during operation and modification	as above	7.6.2	all above, as relevant	software operation and modification procedures
10.6	Software aspects of system safety validation	To ensure that the integrated system complies with the specified requirements for safety-related software at the intended safety integrity level	as above	7.7.2	validation plan for software aspects of system safety	software safety validation results; validated software
–	Software modification	To guide corrections, enhancements or adaptations to the validated software, ensuring that the required software systematic capability is sustained	as above	7.8.2	software modification procedures; software modification request	software modification impact analysis results; software modification log

Table 1 *(continued)*

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 4 box number	Title					
–	Software verification	To test and evaluate the outputs from a given software safety lifecycle phase to ensure correctness and consistency with respect to the outputs and standards provided as input to that phase	depends on phase	7.9.2	appropriate verification plan (depends on phase)	appropriate verification report (depends on phase)
–	Software functional safety assessment	To investigate and arrive at a judgement on the software aspects of the functional safety achieved by the E/E/PE safety-related systems	all above phases	8	software functional safety assessment plan	software functional safety assessment report

7.2 Software safety requirements specification

NOTE This phase is Box 10.1 of Figure 4.

7.2.1 Objectives

7.2.1.1 The first objective of the requirements of this subclause is to specify the requirements for safety-related software in terms of the requirements for software safety functions and the requirements for software systematic capability.

7.2.1.2 The second objective of the requirements of this subclause is to specify the requirements for the software safety functions for each E/E/PE safety-related system necessary to implement the required safety functions.

7.2.1.3 The third objective of the requirements of this subclause is to specify the requirements for software systematic capability for each E/E/PE safety-related system necessary to achieve the safety integrity level specified for each safety function allocated to that E/E/PE safety-related system.

7.2.2 Requirements

NOTE 1 These requirements will in most cases be achieved by a combination of generic embedded software and application specific software. It is the combination of both that provides the features that satisfy the following subclauses. The exact division between generic and application specific software depends on the chosen software architecture (see 7.4.3).

NOTE 2 For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the software safety requirements specification should be considered:

– completeness with respect to the safety needs to be addressed by software;

– correctness with respect to the safety needs to be addressed by software;

– freedom from intrinsic specification faults, including freedom from ambiguity;

– understandability of safety requirements;

– freedom from adverse interference of non-safety functions with the safety needs to be addressed by software;

– capability of providing a basis for verification and validation.

NOTE 3 The safety needs to be addressed by software is the set of safety functions and corresponding safety integrity requirements assigned to software functions by the design of the E/E/PE system. (The complete set of system safety needs is a larger set that includes also safety functions that do not depend on software). The completeness of the software safety requirements specification depends crucially on the effectiveness of earlier system lifecycle phases.

7.2.2.1 If the requirements for safety-related software have already been specified for the E/E/PE safety-related system (see Clause 7 of IEC 61508-2), then the specification of software safety requirements need not be repeated.

7.2.2.2 The specification of the requirements for safety-related software shall be derived from the specified safety requirements of the E/E/PE safety-related system (see IEC 61508-2, 7), and any requirements of safety planning (see Clause 6). This information shall be made available to the software developer.

NOTE 1 This requirement does not mean that there will be no iteration between the developer of the E/E/PE system and the developer of the software (IEC 61508-2 and IEC 61508-3). As the safety-related software requirements and the software architecture become more precise, there may be an impact on the E/E/PE system hardware architecture, and for this reason close co-operation between the hardware and software developer is essential. See Figure 5.

NOTE 2 Where a software design incorporates pre-existing reusable software, that software may have been developed without taking account of the current system requirement specification. See 7.4.2.12 for the requirements on the pre-existing software to satisfy the software safety requirements specification.

7.2.2.3 The specification of the requirements for safety-related software shall be sufficiently detailed to allow the design and implementation to achieve the required safety integrity (including any requirement for independence, see 7.4.3 of IEC 61508-2), and to allow an assessment of functional safety to be carried out.

NOTE The level of detail of the specification may vary with the complexity of the application. An adequate specification of functional behaviour may include requirements for accuracy, timing and performance, capacity, robustness, overload tolerance, and other characterising properties of the specific application.

7.2.2.4 In order to address independence, a suitable common cause failure analysis shall be carried out. Where credible failure mechanisms are identified, effective defensive measures shall be taken.

NOTE See Annex F for techniques for achieving one aspect of independence of software.

7.2.2.5 The software developer shall evaluate the information in 7.2.2.2 to ensure that the requirements are adequately specified. In particular the software developer shall consider the following:

a) safety functions;

b) configuration or architecture of the system;

c) hardware safety integrity requirements (programmable electronics, sensors, and actuators);

d) software systematic capability requirements;

e) capacity and response time;

f) equipment and operator interfaces, including reasonably foreseeable misuse.

NOTE Compatibility with any applications already in existence should be considered.

7.2.2.6 If not already adequately defined in specified safety requirements of the E/E/PE safety-related system, all relevant modes of operation of the EUC, of the E/E/PE system, and of any equipment or system connected to the E/E/PE system shall be detailed in the specified requirements for safety-related software.

7.2.2.7 The software safety requirements specification shall specify and document any safety-related or relevant constraints between the hardware and the software.

7.2.2.8 To the extent required by the E/E/PE hardware architecture design, and considering the possible increase in complexity, the software safety requirements specification shall consider the following:

a) software self-monitoring (for examples see IEC 61508-7);

b) monitoring of the programmable electronics hardware, sensors, and actuators;

c) periodic testing of safety functions while the system is running;

d) enabling safety functions to be testable when the EUC is operational;

e) software functions to execute proof tests and all diagnostic tests in order to fulfil the safety integrity requirement of the E/E/PE safety-related system.

NOTE Increased complexity resulting from the above considerations may require the architecture to be revisited.

7.2.2.9 When the E/E/PE safety-related system is required to perform non-safety functions, then the specified requirements for safety-related software shall clearly identify the non-safety functions.

NOTE See 7.4.2.8 and 7.4.2.9 for requirements on non-interference between safety functions and non-safety functions.

7.2.2.10 The software safety requirements specification shall express the required safety properties of the product, but not of the project as this is covered by safety planning (see Clause 6 of 61508-1). With reference to 7.2.2.1 to 7.2.2.9, the following shall be specified as appropriate:

a) the requirements for the following software safety functions:
   1) functions that enable the EUC to achieve or maintain a safe state;
   2) functions related to the detection, annunciation and management of faults in the programmable electronics hardware;
   3) functions related to the detection, annunciation and management of sensor and actuators faults;
   4) functions related to the detection, annunciation and management of faults in the software itself (software self-monitoring);
   5) functions related to the periodic testing of safety functions on-line (i.e. in the intended operational environment);
   6) functions related to the periodic testing of safety functions off-line (i.e. in an environment where the EUC is not being relied upon for its safety function);
   7) functions that allow the PE system to be safely modified;
   8) interfaces to non safety-related functions;
   9) capacity and response time performance;
   10) interfaces between the software and the PE system;

       NOTE 1 They include both off-line and on-line programming facilities.

   11) safety-related communications (see 7.4.11 of IEC 61508-2).
b) the requirements for the software systematic capability:
   1) the safety integrity level(s) for each of the functions in a) above;

      NOTE 2 See Annex A of IEC 61508-5 for information concerning the allocation of safety integrity to software elements.

   2) independence requirements between functions.

7.2.2.11 Where software safety requirements are expressed or implemented by configuration data, the data shall be:

a) consistent with the system safety requirements;

b) expressed in terms of the permitted range and authorized combinations of its operational parameters;

c) defined in a manner which is compatible with the underlying software (for example sequence of execution, run time, data structures, etc.).

NOTE 1 This requirement on application data is particularly relevant to data-driven applications. These are characterized as follows: the source code is pre-existing and the primary objective of the development activity is to

provide assurance that the configuration data correctly states the behaviour required from the application. There may be complex dependencies between data items, and the validity of data may change over time.

NOTE 2 See Annex G for guidance on data-driven systems.

7.2.2.12 Where data defines the interface between software and external systems, the following performance characteristics shall be considered in addition to 7.4.11 of IEC 61508-2:

a) the need for consistency in terms of data definitions;

b) invalid, out of range or untimely values;

c) response time and throughput, including maximum loading conditions;

d) best case and worst case execution time, and deadlock;

e) overflow and underflow of data storage capacity.

7.2.2.13 Operational parameters shall be protected against:

a) invalid, out of range or untimely values;

b) unauthorized changes;

c) corruption.

NOTE 1 Protection against unauthorized changes should be considered, taking account of both software-based and non-software mechanisms. Note that effective protection against unauthorized software changes can have adverse effects on safety e.g. when changes are needed rapidly and in stressful conditions.

NOTE 2 Although a person can form part of a safety-related system (see Clause 1 of IEC 61508-1), human factor requirements related to the design of E/E/PE safety-related systems are not considered in detail in this standard. However, the following human considerations should be addressed where appropriate:

- An operator information system should use the pictorial layout and the terminology the operators are familiar with. It should be clear, understandable and free from unnecessary details and/or aspects;
- Information about the EUC displayed to the operator should follow closely the physical arrangement of the EUC;
- If several display contents to the operator are feasible and/or if the possible operator actions allow interactions whose consequences cannot be seen at one glance, the information displayed should automatically contain at each state of a display or an action sequence, which state of the sequence is reached, which operations are feasible and which possible consequences can be chosen.

7.3 Validation plan for software aspects of system safety

NOTE 1 This phase is Box 10.2 of Figure 4.

NOTE 2 Software usually cannot be validated separately from its underlying hardware and system environment.

7.3.1 Objective

The objective of the requirements of this subclause is to develop a plan for validating the safety-related software aspects of system safety.

7.3.2 Requirements

7.3.2.1 Planning shall be carried out to specify the steps, both procedural and technical, that will be used to demonstrate that the software satisfies its safety requirements.

7.3.2.2 The validation plan for software aspects of system safety shall consider the following:

a) details of when the validation shall take place;

b) details of those who shall carry out the validation;

c) identification of the relevant modes of the EUC operation including:

   1) preparation for use including setting and adjustment;

   2) start up, teach, automatic, manual, semi-automatic, steady state operation;

3) re-setting, shut down, maintenance;
4) reasonably foreseeable abnormal conditions and reasonably foreseeable operator misuse.

d) identification of the safety-related software which needs to be validated for each mode of EUC operation before commissioning commences;

e) the technical strategy for the validation (for example analytical methods, statistical tests etc.);

f) in accordance with item e), the measures (techniques) and procedures that shall be used for confirming that each safety function conforms with the specified requirements for the safety functions, and the specified requirements for software systematic capability;

g) the required environment in which the validation activities are to take place (for example, for tests this could include calibrated tools and equipment);

h) the pass/fail criteria;

i) the policies and procedures for evaluating the results of the validation, particularly failures.

NOTE These requirements are based on the general requirements given in 7.8 of IEC 61508-1.

7.3.2.3 The validation shall give a rationale for the chosen strategy. The technical strategy for the validation of safety-related software shall include the following information:

a) choice of manual or automated techniques or both;

b) choice of static or dynamic techniques or both;

c) choice of analytical or statistical techniques or both.

d) choice of acceptance criteria based on objective factors or expert judgment or both.

7.3.2.4 As part of the procedure for validating safety-related software aspects, the scope and contents of the validation plan for software aspects of system safety shall be agreed with the assessor or with a party representing the assessor, if required by the safety integrity level (see Clause 8 of IEC 61508-1). This procedure shall also make a statement concerning the presence of the assessor during testing.

7.3.2.5 The pass/fail criteria for accomplishing software validation shall include:

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

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

c) other acceptance criteria, for example memory usage, timing and value tolerances.

7.4 Software design and development

NOTE This phase is box 10.3 of Figure 4.

7.4.1 Objectives

7.4.1.1 The first objective of the requirements of this subclause is to create a software architecture that fulfils the specified requirements for safety-related software with respect to the required safety integrity level.

7.4.1.2 The second objective of the requirements of this subclause is to evaluate the requirements placed on the software by the hardware architecture of the E/E/PE safety-related system, including the significance of E/E/PE hardware/software interactions for safety of the equipment under control.

7.4.1.3 The third objective of the requirements of this subclause is to select a suitable set of tools, including languages and compilers, run-time system interfaces, user interfaces, and data formats and representations for the required safety integrity level, over the whole safety lifecycle of the software which assists verification, validation, assessment and modification.

7.4.1.4 The fourth objective of the requirements of this subclause is to design and implement software that fulfils the specified requirements for safety-related software with respect to the required safety integrity level, which is analysable and verifiable, and which is capable of being safely modified.

7.4.1.5 The fifth objective of the requirements of this subclause is to verify that the requirements for safety-related software (in terms of the required software safety functions and the software systematic capability) have been achieved.

7.4.1.6 The sixth objective of the requirements of this subclause is to ensure, in so far as it is appropriate, that configuration of PE systems by data fulfils the specified requirements for the software systematic capability.

7.4.2 General requirements

7.4.2.1 Depending on the nature of the software development, responsibility for conformance with 7.4 can rest with the supplier of a safety related programming environment (e.g. PLC supplier) alone, or with the user of that environment (e.g. the application software developer) alone, or with both. The division of responsibility shall be determined during safety planning (see Clause 6).

NOTE See 7.4.3 for aspects of system and software architecture that are relevant to deciding on a practical division of responsibility.

7.4.2.2 In accordance with the required safety integrity level and the specific technical requirements of the safety function, the design method chosen shall possess features that facilitate:

a) abstraction, modularity and other features which control complexity;
b) the expression of:
   1) functionality;
   2) information flow between elements;
   3) sequencing and time related information;
   4) timing constraints;
   5) concurrency and synchronized access to shared resources;
   6) data structures and their properties;
   7) design assumptions and their dependencies;
   8) exception handling;
   9) design assumptions (pre-conditions, post-conditions, invariants);
   10) comments.
c) ability to represent several views of the design including structural and behavioural views;
d) comprehension by developers and others who need to understand the design;
e) verification and validation.

7.4.2.3 Testability and the capacity for safe modification shall be considered during the design activities in order to facilitate implementation of these properties in the final safety-related system.

NOTE Examples include maintenance modes in machinery and process plant.

7.4.2.4 The design method chosen shall possess features that facilitate software modification. Such features include modularity, information hiding and encapsulation.

NOTE See F.7.

7.4.2.5 The design representations shall be based on a notation which is unambiguously defined or restricted to unambiguously defined features.

7.4.2.6 As far as practicable the design shall keep the safety-related part of the software simple.

7.4.2.7 The software design shall include, commensurate with the required safety integrity level, self-monitoring of control flow and data flow. On failure detection, appropriate actions shall be taken.

7.4.2.8 Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate design measures ensure that the failures of non-safety functions cannot adversely affect safety functions.

7.4.2.9 Where the software is to implement safety functions of different safety integrity levels, then all of the software shall be treated as belonging to the highest safety integrity level, unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design. It shall be demonstrated either (1) that independence is achieved by both in the spatial and temporal domains, or (2) that any violation of independence is controlled. The justification for independence shall be documented.

NOTE See Annex F for techniques for achieving one aspect of independence of software.

7.4.2.10 Where the systematic capability of a software element is lower than the safety integrity level of the safety function which the software element supports, the element shall be used in combination with other elements such that the systematic capability of the combination equals the safety integrity level of the safety function.

7.4.2.11 Where a safety function is implemented using a combination of software elements of known systematic capability, the systematic capability requirements of 7.4.3 of IEC 61508-2, shall apply to the combination of elements.

NOTE Distinguish consistently between (1) the *end-to-end safety function* that is supported by one or more elements and (2) the *element safety function* of each of the supporting elements. Where two elements combine to achieve a higher systematic capability in combination, each of the paired elements should be capable of preventing/mitigating the hazardous event, but the paired elements are not required to have identical element safety functions, and it is not required that each of the paired elements is independently capable of providing the whole safety functionality demanded from the combination.

EXAMPLE An electronic engine throttle control where the *end-to-end safety function* is “prevent undemanded acceleration”. The *end-to-end safety function* is implemented by two processors. The *element safety function* of the primary controller is the ideal demand/response behaviour of the throttle. The *element safety function* of the secondary processor is a diverse monitor (see IEC 61508-7 C.3.4) and applies an emergency stop if necessary. The combination of the two processors gives higher confidence that the end-to-end safety function “prevent undemanded acceleration” will be achieved.

7.4.2.12 Where a pre-existing software element is reused to implement all or part of a safety function, the element shall meet both requirements a) and b) below for systematic safety integrity:

a) meet the requirements of one of the following compliance routes:
   - Route 1_S: compliant development. Compliance with the requirements of this standard for the avoidance and control of systematic faults in software;
   - Route 2_S: proven in use. Provide evidence that the element is proven in use. See 7.4.10 of IEC 61508-2;
   - Route 3_S:assessment of non-compliant development. Compliance with 7.4.2.13.

NOTE 1 Route 1_S, 2_S and 3_S are the element compliance routes of 7.4.2.2 c) of IEC 61508-2 with particular reference to software elements. They are reproduced here for convenience only, and to minimize references back to IEC 61508-2.

NOTE 2 See 3.2.8 of IEC 61508-4. The pre-existing software could be a commercially available product, or it could have been developed by some organisation for a previous product or system. Pre-existing software may or may not have been developed in accordance with the requirements of this standard.

NOTE 3 Requirements on pre-existing elements apply to a run-time library or an interpreter.

b) provide a safety manual (see Annex D of IEC 61508-2 and Annex D of this standard) that gives a sufficiently precise and complete description of the element to make possible an assessment of the integrity of a specific safety function that depends wholly or partly on the pre-existing software element.

