CASS TEMPLATES FOR SOFTWARE REQUIREMENTS IN RELATION TO IEC 61508 PART 3 SAFETY FUNCTION ASSESSMENT

Version 1.0 (5128)

Report No. T6A01
Prepared for: The CASS Scheme Ltd
By: The 61508 Association

All comment or communications to:

Enquiries: enquiries@cass.uk.net
The CASS Scheme Limited
C/O The GAMBICA Association Ltd
Broadwall House
21 Broadwall
London
SE1 9PL
www.cass.uk.net

Contents List

1 Acknowledgements

The following companies have participated and contributed to the development of this document:

Company name
Analox Sensor Technology
British Energy - part of EDF Energy
Cobham Technical Services
CSE Controls
Cygnet Solutions Ltd
Deep Life Ltd
Det-Tronics
Farside Technology Research
GE Fanuc
HSE[1]
ICS Triplex
Lloyd's Register
Measurement Technology Ltd (MTL)
MIRA Ltd
Moore Industries-International, Inc
National Physical Laboratory[2]
OAC
Sira Test & Certification
The CASS Scheme Ltd[3]
Wittenstein High Integrity Systems

[1] HSE were a contributor to the meetings. HSE has not been formally consulted on this guidance, and does not formally endorse the guidance. HSE participation does not commit any HSE inspectors to accepting that CASS assessment is appropriate and sufficient evidence that software is safe. They may take into account any other factors that they consider relevant.

[2] Part of the development of this work was funded under the Joint Industry Project scheme, managed by the National Physical Laboratory for the Department for Business, Innovation and Skills. This also included funding from Evaluation International and The 61508 Association.

[3] Editorial comment only.

2 Background

Software is increasingly being used in delivering safety functions in systems. IEC 61508-3:1998 states what is required of the software used in delivering functional safety.

Currently there is no clear understanding of what is required of assessors and developers of software. CASS has already developed templates for components (known as Type 1 systems), which focuses on the hardware. There was and is a need for similar templates for software to enable industry to cost-effectively deliver software which meets IEC 61508.

3 Objective

This document is intended to be used as guidance in the consistent compilation of the evidence required by IEC 61508-3:1998, software requirements. This document provides guidance in assessing the conformity, to IEC 61508-3:1998, of an electronic/electrical/ programmable electronic system containing software that is intended to provide a safety function.

The purpose of this document is to provide a template for the assessment and acquisition of evidence for conformity to IEC 61508-3:1998: software requirements of intelligent devices. To this extent the document is also intended to provide the basis for independent verification of the software requirements for electrical / electronic / programmable electronic systems used in safety applications.

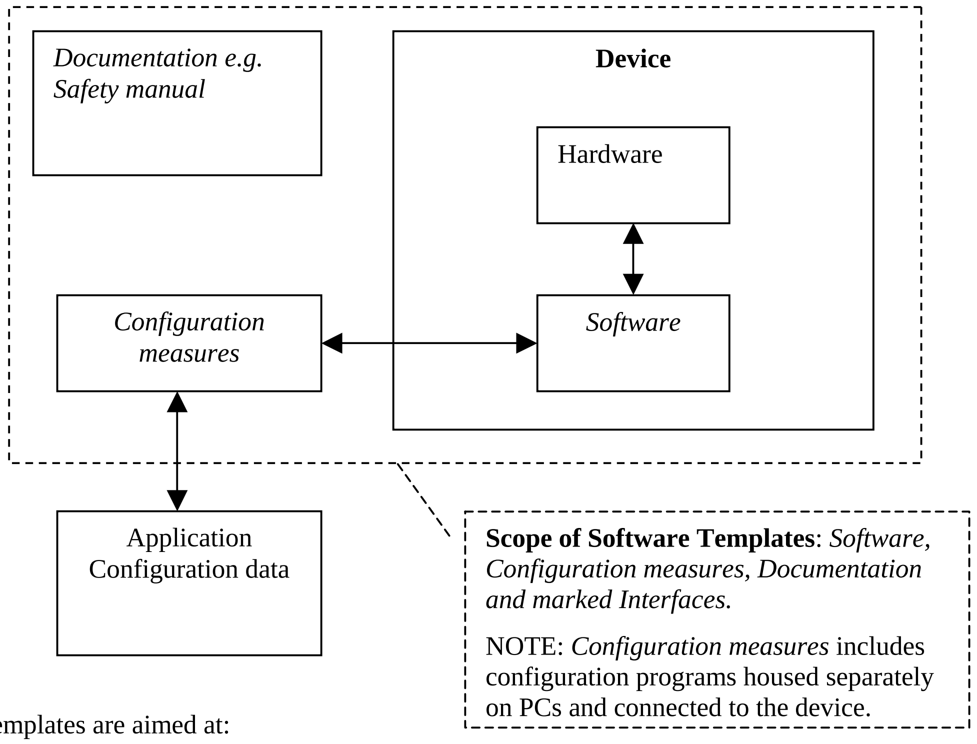
The templates are to complement the CASS Type 1 templates currently being used for hardware certification. Future extension to cover more complex situations should be borne in mind.

