COMMISSION ELECTROTECHNIQUE INTERNATIONALE **CEI IEC 61508-3**

INTERNATIONAL ELECTROTECHNICAL COMMISSION

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

Part 3: Software requirements

Contents

Figures

Tables

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

Part 3: Software requirements

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC national committees). The object of the IEC is to promote international cooperation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes international standards. Their preparation is entrusted to technical committees; any IEC national committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of the IEC on technical matters, prepared by technical committees on which all the national committees having a special interest therein are represented, express, as nearly as possible, an international consensus of opinion on the subjects dealt with.

3) They have the form of recommendations for international use published in the form of standards, technical reports or guides and they are accepted by the national committees in that sense.

4) In order to promote international unification, IEC national committees undertake to apply IEC international standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC standard and the corresponding national or regional standard shall be clearly indicated in the latter.

5) Attention is drawn to the possibility that some of the elements of IEC 61508 may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

6) The IEC has not laid down any procedure concerning marking as an indication of approval and has no responsibility when an item of equipment is declared to comply with one of its standards.

IEC 61508-3 has been prepared by sub-committee 65A: System aspects, of IEC technical committee 65: Industrial process measurement and control.

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

FDIS	Report on voting
65A/xxx	65A/xxx

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

Annexes A and B form a normative part of this standard.

IEC 61508 consists of the following parts, under the general title “Functional safety of electrical/ electronic/programmable electronic safety-related systems”:

— Part 1: General requirements;

— Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems;

— Part 3: Software requirements;

— Part 4: Definitions and abbreviations;

— Part 5: Examples of methods for the determination of safety integrity levels;

— Part 6: Guidelines on the application of parts 2 and 3;

— Part 7: Overview of techniques and measures.

Introduction

Systems comprised of electrical and/or electronic components have been used for many years to perform safety functions in most application sectors. Computer-based systems (generically referred to as programmable electronic systems (PESs)) 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 those decisions.

This standard sets out a generic approach for all safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic components (electrical/electronic/ programmable electronic systems (E/E/PESs)) 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 application sector standards.

In most situations, safety is achieved by a number of protective 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 standard is concerned with electrical/electronic/programmable electronic (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 recognised that there is a great variety of E/E/PES applications in a variety of application sectors and covering a wide range of complexity, hazard and risk potentials. In any particular application, the exact prescription of safety measures will be dependent on many factors specific to the application. This standard, by being generic, will enable such a prescription to be formulated in future application sector international standards.

This standard:

— considers all relevant overall, E/E/PES and software safety lifecycle phases (for example, from initial concept, through design, implementation, operation and maintenance to decommissioning) when E/E/PESs 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 application sector international standards, dealing with safety-related E/E/PESs, to be developed – the development of 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;

— uses 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;

— adopts a risk-based approach for the determination of the safety integrity level requirements;

— sets numerical target failure measures for E/E/PE safety-related systems which are linked to the safety integrity levels;

— sets a lower limit on the target failure measures, in a dangerous mode of failure, that can be claimed for 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 failure of 10^{-5} to perform its design function on demand,

  — a high demand or continuous mode of operation, the lower limit is set at a probability of a dangerous failure of 10^{-9} per hour;

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

— adopts a broad range of principles, techniques and measures to achieve functional safety for E/E/PE safety-related systems, but does not use the concept of fail safe, which may be of value when the failure modes are well defined and the level of complexity is relatively low – the concept of fail safe was considered inappropriate because of the full range of complexity of E/E/PE safety-related systems that are within the scope of the standard.

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

Part 3: Software requirements

1 Scope

1.1 This part of this standard:

a) is intended to be utilised only after a thorough understanding of parts 1 and 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 parts 1 and 2. Such software is termed safety-related software.

— Safety-related software includes operating systems, system software, software in communication networks, human-computer interface functions, support tools and firmware as well as application programs.

— Application programs include high level programs, low level programs and special purpose programs in limited variability languages (see 3.2.7 of part 4).

c) requires that the software safety functions and software safety integrity levels 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 part 2), then it does not have to be repeated in this part.

NOTE 2 Specifying the software safety functions and software safety integrity levels is an iterative procedure – see figures 2 and 6.

NOTE 3 See clause 5 and annex A of part 1 for documentation structure. The documentation structure may take account of company procedures, and of the working practices of specific application sectors.

d) establishes requirements for safety lifecycle phases and activities that are to 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 safety integrity level, for the avoidance of and control of faults and failures in the software.

e) provides requirements for information relating to the software safety validation to be passed to the organisation carrying out the E/E/PES integration.

f) 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.

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

h) provides, in conjunction with parts 1 and 2, requirements for support tools such as development and design tools, language translators, testing and debugging tools, configuration management tools.

NOTE 4 Figures 4 and 6 show the relationship between parts 2 and 3.

1.2 Parts 1, 2, 3 and 4 of this standard are basic safety publications, although this status does not apply in the context of low complexity E/E/PE safety-related systems (see 3.4.4 of part 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 ISO/IEC Guide 104 and ISO/IEC Guide 51. One of the responsibilities of a Technical Committee is, wherever applicable, to make use of basic safety publications in the preparation of its own publications. IEC 61508 is also intended for use as a stand-alone standard.

1.3 Figure 1 shows the overall framework for parts 1 to 7 of this standard and indicates the role that part 3 plays in the achievement of functional safety for E/E/PE safety-related systems. Annex A of part 6 describes the application of parts 2 and 3.

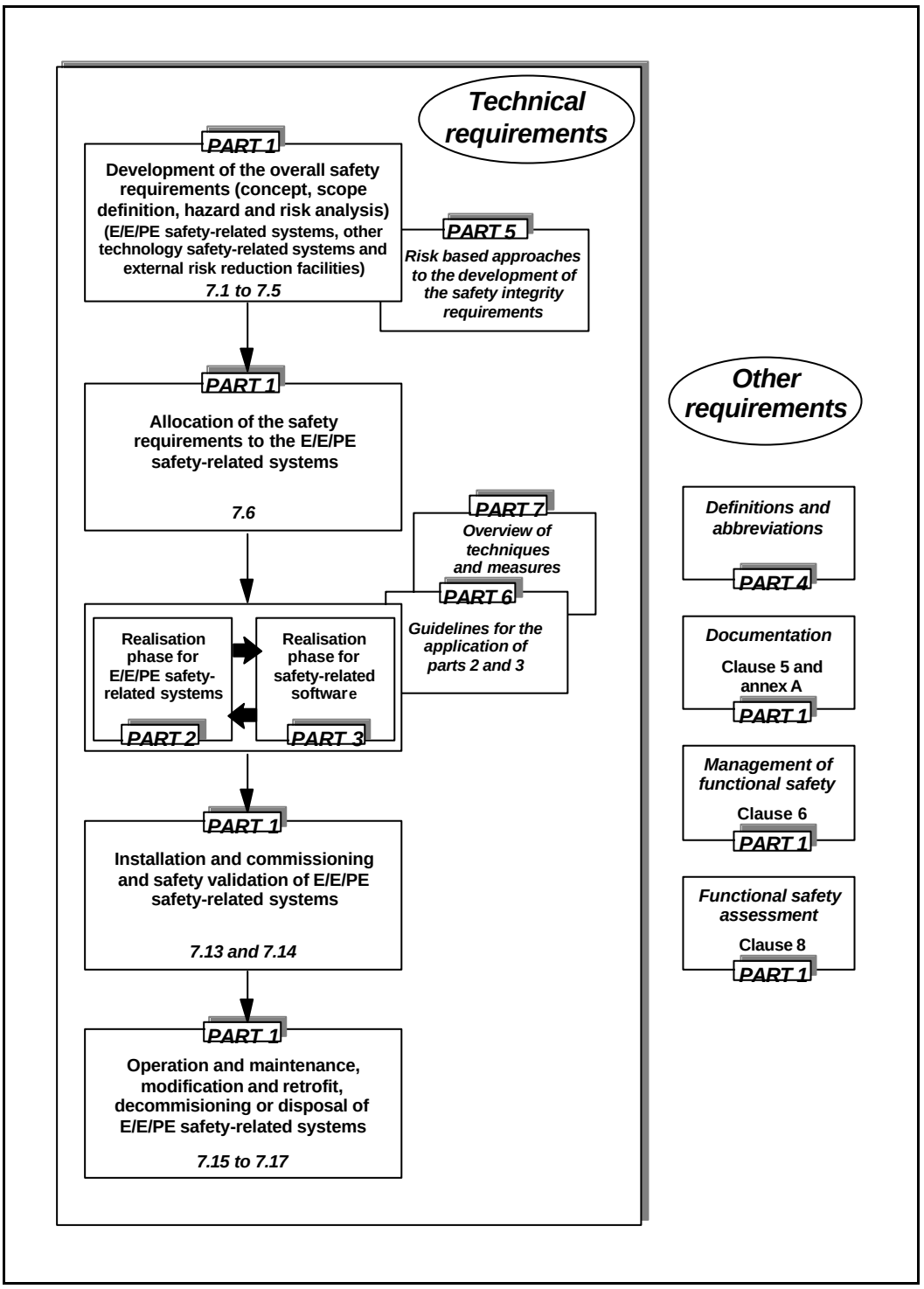

Figure 1 — Overall framework of this standard

2 Normative references

The normative references are given clause 2 of part 1.