NOTE 4 The safety manual may be derived from the element supplier's own documentation and records of the element supplier's development process, or may be created or supplemented by additional qualification activities undertaken by the developer of the safety related system or by third parties. In some cases, reverse engineering may be required to create specification or design documentation adequate to meet the requirements of this clause, subject to the prevailing legal conditions (e.g. copyright or intellectual property rights).

NOTE 5 The justification of the element may be developed during safety planning (see Clause 6).

7.4.2.13 To comply with Route 3_S a pre-existing software element shall meet all of the following requirements a) to i):

a) The software safety requirements specification for the element in its new application shall be documented to the same degree of precision as would be required by this standard for any safety related element of the same systematic capability. The software safety requirements specification shall cover the functional and safety behaviour as applicable to the element in its new application and as specified in 7.2. See Table A.1.

b) The justification for use of a software element shall provide evidence that the desirable safety properties specified in the referenced subclauses (i.e. 7.2.2, 7.4.3, 7.4.4, 7.4.5, 7.4.6, 7.4.7, 7.5.2, 7.7.2, 7.8.2, 7.9.2, and Clause 8) have been considered, taking account of the guidance in Annex C.

c) The element's design shall be documented to a degree of precision, sufficient to provide evidence of compliance with the requirement specification and the required systematic capability. See 7.4.3, 7.4.5 and 7.4.6, and Tables A.2 and A.4 of Annex A.

d) The evidence required in 7.4.2.13 a) and 7.4.2.13 b) shall cover the software's integration with the hardware. See 7.5 and Table A.6 of Annex A.

e) There shall be evidence that the element has been subject to verification and validation using a systematic approach with documented testing and review of all parts of the element's design and code. See 7.4.7, 7.4.8, 7.5, 7.7 and 7.9 and Tables A.5 to A.7 and A.9 of Annex A as well as related tables in Annex B.

NOTE 1 Positive operational experience may be used to satisfy black-box and probabilistic testing requirements [see Tables A.7 and B.3].

f) Where the software element provides functions which are are not required in the safety related system, then evidence shall be provided that the unwanted functions will not prevent the E/E/PE system from meeting its safety requirements.

NOTE 2 Ways to meet this requirement include:

- removing the functions from the build;
- disabling the functions;
- appropriate system architecture (e.g. partitioning, wrappers, diversity, checking the credibility of outputs);
- extensive testing.

g) There shall be evidence that all credible failure mechanisms of the software element have been identified and that appropriate mitigation measures have been implemented.

NOTE 3 Appropriate mitigation measures include:

- appropriate system architecture (e.g. partitioning, wrappers, diversity, credibility of checking of outputs);
- exception handling.

h) The planning for use of the element shall identify the configuration of the software element, the software and hardware run-time environment and if necessary the configuration of the compilation / linking system.

i) The justification for use of the element shall be valid for only those applications which respect the assumptions made in the compliant item safety manual for the element (see Annex D of IEC 61508-2 and Annex D).

7.4.2.14 This Subclause 7.4.2 shall, in so far as it is appropriate, apply to data and data generation languages.

NOTE See Annex G for guidance on data-driven systems.

a) Where a PE system consists of pre-existing functionality that is configured by data to meet specific application requirements, the design of the application software shall be commensurate with the degree of application configurability, pre-delivered existing functionality and complexity of the PE safety-related system.

b) Where the safety-related functionality of a PE system is determined significantly or predominantly by configuration data, appropriate techniques and measures shall be used to prevent the introduction of faults during the design, production, loading and modification of the configuration data and to ensure that the configuration data correctly states the application logic.

c) The specification of data structures shall be:

   1) consistent with the functional requirements of the system, including the application data;

   2) complete;

   3) self consistent;

   4) such that the data structures are protected against alteration or corruption.

d) Where a PE System consists of pre-existing functionality that is configured by data to meet specific application requirements, the configuration process itself shall be documented appropriately.

7.4.3 Requirements for software architecture design

NOTE 1 The software architecture defines the major elements and subsystems of the software, how they are interconnected, and how the required attributes, particularly safety integrity, will be achieved. It also defines the overall behaviour of the software, and how software elements interface and interact. Examples of major software elements include operating systems, databases, EUC input/output subsystems, communication subsystems, application program(s), programming and diagnostic tools, etc.

NOTE 2 In certain industrial sectors the software architecture would be called a function description or functional design specification (although these documents could also include the hardware).

NOTE 3 In some contexts of user application programming, particularly in PLCs (see Annex E of IEC 61508-6), the software architecture is provided by the supplier as a standard feature of the product. The supplier would, under this standard, be required to assure the user of the compliance of his products to the requirements of 7.4. The user tailors the PLC to the application by using the standard programming facilities, for example ladder logic. The requirements of 7.4.3 to 7.4.8 still apply. The requirement to define and document the software architecture can be seen as information that the user would use to select the PLC (or equivalent) for the application.

NOTE 4 From a safety viewpoint, the software architecture phase is where the basic safety strategy is developed for the software.

NOTE 5 Although the IEC 61508 series sets numerical target failure measures for safety functions carried out by E/E/PE safety-related systems, systematic safety integrity is usually unquantified (see 3.5.6 of IEC 61508-4), and software safety integrity (see 3.5.5 of IEC 61508-4) is defined as a systematic capability on a confidence scale of 1-4 (see 3.5.9 of IEC 61508-4). This standard recognizes that a software failure can be safe or unsafe depending on the specific use of the software The system/software architecture needs to be such that unsafe failures of an element are limited by some architectural constraint, and that development methods should take account of these constraints. This standard applies development and validation techniques with rigour that is qualitatively consistent with the required systematic capability.

NOTE 6 For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the software architecture design should be considered:

– completeness with respect to software safety requirements specification;

– correctness with respect to software safety requirements specification;

– freedom from intrinsic design faults;

– simplicity and understandability;
– predictability of behaviour;
– verifiable and testable design;
– fault tolerance;
– defence against common cause failure from external events.

7.4.3.1 Depending on the nature of the software development, responsibility for conformance with 7.4.4 can rest with multiple parties. The division of responsibility shall be documented during safety planning (see Clause 6 of IEC 61508-1).

7.4.3.2 The software architecture design shall be established by the software supplier and/or developer, and shall be detailed. The software architecture design shall:

a) select and justify (see 7.1.2.7) an integrated set of techniques and measures necessary during the software safety lifecycle phases to satisfy the software safety requirements specification at the required safety integrity level. These techniques and measures include software design strategies for both fault tolerance (consistent with the hardware) and fault avoidance, including (where appropriate) redundancy and diversity;
b) be based on a partitioning into elements/subsystems, for each of which the following information shall be provided:
   1) whether the elements/subsystems have been previously verified, and if yes, their verification conditions;
   2) whether each subsystem/element is safety-related or not;
   3) software systematic capability of the subsystem/element.
c) determine all software/hardware interactions and evaluate and detail their significance;

NOTE Were the software/hardware interaction is already determined by the system architecture, it is sufficient to refer to the system architecture.

d) use a notation to represent the architecture which is unambiguously defined or restricted to unambiguously defined features;
e) select the design features to be used for maintaining the safety integrity of all data. Such data may include plant input-output data, communications data, operator interface data, maintenance data and internal database data;
f) specify appropriate software architecture integration tests to ensure that the software architecture satisfies the software safety requirements specification at the required safety integrity level.

7.4.3.3 Any changes required to the E/E/PE System Safety Requirements Specification (see 7.2.2) after applying 7.4.3.2 shall be agreed with the E/E/PE developer and documented.

NOTE There will inevitably be iteration between the hardware and software architecture (see Figure 5) and there is therefore a need to discuss with the hardware developer such issues as the test specification for the integration of the programmable electronics hardware and the software (see 7.5).

7.4.4 Requirements for support tools, including programming languages

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of support tools should be considered:

– the degree to which the tool supports the production of software with the required software properties;
– the clarity of the operation and functionality of the tool;
– the correctness and repeatability of the output.

7.4.4.1 A software on-line support tool shall be considered to be a software element of the safety-related system

NOTE See 3.2.10 and 3.2.11 of IEC 61508-4 for examples of on-line and off-line tools.

7.4.4.2 Software off-line support tools shall be selected as a coherent part of the software development activities.

NOTE 1 See 7.1.2 for software development lifecycle requirements.

NOTE 2 Appropriate off-line tools to support the development of software should be 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 level: 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, and simulators;

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. Such data includes function parameters, instrument ranges, alarm and trip levels, output states to be adopted at failure, geographical layout.

NOTE 3 Off-line support tools should be selected to be integrated. In this context, tools are integrated if they work co-operatively such that the outputs from one tool have suitable content and format for automatic input to a subsequent tool, thus minimising the possibility of introducing human error in the reworking of intermediate results.

NOTE 4 Off-line support tools should be selected to be compatible with the needs of the application, of the safety related system, and of the integrated toolset.

NOTE 5 The availability of suitable tools to supply the services that are necessary over the whole lifetime of the E/E/PE safety-related system (e.g. tools to support specification, design, implementation, documentation, modification) should be considered.

NOTE 6 Consideration should be given to the competence of the users of the selected tools. See Clause 6 of IEC 61508-1 for competence requirements.

7.4.4.3 The selection of the off-line support tools shall be justified.

7.4.4.4 All off-line support tools in classes T2 and T3 shall have a specification or product documentation which clearly defines the behaviour of the tool and any instructions or constraints on its use. See 7.1.2 for software development lifecycle requirements, and 3.2.11 of IEC 61508-4 for categories of software off-line support tool.

NOTE This “specification or product documentation” is not a safety manual for compliant items (see Annex D of 61508-2 and also of this standard) for the tool itself. The safety manual for compliant item relates to a pre-existing element that is incorporated into the executable safety related system. Where a pre-existing element has been generated by a T3 tool and then incorporated into the executable safety related system, then any relevant information (e.g. the documentation for an optimising compiler may indicate that the evaluation order of function parameters is not guaranteed) from the tool’s “specification or product documentation” should be included in the compliant item safety manual that makes possible an assessment of the integrity of a specific safety function that depends wholly or partly on the incorporated element.”

7.4.4.5 An assessment shall be carried out for offline support tools in classes T2 and T3 to determine the level of reliance placed on the tools, and the potential failure mechanisms of the tools that may affect the executable software. Where such failure mechanisms are identified, appropriate mitigation measures shall be taken.

NOTE 1 Software HAZOP is one technique to analyse the consequences of potential software tool failures.

NOTE 2 Examples of mitigation measures include: avoiding known bugs, restricted use of the tool functionality, checking the tool output, use of diverse tools for the same purpose.

7.4.4.6 For each tool in class T3, evidence shall be available that the tool conforms to its specification or documentation. Evidence may be based on a suitable combination of history of successful use in similar environments and for similar applications (within the organisation or other organisations), and of tool validation as specified in 7.4.4.7.

NOTE 1 A version history may provide assurance of maturity of the tool, and a record of the errors / ambiguities that should be taken into account when the tool is used in the new development environment.

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

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

a) a chronological record of the validation activities;

b) the version of the tool product 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.

7.4.4.8 Where the conformance evidence of 7.4.4.6 is unavailable, there shall be effective measures to control failures of the executable safety related system that result from faults that are attributable to the tool.

NOTE An example of a measure would be the generation of diverse redundant code which allows the detection and control of failures of the executable safety related system as a result of faults that have been introduced into the executable safety related system by a translator.

7.4.4.9 The compatibility of the tools of an integrated toolset shall be verified.

Note: tools are integrated if they work co-operatively such that 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. See IEC 61508-7 B.3.5.

7.4.4.10 To the extent required by the safety integrity level, the software or design representation (including a programming language) selected shall:

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

b) use only defined language features;

c) match the characteristics of the application;

d) contain features that facilitate the detection of design or programming mistakes;

e) support features that match the design method.

NOTE 1 A programming language is a class of software or design representations. 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.

NOTE 2 The assessment 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 (the “specification or product manual”, see 7.4.4.4) regarding the intended and appropriate use of the tool should be available to the user of the tool. The assessment of the tool for a specific project may then be reduced to checking general suitability of the tool for the project and compliance with the “specification or product manual” (i.e. proper use of the tool). Proper use might include additional verification activities within the specific project.

NOTE 3 A validation suite (i.e. a set of test programs whose correct translation is known in advance) may be used to evaluate the fitness for purpose of a translator according to defined criteria, which should 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 should be used.

7.4.4.11 Where 7.4.4.10 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.

7.4.4.12 Programming languages for the development of all safety-related software shall be used according to a suitable programming language coding standard.

NOTE See IEC 61508-7 for guidance on coding standard aspects that relate to software safety.

7.4.4.13 A programming language coding standard shall specify good programming practice, proscribe unsafe language features (for example, undefined language features, unstructured designs, etc.), promote code understandability, facilitate verification and testing, and specify procedures for source code documentation. Where practicable, the following information shall be contained in the source code:

a) legal entity (for example company, author(s), etc.);

b) description;

c) inputs and outputs;

d) configuration management history.

7.4.4.14 Where automatic code generation or similar automatic translation takes place, the suitability of the automatic translator for safety-related system development shall be assessed at the point in the development lifecycle where development support tools are selected.

7.4.4.15 Where off-line support tools of classes T2 and T3 generate items in the configuration baseline, configuration management shall ensure that information on the tools is recorded in the configuration baseline. This includes in particular:

a) the identification of the tool and its version;

b) the identification of the configuration baseline items for which the tool version has been used;

c) the way the tool was used (including the tool parameters, options and scripts selected) for each configuration baseline item.

NOTE The objective of this clause is to allow the baseline to be reconstructed.

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

7.4.4.17 Configuration management shall ensure that only tools compatible with each other and with the safety-related system are used.

NOTE The safety-related system hardware may also impose compatibility constraints on software tools e.g. a processor emulator needs to be an accurate model of the real processor electronics.

7.4.4.18 Each new version of off-line support tool shall be qualified. This qualification 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; and

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 may be based on (1) a clear identification of the changes made, (2) an analysis of the verification and validation actions performed on the new version, and (3) any existing operational experience from other users that is relevant to the new version.

7.4.4.19 Depending on the nature of the software development, responsibility for conformance with 7.4.4 can rest with multiple parties. The division of responsibility shall be documented during safety planning (see Clause 6 of IEC 61508-1).

7.4.5 Requirements for detailed design and development – software system design

NOTE 1 Detailed design is defined here to mean software system design: the partitioning of the major elements in the architecture into a system of software modules; individual software module design; and coding. In small applications, software system design and architectural design may be combined.

NOTE 2 The nature of detailed design and development will vary with the nature of the software development activities and the software architecture (see 7.4.3). In some contexts of application programming, for example ladder logic and function blocks, detailed design can be considered as configuring rather than programming.

However it is still good practice to design the software in a structured way, including organising the software into a modular structure that separates out (as far as possible) safety-related parts; including range checking and other features that provide protection against data input mistakes; using previously verified software modules; and providing a design that facilitates future software modifications.

NOTE 3 For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the design and development should be considered:

– completeness with respect to software safety requirements specification;
– correctness with respect to software safety requirements specification;
– freedom from intrinsic design faults;
– simplicity and understandability
– predictability of behaviour;
– verifiable and testable design;
– fault tolerance / fault detection;
– freedom from common cause failure.

7.4.5.1 Depending on the nature of the software development, responsibility for conformance with 7.4.5 can rest with multiple parties. The division of responsibility shall be documented during safety planning (see Clause 6 of IEC 61508-1).

7.4.5.2 The following information shall be available prior to the start of detailed design: the specification of requirements for the E/E/PE safety related system; the software architecture design; the validation plan for software aspects of system safety.

7.4.5.3 The software shall be produced to achieve modularity, testability, and the capability for safe modification.

7.4.5.4 For each major element/subsystem in the software architecture design, further refinement of the design shall be based on a partitioning into software modules (i.e. the specification of the software system design). The design of each software module and the verification to be applied to each software module shall be specified.

NOTE 1 For pre-existing software elements, see 7.4.2.

NOTE 2 Verification includes testing and analysis.

7.4.5.5 Appropriate software system integration tests shall be specified to ensure that the software system satisfies the software safety requirements specification at the required safety integrity level.

7.4.6 Requirements for code implementation

NOTE To the extent required by the safety integrity level, the source code shall possess the following properties (see Annexes A and B for specific techniques, and see Annex C for guidance on interpretation of properties) of code should be considered:

- be readable, understandable and testable;
- satisfy the specified requirements for software module design (see 7.4.5);
- satisfy the specified requirements of the coding standards (see 7.4.4);
- satisfy all relevant requirements specified during safety planning (see Clause 6).

7.4.6.1 Each module of software code shall be reviewed. Where the code is produced by an automatic tool, the requirements of 7.4.4 shall be met. Where the source code consists of reused pre-existing software, the requirements of 7.4.2 shall be met.

NOTE Code review is a verification activity (see 7.9). Code review can be carried out by means of an inspection of the code: (1) by an individual; (2) by a software walk-though (see IEC 61508-7 C.5.15); or (3) by a formal inspection (see IEC 61508-7 C.5.14), in increasing order of rigour.

7.4.7 Requirements for software module testing

NOTE 1 Testing that the software module correctly satisfies its test specification is a verification activity (see 7.9). It is the combination of code review and software module testing that provides assurance that a software module satisfies its associated specification, i.e. it is verified.

NOTE 2 For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the software module testing should be considered:

– completeness of testing with respect to the software design specification;

– correctness of testing with respect to the software design specification (successful completion);

– repeatability;

– precisely defined testing configuration.

7.4.7.1 Each software module shall be verified as required by the software module test specification that was developed during software system design (see 7.4.5).

NOTE Verification includes testing and analysis.