The CASS Scheme Ltd. offer the templates for open public comment, use, and support on the basis of achieving common understanding, common terminology, and common structuring of information.

4 Scope

4.1 Current scope

These templates cover the assessment of device-level software, such as the embedded software and configuration measures generally found in intelligent transmitters and products of similar complexity to IEC 61508-3:1998. Such devices typically allow the setting of user parameters, calibration data, etc. by the integrator/end user but not application programming which changes the algorithms. These templates will cover all of the software in the product under assessment. These templates will also cover the use of software that has been developed elsewhere such as COTS or third party software.


Documentation e.g. Safety manual
Device
Hardware
Configuration measures
Software
Application Configuration data

Scope of Software Templates: *Software, Configuration measures, Documentation and marked Interfaces.*

NOTE: *Configuration measures* includes configuration programs housed separately on PCs and connected to the device.

These templates are aimed at:

- Developers;
- Assessors; and
- Certifiers,

of software to IEC 61508.

This document addresses all of the activities specified in IEC 61508-3:1998 and separately provides tabulated templates for each of these activities. These templates are intended to provide a means of assessing whether these activities have been undertaken and the rigor with which these activities have been performed with respect to the intended SIL.

It does not replace any part of IEC 61508.

4.2 Future scope

These templates do not currently cover:

- SIL3 and SIL 4 – but there is limited guidance for SIL 3 and SIL 4

5 Terminology and Abbreviations

The following table gives expansions of abbreviations used in this document and gives definitions or descriptions of some technical terms. Where relevant, there are references to the original definitions of the terms.

Item	**Expansion/Definition/Description**	**Reference**
Baseline	The identification of a stable revision of a configuration item (or collection of configuration item), which can be subject to review and which serves as a basis of further development	
CASS	= Conformity Assessment of Safety-related Systems	http://www.cass.uk.net/
COTS	= Commercial Off-The-Shelf	[Sinclair1]
E/E/PE	= Electrical/Electronic/Programmable Electronic.	IEC 61508-4:2002, 3.2.6
E/E/PES	= Electrical/Electronic/Programmable Electronic System.	IEC 61508-4:2002, 3.3.3
Embedded software	Software that is part of the system supplied by the manufacturer and is not accessible for modification by the end-user. Embedded software is also referred to as firmware or system software.	IEC 61511-1:2004, 3.2.81.2.2
EUC	= Equipment Under Control.	IEC 61508-4:2002, 3.2.3
FPL	= Fixed program language In this type of language, the user is limited to adjustment of a few parameters (for example, range of the pressure transmitter, alarm levels, network addresses). NOTE Typical examples of devices with FPL are: smart sensor (for example, pressure transmitter), smart valve, sequence of events controller, dedicated smart alarm box, small data logging systems.	IEC 61511-1:2004, 3.2.81.1.1
HR	= Highly Recommended.	IEC 61508-3:2002, Annex A
Independent department	Department that is separate and distinct from the departments responsible for the activities which take place during the specific phase of the software safety lifecycle that is subject to the functional safety assessment or validation,	IEC 61508-4: 2002, 3.8.11 See section 6.3
Independent organization	Organisation that is separate and distinct, through management and other resources, from organisations responsible for the activities which take place during the specific phase of the software safety lifecycle that is subject to the functional safety assessment or validation	IEC 61508-4: 2002, 3.8.12 See section 6.3
Independent person	Person who is separate and distinct from the activities which take place during the specific phase of the software safety lifecycle that is subject to the functional safety assessment or validation, and does not have direct responsibility for those activities	IEC 61508-4: 2002, 3.8.10 See section 6.3

5 Terminology and Abbreviations

Item	**Expansion/Definition/Description**	**Reference**
Modes of operation	The standard describes modes of operation as 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.	IEC 61508-3:2002, 7.3.2.2 (c)
Non-functional requirements	Non-functional requirements specify the manner in which a system implements a function: "how" rather than "what". These can include requirements for: usability, maintainability, operational, performance, security, legal, cultural and political.	[SRJR1]
Process safety time	The process safety time is defined as the period of time between a failure occurring in the EUC or the EUC control system (with the potential to give rise to a hazardous event) and the occurrence of the hazardous event if the safety function is not performed.	IEC 61508-2:2002, 7.4.3.2.5
Review	A process by which an output of the software lifecycle is examined for comment and approval.	See section 6.3
SCM	Software Configuration Management.	IEC 61508-4:2002, 3.7.3, IEC 61508-7:2002, C.5.24
SIL	Safety Integrity Level.	IEC 61508-4:2002, 3.5.6
Software configuration measures	Proposed: measures taken to configure the embedded software for the intended use(s) of the device in which the software is embedded. NOTE Configuration of the software which may be carried out by the integrator or end-user, e.g. for Alarm Annunciators.	
SOUP	= Software of Unknown Pedigree	[BBF1]
TOE	= Target of Evaluation. The term is used by CASS for a specified requirement in IEC 61508. All TOEs have one or more associated IEC 61508 reference clause(s).	–

6 Introduction to the software templates

6.1 Introduction

The next section contains a set of tables of TOEs. The tables are divided into two primary groups, relating to:

- software quality management system
- software safety lifecycle requirements.