3 Definitions and abbreviations

For the purposes of this standard, the definitions and abbreviations given in part 4 apply.

4 Conformance to this standard

The requirements for conformance to this standard are given in clause 4 of part 1.

5 Documentation

The objectives and requirements for documentation are given in clause 5 of part 1.

6 Software quality management system

6.1 Objectives

The objectives are as detailed in 6.1 of part 1.

6.2 Requirements

6.2.1 The requirements are as detailed in 6.2 of part 1 with the following additional requirements.

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

NOTE The philosophy of this approach is to use the functional safety planning as an opportunity to customise this standard to take account of the varying safety integrity which is required in the E/E/PE safety-related system components. 7.4.2.8 of part 3 should be taken into account when E/E/PE safety-related system components of differing safety integrity levels are to be used together.

6.2.3 Software configuration management should:

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 software safety continue to be satisfied.

b) guarantee that all necessary operations have been carried out to demonstrate that the required software safety integrity has been achieved.

c) maintain accurately and with unique identification all configuration items which are necessary to maintain the integrity 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; pre-existing software components 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 unauthorised 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 which justifies the baseline (see 7.8); 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.

e) document the following information to permit a subsequent audit: configuration status, release status, the justification for and approval of all modifications, and the details of the modification.

f) formally document the release of safety-related software. Master copies of the software and all associated documentation should be kept to permit maintenance and modification throughout the operational lifetime of the released software.

NOTE 2 For further information on configuration management, see ISO/IEC DIS 12207-1 "Information Technology - Software - Part 1: Software life-cycle process".

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 2 to 5).

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.

NOTE A safety lifecycle model which satisfies the requirements of clause 7 of part 1 may be suitably customised for the particular needs of the project or organisation.

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

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 1 For further information on lifecycle phases, see ISO/IEC DIS 12207-1 "Information Technology - Software - Part 1: Software life-cycle process".

NOTE 2 Clause 5 of part 1 considers the 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. In simple developments, some safety lifecycle phases may also be merged, see 7.4.5.

7.1.2.4 Provided that the software safety lifecycle satisfies the requirements of figure 3 and table 1, it is acceptable to tailor the depth, number and work-size of the phases of the V-model (see figure 5) to take account of the safety integrity and the complexity of the project.

NOTE 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.

7.1.2.5 It is acceptable to order the software project differently to the organisation of this standard (ie use another software safety lifecycle model) provided all the objectives and requirements of this clause are met.

7.1.2.6 For each lifecycle phase, appropriate techniques and measures shall be used. Annexes A and B (guide to the selection of techniques and measures) give recommendations. Selecting techniques from annexes A and B does not guarantee by itself that the required safety integrity will be achieved.

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

7.1.2.8 If at any stage of the software safety lifecycle, a change is required pertaining to an earlier lifecycle phase, then that earlier safety lifecycle phase and the following phases shall be repeated.

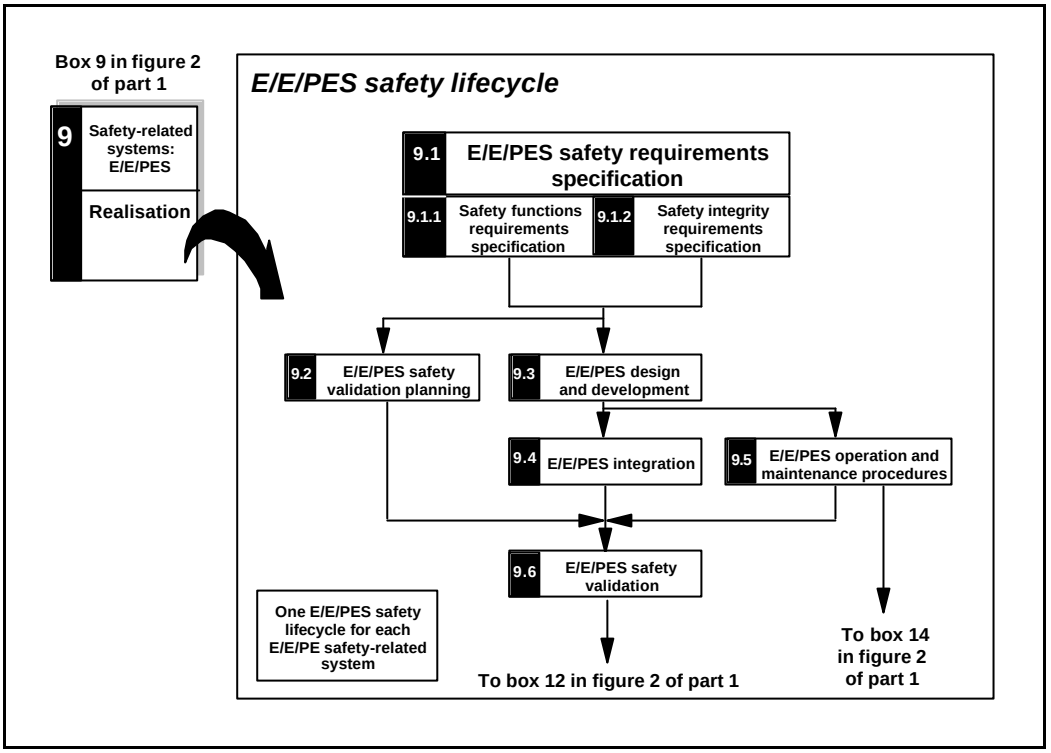

Figure 2 — E/E/PES safety lifecycle (in realisation phase)

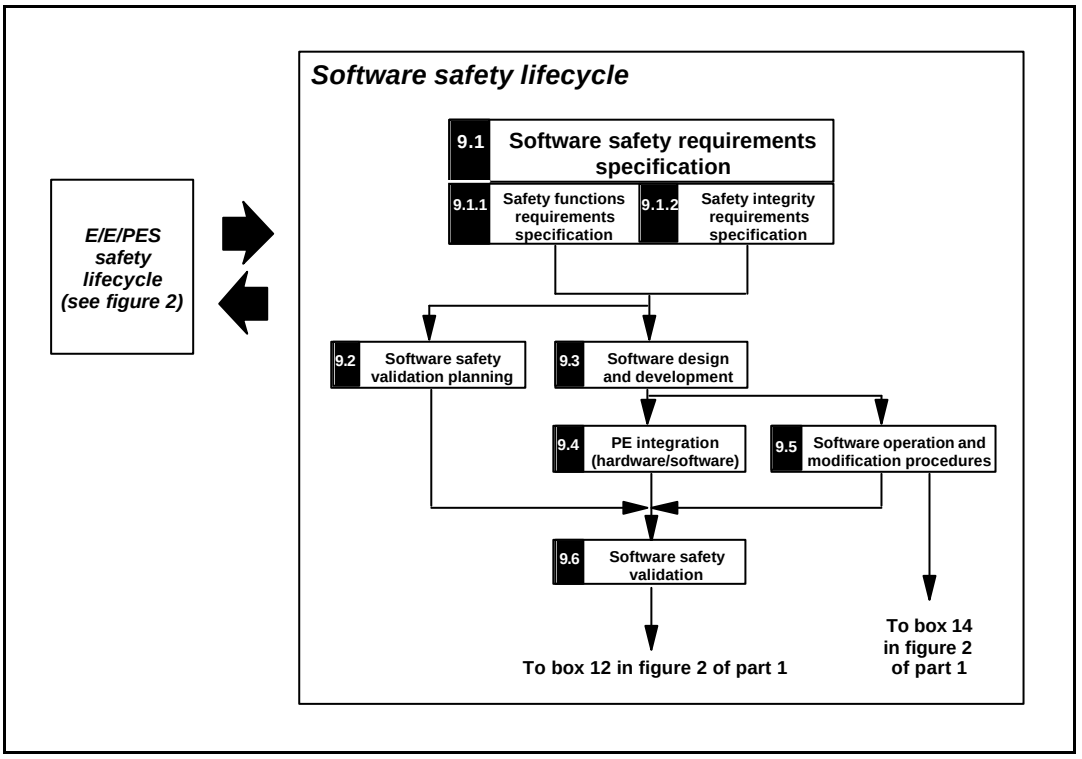

Figure 3 — Software safety lifecycle (in realisation phase)