7.4.7.2 This verification shall show whether or not each software module performs its intended function and does not perform unintended functions.

NOTE 1 This does not imply testing of all input combinations, nor of all output combinations. Testing all equivalence classes or structure based testing may be sufficient. Boundary value analysis or control flow analysis may reduce the test cases to an acceptable number. Analysable programs make the requirements easier to fulfil. See Annex C of IEC 61508-7 for these techniques.

NOTE 2 Where the development uses formal methods, formal proofs or assertions, such tests may be reduced in scope. See Annex C of IEC 61508-7 for these techniques.

NOTE 3 Although systematic safety integrity is usually unquantified (see 3.5.6 of IEC 61508-4), quantified statistical evidence (e.g. statistical testing, reliability growth) is acceptable if all the relevant conditions for statistically valid evidence are satisfied e.g. see Annex D of IEC 61508-7.

NOTE 4 If the module is simple enough to make practicable an exhaustive test, then this can be the most efficient way to demonstrate conformance.

7.4.7.3 The results of the software module testing shall be documented.

7.4.7.4 The procedures for corrective action on not passing the test shall be specified.

7.4.8 Requirements for software integration testing

NOTE Testing that the software is correctly integrated is a verification activity (see 7.9).

7.4.8.1 Software integration tests shall be specified during the design and development phase (see 7.4.5).

7.4.8.2 The software system integration test specification shall state the following:

a) the division of the software into manageable integration sets;

b) test cases and test data;

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) procedures for corrective action on failure of test.

7.4.8.3 The software shall be tested in accordance with the software integration tests specified in the software system integration test specification. These tests shall show that all software modules and software elements/subsystems interact correctly to perform their intended function and do not perform unintended functions.

NOTE 1 This does not imply testing of all input combinations, nor of all output combinations. Testing all equivalence classes or structure based testing may be sufficient. Boundary value analysis or control flow analysis may reduce the test cases to an acceptable number. Analysable programs make the requirements easier to fulfil. See Annex C of IEC 61508-7 for these techniques.

NOTE 2 Where the development uses formal methods, formal proofs or assertions, such tests may be reduced in scope. See Annex C of IEC 61508-7 for these techniques.

NOTE 3 Although systematic safety integrity is usually unquantified (see 3.5.6 of IEC 61508-4), quantified statistical evidence (e.g. statistical testing, reliability growth) is acceptable if all the relevant conditions for statistically valid evidence are satisfied e.g. see Annex D of IEC 61508-7.

7.4.8.4 The results of software integration testing shall be documented, stating the test results, and whether the objectives and the test criteria have been met. If there is a failed integration test, the reasons for the failure shall be documented.

7.4.8.5 During software integration, any modification to the software shall be subject to an impact analysis which shall determine all software modules impacted, and the necessary re-verification and re-design activities.

7.5 Programmable electronics integration (hardware and software)

NOTE This phase is box 10.4 of Figure 4.

7.5.1 Objectives

7.5.1.1 The first objective of the requirements of this subclause is to integrate the software onto the target programmable electronic hardware.

7.5.1.2 The second objective of the requirements of this subclause is to combine the software and hardware in the safety-related programmable electronics to ensure their compatibility and to meet the requirements of the intended safety integrity level.

NOTE 1 Testing that the software is correctly integrated with the programmable electronic hardware is a verification activity (see 7.9).

NOTE 2 Depending on the nature of the application, these activities may be combined with 7.4.8.

7.5.2 Requirements

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the integration should be considered:

– completeness of integration with respect to the design specifications;

– correctness of integration with respect to the design specifications (successful completion);

– repeatability;

– precisely defined integration configuration.

7.5.2.1 Integration tests shall be specified during the design and development phase (see 7.4.3) to ensure the compatibility of the hardware and software in the safety-related programmable electronics.

NOTE Close co-operation with the developer of the E/E/PE system may be required in order to develop the integration tests.

7.5.2.2 The software/PE integration test specification (hardware and software) shall state the following:

a) the split of the system into integration levels;

b) test cases and test data;

c) types of tests to be performed;

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

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

7.5.2.3 The software/PE integration test specification (hardware and software) shall distinguish between those activities which can be carried out by the developer on his premises and those that require access to the user's site.

7.5.2.4 The software/PE integration test specification (hardware and software) shall distinguish between the following activities:

a) merging of the software system on to the target programmable electronic hardware;

b) E/E/PE integration, i.e. adding interfaces such as sensors and actuators;

c) applying the E/E/PE safety-related system to the EUC.

NOTE Items b) and c) are covered by IEC 61508-1 and IEC 61508-2 and are included here to put item a) in context and for completeness. They are not normally the responsibility of the software developers.

7.5.2.5 The software shall be integrated with the safety-related programmable electronic hardware in accordance with the software/PE integration test specification (hardware and software).

7.5.2.6 During the integration testing of the safety-related programmable electronics (hardware and software), any change to the integrated system shall be subject to an impact analysis. The impact analysis shall determine all software modules impacted, and the necessary re-verification activities.

7.5.2.7 Test cases and their expected results shall be documented for subsequent analysis.

7.5.2.8 The integration testing of the safety-related programmable electronics (hardware and software) shall be documented, stating the test results, and whether the objectives and the test criteria have been met. If there is a failure, the reasons for the failure shall be documented. Any resulting modification or change to the software shall be subject to an impact analysis which shall determine all software elements/modules impacted, and the necessary re-verification and re-design activities.

7.6 Software operation and modification procedures

NOTE This phase is box 10.5 of Figure 4.

7.6.1 Objective

The objective of the requirements of this subclause is to provide information and procedures concerning software necessary to ensure that the functional safety of the E/E/PE safety-related system is maintained during operation and modification.

7.6.2 Requirements

The requirements are given in 7.6 of IEC 61508-2 and in 7.8 of this standard.

NOTE In this standard software (unlike hardware) is not capable of being maintained: it is always modified.

7.7 Software aspects of system safety validation

NOTE 1 This phase is box 10.6 of Figure 4.

NOTE 2 Software usually cannot be validated separately from its underlying hardware and system environment.

7.7.1 Objective

The objective of the requirements of this subclause is to ensure that the integrated system complies with the software safety requirements specification at the required safety integrity level.

7.7.2 Requirements

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of safety validation should be considered:

– completeness of validation with respect to the software design specification;

– correctness of validation with respect to the software design specification (successful completion);

– repeatability;

– precisely defined validation configuration.

7.7.2.1 If the compliance with the requirements for safety-related software has already been established in the safety validation planning for the E/E/PE safety-related system (see 7.7 of IEC 61508-2), then the validation need not be repeated.

7.7.2.2 The validation activities shall be carried out as specified the in validation plan for software aspects of system safety.

7.7.2.3 Depending on the nature of the software development, responsibility for conformance with 7.7 can rest with multiple parties. The division of responsibility shall be documented during safety planning (see Clause 6 of IEC 61508-1).

7.7.2.4 The results of validating the software aspects of system safety shall be documented.

7.7.2.5 For each safety function, software safety validation shall document the following results:

a) a chronological record of the validation activities that will permit the sequence of activities to be retraced;

   NOTE When recording test results, it is important to be able to retrace the sequence of activities. The emphasis of this requirement is on retracing a sequence of activities, and not on producing a timed/dated list of documents.

b) the version of the validation plan for software aspects of system safety (see 7.3) being used;

c) the safety function being validated (by test or analysis), together with reference to the validation plan for software aspects of system safety;

d) tools and equipment used together with calibration data;

e) the results of the validation activity;

f) discrepancies between expected and actual results.

7.7.2.6 When discrepancies occur between expected and actual results, the analysis made and the decisions taken on whether to continue the validation, or to issue a change request and return to an earlier part of the development lifecycle, shall be documented as part of the results of validating the software aspects of system safety.

NOTE The requirements of 7.7.2.2 to 7.7.2.6 are based on the general requirements given in 7.14 of IEC 61508-1.

7.7.2.7 The validation of safety-related software aspects of system safety shall meet the following requirements:

a) testing shall be the main validation method for software; analysis, animation and modelling may be used to supplement the validation activities;

b) the software shall be exercised by simulation of:

   1) input signals present during normal operation;

   2) anticipated occurrences;

   3) undesired conditions requiring system action;

c) the supplier and/or developer (or the multiple parties responsible for compliance) shall make available the documented results of the validation of software aspects of system safety and all pertinent documentation to the system developer to enable his product to meet the requirements of IEC 61508-1 and IEC 61508-2.

7.7.2.8 Software tools shall meet the requirements of 7.4.4.

7.7.2.9 The results of the validation of safety-related software aspects of system safety shall meet the following requirements:

a) the tests shall show that all of the specified requirements for safety-related software (see 7.2) are correctly met and the software does not perform unintended functions;

b) test cases and their results shall be documented for subsequent analysis and independent assessment (see Clause 8 of IEC 61508-1) as required by the safety integrity level;

c) the documented results of validating the software aspects of system safety shall state either (1) that the software has passed the validation or (2) the reasons for not passing the validation.

7.8 Software modification

NOTE This phase is Box 10.5 of Figure 4.

7.8.1 Objective

The objective of the requirements of this subclause is to guide corrections, enhancements or adaptations to the validated software, ensuring that the required software systematic capability is sustained.

7.8.2 Requirements

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the software modification should be considered:

– completeness of modification with respect to its requirements;

– correctness of modification with respect to its requirements;

– freedom from introduction of intrinsic design faults;

– avoidance of unwanted behaviour;

– verifiable and testable design;

– regression testing and verification coverage.

7.8.2.1 Prior to carrying out any software modification, software modification procedures shall be made available (see 7.16 of IEC 61508-1).

NOTE 1 Subclauses 7.8.2.1 to 7.8.2.9 apply primarily to changes occurring during the operational phase of the software. They may also apply during the programmable electronics integration and overall installation and commissioning phases (see 7.13 of IEC 61508-1).

NOTE 2 An example of a modification procedure model is shown in Figure 9 of IEC 61508-1.

7.8.2.2 A modification shall be initiated only on the issue of an authorized software modification request under the procedures specified during safety planning (see Clause 6) which details the following:

a) the hazards which may be affected;

b) the proposed modification;

c) the reasons for modification.

NOTE A request for modification could arise from, for example

- functional safety is found to be less than required by the safety requirements specification;
- systematic fault experience;

- new or amended safety legislation;
- modifications to the EUC or its use;
- modification to the overall safety requirements;
- analysis of operations and maintenance performance, indicating that the performance is below target;
- routine functional safety audits.

7.8.2.3 An analysis shall be carried out on the impact of the proposed software modification on the functional safety of the E/E/PE safety-related system:

a) to determine whether or not a hazard and risk analysis is required;

b) to determine which software safety lifecycle phases will need to be repeated.

7.8.2.4 The impact analysis results obtained in 7.8.2.3 shall be documented.

7.8.2.5 All modifications which have an impact on the functional safety of the E/E/PE safety-related system shall initiate a return to an appropriate phase of the software safety 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. Safety planning (see Clause 6) shall detail all subsequent activities.

NOTE It may be necessary to implement a full hazard and risk analysis, which may generate a need for different safety integrity levels than currently specified for the safety functions implemented by the E/E/PE safety-related systems.

7.8.2.6 The safety planning for the modification of safety-related software shall meet the requirements given in Clause 6 of IEC 61508-1. In particular:

a) identification of staff and specification of their required competency;

b) detailed specification for the modification;

c) verification planning;

d) scope of revalidation and testing of the modification to the extent required by the safety integrity level.

NOTE Depending on the nature of the application, involvement of domain experts may be important.

7.8.2.7 Modification shall be carried out as planned.

7.8.2.8 Details of all modifications shall be documented, including references to:

a) the modification/retrofit request;

b) the results of the impact analysis which assesses the impact of the proposed software modification on the functional safety, and the decisions taken with associated justifications;

c) software configuration management history;

d) deviation from normal operations and conditions;

e) all documented information affected by the modification activity.

7.8.2.9 Information on the details of all modifications shall be documented. The documentation shall include the re-verification and re-validation of data and results.

7.8.2.10 The assessment of the required modification or retrofit activity shall be dependent on the results of the impact analysis and the software systematic capability.

7.9 Software verification

7.9.1 Objective

The objective of the requirements of this subclause is, to the extent required by the safety integrity level, to test and evaluate the outputs from a given software safety lifecycle phase to ensure correctness and consistency with respect to the inputs to that phase.

NOTE 1 This subclause considers the generic aspects of verification which are common to several safety lifecycle phases. This subclause does not place additional requirements for the testing element of verification in 7.4.7 (software module testing), 7.4.8 (software integration) and 7.5 (programmable electronics integration) because these are verification activities in themselves. Nor does this subclause require verification in addition to software validation (see 7.7), because in this standard software validation is the demonstration of conformance to the safety requirements specification. Checking whether the safety requirements specification is itself correct is carried out by domain experts.

NOTE 2 Depending on the software architecture, responsibility for the verification activity may be split between all organisations involved in the development and modification of the software.

7.9.2 Requirements

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the data verification should be considered:

– completeness of verification with respect to the previous phase;

– correctness of verification with respect to the previous phase (successful completion);

– repeatability;

– precisely defined verification configuration.

7.9.2.1 The verification of software shall be planned (see 7.3) concurrently with the development, for each phase of the software safety lifecycle, and shall be documented.

7.9.2.2 The software verification planning shall refer to the criteria, techniques and tools to be used in the verification activities, and shall address:

a) the evaluation of the safety integrity requirements;

b) the selection and documentation of verification strategies, activities and techniques;

c) the selection and utilisation of verification tools (test harness, special test software, input/output simulators etc.);

d) the evaluation of verification results;

e) the corrective actions to be taken.

7.9.2.3 The software verification shall be performed as planned.

NOTE Selection of techniques, measures for verification and the degree of independence of the verification activities will depend upon a number of factors and may be specified in application sector standards. The factors could include, for example:

- size of project;
- degree of complexity;
- degree of novelty of design;
- degree of novelty of technology.

7.9.2.4 Evidence shall be documented to show that the phase being verified has, in all respects, been satisfactorily completed.

7.9.2.5 After each verification, the verification documentation shall include:

a) identification of items to be verified;

b) identification of the information against which the verification has been done;

NOTE 1 Information against which the verification has been performed includes but is not limited to input from the previous lifecycle phase, design standards, coding standards and tools used.

c) non-conformances.

NOTE 2 Examples of non-conformances include software modules, data structures, and algorithms poorly adapted to the problem.

7.9.2.6 All essential information from phase N of the software safety lifecycle needed for the correct execution of the next phase N+1 shall be available and shall be verified. Outputs from phase N include:

a) adequacy of the specification, design, or code in phase N for:
   1) functionality;
   2) safety integrity, performance and other requirements of safety planning (see Clause 6);
   3) readability by the development team;
   4) testability for further verification;
   5) safe modification to permit further evolution;
b) adequacy of the validation planning and/or tests specified for phase N for specifying and describing the design of phase N;
c) check for incompatibilities between:
   1) the tests specified in phase N, and the tests specified in the previous phase N–1;
   2) the outputs within phase N.

7.9.2.7 Subject to the choice of software development lifecycle (see 7.1), the following verification activities shall be performed:

a) verification of software safety requirements;
b) verification of software architecture;
c) verification of software system design;
d) verification of software module design;
e) verification of code;
f) verification of data;
g) verification of timing performance;
h) software module testing (see 7.4.7);
i) software integration testing (see 7.4.8);
j) programmable electronics integration testing (see 7.5);
k) software aspects of system safety validation (see 7.7).

NOTE For requirements a) to g) see below.

7.9.2.8 Verification of software safety requirements: after the software safety requirements specification has been completed, and before the next phase of software design and development begins, verification shall:

a) consider whether the software safety requirements specification adequately fulfils the E/E/PE system safety requirements specification (see 7.10 of IEC 61508-1 and 7.2 of IEC 61508-2) for functionality, safety integrity, performance, and any other requirements of safety planning;
b) consider whether the validation plan for software aspects of system safety adequately fulfils the software safety requirements specification;
c) check for incompatibilities between:
   1) the software safety requirements specification, and the E/E/PE system safety requirements specification (see 7.10 of IEC 61508-1 and 7.2 of IEC 61508-2);
   2) the software safety requirements specification, and the validation plan for software aspects of system safety.

7.9.2.9 Verification of software architecture: after the software architecture design has been completed, verification shall:

a) consider whether the software architecture design adequately fulfils the software safety requirements specification;

b) consider whether the integration tests specified in the software architecture design are adequate;

c) consider whether the attributes of each major element/subsystem are adequate with reference to:

   1) feasibility of the safety performance required;

   2) testability for further verification;

   3) readability by the development and verification team;

   4) safe modification to permit further evolution.

d) check for incompatibilities between the following:

   1) the software architecture design, and the software safety requirements specification;

   2) the software architecture design and its integration tests;

   3) the software architecture design integration tests and the validation plan for software aspects of system safety.

7.9.2.10 Verification of software system design: after the software system design has been completed, verification shall:

a) consider whether the software system design (see 7.4.5) adequately fulfils the software architecture design;

b) consider whether the specified tests of the software system integration (see 7.4.5) adequately fulfil the software system design (see 7.4.5);

c) consider whether the attributes of each major element of the software system design specification (see 7.4.5) are adequate with reference to:

   1) feasibility of the safety performance required;

   2) testability for further verification;

   3) readability by the development and verification team;

   4) safe modification to permit further evolution.

   NOTE The software system integration tests may be specified as part of the software architecture integration tests.

d) check for incompatibilities between:

   1) the software system design specification (see 7.4.5), and the software architecture design;

   2) the software system design specification (see 7.4.5), and the software system integration test specification (see.4.5);

   3) the tests required by the software system integration test specification (see 7.4.5) and the software architecture integration test specification (see 7.4.3).

7.9.2.11 Verification of software module design: after the design of each software module has been completed, verification shall:

a) consider whether the software module design specification (see 7.4.5) adequately fulfils the software system design specification (see 7.4.5);

b) consider whether the software module test specification (see 7.4.5) is adequate for the software module design specification (see 7.4.5);

c) consider whether the attributes of each software module are adequate with reference to:

   1) feasibility of the safety performance required (see software safety requirements specification);

2) testability for further verification;
3) readability by the development and verification team;
4) safe modification to permit further evolution.

d) check for incompatibilities between:
1) the software module design specification (see 7.4.5), and the software system design specification (see 7.4.5);
2) (for each software module) the software module design specification (see 7.4.5), and the software module test specification (see 7.4.5);
3) the software module test specification (see 7.4.5), and the software system integration test specification (see 7.4.5).

7.9.2.12 Verification of code: the source code shall be verified by static methods to ensure conformance to the software module design specification (see 7.4.5), the required coding standards (see 7.4.4), and the validation plan for software aspects of system safety.

NOTE In the early phases of the software safety lifecycle, verification is static (for example inspection, review, formal proof, etc). Code verification includes such techniques as software inspections and walk-throughs. It is the combination of the results of code verification and software module testing that provides assurance that each software module satisfies its associated specification. From then onwards testing becomes the primary means of verification.

7.9.2.13 Verification of data.

a) The data structures shall be verified.
b) The application data shall be verified for:
1) consistency with the data structures;
2) completeness against the application requirements;
3) compatibility with the underlying system software (for example, sequence of execution, run-time, etc.); and
4) correctness of the data values.
c) All operational parameters shall be verified against the application requirements.
d) All plant interfaces and associated software (i.e. sensors and actuators and off-line interfaces: see 7.2.2.12) shall be verified for:
1) detection of anticipated interface failures;
2) tolerance to anticipated interface failures.
e) All communication interfaces and associated software shall be verified for an adequate level of:
1) failure detection;
2) protection against corruption;
3) data validation.

7.9.2.14 Verification of timing performance: predictability of behaviour in the time domain shall be verified.

NOTE Timing behaviour may include: performance, resources, response time, worst case execution time, thrashing, dead-lock free, run-time system.

8 Functional safety assessment

NOTE For the selection of appropriate techniques and measures (see Annexes A and B) to implement the requirements of this clause, the following properties (see Annex C for guidance on interpretation of properties, and Annex F of IEC 61508-7 for informal definitions) of the functional safety assessment should be considered:

– completeness of functional safety assessment with respect to this standard;
– correctness of functional safety assessment with respect to the design specifications (successful completion);

– traceable closure of all identified issues;
– the ability to modify the functional safety assessment after change without the need for extensive re-work of the assessment;
– repeatability;
– timeliness;
– precisely defined configuration.

8.1 The objective and requirements of Clause 8 of IEC 61508-1 apply to the assessment of safety-related software.

8.2 Unless otherwise stated in application sector international standards, the minimum level of independence of those carrying out the functional safety assessment shall be as specified in Clause 8 of IEC 61508-1.

8.3 An assessment of functional safety may make use of the results of the activities of Table A.10.

NOTE Selecting techniques from Annexes A and B does not guarantee by itself that the required safety integrity will be achieved (see 7.1.2.7). The assessor should also consider:

- the consistency and the complementary nature of the chosen methods, languages and tools for the whole development cycle;
- whether the developers use methods, languages and tools they fully understand;
- whether the methods, languages and tools are well-adapted to the specific problems encountered during development.

Annex A
(normative)

Guide to the selection of techniques and measures

Some of the subclauses of this standard have an associated table, for example 7.2 (software safety requirements specification) is associated with Table A.1. More detailed tables in Annex B expand upon some of the entries in the tables of Annex A. For example, Table B.2 expands on the topic of dynamic analysis and testing in Table A.5.

See IEC 61508-7 for an overview of the specific techniques and measures referenced in Annexes A and B.

With each technique or measure in the tables there is a recommendation for safety integrity levels 1 to 4. These recommendations are as follows.

HR	the technique or measure is highly recommended for this safety integrity level. If this technique or measure is not used then the rationale behind not using it should be detailed with reference to Annex C during the safety planning and agreed with the assessor.
R	the technique or measure is recommended for this safety integrity level as a lower recommendation to a HR recommendation.
---	the technique or measure has no recommendation for or against being used.
NR	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 should be detailed with reference to Annex C during the safety planning and agreed with the assessor.

Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

Other measures and techniques may be applied providing that the requirements and objectives have been met. See Annex C for guidance on selecting techniques.

The ranking of the techniques and measures is linked to the concept of *effectiveness* used in IEC 61508-2. For all other factors being equal, techniques which are ranked HR will be more effective in either preventing the introduction of systematic faults during software development, or (for the case of the software architecture) more effective in controlling residual faults in the software revealed during execution than techniques ranked as R.

Given the large number of factors that affect software systematic capability it is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application. Guidance on a rationale for selecting specific techniques to achieve software systematic capability is given in Annex C.

For a particular application, the appropriate combination of techniques or measures are to be stated during safety planning, with appropriate techniques or measures being selected unless the note attached to the table makes other requirements.

Initial guidance in the form of two worked examples on the interpretation of the tables is given in IEC 61508-6.

Table A.1 – Software safety requirements specification

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

NOTE 1 The software safety requirements specification will always require a description of the problem in natural language and any necessary mathematical notation that reflects the application.

NOTE 2 The table reflects additional requirements for specifying the software safety requirements clearly and precisely.

NOTE 3 See Table C.1.

NOTE 4 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table A.2 – Software design and development – software architecture design

(see 7.4.3)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
	Architecture and design feature					
1	Fault detection	C.3.1	---	R	HR	HR
2	Error detecting codes	C.3.2	R	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	R	HR
3b	Diverse monitor techniques (with independence between the monitor and the monitored function in the same computer)	C.3.4	---	R	R	----
3c	Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	C.3.4	---	R	R	HR
3d	Diverse redundancy, implementing the same software safety requirements specification	C.3.5	---	---	---	R
3e	Functionally diverse redundancy, implementing different software safety requirements specification	C.3.5	---	---	R	HR
3f	Backward recovery	C.3.6	R	R	---	NR
3g	Stateless software design (or limited state design)	C.2.12	---	---	R	HR
4a	Re-try fault recovery mechanisms	C.3.7	R	R	---	---
4b	Graceful degradation	C.3.8	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.9	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.10	---	NR	NR	NR
7	Modular approach	Table B.9	HR	HR	HR	HR
8	Use of trusted/verified software elements (if available)	C.2.10	R	HR	HR	HR
9	Forward traceability between the software safety requirements specification and software architecture	C.2.11	R	R	HR	HR
10	Backward traceability between the software safety requirements specification and software architecture	C.2.11	R	R	HR	HR
11a	Structured diagrammatic methods **	C.2.1	HR	HR	HR	HR
11b	Semi-formal methods **	Table B.7	R	R	HR	HR
11c	Formal design and refinement methods **	B.2.2, C.2.4	---	R	R	HR
11d	Automatic software generation	C.4.6	R	R	R	R
12	Computer-aided specification and design tools	B.2.4	R	R	HR	HR
13a	Cyclic behaviour, with guaranteed maximum cycle time	C.3.11	R	HR	HR	HR
13b	Time-triggered architecture	C.3.11	R	HR	HR	HR
13c	Event-driven, with guaranteed maximum response time	C.3.11	R	HR	HR	-
14	Static resource allocation	C.2.6.3	-	R	HR	HR
15	Static synchronisation of access to shared resources	C.2.6.3	-	-	R	HR

NOTE 1 Some of the methods given in Table A.2 are about design concepts, others are about how the design is represented.

NOTE 2 The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in IEC 61508-2.

NOTE 3 See Table C.2.

NOTE 4 The group 13 measures apply only to systems and software with safety timing requirements.

NOTE 5 Measure 14. The use of dynamic objects (for example on the execution stack or on a heap) may impose requirements on both available memory and also execution time. Measure 14 does not need to be applied if a compiler is used which ensures a) that sufficient memory for all dynamic variables and objects will be allocated before runtime, or which guarantees that in case of memory allocation error, a safe state is achieved; b) that response times meet the requirements.

NOTE 6 Measure 4a. Re-try fault recovery is often appropriate at any SIL but a limit should be set on the number of retries.

NOTE 7 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

** Group 11, “Structured methods”. Use measure 11a only if 11b is not suited to the domain for SIL 3+4.

Table A.3 – Software design and development – support tools and programming language

(See 7.4.4)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Suitable programming language	C.4.5	HR	HR	HR	HR
2	Strongly typed programming language	C.4.1	HR	HR	HR	HR
3	Language subset	C.4.2	---	---	HR	HR
4a	Certified tools and certified translators	C.4.3	R	HR	HR	HR
4b	Tools and translators: increased confidence from use	C.4.4	HR	HR	HR	HR

NOTE 1 See Table C.3.

NOTE 2 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table A.4 – Software design and development – detailed design

(See 7.4.5 and 7.4.6)

(Includes software system design, software module design and coding)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Structured methods **	C.2.1	HR	HR	HR	HR
1b	Semi-formal methods **	Table B.7	R	HR	HR	HR
1c	Formal design and refinement methods **	B.2.2, C.2.4	---	R	R	HR
2	Computer-aided design tools	B.3.5	R	R	HR	HR
3	Defensive programming	C.2.5	---	R	HR	HR
4	Modular approach	Table B.9	HR	HR	HR	HR
5	Design and coding standards	C.2.6 Table B.1	R	HR	HR	HR
6	Structured programming	C.2.7	HR	HR	HR	HR
7	Use of trusted/verified software elements (if available)	C.2.10	R	HR	HR	HR
8	Forward traceability between the software safety requirements specification and software design	C.2.11	R	R	HR	HR

NOTE 1 See Table C.4.

NOTE 2 There is still debate about the suitability of OO software development for safety-related systems. See Annex G of IEC 61508-7 for guidance on object oriented architecture and design.

NOTE 3 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

** Group 1, “Structured methods”. Use measure 1a only if 1b is not suited to the domain for SIL 3+4.

Table A.5 – Software design and development – software module testing and integration

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

NOTE 1 Software module and integration testing are verification activities (see Table B.9).

NOTE 2 See Table C.5.

NOTE 3 Technique 9. Formal verification may reduce the amount and extent of module and integration testing required.

NOTE 4 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table A.6 – Programmable electronics integration (hardware and software)

(See 7.5)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
2	Performance testing	Table B.6	R	R	HR	HR
3	Forward traceability between the system and software design requirements for hardware/software integration and the hardware/software integration test specifications	C.2.11	R	R	HR	HR

NOTE 1 Programmable electronics integration is a verification activity (see Table A.9).

NOTE 2 See Table C.6.

NOTE 3 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table A.7 – Software aspects of system safety validation

(See 7.7)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	HR
2	Process simulation	C.5.18	R	R	HR	HR
3	Modelling	Table B.5	R	R	HR	HR
4	Functional and black-box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Forward traceability between the software safety requirements specification and the software safety validation plan	C.2.11	R	R	HR	HR
6	Backward traceability between the software safety validation plan and the software safety requirements specification	C.2.11	R	R	HR	HR

NOTE 1 See Table C.7.

NOTE 2 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table A.8 – Modification

(See 7.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Impact analysis	C.5.23	HR	HR	HR	HR
2	Reverify changed software module	C.5.23	HR	HR	HR	HR
3	Reverify affected software modules	C.5.23	R	HR	HR	HR
4a	Revalidate complete system	Table A.7	---	R	HR	HR
4b	Regression validation	C.5.25	R	HR	HR	HR
5	Software configuration management	C.5.24	HR	HR	HR	HR
6	Data recording and analysis	C.5.2	HR	HR	HR	HR
7	Forward traceability between the Software safety requirements specification and the software modification plan (including reverification and revalidation)	C.2.11	R	R	HR	HR
8	Backward traceability between the software modification plan (including reverification and revalidation)and the software safety requirements specification	C.2.11	R	R	HR	HR

NOTE 1 See Table C.8.

NOTE 2 Techniques group 4. Impact analysis is a necessary part of regression validation. See IEC 61508-7.

NOTE 3 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table A.9 – Software verification

(See 7.9)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Formal proof	C.5.12	---	R	R	HR
2	Animation of specification and design	C.5.26	R	R	R	R
3	Static analysis	B.6.4 Table B.8	R	HR	HR	HR
4	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
5	Forward traceability between the software design specification and the software verification (including data verification) plan	C.2.11	R	R	HR	HR
6	Backward traceability between the software verification (including data verification) plan and the software design specification	C.2.11	R	R	HR	HR
7	Offline numerical analysis	C.2.13	R	R	HR	HR
Software module testing and integration		See Table A.5				
Programmable electronics integration testing		See Table A.6				
Software system testing (validation)		See Table A.7				

NOTE 1 For convenience all verification activities have been drawn together under this table. However, this does not place additional requirements for the dynamic testing element of verification in Table A.5 and Table A.6 which are verification activities in themselves. Nor does this table require verification testing in addition to software validation (see Table B.7), which in this standard is the demonstration of conformance to the safety requirements specification (end-end verification).

NOTE 2 Verification crosses the boundaries of IEC 61508-1, IEC 61508-2 and IEC 61508-3. Therefore the first verification of the safety-related system is against the earlier system level specifications.

NOTE 3 In the early phases of the software safety lifecycle verification is static, for example inspection, review, formal proof. When code is produced dynamic testing becomes possible. It is the combination of both types of information that is required for verification. For example code verification of a software module by static means includes such techniques as software inspections, walk-throughs, static analysis, formal proof. Code verification by dynamic means includes functional testing, white-box testing, statistical testing. It is the combination of both types of evidence that provides assurance that each software module satisfies its associated specification.

NOTE 4 See Table C.9.

NOTE 5 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table A.10 – Functional safety assessment

(see Clause 8)

Assessment/Technique *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Checklists	B.2.5	R	R	R	R
2	Decision/truth tables	C.6.1	R	R	R	R
3	Failure analysis	Table B.4	R	R	HR	HR
4	Common cause failure analysis of diverse software (if diverse software is actually used)	C.6.3	---	R	HR	HR
5	Reliability block diagram	C.6.4	R	R	R	R
6	Forward traceability between the requirements of Clause 8 and the plan for software functional safety assessment	C.2.11	R	R	HR	HR
NOTE 1 See Table C.10. NOTE 2 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.						
* Appropriate techniques/measures shall be selected according to the safety integrity level.						

Annex B
(informative)

Detailed tables

Table B.1 – Design and coding standards

(Referenced by Table A.4)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Use of coding standard to reduce likelihood of errors	C.2.6.2	HR	HR	HR	HR
2	No dynamic objects	C.2.6.3	R	HR	HR	HR
3a	No dynamic variables	C.2.6.3	---	R	HR	HR
3b	Online checking of the installation of dynamic variables	C.2.6.4	---	R	HR	HR
4	Limited use of interrupts	C.2.6.5	R	R	HR	HR
5	Limited use of pointers	C.2.6.6	---	R	HR	HR
6	Limited use of recursion	C.2.6.7	---	R	HR	HR
7	No unstructured control flow in programs in higher level languages	C.2.6.2	R	HR	HR	HR
8	No automatic type conversion	C.2.6.2	R	HR	HR	HR

NOTE 1 Measures 2, 3a and 5. The use of dynamic objects (for example on the execution stack or on a heap) may impose requirements on both available memory and also execution time. Measures 2, 3a and 5 do not need to be applied if a compiler is used which ensures a) that sufficient memory for all dynamic variables and objects will be allocated before runtime, or which guarantees that in case of memory allocation error, a safe state is achieved; b) that response times meet the requirements.

NOTE 2 See Table C.11.

NOTE 3 The references (which are informative, not normative) “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table B.2 – Dynamic analysis and testing

(Referenced by Tables A.5 and A.9)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Test case execution from boundary value analysis	C.5.4	R	HR	HR	HR
2	Test case execution from error guessing	C.5.5	R	R	R	R
3	Test case execution from error seeding	C.5.6	---	R	R	R
4	Test case execution from model-based test case generation	C.5.27	R	R	HR	HR
5	Performance modelling	C.5.20	R	R	R	HR
6	Equivalence classes and input partition testing	C.5.7	R	R	R	HR
7a	Structural test coverage (entry points) 100 % **	C.5.8	HR	HR	HR	HR
7b	Structural test coverage (statements) 100 %**	C.5.8	R	HR	HR	HR
7c	Structural test coverage (branches) 100 %**	C.5.8	R	R	HR	HR
7d	Structural test coverage (conditions, MC/DC) 100 %**	C.5.8	R	R	R	HR

NOTE 1 The analysis for the test cases is at the subsystem level and is based on the specification and/or the specification and the code.

NOTE 2 See Table C.12.

NOTE 3 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

** Where 100 % coverage cannot be achieved (e.g. statement coverage of defensive code), an appropriate explanation should be given.

Table B.3 – Functional and black-box testing

(Referenced by Tables A.5, A.6 and A.7)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Test case execution from cause consequence diagrams	B.6.6.2	---	---	R	R
2	Test case execution from model-based test case generation	C.5.27	R	R	HR	HR
3	Prototyping/animation	C.5.17	---	---	R	R
4	Equivalence classes and input partition testing, including boundary value analysis	C.5.7 C.5.4	R	HR	HR	HR
5	Process simulation	C.5.18	R	R	R	R

NOTE 1 The analysis for the test cases is at the software system level and is based on the specification only.