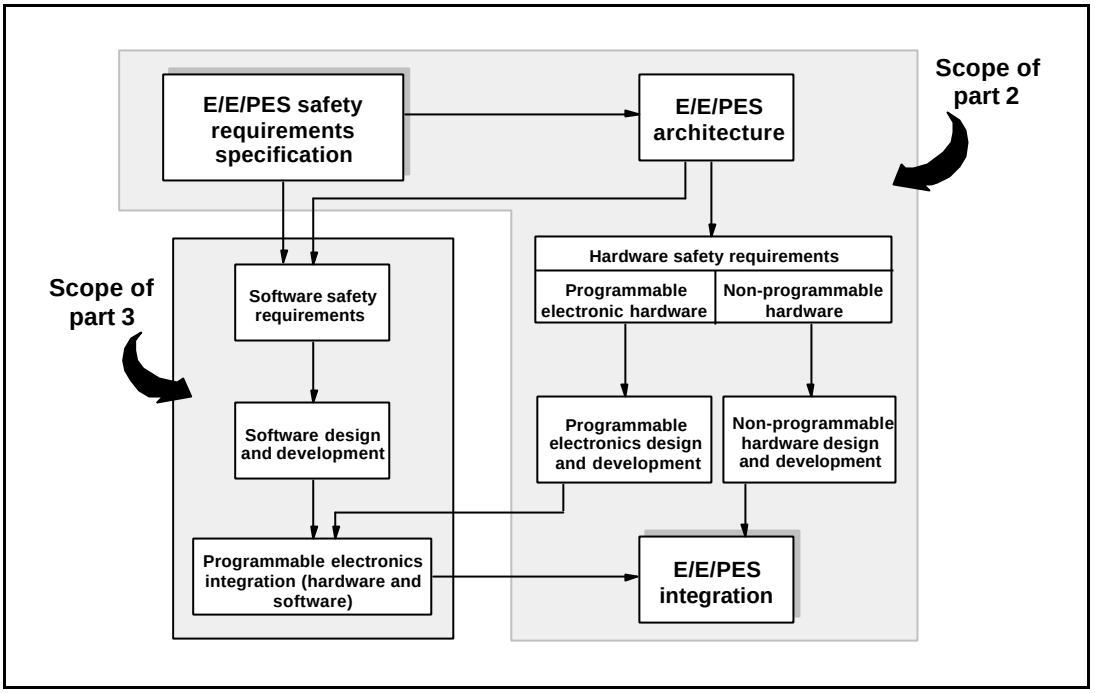

Figure 4 — Relationship and scope for parts 2 and 3

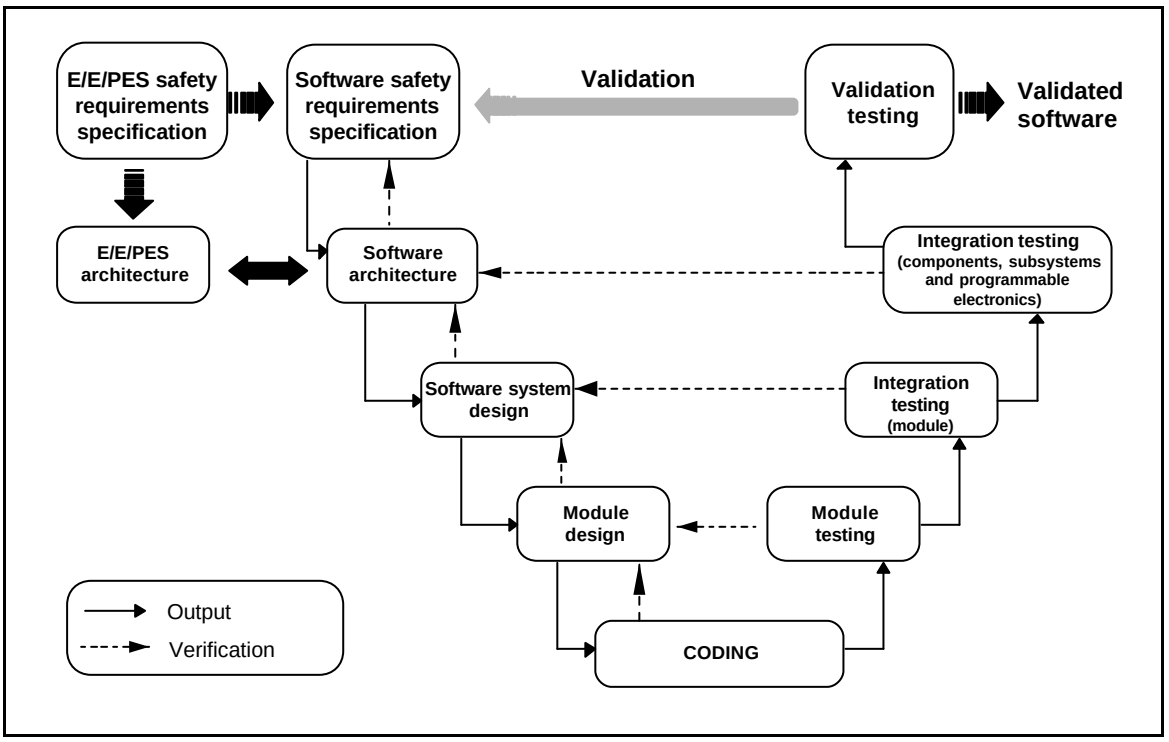

Figure 5 — Software safety integrity and the development lifecycle (the V-model)

Table 1 — Software safety lifecycle: overview

Safety lifecycle phase		Objectives	Scope	Require -ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
9.1	Software safety requirements specification	To specify the requirements for software safety in terms of (1) the requirements for software safety functions and (2) the requirements for software safety integrity; 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 safety integrity 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.	PES; Software system.	7.2.2	E/E/PES safety requirements specification (part 2).	Software safety requirements specification.
9.2	Software safety validation planning	To develop a plan for validating the software safety.	PES; Software system.	7.3.2	Software safety requirements specification.	Software safety validation plan.
9.3	Software design and development	**Architecture:** To create a software architecture that fulfils the specified requirements for software safety with respect to the required safety integrity level; To review and 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.	PES; Software system.	7.4.3	Software safety requirements specification; E/E/PES hardware architecture design (from part 2).	Software architecture design description; Software architecture integration test specification; Software/ programmable electronics integration test specification (the same as part 2 requires).
9.3	Software design and development	**Support tools and programming languages:** To select a suitable set of tools, including languages and compilers, for the required safety integrity level, over the whole safety lifecycle of the software which assists verification, validation, assessment and modification.	PES; Software system; Support tools; Programming language.	7.4.4	Software safety requirements specification; Software architecture design description.	Development tools and coding standards; Selection of development tools.

Safety lifecycle phase		Objectives	Scope	Require -ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
9.3	Software design and development	**Detailed design and development (software system design):** To design and implement software that fulfils the specified requirements for software safety with respect to the required safety integrity level, which is analysable and verifiable, and which is capable of being safely modified.	Major components and subsystems of software architectural design.	7.4.5	Software architecture design description; Support tools and coding standards.	Software system design specification; Software system integration test specification.
9.3	Software design and development	**Detailed design and development (individual software module design):** To design and implement software that fulfils the specified requirements for software safety 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.

Table 1 *(continued)*

Safety lifecycle phase		Objectives	Scope	Require -ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
9.3	Software design and development	**Detailed code implementation:** To design and implement software that fulfils the specified requirements for software safety 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.
9.3	Software design and development	**Software module testing:** To verify that the requirements for software safety (in terms of the required software safety functions and the software safety integrity) have been achieved – to show that each software module performs its intended function and does not perform unintended functions.	Software modules.	7.4.7	Software module test specification; Source Code listing; Code review report.	Software module test results; Verified and tested software modules.

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
9.3	Software design and development	**Software integration testing:** To verify that the requirements for software safety (in terms of the required software safety functions and the software safety integrity) have been achieved – to show that all software modules, components and subsystems interact correctly to perform their intended function and do not perform unintended functions.	Software architecture; Software system.	7.4.8	Software system integration test specification.	Software system integration test results. Verified and tested software system.
9.4	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.	Programmable electronics hardware; Integrated software.	7.5.2	Software architecture integration test specification; Programmable electronics integration test specification (the same as part 2 requires); Integrated programmable electronics.	Software architecture integration test results; Programmable electronics integration test results; Verified and tested integrated programmable electronics.
9.5	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.

Table 1 *(concluded)*

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
9.6	Software safety validation	To ensure that the integrated system complies with the specified requirements for software safety at the intended safety integrity level.	As above	7.7.2	Software safety validation plan.	Software safety validation results; Validated software.
9.5	Software modification	To make corrections, enhancements or adaptations to the validated software, ensuring that the required software safety integrity level is sustained.	As above	7.8.2	Software modification procedures; Software modification request.	Software modification impact analysis results; Software modification log.

Safety lifecycle phase		Objectives	Scope	Require-ments subclause	Inputs (information required)	Outputs (information produced)
Figure 3 box number	Title					
-	Software verification	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 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 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 1 See also tables A.1 and B.7.

NOTE 2 This phase is box 9.1 of figure 3.

7.2.1 Objectives

7.2.1.1 The first objective of the requirements of this subclause is to specify the requirements for software safety in terms of (1) the requirements for software safety functions and (2) the requirements for software safety integrity.

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 to specify the requirements for software safety integrity 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.

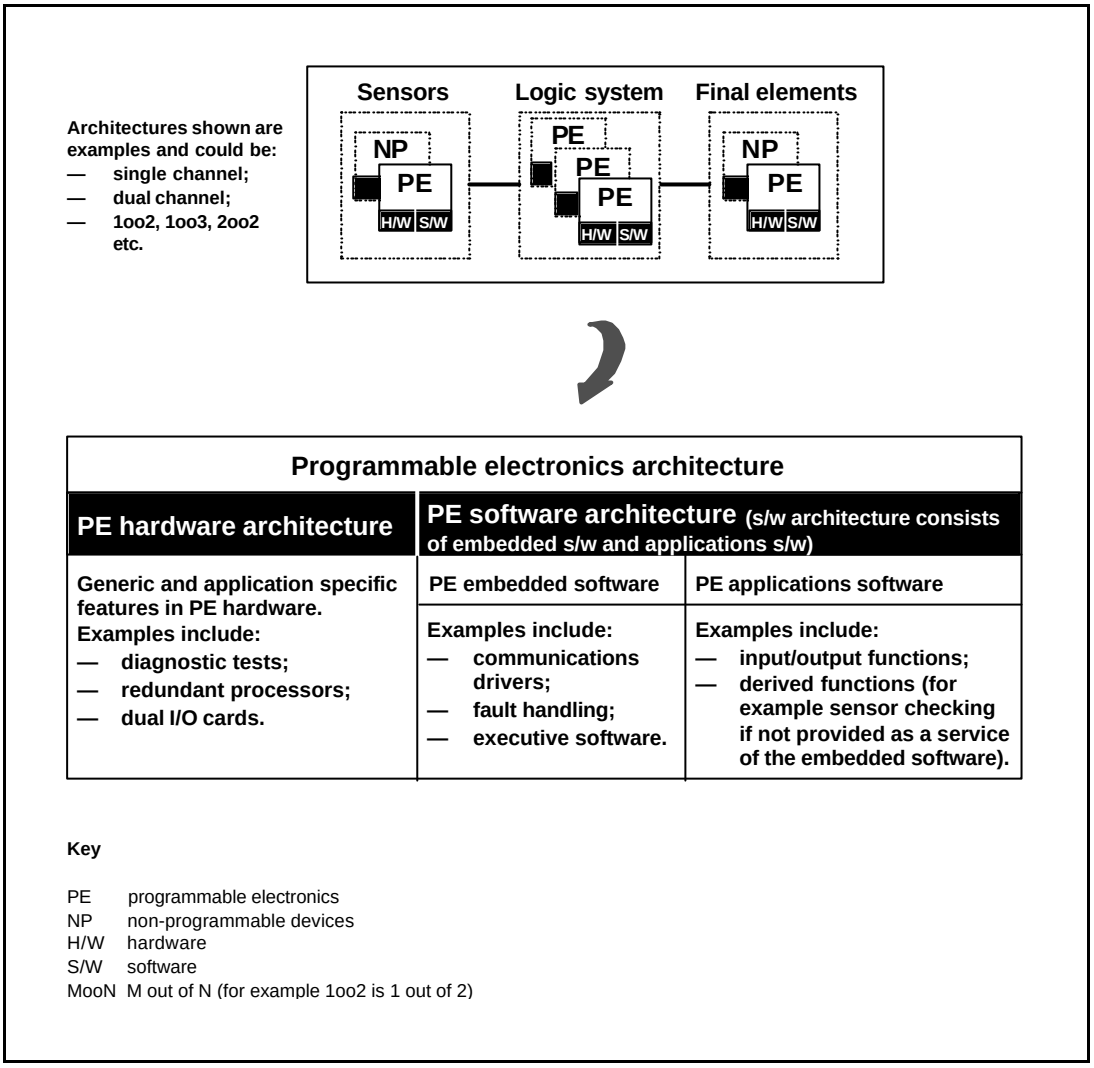

Figure 6 — Relationship between the hardware and software architectures of programmable electronics

7.2.2 Requirements

NOTE 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 is required to provide 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 and figure 6).

7.2.2.1 If the requirements for software safety have already been specified in the requirements for the E/E/PE safety-related system (see 7.2 of part 2), then the specification of software safety requirements need not be repeated.

7.2.2.2 The specification of the requirements for software safety shall be derived from (1) the specified safety requirements of the E/E/PE safety-related system (see part 2), and (2) any requirements of safety planning (see clause 6). This information shall be made available to the software developer.

NOTE This requirement does not mean that there will be no iteration between the developer of the E/E/PES and the developer of the software (parts 2 and 3). As the software safety requirements and the software architecture (see 7.4.3) become more precise, there may be an impact on the E/E/PES hardware architecture, and for this reason close co-operation between the hardware and software developer is essential. See figure 4.

7.2.2.3 The specification of the requirements for software safety shall be sufficiently detailed to allow the design and implementation to achieve the required safety integrity, 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.

7.2.2.4 The software developer shall review 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 safety integrity requirements;

e) capacity and response time performance;

f) equipment and operator interfaces.

7.2.2.5 The software developer shall establish procedures for resolving any disagreements over the assignment of the software safety integrity level.

7.2.2.6 To the extent required by the safety integrity level, the specified requirements for software safety shall be expressed and structured such that they are:

a) clear, precise, unequivocal, verifiable, testable, maintainable and feasible, commensurate with the safety integrity level;

b) traceable back to the specification of the safety requirements of the E/E/PE safety-related system;

c) free of terminology and descriptions which are ambiguous and/or not understood by those who will utilise the document at any stage of the software safety lifecycle.

7.2.2.7 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 shall be detailed in the specified requirements for software safety.

NOTE This requirement will in most cases be achieved by a combination of generic embedded software and application specific software. It is the combination of both that is required to provide the features that satisfy the requirement. The exact division between generic and application specific software depends on the chosen software architecture (see 7.4.3 and figure 6).