NOTE 2 The completeness of the simulation will depend upon the safety integrity level, complexity and application.

NOTE 3 See Table C.13.

NOTE 4 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table B.4 – Failure analysis

(Referenced by Table A.10)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1a	Cause consequence diagrams	B.6.6.2	R	R	R	R
1b	Event tree analysis	B.6.6.3	R	R	R	R
2	Fault tree analysis	B.6.6.5	R	R	R	R
3	Software functional failure analysis	B.6.6.4	R	R	R	R

NOTE 1 Preliminary hazard analysis should have already taken place in order to categorize the software into the most appropriate safety integrity level.

NOTE 2 See Table C.14.

NOTE 3 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table B.5 – Modelling

(referenced by Table A.7)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Data flow diagrams	C.2.2	R	R	R	R
2a	Finite state machines	B.2.3.2	---	R	HR	HR
2b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2c	Time Petri nets	B.2.3.3	---	R	HR	HR
3	Performance modelling	C.5.20	R	HR	HR	HR
4	Prototyping/animation	C.5.17	R	R	R	R
5	Structure diagrams	C.2.3	R	R	R	HR

NOTE 1 If a specific technique is not listed in the table, it should not be assumed that it is excluded from consideration. It should conform to this standard.

NOTE 2 Quantification of probabilities is not required.

NOTE 3 See Table C.15.

NOTE 4 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table B.6 – Performance testing

(referenced by Tables A.5 and A.6)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Avalanche/stress testing	C.5.21	R	R	HR	HR
2	Response timings and memory constraints	C.5.22	HR	HR	HR	HR
3	Performance requirements	C.5.19	HR	HR	HR	HR

NOTE 1 See Table C.16.

NOTE 2 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table B.7 – Semi-formal methods

(Referenced by Tables A.1, A.2 and A.4)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Logic/function block diagrams	See Note 1	R	R	HR	HR
2	Sequence diagrams	see Note 1	R	R	HR	HR
3	Data flow diagrams	C.2.2	R	R	R	R
4a	Finite state machines/state transition diagrams	B.2.3.2	R	R	HR	HR
4b	Time Petri nets	B.2.3.3	R	R	HR	HR
5	Entity-relationship-attribute data models	B.2.4.4	R	R	R	R
6	Message sequence charts	C.2.14	R	R	R	R
7	Decision/truth tables	C.6.1	R	R	HR	HR
8	UML	C.3.12	R	R	R	R

NOTE 1 Logic/function block diagrams and sequence diagrams are described in IEC 61131-3.

NOTE 2 See Table C.17.

NOTE 3 The references "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table B.8 – Static analysis

(Referenced by Table A.9)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Boundary value analysis	C.5.4	R	R	HR	HR
2	Checklists	B.2.5	R	R	R	R
3	Control flow analysis	C.5.9	R	HR	HR	HR
4	Data flow analysis	C.5.10	R	HR	HR	HR
5	Error guessing	C.5.5	R	R	R	R
6a	Formal inspections, including specific criteria	C.5.14	R	R	HR	HR
6b	Walk-through (software)	C.5.15	R	R	R	R
7	Symbolic execution	C.5.11	---	---	R	R
8	Design review	C.5.16	HR	HR	HR	HR
9	Static analysis of run time error behaviour	B.2.2, C.2.4	R	R	R	HR
10	Worst-case execution time analysis	C.5.20	R	R	R	R

NOTE 1 See Table C.18.

NOTE 2 The references "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

Table B.9 – Modular approach

(Referenced by Table A.4)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Software module size limit	C.2.9	HR	HR	HR	HR
2	Software complexity control	C.5.13	R	R	HR	HR
3	Information hiding/encapsulation	C.2.8	R	HR	HR	HR
4	Parameter number limit / fixed number of subprogram parameters	C.2.9	R	R	R	R
5	One entry/one exit point in subroutines and functions	C.2.9	HR	HR	HR	HR
6	Fully defined interface	C.2.9	HR	HR	HR	HR

NOTE 1 See Table C.19.

NOTE 2 The references "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. No single technique is likely to be sufficient. All appropriate techniques shall be considered.

Annex C
(informative)

Properties for software systematic capability

C.1 Introduction

Given the large number of factors that affect software systematic capability it is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application. The purpose of Annex C is:

– to give guidance on selecting specific techniques from Annexes A and B to achieve software systematic capability;

– to outline a rationale for justifying the use of techniques that are not explicitly listed in Annexes A and B.

Annex C is supplementary to Annexes A and B tables.

C.1.1 Structure of Annex C, relating to Annexes A and B

The outputs from each phase of the software safety lifecycle are defined in Table 1. For example, consider the software safety requirements specification.

Table A.1 (“Software safety requirements specification”) of Annex A recommends specific techniques for developing the software safety requirements specification.

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

Annex C Table C.1 (“Properties for systematic safety integrity – Software safety requirements specification”) states that the software safety requirements specification is characterized by the following desirable properties (which are informally defined in Annex F of IEC 61508-7):

Properties					
Completeness with respect to the safety needs to be addressed by software	Correctness with respect to the safety needs to be addressed by software	Freedom from intrinsic specification faults, including freedom from ambiguity	Understandability of safety requirements	Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software	Capability of providing a basis for verification and validation

Annex C Table C.1 also ranks on an informal scale R1/R2/R3 the effectiveness of specific techniques in achieving these desirable properties.

Technique/ Measure		Properties					
		Completeness with respect to the safety needs to be addressed by software	Correctness with respect to the safety needs to be addressed by software	Freedom from intrinsic specification faults, including freedom from ambiguity	Understandability of safety requirements	Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software	Capability of providing a basis for verification and validation
1a	Semi-formal methods	R1 Application-friendly or domain specific specification method and notation used by domain experts	R1 Application-friendly or domain specific specification method and notation used by domain experts R2 Verification of specification according to coverage criteria	R1 Method and notation that helps avoid or detect internal inconsistency, missing behaviour or mathematically inconsistent expressions. R2 Verification of specification according to coverage criteria R3 Verification of specification based on systematic analysis, and / or systematic avoidance of particular types of intrinsic specification faults	R1 Defined notation that restricts opportunity for misunderstanding R2 Application of complexity limits in specification	_	R2 Defined notation that reduces ambiguity in specification

The confidence that can be placed in the software safety requirements specification as a basis for safe software depends on the rigour of the techniques by which the desirable properties of the software safety requirements specification have been achieved. The rigour of a technique is informally ranked on a scale R1 to R3, where R1 is the least rigorous and R3 the most rigorous.

R1	without objective acceptance criteria, or with limited objective acceptance criteria. E.g., black-box testing based on judgement, field trials.
R2	with objective acceptance criteria that can give a high level of confidence that the required property is achieved (exceptions to be identified & justified); e.g., test or analysis techniques with coverage metrics, coverage of checklists.
R3	with objective, systematic reasoning that the required property is achieved. E.g. formal proof, demonstrated adherence to architectural constraints that guarantee the property.
_	this technique is not relevant to this property.

A technique may achieve one of several R1/R2/R3 rankings relating to a particular property, depending on the level of rigour that the technique satisfies.

Technique/ Measure		Properties					
		Completeness with respect to the safety needs to be addressed by software	Correctness with respect to the safety needs to be addressed by software	Freedom from intrinsic specification faults, including freedom from ambiguity	Understandability of safety requirements	Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software	Capability of providing a basis for verification and validation
1a	Semi-formal methods				R1 Defined notation that restricts opportunity for misunderstanding R2 Application of complexity limits in specification		

In this example, a semi-formal method achieves rigour R1 by providing a restricted notation that improves accurate expression, and achieves R2 by further restricting the complexity of specification which might otherwise cause confusion.

C.1.2 Method of use – 1

For guidance purposes, if it can be convincingly demonstrated that the desirable properties have been achieved in the development of the software safety requirements specification, then confidence is justified that the software safety requirements specification is an adequate basis for developing software that has sufficient systematic safety integrity.

Annex C Table C.1 says that each of the Annex A Table A.1 techniques typically achieves, to a greater or lesser extent, one or more of the above Table C.1 properties that are relevant to the software safety requirements specification.

However, it is important to note that although Annex A Table A.1 recommends specific techniques, these recommendations are not prescriptive, and in fact Annex A states clearly that “Given the large number of factors that affect software systematic capability it is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application”.

In practice the techniques by which the software safety requirements specification is developed are selected subject to several practical constraints (see 7.1.2.7) in addition to the inherent capabilities of the techniques. Such constraints may include:

– the consistency and the complementary nature of the chosen methods, languages and tools for the whole development cycle;
– whether the developers use methods, languages and tools they fully understand;
– whether the methods, languages and tools are well-adapted to the specific problems encountered during development.

Table C.1 may be used to compare the relative effectiveness of the specific Annex A Table A.1 techniques in achieving the desirable properties of the software safety requirements specification lifecycle, while at the same time factoring in the practical constraints of the particular development project.

For example, a formal method is capable of giving a better basis (R3) for verification and validation than is a semi-formal method (R2), but other project constraints (e.g. the availability of sophisticated computer support tools, or the very specialized expressiveness of a formal notation) may favour a semi-formal approach.

In this way, the Table C.1 desirable properties can provide the basis of a reasoned and practical comparison of the alternative techniques that Annex A Table A.1 recommends for developing the software safety requirements specification. Or more generally, a reasoned selection from the several alternative techniques recommended by Annex A for a particular lifecycle phase can be made by considering the desirable properties listed in the corresponding Annex C table.

But note carefully that due to the nature of systematic behaviour, these Annex C properties may not be achievable or demonstrable with the highest rigour. Rather, they are goals to be aimed for. Their achievement may even necessitate trade-offs between different properties e.g. between defensive design and simplicity.

Finally, in addition to defining R1/R2/R3 criteria, it is useful for guidance purposes to make an informal link between (1) the increasing level of rigour of the R1 to R3 progression and (2) an increased confidence in the correctness of the software. As a general and informal recommendation, the following minimum levels of rigour should be aimed for when Annex A requires the corresponding SIL performance:

SIL	Rigour R
1 / 2	R1
3	R2 where available
4	highest rigour available

C.1.3 Method of use – 2

Although Annex A recommends specific techniques, it is also permitted to apply other measures and techniques, providing that the requirements and objectives of the lifecycle phase have been met.

It has already been noted that many factors affect software systematic capability, and it is not possible to give an algorithm for selecting and combining the techniques in a way that is guaranteed in any given application to achieve the desirable properties.

There may be several effective ways to achieve the desirable properties, and it should be recognized that system developers may be able to provide alternative evidence. The information in these Annex C tables can be used as the basis of a reasoned argument to justify the selection of techniques other than those given in the Annex A tables.

C.2 Properties for systematic safety integrity

The guidance here and in IEC 61508-7 indicates specific techniques for achieving the systematic safety integrity properties and for generating convincing evidence. Where a method does not contribute to the achievement of a property, this is shown in the following tables by a dash. Where a method may have adverse effects on some properties and positive effects on others, a note is provided in the relevant table below.

Table C.1 – Properties for systematic safety integrity – Software safety requirements specification

(See 7.2. Referenced by Table A.1)

Technique/Measure		Properties					
		Completeness with respect to the safety needs to be addressed by software	**Correctness with respect to the safety needs to be addressed by software**	**Freedom from intrinsic specification faults, including freedom from ambiguity**	**Understandability of safety requirements**	**Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software**	**Capability of providing a basis for verification and validation**
1a	Semi-formal methods	R1 Application-friendly or domain specific specification method and notation used by domain experts	R1 Application-friendly or domain specific specification method and notation used by domain experts R2 Verification of specification according to coverage criteria	R1 Method and notation that helps avoid or detect internal inconsistency, missing behaviour or mathematically inconsistent expressions. R2 Verification of specification according to coverage criteria R3 Verification of specification based on systematic analysis, and / or systematic avoidance of particular types of intrinsic specification faults	R1 Defined notation that restricts opportunity for misunderstanding R2 Application of complexity limits in specification	–	R2 Defined notation that reduces ambiguity in specification

Technique/Measure		Properties					
		Completeness with respect to the safety needs to be addressed by software	**Correctness with respect to the safety needs to be addressed by software**	**Freedom from intrinsic specification faults, including freedom from ambiguity**	**Understandability of safety requirements**	**Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software**	**Capability of providing a basis for verification and validation**
1b	Formal methods	R1 Application-friendly or domain specific specification method and notation used by domain experts	R1 Application-friendly or domain specific specification method and notation used by domain experts. R2 Verification of specification according to coverage criteria R3 Guarantee of correctness on limited aspects of behaviour	R1 Method and notation that help avoid or detect internal inconsistency, missing behaviour or mathematically inconsistent expressions. R2 Verification of specification according to coverage criteria R3 Verification of specification based on systematic analysis, and / or Systematic avoidance of particular types of intrinsic specification faults	– Note: May complicate the achievement of this property if the method is not application-friendly or domain specific.	–	R3 Reduces ambiguity in specification.
2	Forward traceability between the system safety requirements specification and the software safety requirements	R1 Confidence that the Software safety requirements specification addresses the system safety requirements	–	–	–	–	–
3	Backward traceability between the software safety requirements specification and the perceived safety needs	–	R1 Confidence that the software safety requirements specification contains no unnecessary complexity	–	RI Traceability to the EUC safety needs enhances understandability	R1	R1

Technique/Measure		Properties					
		Completeness with respect to the safety needs to be addressed by software	**Correctness with respect to the safety needs to be addressed by software**	**Freedom from intrinsic specification faults, including freedom from ambiguity**	**Understandability of safety requirements**	**Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software**	**Capability of providing a basis for verification and validation**
4	Computer-aided specification tools to support appropriate techniques/measures above	R1 Encapsulation of domain knowledge of the EUC and of the software environment R2 If checklist of issues to be taken into consideration is defined, justified and covered	R1 Functional simulation techniques R2 Functional simulation according to defined and justified coverage criteria	R2 Semantic and syntactic checks to ensure that the relevant rules are satisfied	R1 Animation of, or browsing through the specification	R1 Identification of safety and non-safety functions	R1 Assists traceability and coverage R2 Measurement of traceability and coverage

Table C.2 – Properties for systematic safety integrity – Software design and development – software Architecture Design

(See 7.4.3. Referenced by Table A.2)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
1	Fault detection	_	_	_	_ May complicate the achievement of this property	R1 Logical program flow monitoring provides for predictability	_	R1 (R2 if coverage targets are defined, justified and met)	R1 or _
2	Error detecting codes	_	_	_	_ May complicate the achievement of this property.	_ May complicate the achievement of this property.	_	R1 (R2 if coverage targets are defined, justified and met) Effective for specific application areas e.g. data comms	R1 Effective for specific application areas e.g. data comms
3a	Failure assertion programming	_	R2 Post-assertions may check compliance with detailed requirements	_	R2 Pre-assertions limit the input space	R2 Post-assertions check for expected / acceptable outputs	R2 Pre-assertions limit the input space and hence the required test space	R3 Effective for the targeted failures	R3 Effective for the targeted failures
3b	Diverse monitor techniques (with independence between the monitor and the monitored function in the same computer)	_	_	R2 Diverse monitor implements only the minimum safety requirements	R2 Diverse monitor provides for implicit diversity	R2 Diverse monitor implements in a simple manner only the minimum safety requirements	R2 Diverse monitor implements only the minimum safety requirements	R1 (R2 if coverage targets are defined, justified and met)	R1 (R2 if coverage targets are defined, justified and met)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
3c	Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	-	_	R2 Diverse monitor implements only the minimum safety requirements	R2 Diverse monitor provides for implicit diversity	R2 Diverse monitor implements in a simple manner only the minimum safety requirements)	R2 Diverse monitor implements only the minimum safety requirements	R1 (R2 if coverage targets are defined, justified and met)	R1 (R2 if coverage targets are defined, justified and met)
3d	Diverse redundancy, implementing the same software safety requirements specification	_	_	_	_ Note: May complicate the achievement of this property if done within the same executable software.	_	_	R1 If the failure of one program does not adversely affect the others R2 If coverage targets are defined, justified and met Does not protect against requirements specification faults	R1 If the failure of one program does not adversely affect the others R2 If coverage targets are defined, justified and met Does not protect against requirements specification faults

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
3e	Functionally diverse redundancy, implementing different software safety requirements specification. This will typically require sensors operating on different physical principles	–	–	R1	– Note: May complicate the achievement of this property if done within the same executable software.	–	–	R1 If the failure of one program does not adversely affect the others.	R1 If the failure of one program does not adversely affect the others. Protects against specification faults
3f	Backward recovery	–	–	– Note: May complicate the achievement of this property.	–	– Note: May complicate the achievement of this property.	–	R2	R1 (R2 if coverage targets are defined, justified and met)
3g	Stateless design (or limited state design)	R2 Provided safety requirements are also stateless or limited state	R2 Provided safety requirements are also stateless or limited state	R2 Provided safety requirements are also stateless or limited state	R1 R2 If limits are defined, justified and met regarding the possible number of states	R1 R2 If limits are defined, justified and met regarding the possible number of states	R1 R2 If targets are defined, justified and met for the verification / test coverage of the possible states	R1 If this leads to a self-healing design R2 If targets are defined, justified and met for self-healing	R1 If this leads to a self-healing design R2 If targets are defined, justified and met for self-healing
4a	Re-try fault recovery mechanisms	–	–	–	–	May complicate the achievement of this property	–	R1 (R2 if coverage targets are defined, justified and met)	R1 (R2 if coverage targets are defined, justified and met)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
4b	Graceful degradation	–	–	– Note: May complicate the achievement of this property.	–	–	–	R1 R2 if coverage targets are defined, justified and met	R1 R2 if coverage targets are defined, justified and met
5	Artificial intelligence - fault correction	–	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	–	–
6	Dynamic reconfiguration	–	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	–	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
7	Modular approach	_	R1 R2 R2 is achieved If modularity targets are defined, justified and met. Otherwise, only R1 is achieved.	R1 If freedom from particular types of intrinsic design faults can be verified independently for each module R3 If freedom from particular types of intrinsic design faults can be supported by a rigorous reasoning based on modular design	R1 R2 If modularity targets are defined, justified and met	R1 R2 If modularity targets are defined, justified and met	R1 R2 If modularity targets are defined, justified and met	R1 If modules not affected by the failure of a module contribute in mitigation / recovery R3 If tolerance to particular faults can be supported by a rigorous reasoning	R1 If modules that can be influenced by external events that can affect multiple channels concurrently, are identified and subject to thorough verification R3 If tolerance to particular external events can be supported by a rigorous reasoning