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

7.2.2.9 To the extent required by the description of the E/E/PE hardware architecture design, the software safety requirements specification shall consider the following:

a) software self-monitoring (for examples see C.2.5 and C.3.10 of part 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.

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

7.2.2.11 The software safety requirements specification shall express the required safety properties of the product, but not of the project. With reference to 7.2.2.1 to 7.2.2.10, the following shall be specified as appropriate:

a) the requirements for the software safety functions:

- functions that enable the EUC to achieve or maintain a safe state,
- functions related to the detection, annunciation and management of faults in the programmable electronics hardware,
- functions related to the detection, annunciation and management of sensor and actuators faults,
- functions related to the detection, annunciation and management of faults in the software itself (software self-monitoring),
- functions related to the periodic testing of safety functions on-line,
- functions related to the periodic testing of safety functions off-line,
- functions that allow the PES to be safely modified,
- interfaces to non safety-related functions,
- capacity and response time performance,
- interfaces between the software and the PES.

NOTE 1 Interfaces include both off-line and online programming facilities.

b) the requirement for the software safety integrity:

- the safety integrity level(s) for each of the functions in a) above.

NOTE 2 See annex A of part 5 for information concerning the allocation of safety integrity to software components.

7.3 Software safety validation planning

NOTE This phase is box 9.2 of figure 3.

7.3.1 Objective

The objective of the requirements of this subclause is to develop a plan for validating the software 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 (see 7.2).

7.3.2.2 The plan for validating the software 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:

- preparation for use including setting and adjustment,
- start up; teach; automatic; manual; semi-automatic; steady state of operation,
- re-setting; shut down; maintenance,

— reasonably foreseeable abnormal conditions;

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. (see 7.3.2.3);

f) In accordance with (e), the measures (techniques) and procedures that shall be used for confirming that each safety function conforms with (1) the specified requirements for the software safety functions (see 7.2), and (2) the specified requirements for software safety integrity (see 7.2);

g) specific reference to the specified requirements for software safety (see 7.2);

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

i) the pass/fail criteria (see 7.3.2.5);

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

NOTE These requirements are based on the general requirements of 7.8 of part 1.

7.3.2.3 The technical strategy for the validation of safety-related software (see table A.7 — software safety validation) 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.

7.3.2.4 As part of the procedure for validating safety-related software, the scope and contents of the planning for validating the software safety shall be reviewed with the assessor or with a party representing the assessor, if required by the safety integrity level (see 8.2.12 of part 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 9.3 of figure 3.

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 software safety (see 7.2) with respect to the required safety integrity level.

7.4.1.2 The second objective of the requirements of this subclause is to review and 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, 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 software safety (see 7.2) 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 software safety (in terms of the required software safety functions and the software safety integrity) have been achieved.

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 alone, or with the user alone, or with both. The division of responsibility shall be determined during safety planning (see clause 6).

7.4.2.2 In accordance with the required safety integrity level, the design method chosen shall possess features that facilitate:

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

b) the expression of:

- functionality,
- information flow between components,
- sequencing and time related information,
- timing constraints,
- concurrency,
- data structures and their properties,
- design assumptions and their dependencies;

c) comprehension by developers and others who need to understand the design;

d) verification and validation.

NOTE See also the tables in annexes A and B.

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.

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 minimise the safety-related part of the software.

7.4.2.7 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 independence between the functions can be demonstrated in the design.

7.4.2.8 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. The justification for independence shall be documented.

NOTE The software safety integrity level has to be at least as high as the safety integrity level of the entity (system, subsystem or component), to which it belongs. However the safety integrity level of a software component can be lower than the safety integrity level of the subsystem or system to which the component belongs, if the component is used in combination with other components such that the safety integrity level of the combination at least equals that of the system or subsystem.

7.4.2.9 As far as practicable, the design shall include 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 (as set out in part 2).

7.4.2.10 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. See table A.2 (memorising executed cases) and table A.4 (defensive programming).

7.4.2.11 If standard or previously developed software is to be used as part of the design (see tables A.3 and A.4) then it shall be clearly identified. The software's suitability in satisfying the specification of requirements for software safety (see 7.2) shall be justified. Suitability shall be based upon evidence of satisfactory operation in a similar application or having been subject to the same verification and validation procedures as would be expected for any newly developed software. Constraints from the previous software environment (for example operating system and compiler dependencies) should be evaluated.

NOTE The justification may be developed during safety planning (see clause 6).

7.4.2.12 This subclause (7.4) shall, in so far as it is appropriate, apply to data including any data generation languages.

7.4.3 Requirements for software architecture

NOTE 1 See also tables A.2 and B.7.

NOTE 2 The software architecture defines the major components and subsystems of the software, how they are interconnected, and how the required attributes, particularly safety integrity, will be achieved. Examples of major software components include operating systems, databases, plant input/output subsystems, communication subsystems, application program(s), programming and diagnostic tools etc.

NOTE 3 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 4 For user application programming in limited variability languages particularly in PLCs (see annex E of part 6) the architecture is provided by the supplier as a standard feature of the PLC. The supplier would, under this standard, be required to assure the user of his products compliance 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 architecture can be seen as information that the user would use to select the PLC (or equivalent) for the application.

NOTE 5 At the other extreme in certain embedded applications using a full variability language, for example a microprocessor controlled machine, the architecture will need to be created especially for the application (or class of application) by the supplier. The user usually has no programming facilities. In these circumstances responsibility for assuring compliance with 7.4 rests with the supplier.

NOTE 6 There are systems falling in between the two types of systems mentioned in notes 4 and 5, and responsibility for compliance will therefore be shared between the user and the supplier.

NOTE 7 From a safety viewpoint, the software architecture phase is where the basic safety strategy is developed for the software.

7.4.3.1 Depending on the nature of the software development, responsibility for conformance with 7.4.3 can rest with the supplier alone, or with the user alone, or with both (see notes above). The division of responsibility shall be documented during safety planning (see clause 6).

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

a) select and justify an integrated set of techniques and measures necessary during the software safety lifecycle phases to satisfy the specification of requirements for software safety at the required safety integrity level (see 7.2). 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 components/subsystems, for each of which the following information shall be provided:

   — whether they are new, existing or proprietary;

   — whether they have been previously verified, and if yes, their verification conditions;

   — whether each subsystem/component is safety-related or not; and

   — the software safety integrity level of the subsystem/component.

c) determine all software/hardware interactions and evaluate and detail their significance.

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 specification of requirements for software safety at the required safety integrity level (see 7.2).

7.4.3.3 Any changes required to the specified safety requirements of the E/E/PE safety-related system 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 and programming languages

NOTE 1 See also table A.3.

NOTE 2 The selection of development tools will depend on the nature of the software development activities and the software architecture (see 7.4.3).

— For user application programming in a limited variability language, at low levels of safety integrity, the required tools and programming languages may be limited to the standard PLC languages, editors and loaders. Responsibility for compliance with 7.4.4 will therefore mainly rest with the supplier.

— At higher levels of safety integrity restricted subsets of the PLC language may be necessary and verification and validation tools such as code analysers, and simulators needed. Responsibility will rest with both the supplier and the user in these circumstances.

— Tools for embedded applications using a full variability language will necessarily have to be more comprehensive even at low levels of safety integrity. Responsibility for conformance with 7.4.4 will rest here mainly with the software developer. This includes the PLC supplier who would use full variability languages in providing the low variability language for user application programming.

7.4.4.1 Depending on the nature of the software development, responsibility for conformance with 7.4.4 can rest with the supplier alone, or with the user alone, or with both (see note 2 above). The division of responsibility shall be documented during safety planning (see clause 6).

7.4.4.2 A suitable set of integrated tools, including languages, compilers, configuration management tools, and when applicable automatic testing tools, shall be selected for the required safety integrity level. The availability of suitable tools (not necessarily those used during initial system development) to supply the relevant services over the whole lifetime of the E/E/PE safety-related system should be considered.

7.4.4.3 To the extent required by the safety integrity level, the programming language selected shall:

a) have a translator/compiler which has either a certificate of validation to a recognised national or international standard, or it shall be assessed to establish its fitness for purpose;

b) be completely and unambiguously defined or restricted to unambiguously defined features;

c) match the characteristics of the application;

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

e) support features that match the design method.