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
8	Use of trusted/verified software modules and elements (if available)	–	R1 R2 R3 If the element significantly contributes to particular safety requirements, and is correctly used	R1 R2 R3 Re-uses proven elements. Such capability shall be justified for the element	R1 Modular approach decomposes overall complexity into understandable units	R1 R2 R3 Reuses proven elements	–	R1R2 R2 If fault tolerance capabilities are readily provided by the element and are correctly used, or if a fault tolerance layer is built around the element	R1R2 R2 If defences against external events that could affect concurrently multiple channels are readily provided by the element and are correctly used, or if a defensive layer is built around the element
9	Forward traceability between the software safety requirements specification and software architecture	R1 Confidence that the architecture addresses the software safety requirements	–	–	–	–	–	–	–
10	Backward traceability between the software architecture and the software safety requirements specification	–	R1 Confidence that the architecture contains no unnecessary complexity	–	–	–	–	–	–
11a	Structured diagrammatic methods	–	R1	–	R1 (Graphical descriptions are easier to understand)	–	R1 (Structured designs are easier to verify and test)	–	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
11b	Semi-formal methods	R1 An application-friendly or domain specific specification method and notation	R1 An application-friendly or domain specific specification method and notation	R2 Can detect internal inconsistency or missing behaviour or mathematically inconsistent expressions	–	R2 (Provides evidence for predictability)	R2 (Provides evidence for inner consistence of the design model)	–	–
11c	Formal design and refinement methods	R1 An application-friendly or domain specific specification method and notation	R1 Provides precise definition of limited aspects of behaviour which needs to be appropriate to the domain	R3 Can detect internal inconsistency or missing behaviour or mathematically inconsistent expressions	– Note: May complicate the achievement of this property.	R2 Provides proof for predictability	R2	–	–
11d	Automatic software generation	R1 If executable software is automatically generated from requirements specification, or from a design that has been shown to be complete R2 If the generation tools are shown to have appropriate pedigree	R1 If executable software is automatically generated from requirements specification, or from a design that has been shown to be correct R2 If the generation tools are shown to have appropriate pedigree	R1 If the generation tools guarantee avoidance of particular intrinsic design faults R2 If the generation tools are shown to have appropriate pedigree	–	–	–	R1 R2 R3 If fault tolerance capabilities are automatically generated	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
12	Computer-aided specification and design tools	R1 Encapsulation of domain knowledge of the EUC and of the software environment R2 If checklist of issues to be taken into consideration are defined, justified and covered	R1 Enforcement of backward requirements traceability Functional simulation techniques R2 Functional simulation according to defined and justified coverage criteria	R2 Semantic and syntactic checks to ensure that the relevant rules are satisfied	R1 Animation and browsing	–	R2 Semantic and syntactic checks to ensure that the relevant rules are satisfied	–	–
13a	Cyclic behaviour, with guaranteed maximum cycle time	–	R1 for timing aspects of specification R3 If maximum cycle time established by rigorous reasoning	R1 for timing aspects of specification R3 If maximum cycle time established by rigorous reasoning	–	R1 for timing aspects of specification R3 If maximum cycle time established by rigorous reasoning	R1 for timing aspects of specification R3 If maximum cycle time established by rigorous reasoning	–	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance**	**Defence against common cause failure from external events**
13b	Time-triggered architecture	R3 Completeness is guaranteed by allocation (only for timing properties)	R3 Correctness is guaranteed by allocation (only for timing properties)	R3 Rigorous guarantee against intrinsic timing faults	R1 Defined notation reduces misunderstanding considerably, predictability as approach	R3 Adverse interference: total separation in time domain, no interference	R3 Greatly reduces the effort required for testing and certifying the system	R2 Transparent implementation of fault-tolerance	R3 External interrupts cannot interfere with the time-triggered schedule which gives priority to safety-critical tasks
13c	Event-driven, with guaranteed maximum response time	–	–	–	R1 Event driven architectures may hinder understandibility	R2 Event driven architectures may hinder understandibility	R1 Makes testing more predictable	–	–
14	Static resource allocation	R1	R1	R1	R1 Makes the design more understandable	R2 With architecture defining resource usage	R1 Makes testing more predictable	–	–
15	Static synchronisation of access to shared resources	–	R1 Gives predictability in resource access	R1 R3 if supported by rigorous reasoning as to correctness of synchronisation	R1 Makes the design more understandable	R1 R3 if supported by rigorous reasoning as to correctness of synchronisation	–	–	–

Table C.3 – Properties for systematic safety integrity – Software design and development – support tools and programming language

(See 7.4.4. Referenced by Table A.3)

Technique/Measure		Properties		
		Support the production of software with the required software properties	**Clarity of the operation and functionality of the tool**	**Correctness and repeatability of output**
1	Suitable programming language	R2 if strong typing, restricted type conversion. R3 if defined semantics for rigorous reasoning	–	–
2	Strongly typed programming language	R2	–	–
3	Language subset	R2 Depending on chosen subset	R1	R2 Depending on chosen subset
4a	Certificated tools	–	R2	R2
4b	Tools: increased confidence from use	R1 If the class of detected program errors is systematically defined R2 If there is objective validation evidence for the tool performance.	R1 If the tool support is non-specific to the problem domain. R2 If the tool support is significantly specialized to the problem domain.	R1 R2 If there is objective validation evidence for the tool performance e.g. a complier validation suite.

Table C.4 – Properties for systematic safety integrity – Software design and development – detailed design (includes software system design, software module design and coding)

(See 7.4.5 and 7.4.6. Referenced by Table A.4)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance / Fault detection**	**Freedom from common cause failure**
1a	Structured methods	R2	R1	R1	–	–	R1 Structured designs are more readily verifiable and testable	–	–
1b	Semi-formal methods	R2	R2	R2	–	R2	R2	–	–
1c	Formal design and refinement methods	–	R3	R3	– Note: May complicate the achievement of this property.	R3 Provides evidence for predictability	R2	–	–
2	Computer-aided design tools	R2 Dependent upon the Computer aided specification tool applying semantic and syntactic checks to ensure that the relevant rules are satisfied	R1	R2 Dependent upon the computer aided specification tool applying semantic and syntactic checks to ensure that the relevant rules are satisfied	–	–	R2 Dependant upon CASE tool to support test coverage and static verification	–	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance / Fault detection**	**Freedom from common cause failure**
3	Defensive programming	–	–	–	– Note: May complicate the achievement of this property.	– Note: May complicate the achievement of this property.	–	R1 (R2 if coverage targets are defined, justified and met)	R1 (R2 if coverage targets are defined, justified and met)
4	Modular approach	–	–	R1	R1	R1	R1	–	–
5	Design and coding standards	–	–	R1	R1	R1	R1	–	–
6	Structured programming	–	R1	R1	R1	R1	R1	–	–
7	Use of trusted/verified software modules and elements (if available)	–	–	R1 Reuses proven elements	R1 Modular approach decomposes overall complexity into understandable units	R1 The behaviour of the element is already known	–	–	–
8	Forward traceability between the software safety requirements specification and software design	R1 Confidence that the design addresses the software safety requirements	–	–	–	–	–	–	–

Table C.5 – Properties for systematic safety integrity – Software design and development – software module testing and integration

(See 7.4.7 and 7.4.8. Referenced by Table A.5)

Technique/Measure		Properties			
		Completeness of testing and integration with respect to the software design specification	**Correctness of testing and integration with respect to the software design specification (successful completion)**	**Repeatability**	**Precisely defined testing configuration**
1	Probabilistic testing	R1 (R2 if operational profile coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
2	Dynamic analysis and testing	R1 (R2 if structural coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
3	Data recording and analysis	–	R1	R1 Promotes consistency in testing procedures	R2 If fault records/test logs include details of software baseline
4	Functional and black box testing	R1 (R2 if operational profile coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
5	Performance testing	–	R1 (R2 if required outputs are defined, justified and met)	–	–

Technique/Measure		Properties			
		Completeness of testing and integration with respect to the software design specification	**Correctness of testing and integration with respect to the software design specification (successful completion)**	**Repeatability**	**Precisely defined testing configuration**
6	Model based testing (MBT)	R2 MBT allows early exposure of ambiguities in specification and design, the MBT process starts with requirements R3 If rigorous reasoning is applied to modelling, and test case generation (TCG) is used	R2 Evaluation of results and regression test suites is a key benefit of MBT R3 If rigorous modelling approach is applied, then objective evidence of coverage is possible	R3 MBT (with TCG) aims at automatic execution of generated tests	R2 MBT is automated, testing configuration has to be precisely defined; execution of the generated tests is similar to black box testing with the possibility to be combined with source code level coverage measurement
7	Interface testing	–	R1 (R2 if required outputs are defined, justified and met)	–	–
8	Test management and automation tools	R1 (R2 if test coverage targets are defined, justified and met)	–	R1 Automation promotes consistency	R2 Gives repeatability of testing
9	Forward traceability between the software safety requirements specification and the module and integration test specifications	R1 Confidence that the test specification addresses the software safety requirements	–	–	R2 Confidence in a clear baseline of requirements under test
10	Formal verification	R3 If rigorous reasoning is applied to construction of test cases to show that all aspects of design have been exercised	R3 Gives objective evidence of meeting all of the software safety requirements	R1 If support tools unavailable R2 If tool supported	–

Table C.6 – Properties for systematic safety integrity – Programmable electronics integration (hardware and software)

(See 7.5. Referenced by Table A.6)

Technique/Measure		Properties			
		Completeness of integration with respect to the design specifications	**Correctness of integration with respect to the design specifications (successful completion)**	**Repeatability**	**Precisely defined integration configuration**
1	Functional and black box testing	R1 (R2 if operational profile coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
2	Performance testing	–	R1 (R2 if required outputs are defined, justified and met)	–	–
3	Forward traceability between the system and software design requirements for hardware/software integration and the hardware/software integration test specifications	R1 Confidence that the hardware/software integration test specifications addresses the integration requirements	–	–	R2 Confidence with a clear baseline of requirements under test

Table C.7 – Properties for systematic safety integrity – Software aspects of system safety validation

(See 7.7. Referenced by Table A.7)

Technique/Measure		Properties			
		Completeness of validation with respect to the software Design Specification	**Correctness of validation with respect to the software Design Specification (successful completion)**	**Repeatability**	**Precisely defined validation configuration**
1	Probabilistic testing	R1 (R2 if operational profile coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
2	Process simulation	R1	R1 (R2 if required outputs are defined, justified and met)	–	R2 Gives a definition of the external environment
3	Functional and black-box testing	R1 (R2 if operational profile coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
4	Forward traceability between the software safety requirements specification and the software safety validation plan	R1 Confidence that the software safety validation plan addresses the software safety requirements	–	–	R2 Confidence with a clear baseline of requirements under test
5	Backward traceability between the software safety validation plan and the software safety requirements specification	–	R1 Confidence that software safety validation plan contains no unnecessary complexity	–	R2 Confidence with a clear baseline of requirements under test

Table C.8 – Properties for systematic safety integrity – Software modification

(See 7.8. Referenced by Table A.8)

Technique/Measure		Properties					
		Completeness of modification with respect to its requirements	**Correctness of modification with respect to its requirements**	**Freedom from introduction of intrinsic design faults**	**Avoidance of unwanted behaviour**	**Verifiable and testable design**	**Regression testing and verification coverage**
1	Impact analysis	–	–	–	R1	R1	R1
2	Re-verify changed software module	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	–	–	R1R2 (R2 if objective verification targets)
3	Re-verify affected software modules	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	–	–	R1 (R2 if objective verification targets)
4a	Revalidate complete system	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	–	R1 (R2 if objective verification targets)	–	R1 (R2 if objective verification targets)
4b	Regression validation	R1 (R2 if objective verification targets)	R1 (R2 if objective verification targets)	–	R1 (R2 if objective verification targets)	–	R1 (R2 if objective verification targets)
5	Software configuration management	–	–	–	–	–	R1
6	Data recording and analysis	R1	R1	–	–	–	–

Technique/Measure		Properties					
		Completeness of modification with respect to its requirements	**Correctness of modification with respect to its requirements**	**Freedom from introduction of intrinsic design faults**	**Avoidance of unwanted behaviour**	**Verifiable and testable design**	**Regression testing and verification coverage**
7	Forward traceability between the software safety requirements specification and the software modification plan (including reverification and revalidation)	R1 Confidence that the software modification plan (including re-verification and revalidation) addresses the software safety requirements	–	–	–	–	–
8	Backward traceability between the software modification plan (including reverification and revalidation) and the software safety requirements specification	–	R1 Confidence that software modification plan (including re-verification and revalidation) contains no unnecessary complexity	–	–	–	–

Table C.9 – Properties for systematic safety integrity – Software verification

(See 7.9. Referenced by Table A.9)

Technique/Measure		Properties			
		Completeness of verification with respect to the previous phase	**Correctness of verification with respect to the previous phase (successful completion)**	**Repeatability**	**Precisely defined verification configuration**
1	Formal proof	–	R3	–	–
2	Animation of specification and design	R1	R1	–	–
3	Static analysis	–	R1/R2/R3 (Rigour may range from language subset enforcement to mathematical formal analysis)	–	–
4	Dynamic analysis and testing	R1 (R2 if structural coverage targets are defined, justified and met)	R1 (R2 if required outputs are defined, justified and met)	–	–
5	Forward traceability between the software Design Specification and the software verification (including data verification) plan.	R1 Confidence that the software verification (including data verification) plan addresses the software safety requirements	–	–	R2 Confidence with a clear baseline of requirements under test
6	Backward traceability between the software verification (including data verification) plan and the software design specification	–	R1 Confidence that software verification (including data verification) plan contains no unnecessary complexity	–	R2 Confidence with a clear baseline of requirements under test
7	Offline numerical analysis	–	R1 Increased confidence in the expected numerical accuracy of well-conditioned calculations (R2 with objective acceptance criteria. R3 if used in conjunction with objective systematic reasoning to justify the acceptance criteria)	–	–

Table C.10 – Properties for systematic safety integrity – Functional safety assessment

(See Clause 8. Referenced by Table A.10)

Technique/Measure		Properties						
		Completeness of functional safety assessment with respect to this standard	**Correctness of functional safety assessment with respect to the design specifications (successful completion)**	**Traceable closure of all identified issues**	**The ability to modify the functional safety assessment after change without the need for extensive re-work of the assessment**	**Repeatability**	**Timeliness**	**Precisely defined configuration**
1	Checklists	R1	R1	R1	–	R1	–	–
2	Decision/truth tables	R1	R2	–	–	R2	–	–
3	Failure analysis	R2	R2	R1 (The failure analysis is based on agreed failure lists)	–	R1 (The failure analysis is based on agreed failure lists)	–	–
4	Common cause failure analysis of diverse software (if diverse software is actually used)	R2	R2	R1 (Provided the CCF analysis is based on agreed CC initiator lists)	–	R1 (Provided the CCF analysis is based on agreed CC initiator lists)	–	–
5	Reliability block diagram	R1	R1	–	–	–	–	–
6	Forward traceability between the requirements of IEC 61508-3 Clause 8 and the plan for software functional safety assessment	R1 Confidence that the plan for software functional safety assessment addresses the requirements of 61508-3 Clause 8	–	–	–	–	–	–

C.3 Properties for systematic safety integrity – Detailed tables

Table C.11 – Detailed properties – Design and coding standards

(Referenced by Table B.1)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance / Fault detection**	**Freedom from common cause failure**
1	Use of coding standard to reduce likelihood of errors	–	–	R1	R1 Eliminates selected language constructs	R1	R1	–	–
2	No dynamic objects	–	–	R1/R2/R3 Depending on language used	–	R1/R2/R3 Depending on language used	R1/R2 Depending on language used	–	–
3a	No dynamic variables	–	–	R1/R2/R3 Depending on language used	–	R1/R2/R3 Depending on language used	R1/R2 Depending on language used	–	–
3b	Online checking of the installation of dynamic variables	–	–	R1/R2/R3 Depending on language used	–	R1/R2/R3 Depending on language used	R1/R2 Depending on language used	–	–
4	Limited use of interrupts	–	–	R1/R2 Depending on language used	R1 Increases clarity of logic and event sequences	R1/R2 Depending on language used	R1/R2 Depending on language used	–	–
5	Limited use of pointers	–	–	R1/R2 Depending on language used	R1 Increases clarity of logic	R1/R2 Depending on language used	R1/R2 Depending on language used	–	–
6	Limited use of recursion	–	–	R1/R2 Depending on language used	–	R1/R2 Depending on language used	R1/R2 Depending on language used	–	–

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance / Fault detection**	**Freedom from common cause failure**
7	No unstructured control flow in programs in higher level languages	–	–	R1/R2 Depending on language used	R1 Increases clarity of logic	R1/R2 Depending on language used	R1/R2 Depending on language used	–	–
8	No automatic type conversion	–	R2 Prevents rounding errors	R2 Prevents rounding errors	R1	R1	–	–	–