7.4.4.4 When 7.4.4.3 cannot be satisfied, then a justification for an alternative language used shall be documented during software architecture design description (see 7.4.3). The justification shall detail the fitness for purpose of the language, and any additional measures which address any identified shortcomings of the language.

7.4.4.5 Coding standards shall be:

a) reviewed as fit for purpose by the assessor; and

b) used for the development of all safety-related software.

7.4.4.6 The coding standards shall specify good programming practice, proscribe unsafe language features (for example, undefined language features, unstructured designs, etc) and specify procedures for source code documentation. As a minimum, the following information should be contained in the source code documentation:

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

b) description;

c) inputs and outputs; and

d) configuration management history.

7.4.5 Requirements for detailed design and development

NOTE 1 See also tables A.4, B.1, B.7 and B.9.

NOTE 2 Detailed design is defined here to mean software system design - the partitioning of the major components 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 3 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). For user application programming using a limited variability language, 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.

7.4.5.1 Depending on the nature of the software development responsibility for conformance with 7.4.5 can rest with the supplier alone, or with the user alone, or with both (see note 3 above). The division of responsibility shall be documented during safety planning (see clause 6).

7.4.5.2 The following information should be available prior to the start of detailed design: (1) the specification of requirements for software safety (see 7.2); (2) the description of the software architecture design (see 7.4.3); (3) the plan for validating the software safety (see 7.3).

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

7.4.5.4 For each major component/subsystem in the description of the software architecture design (see 7.4.3), further refinement of the design shall be based on a partitioning into software modules (ie the specification of the software system design). The design of each software module and the tests to be applied to each software module shall be specified.

NOTE For standard or previously developed software components or software modules, no design or test specification is needed if it can be shown that they fulfil the requirements of 7.4.2.11.

7.4.5.5 Appropriate software system integration tests should be specified to ensure that the software system satisfies the specified requirements for software safety at the required safety integrity level (see 7.2).

7.4.6 Requirements for code implementation

NOTE See also tables A.4, B.1, B.7 and B.9.

7.4.6.1 The source code shall:

a) be readable, understandable and testable;

b) satisfy the specified requirements for software module design (see 7.4.5);

c) satisfy the specified requirements of the coding standards (see 7.4.4);

d) satisfy all relevant requirements specified during safety planning (see clause 6).

7.4.6.2 Each module of software code should be reviewed.

NOTE Code review is a verification activity - see 7.9.

7.4.7 Requirements for software module testing

NOTE 1 See also tables A.5, B.2, B.3 and B.6.

NOTE 2 Testing that the software module correctly satisfies its test specification is a verification activity - see also 7.9. It is the combination of code review and software module testing that provides assurance that a software module satisfies its associated specification, ie it is verified.

7.4.7.1 Each software module shall be tested as specified during software design (see 7.4.5).

7.4.7.2 These tests shall show that 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 (C.5.7 of part 7) or structure based testing (C.5.8 of part 7) may be sufficient. Boundary value analysis (C.5.4 of part 7), control flow analysis (C.5.9 of part 7) or sneak circuit analysis (C.5.11 of part 7) may reduce the amount of test cases to an acceptable number. Analysable programs (C.2.7 of part 7) make the requirements easier to fulfil.

NOTE 2 Where the development uses formal methods (C.2.4 of part 7), formal proofs (C.5.13 of part 7) or assertions (C.3.3 of part 7), such tests may be reduced in scope.

NOTE 3 Statistical evidence may be used as well (see annex D of part 7).

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

7.4.7.4 The procedures for corrective action on failure of test shall be specified.

7.4.8 Requirements for software integration testing

NOTE 1 See also tables A.5, B.2, B.3 and B.6.

NOTE 2 Testing that the software is correctly integrated is a verification activity - see also 7.9.

7.4.8.1 Software integration tests shall be specified concurrently during the design and development phase.

7.4.8.2 The specified software integration tests shall specify 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; and

f) procedures for corrective action on failure of test.

7.4.8.3 The software shall be tested in accordance with the specified software integration tests. These tests shall show that all software modules and software components/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 (C.5.7 of part 7) or structure based testing (C.5.8 of part 7) may be sufficient. Boundary value analysis (C.5.4 of part 7), control flow analysis (C.5.9 of part 7) or sneak circuit analysis (C.5.11 of part 7) may reduce the amount of test cases to an acceptable number. If development is carried out leading to analysable programs (C.2.7 of part 7), the requirements are easier to fulfil.

NOTE 2 Where the development uses formal methods (C.2.4 of part 7), formal proofs (C.5.13 of part 7) or assertions (C.3.3 of part 7), such tests may be reduced in scope.

NOTE 3 Statistical evidence may be used as well (see annex D of part 7).

7.4.8.4 The results of software integration testing shall be documented, stating (1) the test results, and (2) whether the objectives and criteria of the test criteria have been met. If there is a failure, the reasons for the failure shall be documented.

7.4.8.5 During software integration, any modification or change to the software shall be subject to an impact analysis which shall determine (1) all software modules impacted, and (2) the necessary re-verification and re-design activities.

7.5 Programmable electronics integration (hardware and software)

NOTE 1 See also tables A.6, B.3 and B.6.

NOTE 2 This phase is box 9.4 of figure 3.

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

7.5.2.1 Integration tests shall be specified during the design and development phase 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/PES may be required in order to develop the integration tests.

7.5.2.2 The integration tests for programmable electronics (hardware and software) shall specify 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; and

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

7.5.2.3 The specified integration tests for programmable electronics (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 specified integration tests for programmable electronics (hardware and software) shall distinguish between the following activities:

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

b) E/E/PE integration, ie adding interfaces such as sensors and actuators; and

c) total integration of the EUC and E/E/PE safety-related system.

NOTE Items b) and c) are covered by parts 1 and 2 and are included here to put item a) in context and for completeness.

7.5.2.5 The software shall be integrated with the safety-related programmable electronic hardware in accordance with the specified integration tests for programmable electronics (hardware and software).

7.5.2.6 During the integration testing of the safety-related programmable electronics (hardware and software), any modification or change to the integrated system shall be subject to an impact analysis which shall determine (1) all software modules impacted, and (2) the necessary re-verification activities.

7.5.2.7 Test cases and their 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 (1) the test results, and (2) whether the objectives and criteria of 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 (1) all software components/modules impacted, and (2) the necessary re-verification and re-design activities.

7.6 Software operation and modification procedures

NOTE 1 See also table A.8.

NOTE 2 This phase is box 9.5 of figure 3.

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 part 2 and 7.8 of part 3.

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

7.7 Software safety validation

NOTE 1 See also tables A.7, B.3 and B.5.

NOTE 2 This phase is box 9.6 of figure 3.

7.7.1 Objective

7.7.1.1 The objective of the requirements of this subclause is to ensure that the integrated system complies with the specified requirements for software safety (see 7.2) at the intended safety integrity level.

7.7.2 Requirements

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

7.7.2.2 The validation activities shall be carried out as specified during software safety validation planning (see 7.3).

7.7.2.3 The results of software safety validation shall be documented.

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

a) a chronological record of the validation activities;

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

c) the safety function being validated (by test or analysis), along with reference to the software safety validation plan (see 7.3);

d) tools and equipment used together with calibration data;

e) the results of the validation activity;

f) discrepancies between expected and actual results.

7.7.2.5 When discrepancies occur between expected and actual results, the analysis made and the decisions taken on whether (1) to continue the validation, or (2) to issue a change request and return to an earlier part of the development lifecycle, shall be documented as part of the results of the software safety validation.

NOTE The requirements of 7.7.2.2 to 7.7.2.5 are based on the general requirements of 7.14 of part 1.

7.7.2.6 The validation of safety-related software shall meet the following requirements:

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

b) the software shall be exercised by simulation of:

— input signals present during normal operation,

— anticipated occurrences, and

— undesired conditions requiring system action;

c) the supplier and/or developer shall make available the documented results of the software safety validation and all pertinent documentation to the system developer to enable him to meet the requirements of parts 1 and 2 of this standard.

7.7.2.7 Software tool qualification requirements are as follows:

a) all equipment used for validation shall be qualified according to a specification traceable to an international standard (if available), or to a national standard (if available), or to a well recognized procedure;

b) equipment used for software validation shall be qualified appropriately and any tools used, hardware or software, shall be shown to be suitable for purpose.

NOTE In this standard, qualification is the activity which illustrates that a particular specification is met, rather than the generic conformance testing procedures which would apply to any specification.

7.7.2.8 Software validation result requirements are as follows:

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

b) test cases and their results shall be documented for subsequent analysis and independent assessment as required by the safety integrity level (see 8.2.12 of part 1);

c) the documented results of software safety validation shall state either (1) that the software has passed the validation or (2) the reasons for its failure.

7.8 Software modification

NOTE 1 See also table A.8.

NOTE 2 This phase is box 9.5 of figure 3.

7.8.1 Objective

The objective of the requirements of this subclause is to make corrections, enhancements or adaptations to the validated software, ensuring that the required software safety integrity level is sustained.

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

7.8.2 Requirements

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

NOTE 1 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 part 1).

NOTE 2 An example of a modification procedure model is shown in figure 9 of part 1.

7.8.2.2 A modification shall be initiated only on the issue of an authorised 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 change;

c) the reasons for change.

NOTE The reason for the request for the modification could arise from, for example:

— functional safety below that specified;

— 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) should 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-related systems and external risk reduction facilities.

7.8.2.6 The safety planning for the modification of safety-related software shall include the following information:

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

b) a detailed specification for the modification;

c) verification planning;

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

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; and

e) all documented information affected by the modification activity.

7.8.2.9 Information (for example a log) on the details of all modifications shall be documented. The documentation shall include the re-verification and revalidation of data and results.

NOTE 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 part 1).

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 safety integrity level.

7.9 Software verification

NOTE See also tables A.9, B.2 and B.8.

7.9.1 Objective

7.9.1.1 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 outputs and standards provided as input 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) which are verification activities in themselves. Nor does this subclause require verification in addition to software validation (see 7.7), which in this standard is the demonstration of conformance to the safety requirements specification (end-end verification). 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

7.9.2.1 The verification of software shall be planned (see 7.4) concurrently with the development, for each phase of the software safety lifecycle, and this information 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 the following:

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 should include the following:

a) identification of items to be verified;

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

c) non-conformances.

NOTE 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 should be verified. Outputs from phase N include:

a) adequacy of the specification, design description, or code in phase N for:

   — functionality,

   — safety integrity, performance and other requirements of safety planning (see clause 6),

   — readability by the development team,

   — testability for further verification, and

   — 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:

   — the tests specified in phase N, and the tests specified in the previous phase N-1,

   — the outputs within phase N.

7.9.2.7 Subject to 7.1.2.1 the following verification activities shall be performed:

a) verification of software safety requirements (see 7.9.2.8);

b) verification of software architecture (see 7.9.2.9);

c) verification of software system design (see 7.9.2.10);

d) verification of software module design (see 7.9.2.11);

e) verification of code (see 7.9.2.12);

f) data verification (see 7.9.2.13).

g) software module testing (see 7.4.7);

h) software integration testing (see 7.4.8);

i) programmable electronics integration testing (see 7.5);

j) software safety requirements testing (software validation) (see 7.7);

7.9.2.8 Software safety requirements verification: once the software safety requirements have been specified (see 7.2), and before the next phase, software design and development begins, verification shall:

a) consider whether the specified software safety requirements (see 7.2) adequately fulfil the specified E/E/PES safety requirements (see part 2) for functionality, safety integrity, performance, and any other requirements of safety planning;

b) consider whether the software safety validation planning (see 7.3) adequately fulfils the specified software safety requirements (see 7.2);

c) check for incompatibilities between the following:

— the specified software safety requirements (see 7.2), and the specified E/E/PES safety requirements (see part 2),

— the specified software safety requirements (see 7.2), and the software safety validation planning (see 7.3).

7.9.2.9 Software architecture verification: after the software architecture design has been established, verification shall:

a) consider whether the description of the software architecture design (see 7.4.3) adequately fulfils the specified software safety requirements (see 7.2);

b) consider whether the specified tests of the software architecture integration (see 7.4.3) are adequate for the description of the software architecture design (see 7.4.3);

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

— feasibility of the safety performance required,

— testability for further verification,

— readability by the development and verification team, and

— safe modification to permit further evolution;

d) check for incompatibilities between the following:

— the description of the software architecture design (see 7.4.3), and the specified software safety requirements (see 7.2),

— the description of the software architecture design (see 7.4.3), and the specified tests of the software architecture integration (see 7.4.3),

— the specified tests of the software architecture integration (see 7.4.3), and the software safety validation planning (see 7.3).

7.9.2.10 Software system design verification: after the software system design has been specified, verification shall:

a) consider whether the specified software system design (see 7.4.5) adequately fulfils the software architecture design (see 7.4.3);

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

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

— feasibility of the safety performance required,

— testability for further verification,

— readability by the development and verification team, and

— 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 the following:

— the specified software system design (see 7.4.5), and the description of the software architecture design (see 7.4.3),

— the description of the software system design (see 7.4.5), and the specified tests of the software system integration (see 7.4.5),

— the specified tests of the software system integration (see 7.4.5) and the specified tests of the architecture integration (see 7.4.3).

7.9.2.11 Software module design verification: after the design of each software module has been specified, verification shall:

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

b) consider whether the specified tests for each software module (see 7.4.5) are adequate for the specified software module design (see 7.4.5);

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

— feasibility of the safety performance required (see 7.2),

— testability for further verification,

— readability by the development and verification team, and

— safe modification to permit further evolution;

d) check for incompatibilities between:

— the specified software module design (see 7.4.5), and the specified software system design (see 7.4.5),

— (for each software module) the specified software module design (see 7.4.5), and the specified software module tests (see 7.4.5),

— the specified software module tests (see 7.4.5), and the specified tests of the software system integration (see 7.4.5).

7.9.2.12 Code verification: the source code shall be verified by static methods to ensure conformance to the specified design of the software module (see 7.4.5), the required coding standards (see 7.4.4), and the requirements of safety planning (see 7.3).

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 Data verification:

a) the data structures specified during design shall be verified for:
   - completeness,
   - self-consistency,
   - protection against alteration or corruption, and
   - consistency with the functional requirements of the data-driven system;

b) the application data shall be verified for:
   - consistency with the data structures,
   - completeness,
   - compatibility with the underlying system software (for example sequence of execution, run-time, etc), and
   - correctness of the data values;

   NOTE An example of application data is a parts program for a numerically controlled machine. System software (typically a collection of subroutines) acts as an interpreter to the application data. In other contexts this application data would be considered an application program.

c) all modifiable parameters shall be verified for protection against:
   - invalid or undefined initial values,
   - erroneous, inconsistent or unreasonable values,
   - unauthorised changes,
   - data corruption;

d) all plant interfaces and associated software (ie sensors and actuators, and off-line interfaces: see 7.2.2.11) shall be verified for:
   - detection of anticipated interface failures,
   - tolerance to anticipated interface failures;

e) all communications interfaces and associated software shall be verified for an adequate level of:
   - failure detection,
   - protection against corruption, and
   - data validation.

8 Functional safety assessment

8.1 The objective and requirements of clause 8 of part 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 8.2.12 of part 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 7.1.2.6). The assessor should also consider:

— the consistency and the complementary 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 part 7 for an overview of the specific techniques and measures listed 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 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 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.

The ranking of the techniques and measures is linked to the concept of *effectiveness* used in part 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 safety integrity it is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application. However guidance on the use of the tables by way of two worked examples are given in part 6.

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 on the interpretation of the tables for user application programming in a limited variability language is given in part 6.

Table A.1 — Software safety requirements specification (see 7.2)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Computer-aided specification tools	B.2.4	R	R	HR	HR
2a Semi-formal methods	Table B.7	R	R	HR	HR
2b Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR

a) 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.

b) The table reflects additional requirements for specifying the software safety requirements clearly and precisely.

c) 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.