Table C.12 – Detailed properties – Dynamic analysis and testing

(Referenced by Table B.2)

Technique/Measure		Properties			
		Completeness of testing and verification with respect to the software design specifications	**Correctness of testing and verification with respect to the software design specifications (successful completion)**	**Repeatability**	**Precisely defined testing and verification configuration**
1	Test case execution from boundary value analysis	–	R1 (R2 if objective criteria for boundary results)	–	–
2	Test case execution from error guessing	–	R1	–	–
3	Test case execution from error seeding	–	R1	–	–
4	Test case execution from model-based test case generation	R2 The MBT process starts with requirements and facilitates early finding of errors during software design and development R3 If rigorous reasoning is applied to modelling, and TCG (Test Case Generation) is used	R2 Evaluation of results and regression test suites is a key benefit of MBT, it further facilitates understanding of consequences of specified requirements R3 If rigorous modelling approach is applied, then objective evidence of coverage is possible	R3 MBT (with TCG) aims at automatic execution of generated tests	R2 MBT is automated, testing configuration has to be precisely defined; execution of the generated tests is similar to black box testing with the possibility to be combined with source code level coverage measurement
5	Performance modelling	–	R1 (R2 if objective performance requirements)	–	–
6	Equivalence classes and input partition testing	R1 (If the input data profile is well defined and is manageably simple in structure)	R1 (If the partitions plausibly contain no non-linearities i.e. all members of a class are truly equivalent)	–	–
7	Structure-based testing	–	R1 (R2 is objective structural coverage targets)	–	–

Table C.13 – Detailed properties – Functional and black-box testing

(Referenced by Table B.3)

Technique/Measure		Properties			
		Completeness of testing, integration and validation with respect to the design specifications	**Correctness of testing, integration and validation with respect to the design specifications (successful completion)**	**Repeatability**	**Precisely defined testing, integration and validation configuration**
1	Test case execution from cause consequence diagrams	R1	R1	–	–
2	Test case execution from model-based test case generation	R2 MBT Model-based Testing is the automatic generation of efficient test cases/procedures using models of system requirements and specified functionality, it facilitates early error disclosure and understanding of consequences of specified requirements R3 If rigorous reasoning is applied to modelling, and TCG is used	R2 MBT is based on system models derived from (mainly functional/behavioural) requirements. R3 If rigorous modelling approach is applied, then objective evidence of coverage is possible	R3 MBT (with TCG) aims at automatic execution of generated tests	R2 MBT is automated, testing configuration has to be precisely defined
3	Prototyping/animation	–	R1	–	–
4	Equivalence classes and input partition testing, including boundary value analysis	R1 (If the input data profile is well defined and is manageably simple in structure)	R1 (If the partitions plausibly contain no non-linearities i.e. all members of a class are truly equivalent)	–	–
5	Process simulation	–	R1	–	R2 Gives a definition of the external environment

Table C.14 – Detailed properties – Failure analysis

(Referenced by Table B.4)

Technique/Measure		Properties						
		Completeness of functional safety assessment with respect to this standard	**Correctness of functional safety assessment with respect to the design specifications (successful completion)**	**Traceable closure of all identified issues**	**The ability to modify the functional safety assessment after change without the need for extensive re-work of the assessment**	**Repeatability**	**Timeliness**	**Precisely defined configuration**
1a	Cause consequence diagrams	R2	R2	–	–	–	–	–
1b	Event tree analysis	R2	R2	–	–	–	–	–
2	Fault tree analysis	R2	R2	–	–	–	–	–
3	Software functional failure analysis	R2	R2	–	–	–	–	–

Table C.15 – Detailed properties – Modelling

(Referenced by Table B.5)

Technique/Measure		Properties			
		Completeness of validation with respect to the software design specification	Correctness of validation with respect to the software design specification (successful completion)	Repeatability	Precisely defined validation configuration
1	Data flow diagrams	_	R1	_	_
2a	Finite state machines	R3	R3	_	_
2b	Formal methods	R3	R3	_	_
2c	Time Petri nets	_	R1	_	_
3	Performance modelling	_	R1	_	_
4	Prototyping/animation	_	R1	_	_
5	Structure diagrams	_	R1	_	_

Table C.16 – Detailed properties – Performance testing

(Referenced by Table B.6)

Technique/Measure		Properties			
		Completeness of testing and integration with respect to the design specifications	**Correctness of testing and integration with respect to the design specifications (successful completion)**	**Repeatability**	**Precisely defined testing and integration configuration**
1	Avalanche/stress testing	–	R1 (R2 if objective targets are set)	–	–
2	Response timings and memory constraints	–	R1 (R2 if objective targets are set)	–	–
3	Performance requirements	–	R1 (R2 if objective targets are set)	–	–

Table C.17 – Detailed properties – Semi-formal methods

(Referenced by Table B.7)

Technique/Measure		Properties									
		Completeness with respect to software safety requirements specification	**Correctness with respect to software safety requirements specification**	**Freedom from intrinsic design faults**	**Understandable safety requirements**	**Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software**	**Simplicity and understandability**	**Predictability of behaviour**	**Verifiable and testable design**	**Fault tolerance / Fault detection**	**Freedom from common cause failure from external events**
1	Logic/function block diagrams	R2	R2	R2	–	–	R1	R2	–	–	R1
2	Sequence diagrams	R2	R2	R2	–	–	R1	R2	–	–	R2
3	Data flow diagrams	R1	R1	R1	–	–	R1 Suitable for transaction processing	–	–	–	R1
4a	Finite state machines/state transition diagrams	R2	R2	R2	–	–	R1 Mathematically complete specification of event sequences	R2	–	–	R2
4b	Time Petri nets	R2	R2	R2	–	–	R1 Specifies real-time interactions	R2	–	–	R2
5	Entity-relationship-attribute data models	R1	R1	R1	–	–	R1	–	–	–	R1
6	Message sequence charts	R2	R2	R2	–	–	R1	R2	–	–	R2
7	Decision/truth tables	R2	R2	R2	–	–	R1 For combinatorial logic	R2	–	–	R2

Table C.18 – Properties for systematic safety integrity – Static analysis

(Referenced by Table B.8)

Technique/Measure		Properties			
		Completeness of verification with respect to the previous phase	Correctness of verification with respect to the previous phase (successful completion)	Repeatability	Precisely defined verification configuration
1	Boundary value analysis	–	R1 (R2 if objective criteria for boundary results)	–	–
2	Checklists	–	R1	–	R1
3	Control flow analysis	–	R1	–	–
4	Data flow analysis	–	R1	–	–
5	Error guessing	–	R1	–	–
6a	Formal inspections, including specific criteria	R2	R2	–	R2
6b	Walk-through (software)	R1	R1	–	R1
7	Symbolic execution	–	R2 R3 if used in the context formally defined preconditions and postconditions and performed by a tool using a mathematically rigorous algorithm	–	–
8	Design review	R2	R1 R2 (with objective criteria)	–	R2
9	Static analysis of run time error behaviour	–	R1 R3 for certain classes of error if performed by a tool using a mathematically rigorous algorithm	–	–
10	Worst-case execution time analysis	R1	R3	–	R2

Table C.19 – Detailed properties – Modular approach

(Referenced by Table B.9)

Technique/Measure		Properties							
		Completeness with respect to software safety requirements specification	Correctness with respect to software safety requirements specification	Freedom from intrinsic design faults	Simplicity and understandability	Predictability of behaviour	Verifiable and testable design	Fault tolerance / Fault detection	Freedom from common cause failure
1	Software module size limit	–	–	R1	R1	R1	R1	–	–
2	Software complexity control	–	–	R1	R1	R1	R1	–	–
3	Information hiding/encapsulation	–	–	R1	R1	R1	R1	–	–
4	Parameter number limit / fixed number of subprogram parameters	–	–	R1	R1	R1	R1	–	–
5	One entry/one exit point in subroutines and functions	–	–	R1	R1	R1	R1	–	–
6	Fully defined interface	–	–	R2	R1	R1	R1	–	–

Annex D
(normative)

Safety manual for compliant items – additional requirements for software elements

D.1 Purpose of the safety manual

D.1.1 When an element is re-used or is intended to be re-used in one or more other system developments, it is necessary to ensure that the element is accompanied by a sufficiently precise and complete description (i.e. functions, constraints and evidence), to make possible an assessment of the integrity of a specific safety function that depends wholly or partly on the element. This shall be implemented by means of a safety manual.

D.1.2 The safety manual may consist of the element supplier's documentation if this is adequate to meet the requirements of Annex D of IEC 61508-2 and of this annex. Otherwise it should be created as part of the design of the safety related system.

D.1.3 The safety manual shall define the attributes of an element, which may comprise hardware constraints and/or software of which the integrator shall be aware and take into consideration during application. In particular it forms the vehicle for informing the integrator of its properties and what the element was designed for, its behaviour and characteristics.

NOTE 1 The scope and time of delivery of the safety manual will be dependent upon who it applies to, the type of integrator, the purpose of the element and who provides and maintains it.

NOTE 2 The person or department or organization that integrates software is called the integrator.

D.2 Contents of the safety manual for a software element

D.2.1 The safety manual shall contain all the information required by IEC 61508-2 Annex D, that is relevant to the element. E.g. the hardware-related items of IEC 61508-2 Annex D are not relevant to a purely software element.

D.2.2 The element shall be identified and all necessary instructions for its use shall be available to the integrator.

NOTE For software this can be demonstrated by clearly identifying the element and demonstrating that its content is unchanged.

D.2.3 Element configuration:

a) The configuration of the software element, the software and hardware run-time environment and if necessary the configuration of the compilation / link system shall be documented in the safety manual.
b) The recommended configuration of the software element shall be documented in the safety manual and that configuration shall be used in safety application.
c) The safety manual shall include all the assumptions made on which the justification for use of the element depends.

D.2.4 The following shall be included in the safety manual:

a) Competence: The minimum degree of knowledge expected of the integrator of the element should be specified, i.e. knowledge of specific application tools.
b) Degree of reliance placed on the element: Details of any certification of the element, independent assessment performed, integrity to which the integrator may place on the

pre-existing element. This should include the integrity to which the element was designed, the standards that were followed during the design process, and any constraints passed to the integrator which shall be implemented in support of the systematic capability claimed. (depending on the functionality of the element, it is conceivable that some requirements may only be met at the integration phase of a system. In such circumstances, these requirements shall be identified for further progression by the integrator. Requirements pertaining to response times and performance are two such examples).

NOTE Unlike IEC 61508-2, IEC 61508-3 does not require software failure modes or quantitative failure rates in safety manual for compliant items, because the causes of software errors are fundamentally different from the causes of the random hardware failures of interest in IEC 61508-2 Annex D.

c) Installation instructions: Details of, or reference to, how to install the pre-existing element into the integrated system.

d) The reason for release of the element: Details of whether the pre-existing element has been subject to release to clear outstanding anomalies, or inclusion of additional functionality.

e) Outstanding anomalies: Details of all outstanding anomalies should be given, with explanation of the anomaly, how it occurs and the mechanisms that the integrator shall take to mitigate the anomaly should the particular functions be used.

f) Backward compatibility: Details of whether the element is compatible with previous releases of the sub-system, and if not, details of the process providing the upgrade path to be followed.

g) Compatibility with other systems: A pre-existing element may be dependent upon a specially developed operating system. In such circumstances, details of the version of the specially developed operating system should be detailed.

The build standard should also be specified incorporating compiler identification and version, tools used in creation of the pre-existing element (identification and version), and test pre-existing element used (again identification and version).

h) Element configuration: Details of the pre-existing element name(s) and description(s) should be given, including the version / issue / modification state.

i) Change control: The mechanism by which the integrator can initiate a change request to the producer of the software.

j) Requirements not met: It is conceivable that there may exist specific requirements that have been specified, but have not been met in the current revision of the element. In such circumstances, these requirements should be identified for the integrator to consider.

k) Design safe state: In certain circumstances, upon controlled failure of the system application, the element may revert to a design safe state. In such circumstances, the precise definition of design safe state should be specified for consideration by the integrator.

l) Interface constraints: Details of any specific constraints, in particular user interface requirements shall be identified.

m) Details of any security measures that may have been implemented against listed threats and vulnerabilities.

n) Configurable elements: details of the configuration method or methods available for the element, their use and any constraints on their use shall be provided.

D.3 Justification of claims in the safety manual for compliant items

D.3.1 All claims in the safety manual for compliant items shall be justified by adequate supporting evidence. See 7.4.9.7 of IEC 61508-2.

NOTE 1 It is essential that the claimed safety performance of an element is supported by sufficient evidence. Unsupported claims do not help establish the correctness and integrity of the safety function to which the element contributes.

NOTE 2 The supporting evidence may be derived from the element supplier's own documentation and records of the element supplier's development process, or may be created or supplemented by additional qualification activities by the developer of the safety related system or by third parties.

NOTE 3 There may be commercial or legal restrictions on the availability of the evidence (e.g. copyright or intellectual property rights). These restrictions are outside the scope of this standard.

D.3.2 The supporting evidence that justifies the claims in the safety manual for compliant items is distinct from the element safety manual.

D.3.3 Where the evidence cannot be made available to facilitate functional safety assessment, then the element is not suitable for use in E/E/PE safety-related systems.

Annex E
(informative)

Relationships between IEC 61508-2 and IEC 61508-3

The following table helps finding which clauses of IEC 61508-2 need consideration by those who are dealing with software only and which clauses can be neglected. It is well known that almost all clauses address hardware issues. Therefore this is not repeated here. Important software aspects are treated by IEC 61508-3, many software-related requirements do however also occur in IEC 61508-2, mostly overlapping IEC 61508-3 requirements. Knowledge of IEC 61508-2 is mainly needed for those software specialists who seek compatibility between hardware and software. The IEC 61508-2 requirements are grouped into the following categories:

Table E.1 – Categories of IEC 61508-2 requirements

Software	Both for users of the standard dealing with hardware and for users dealing with software.
Application software	Users dealing with software that is for solving a related safety function as such; not for operating system software or library functions.
System software	For users dealing primarily with operating system software, library functions and the like.
Hardware only	Not for those interested in software only.
Mainly hardware	Concerns software only marginally.

Table E.2 – Requirements of IEC 61508-2 for software and their typical relevance to certain types of software

IEC 61508-2 Requirement	Important to users dealing with	Remarks
7.2	Software	
7.2.3.1	Application software	
7.2.3.2 to 7.2.3.6	Software	
7.2.3.3	Hardware only	
7.3	Software	7.3.2.2 f) Hardware only
7.4	Software	
7.4.2.1 to 7.4.2.12	Software	
7.4.2.13, 7.4.2.14	Hardware only	
7.4.3.1 to 7.4.3.3	Software	
7.4.3.4	Hardware only	
7.4.4	Hardware only	
7.4.5	Hardware only	
7.4.6	Software	7.4.6.7 Hardware only
7.4.7	Software	7.4.7.1 a), b) Hardware only
7.4.8	Hardware only	
7.4.9.1 to 7.4.9.3	Software	
7.4.9.4, 7.4.9.5	Hardware only	
7.4.9.6, 7.4.9.7	Software	
7.4.10	Software	Mainly system software

IEC 61508-2 Requirement	Important to users dealing with	Remarks
7.4.11	Hardware only	
7.5	Software	
7.6	Software	
7.6.2.1 a)	Hardware	
7.6.2.4	Mainly hardware	
7.7	Software	7.7.2.3, 7.7.2.4 Mainly application software
7.8	Software	
7.9	Mainly Application software	
8	Software	
Annex A.1	Mainly hardware	
Annex A.2 and tables	Mainly hardware	Table A.10 Software
Annex A.3	Mainly hardware	Tables A.16, A.17, A.18 Contain some software aspects
Annex B, all tables	Software	
Annex C	Hardware	
Annex D	Software	D.2.3 Hardware only
Annex E	Hardware only	
Annex F	Hardware only	

Annex F
(informative)

Techniques for achieving non-interference between software elements on a single computer

F.1 Introduction

Independence of execution between software elements which are hosted on a single computer system (consisting of one or more processors together with memory and other hardware devices shared between those processors) can be achieved and demonstrated by means of a number of different methods. This annex sets out some techniques which can be used to achieve non-interference (between elements of differing systematic capability, between elements which are designed to achieve or contribute to the same safety function, or between software contributing to a safety function and non-safety related software on the same computer).

NOTE The term "independence of execution" means that elements will not adversely interfere with each other's execution behaviour such that a dangerous failure would occur. It is used to distinguish other aspects of independence which may be required between elements, in particular diversity, to meet other requirements of the standard.

F.2 Domains of behaviour

Independence of execution should be achieved and demonstrated both in the spatial and temporal domains.

Spatial: the data used by a one element shall not be changed by a another element. In particular, it shall not be changed by a non-safety related element.

Temporal: one element shall not cause another element to function incorrectly by taking too high a share of the available processor execution time, or by blocking execution of the other element by locking a shared resource of some kind.

F.3 Causal factor analysis

To demonstrate independence of execution, an analysis of the proposed design should be undertaken to identify all possible causes of execution interference between the notionally independent (non-interfering) elements in the spatial and temporal domains. The analysis should consider both normal operation and operation under failure conditions, and should include (but need not be limited to) the following:

a) shared use of random access memory;

b) shared use of peripheral devices;

c) shared use of processor time (where two or more elements are executed by a single processor);

d) communications between the elements necessary to achieve the overall design;

e) the possibility that a failure in one element (such as an overflow, or divide by zero exception, or an incorrect pointer calculation) may cause a consequent failure in other elements.

The achievement and justification of independence of execution will then have to address all these identified sources of interference.

F.4 Achieving spatial independence

Techniques for achieving and demonstrating spatial independence include the following:

a) Use of hardware memory protection between different elements, including elements of differing systematic capability.

b) Use of an operating system which permits each element to execute in its own process with its own virtual memory space, supported by hardware memory protection.

c) Use of rigorous design, source code and possibly object code analysis to demonstrate that no explicit or implicit memory references are made from between software elements which can result in data belonging to another element being overwritten (for the case where hardware memory protection is not available).

d) Software protection of the data of a higher integrity element from illegal modification by a lower integrity element.

Data should not be passed from a lower to a higher integrity element unless the higher integrity element can verify that the data is of sufficient integrity.

Where data has to be passed between elements which are required to be independent, uni-directional interfaces such as messages or pipes should be used in preference to shared memory.

NOTE Ideally the independent elements would not communicate with each other. However, where the design of the system requires that one element should send data to another element, the design of the communication mechanism should be such that neither the sending nor the receiving elements should fail or be blocked in execution if data transmission ceases or is delayed.

Any data resident on permanent storage devices such as magnetic discs shall be taken into account for spatial partitioning, in addition to transient data in random access memory. For example, file access protection implemented by an operating system could be used to prevent one element writing to data areas belonging to another element.

F.5 Achieving temporal independence

Techniques for ensuring temporal independence include

a) Deterministic scheduling methods. For example,
   - a cyclic scheduling algorithm which gives each element a defined time slice supported by worst case execution time analysis of each element to demonstrate statically that the timing requirements for each element are met;
   - time triggered architectures.

b) Strict priority based scheduling implemented by a real-time executive with a means of avoiding priority inversion.

c) Time fences which will terminate the execution of an element if it over-runs its allotted execution time or deadline (in such a case, hazard analysis shall be undertaken to show that termination of an element will not result in a dangerous failure, so this technique may be best employed for a non-safety related element).

d) An operating system which guarantees that no process can be starved of processor time, for example by means of time slicing. Such an approach may only be applicable where there are no hard real time requirements to be met by the safety related elements, and it is shown that the scheduling algorithm will not result in undue delays to any element.

Where a resource (such as a peripheral device) is shared between elements, the design shall ensure that the elements will not function incorrectly because the shared resource is locked by another element. The time required to access a shared resource shall be taken into account in determining temporal non-interference.

F.6 Requirements for supporting software

If an operating system, a real-time executive, memory management, timer management or any other such software is to be used to provide spatial or temporal independence, or both, then such software shall be of the highest systematic capability of any of the elements which are required to be independent.

NOTE It is clear that any such software represents a potential common cause of failure of the independent elements.

F.7 Independence of software modules – programming language aspects

The following Table F.1 is an informal definition of relevant terms.

Table F.1 – Module coupling – definition of terms

Term	Informal definition
Cohesion	measure of tightness of the connections between data and subprograms within one module
Coupling	measure for the tightness of connections between modules
Encapsulation	hiding of internal (private) data and subprograms from external access; term primarily used with object oriented programs
Independence	measure of decoupling of software parts; complement of coupling
Module	confined software part that performs something and that may have data of its own; `Class`, hierarchy of classes, subprogram, `unit`, `module`, `package`, ... according to programming language
Interface	well defined set of heads of subprograms that provide access to a module
Tramp data	data that is not used in the receiving module, but only transferred to another module

As a general rule, module independence is enhanced if there is loose coupling between modules and high cohesion within modules. High cohesion encourages the situation where identifiable units of functionality correspond clearly with identifiable units of implementing code, while loose module coupling promotes low interaction and thus high independence between functionally unrelated modules.

Loose module coupling usually results from achieving high cohesion within modules by putting the code and data together that are used to perform one particular function. Low cohesion results, if code and data are assembled in modules only arbitrarily, or because of some timing sequence or due to some sequence in the control flow.

Several aspects of module coupling can be distinguished, see Table F.2 below.

Table F.2 – Types of module coupling

Coupling	Definition	Explanation	Rationale	Remark
Interface coupling, encapsulation	Coupling only via a well defined set of subprograms.	Access to the module or its data only via subprograms; any change of a value of a variable, any question about the value of such a variable, or any other service required from the module is routed via a subprogram call.	The heads of the subprograms (signatures) of a module explain the available services. If any changes of a module are required, a large amount of these changes can be done within that module, without affecting other modules. Promotes loose coupling, recommended in general.	Mainly for object oriented programs, classes, hierarchies of classes, packages of libraries; not for subprograms.
Data coupling via parameter list	Data transfer only via the parameter list or the identifier of subprograms.	Access to the module or its data only via variables or objects that are indicated in the head of the subprogram; any change of a value of a variable, any question about the value of such a variable is visible.	The head of the subprograms exhibits the data or objects involved with a call of that subprogram. Promotes loose coupling, recommended in general.	Within classes of object oriented programs this principle is normally not observed. Local variables may be accessed directly. Strict adherence to that principle may also lead to tramp data. The principle should be violated to avoid this type of data.
Structure coupling	Data transfer contains more data than necessary.	More data are transferred to the receiving subprogram than necessary for performing the required function.	The superfluous data provide another module with information that it does not require for fulfilling its purpose. These data may lead to misunderstanding the cooperation between the modules. It is, however, not deprecated.	The deficiency can normally easily be corrected.
Control coupling	Coupling that exercises immediate control on the receiving module.	Data transfer that can only cause a branching reaction in the other module; in many cases characterized by transfer of a single bit.	Tighter than the couplings above, as it requires immediate action, prescribing the receiving subprogram to do something. To be handled cautiously; to be avoided, if possible. Not recommended in general.	Cannot always be avoided. May be necessary, e.g. if the completion of an action is announced, or the validity of a value.
Global coupling	Coupling via global data.	Modules can access data that are directly accessible by other modules, or one module can directly access data belonging to another module.	The heads of the subprograms do not indicate, which data are used and from where. It is difficult to understand the subprograms' functions and to predict the effects of any changes to code.	Deprecated in general. May be necessary exceptionally, e.g. to avoid tramp data. To be used only in very limited way that conforms to a clearly defined and documented coding standard.
Content coupling	Jumping directly into other modules, influencing branching goals in other modules, or accessing data in other modules directly.	Feasible in assembly language programs; not possible in all higher level languages. Can accelerate program execution and reduce coding effort.	Deprecated. One module can only be understood by understanding its connected modules as well. Makes a program extremely difficult to understand and extremely difficult to change.	In some programming languages not even possible. Can always be avoided.

Code reading or code review (see 7.9.2.12) should verify whether or not the program modules are loosely coupled. This analysis normally requires some sort of understanding of the modules' purpose and their way of working. Proper coupling can therefore be assessed only by reading the code and its documentation.

Content coupling should be avoided. Global coupling may be used only exceptionally. Control coupling and procedural coupling should be avoided. If ever possible, modules should be connected by interface coupling (encapsulation) and/or data coupling.

Annex G
(informative)

Guidance for tailoring lifecycles associated with data driven systems

G.1 Data driven – system part and application part

Many systems are written in two parts. One part provides the underlying system capability. The other part adapts the system to the specific requirements of the intended application. The application part may be written in the form of data, that configures the system part. This is termed “data driven” in this Annex.

The application specific part of the software, may be developed using a variety of programming tools and programming languages. These languages and tools may constrain the way the application program can be written.

For instance, where a programming language supports the developer/configurer in describing the functionality (e.g. the use of ladder logic for simple interlock systems), then the application software programming task is likely to be fairly simple. However, where the programming language allows the developer/configurer to describe complex application behaviour, then the application software programming task is likely to be complex. Where very simple application software is developed, detailed design may be considered as configuring rather than programming.

The degree of rigour necessary to achieve the required safety integrity is dependent upon the degree of configuration complexity available to the developer/configurer and the complexity of behaviour to be represented in the application. This is represented diagrammatically on the axes of Figure G.1.

For simplicity the axes have been further divided into classes of complexity as:

a) Variability allowed by the language:
   - fixed program;
   - limited variability (some industries view the application program as ‘data’ which is interpreted by the system part);
   - full variability (whilst not normally considered as data driven this type of system may also be used for application development and is included in this annex for completeness).

b) Ability to configure application:
   - limited;
   - full.

In reality a particular system may comprise different levels of complexity and configurability. Further, the complexity may exhibit a sliding scale along the continuum of the two axes. When attempting to tailor the software lifecycle, the relevant level of complexity should be identified and the degree of tailoring should be justified.

A description of the typical types of system for each level of complexity is given below. Guidance on suggested techniques for implementing each type of system is given in IEC 61508-7.

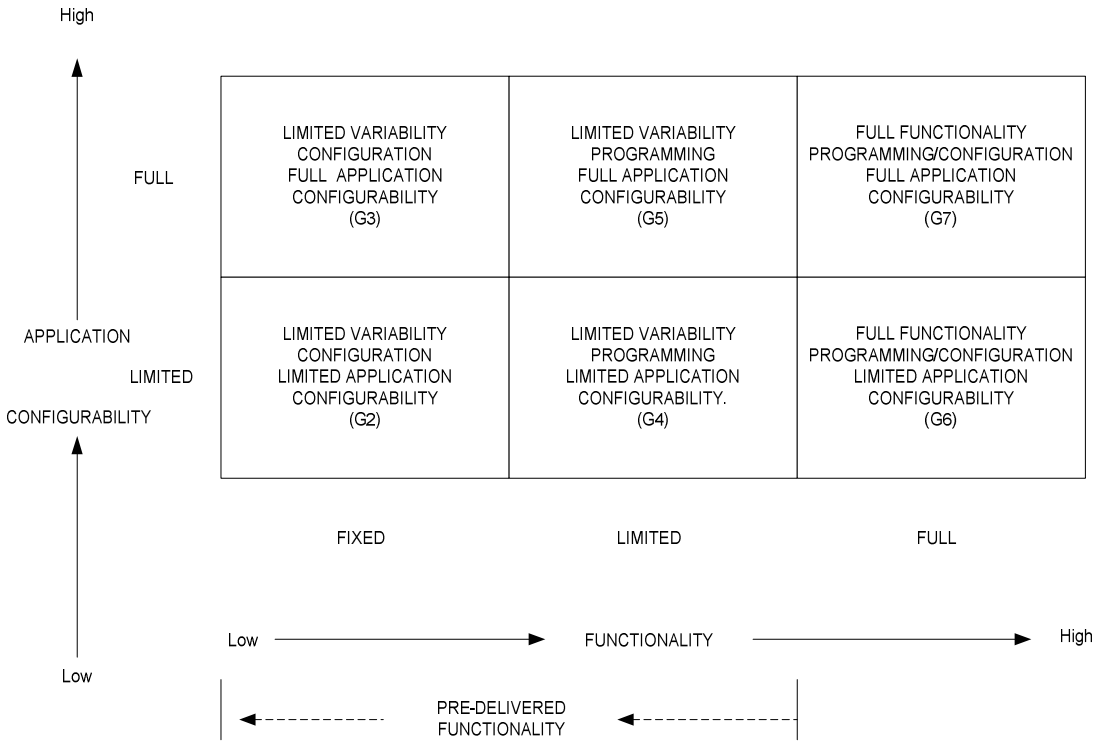

Figure G.1 – Variability in complexity of data driven systems

Typical systems in each class of complexity are described in G.2.

G.2 Limited variability configuration, limited application configurability

A proprietary configuration language used with an IEC 61508 compliant system with fixed pre-delivered functionality.

The configuration language does not allow the programmer to alter the function of the system. Instead configuration is limited to adjustment of a few (data) parameters to enable the system to be matched to its application. Examples may include smart sensors and actuators whereupon specific parameters are entered, network controllers, sequence controllers, small data logging systems and smart instruments.

The justification of the tailoring of the safety lifecycle should include, but not be limited to, the following:

a) specification of the input parameters for this application;

b) verification that the parameters have been correctly implemented in the operational system;

c) validation of all combinations of input parameters;

d) consideration of special and specific modes of operation during configuration;

e) human factors / ergonomics;

f) interlocks, e.g. ensuring that operational interlocks are not invalidated during the configuration process;

g) Inadvertent re-configuration, e.g. key switch access, protection devices.

G.3 Limited variability configuration, full application configurability

A proprietary configuration language used with an IEC 61508 compliant system with fixed pre-delivered functionality.

The configuration language does not allow the programmer to alter the function of the system. Instead, configuration is constrained to creation of extensive static data parameters to enable the system to be matched to its application. An example may be an air traffic control system consisting of data with large numbers of data entities each with one or more attributes. An essential characteristic of the data is that it contains no explicit sequencing, ordering or branching constructs in the data and does not contain any representation of the combinatorial states of the application.

In addition to the considerations given in G.2, the justification of the tailoring of the safety lifecycle should include, but not be limited to, the following:

a) automation tools for creation of data;

b) consistency checking, e.g. the data is self compatible;

c) rules checking, e.g. to ensure the generation of the data meets the defined constraints;

d) validity of interfaces with the data preparation systems.

G.4 Limited variability programming, limited application configurability

A problem-oriented language, used with an IEC 61508 compliant system, where the language statements contain or resemble the terminology of the application of the user for systems with limited pre-delivered functionality.

These languages allow the user limited flexibility to customize the functions of the system to their own specific requirements, based on a range of hardware and software elements.

An essential characteristic of limited variability programming is that data may contain explicit sequencing, ordering or branching constructs and may invoke combinatorial states of the application. Examples may include functional block programming, ladder logic, spreadsheet based systems, and graphical systems.

In addition to the considerations given in G.3, the following elements should be included, but not limited to:

a) the specification of the application requirements;

b) the permitted language sub-sets for this application;

c) the design methods for combining the language sub-sets;

d) the coverage criteria for verification addressing the combinations of potential system states.

G.5 Limited variability programming, full application configurability

A problem-oriented language, used with an IEC 61508 compliant system, where the language statements contain or resemble the terminology of the application of the user for system with limited pre-delivered functionality.

The essential difference from limited variability programming, limited application configurability is the complexity of the configuration of the application. Examples may include graphical systems and SCADA-based batch control systems.

In addition to the considerations given in G.4, the following elements should be included but not limited to:

a) the architectural design of the application;

b) the provision of templates;

c) the verification of the individual templates;

d) the verification and validation of the application.

The aspect of the lifecycle outlined in this standard which is most likely to be unnecessary (depending on the language used) is the lowest level module implementation and testing.

G.6 Full functionality programming/configuration, limited application configurability

See G.7 below.

G.7 Full functionality programming/configuration, full application configurability

For these systems the full lifecycle requirements of this standard apply.

Full variability parts of systems are based on general purpose programming languages or general purpose database languages, or general scientific and simulation packages. Typically, these parts will be used in conjunction with a computer-based system, equipped with an operating system which provides system resource allocation and a real time multi-programming environment. Examples of systems that may be written in full variability languages may include for example: a dedicated machinery control system, specially developed flight control systems, or web services for management of safety related services.

Bibliography

[1] IEC 61511 (all parts), *Functional safety – Safety instrumented systems for the process industry sector*

[2] IEC 62061, *Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems*

[3] IEC 61800-5-2, *Adjustable speed electrical power drive systems – Part 5-2: Safety requirements – Functional*

[4] IEC 61508-5: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 5: Examples of methods for the determination of safety integrity levels*

[5] IEC 61508-6: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3*

[6] IEC 61508-7: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 7: Overview of techniques and measures*

[7] IEC 60601 (all parts), *Medical electrical equipment*

[8] IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

_____

This page deliberately left blank

This page deliberately left blank

British Standards Institution (BSI)

BSI is the independent national body responsible for preparing British Standards and other standards-related publications, information and services.
It presents the UK view on standards in Europe and at the international level.

It is incorporated by Royal Charter.

Revisions

British Standards are updated by amendment or revision. Users of British Standards should make sure that they possess the latest amendments or editions.

It is the constant aim of BSI to improve the quality of our products and services. We would be grateful if anyone finding an inaccuracy or ambiguity while using this British Standard would inform the Secretary of the technical committee responsible, the identity of which can be found on the inside front cover.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001

BSI offers Members an individual updating service called PLUS which ensures that subscribers automatically receive the latest editions of standards.

Tel: +44 (0)20 8996 7669 Fax: +44 (0)20 8996 7001
Email: plus@bsigroup.com

Buying standards

You may buy PDF and hard copy versions of standards directly using a credit card from the BSI Shop on the website **www.bsigroup.com/shop.** In addition all orders for BSI, international and foreign standards publications can be addressed to BSI Customer Services.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001
Email: orders@bsigroup.com

In response to orders for international standards, it is BSI policy to supply the BSI implementation of those that have been published as British Standards, unless otherwise requested.

Information on standards

BSI provides a wide range of information on national, European and international standards through its Knowledge Centre.

Tel: +44 (0)20 8996 7004 Fax: +44 (0)20 8996 7005
Email: knowledgecentre@bsigroup.com

Various BSI electronic information services are also available which give details on all its products and services.

Tel: +44 (0)20 8996 7111 Fax: +44 (0)20 8996 7048
Email: info@bsigroup.com

BSI Subscribing Members are kept up to date with standards developments and receive substantial discounts on the purchase price of standards. For details of these and other benefits contact Membership Administration.

Tel: +44 (0)20 8996 7002 Fax: +44 (0)20 8996 7001
Email: membership@bsigroup.com

Information regarding online access to British Standards via British Standards Online can be found at **www.bsigroup.com/BSOL**

Further information about BSI is available on the BSI website at **www.bsigroup.com/standards**

Copyright

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Tel +44 (0)20 8996 9001
Fax +44 (0)20 8996 7001
www.bsigroup.com/standards

raising standards worldwide™