Table A.2 — Software design and development: software architecture design (see 7.4.3)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Fault detection and diagnosis	C.3.1	---	R	HR	HR
2 Error detecting and correcting codes	C.3.2	R	R	R	HR
3a Failure assertion programming	C.3.3	R	R	R	HR
3b Safety bag techniques	C.3.4	---	R	R	R
3c Diverse programming	C.3.5	R	R	R	HR
3d Recovery block	C.3.6	R	R	R	R
3e Backward recovery	C.3.7	R	R	R	R
3f Forward recovery	C.3.8	R	R	R	R
3g Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h Memorising executed cases	C.3.10	---	R	R	HR
4 Graceful degradation	C.3.11	R	R	HR	HR
5 Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6 Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a Structured methods including for example, JSD, MASCOT, SADT and Yourdon.	C.2.1	HR	HR	HR	HR
7b Semi-formal methods	Table B.7	R	R	HR	HR
7c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8 Computer-aided specification tools	B.2.4	R	R	HR	HR

a) 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.

b) 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 part 2 of this standard.

Table A.3 — Software design and development: support tools and programming language (see 7.4.4)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Suitable programming language	C.4.6	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 Certificated tools	C.4.3	R	HR	HR	HR
4b Tools: increased confidence from use	C.4.4	HR	HR	HR	HR
5a Certificated translator	C.4.3	R	HR	HR	HR
5b Translator: increased confidence from use	C.4.4	HR	HR	HR	HR
6 Library of trusted/verified software modules and components	C.4.5	R	HR	HR	HR

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.

Table A.4 — Software design and development: detailed design (see 7.4.5 and 7.4.6)

(This includes software system design, software module design and coding)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1a Structured methods including for example, JSD, MASCOT, SADT and Yourdon	C.2.1	HR	HR	HR	HR
1b Semi-formal methods	Table B.7	R	HR	HR	HR
1c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	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	Table B.1	R	HR	HR	HR
6 Structured programming	C.2.7	HR	HR	HR	HR
7 Use of trusted/verified software modules and components (if available)	C.2.10 C.4.5	R	HR	HR	HR

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.

Table A.5 — Software design and development: software module testing and integration (see 7.4.7 and 7.4.8)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Probabilistic testing	C.5.1	---	R	R	HR
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	C.5.20 Table B.6	R	R	HR	HR
6 Interface testing	C.5.3	R	R	HR	HR

a) Software module and integration testing are verification activities (see table A.9).

b) A numbered technique/measure shall be selected according to the safety integrity level.

c) 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	SIL1	SIL2	SIL3	SIL4
1 Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
2 Performance testing	C.5.20 Table B.6	R	R	HR	HR

a) Programmable electronics integration is a verification activity - see table A.9.

b) A numbered technique/measure shall be selected according to the safety integrity level.

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

Table A.7 — Software safety validation (see 7.7)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Probabilistic testing	C.5.1	---	R	R	HR
2 Simulation/modelling	Table B.5	R	R	HR	HR
3 Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR

a) A numbered technique/measure shall be selected according to the safety integrity level.

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

Table A.8 — Modification (see 7.8)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
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
4 Revalidate complete system	C.5.23	---	R	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

a) A numbered technique/measure shall be selected according to the safety integrity level.

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

Table A.9 — Software verification (see 7.9)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Formal proof	C.5.13	---	R	R	HR
2 Probabilistic testing	C.5.1	---	R	R	HR
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 Software complexity metrics	C.5.14	R	R	R	R
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				

a) 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 (software module testing and integration) and table A.6 (programmable electronics integration) which are verification activities in themselves. Nor does this table require verification testing in addition to software validation (see table A.7), which in this standard is the demonstration of conformance to the safety requirements specification (end-end verification).

b) Verification crosses the boundaries of parts 1, 2 and 3 of this standard. Therefore the first verification of the safety-related system is against the earlier system level specifications.

c) 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.

d) A numbered technique/measure shall be selected according to the safety integrity level.

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

Table A.10 — Functional safety assessment (see clause 8)

Assessment/Technique	Ref	SIL1	SIL2	SIL3	SIL4
1 Checklists	B.2.5	R	R	R	R
2 Decision/truth tables	C.6.1	R	R	R	R
3 Software complexity metrics	C.5.14	R	R	R	R
4 Failure analysis	Table B.4	R	R	HR	HR
5 Common cause failure analysis of diverse software (if diverse software is actually used)	C.6.3	---	R	HR	HR
6 Reliability block diagram	C.6.5	R	R	R	R

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

Annex B
(normative)

Detailed tables

NOTE The references indicate detailed descriptions of techniques/measures in part 7 of this standard.

Table B.1 — Design and coding standards (referenced by table A.4)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Use of coding standard	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 unconditional jumps in programs in higher level languages	C.2.6.2	R	HR	HR	HR

a) Measures 2 and 3a do not need to be applied if a compiler is used (1) which ensures that sufficient memory for all dynamic variables and objects will be allocated before runtime, or (2) which inserts runtime checks for the correct online allocation of memory.

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

Table B.2 — Dynamic analysis and testing (referenced by tables A.5 and A.9)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
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 Performance modelling	C.5.20	R	R	R	HR
5 Equivalence classes and input partition testing	C.5.7	R	R	R	HR
6 Structure-based testing	C.5.8	R	R	HR	HR

a) The analysis for the test cases is at the sub-system level and is based on the specification and/or the specification and the code.

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

Table B.3 — Functional and black box testing (referenced by tables A.5, A.6 and A7)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Test case execution from cause consequence diagrams	B.6.6.2	---	---	R	R
2 Prototyping/animation	C.5.17	---	---	R	R
3 Boundary value analysis	C.5.4	R	HR	HR	HR
4 Equivalence classes and input partition testing	C.5.7	R	HR	HR	HR
5 Process simulation	C.5.18	R	R	R	R

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

b) The completeness of the simulation will depend upon the extent of the safety integrity level, complexity and application.

c) 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	SIL1	SIL2	SIL3	SIL4
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	HR	HR
3 Failure modes, effects and criticality analysis	B.6.6.4	R	R	HR	HR
4 Monte-Carlo simulation	C.6.6	R	R	R	R

a) Preliminary hazard analysis should have already taken place in order to categorise the software into the most appropriate safety integrity level.

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

Table B.5 — Modelling (referenced by table A.7)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Data flow diagrams	C.2.2	R	R	R	R
2 Finite state machines	B.2.3.2	---	R	HR	HR
3 Formal methods	C.2.4	---	R	R	HR
4 Performance modelling	C.5.20	R	HR	HR	HR
5 Time Petri nets	B.2.3.3	---	R	HR	HR
6 Prototyping/animation	C.5.17	R	R	R	R
7 Structure diagrams	C.2.3	R	R	R	HR

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

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

Table B.6 — Performance testing (referenced by tables A.5 and A.6)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
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

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	SIL1	SIL2	SIL3	SIL4
1 Logic/function block diagrams	see a) below	R	R	HR	HR
2 Sequence diagrams	see a) below	R	R	HR	HR
3 Data flow diagrams	C.2.2	R	R	R	R
4 Finite state machines/state transition diagrams	B.2.3.2	R	R	HR	HR
5 Time Petri nets	B.2.3.3	R	R	HR	HR
6 Decision/truth tables	C.6.1	R	R	HR	HR

a) Logic/function block diagrams and sequence diagrams are described in IEC 61131-3: 1993 Programmable controllers - Part 3: Programming Languages.

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

Table B.8 — Static analysis
(referenced by table A.9)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
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
6 Fagan inspections	C.5.15	---	R	R	HR
7 Sneak circuit analysis	C.5.11	---	---	R	R
8 Symbolic execution	C.5.12	R	R	HR	HR
9 Walk-throughs/design reviews	C.5.16	HR	HR	HR	HR

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

Table B.9 — Modular approach
(referenced by table A.4)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Software module size limit	C.2.9	HR	HR	HR	HR
2 Information hiding/encapsulation	C.2.8	R	HR	HR	HR
3 Parameter number limit	C.2.9	R	R	R	R
4 One entry/one exit point in subroutines and functions	C.2.9	HR	HR	HR	HR
5 Fully defined interface	C.2.9	HR	HR	HR	HR

a) For information on all these techniques except information hiding/encapsulation, see C.2.9 of part 7 (modular approach).

b) No single technique is likely to be sufficient. All appropriate techniques shall be considered.