Each table of TOEs correlates to a particular management or lifecycle activity specified in the IEC 61508 part 3 software requirements. These tables are representative of the order of requirements specified in the standard.

A TOE exists for each requirement specified for each management / lifecycle activity. For each requirement covered, the TOE specifies whether the TOE is relevant at the intended SIL level and, if the TOE is relevant, what is required in evidence to verify that the requirement has been met. All of the activity related to establishing the appropriate data to use in relation to the standard is required to be undertaken by persons who are competent to undertake those activities For systems/subsystems specifically intended for safety applications, the obligations on suppliers, designers, and users related to the presentation and maintenance of this data are best considered within the context of IEC 61508 Part 1 clause 6, Management of Functional Safety.

6.2 Structure

The “Objective” section of each table stipulating the TOEs, is both definitive and prescriptive. Each TOE specifies what must be achieved in order to attain credit for compliance with IEC61508 part 3. The TOEs relate to specific clauses within IEC61508 part 3.

The “Comments” section of each table provides commentary on the TOE.

The “Guideline” section of each table provides guidance on how TOEs can be achieved. Typically, a TOE can be met by a variety of methods. Furthermore, meeting a TOE is dependent on the targeted SIL level. Higher SIL levels require more demanding and rigorous practices to be undertaken to achieve the SIL. The “Guideline” section primarily gives guidance on how SIL 1 and 2 can be achieved. There is also additional but limited guidance for SIL 3 and SIL 4.

Because the “Guideline” section provides guidance rather than stipulating requirements a less definitive narrative is used. The word ‘should’ is use instead of ‘shall’ to reflect that a developer may use a particular technique or method described in the guidance section. However, to meet the TOE, the developer must use either: the techniques and methods described in the guidance or alterative techniques and methods that meet the spirit of the IEC61508 standard. When alternative techniques and methods are employed the assessor must ensure that they are appropriate for the TOE / SIL. IEC6105 part 3 annex ‘A’ lists a set of tables that provide guidance on the selection of techniques and measures that should be employed to achieve the required SIL level.

The “Assessment requirements” section of each table states what an assessor needs to check in terms of the evidence that the TOE has been successfully met.

6.3 Reviews

[Based on IEC 61508-1 Subclauses 8.2.11, 8.2.12, 8.2.13 and Table 5.]

Reviews are required throughout the software development lifecycle and the reviewers must be competent for the activities to be undertaken. The reviewers of each document should be identified within the software safety management plan. The minimum level of independence of those carrying out the software lifecycle reviews depends on the safety integrity level and is specified in the tables below. The recommendations in the tables are as follows.

6 Introduction to the software templates

- HR: the level of independence specified is highly recommended as a minimum for the safety integrity level. If a lower level of independence is adopted then the rationale for not using the HR level should be detailed.
- NR: the level of independence specified is considered insufficient and is positively not recommended for the safety integrity level. If this level of independence is adopted then the rationale for using it should be detailed.
- –: the level of independence specified has no recommendation for or against being used.

NOTE – Depending upon the company organization and expertise within the company, the requirement for independent persons and departments may have to be met by using an external organization. Conversely, companies that have internal organizations skilled in risk assessment and the application of safety-related systems, which are independent of and separate (by ways of management and other resources) from those responsible for the main development, may be able to use their own resources to meet the requirements for an independent organization.

Table 5. Minimum levels of independence of those carrying out software safety lifecycles

Minimum level of Independence	**Safety integrity level**			
	1	**2**	**3**	**4**
Independent person	HR	HR^1	NR	NR
Independent department	–	HR^2	HR^1	NR
Independent organization (see NOTE above)	–	–	HR^2	HR

In the context of the table, either HR^1 or HR^2 is applicable (not both), depending on a number of factors specific to the application. If HR^1 is applicable then HR^2 should be read as no requirement; if HR^2 is applicable then HR^1 should be read as NR (not recommended). If no application sector standard exists, the rationale for choosing HR^1 or HR^2 should be detailed. Factors that will tend to make HR^2 more appropriate than HR^1 are

- lack of previous experience with a similar design;
- greater degree of complexity;
- greater degree of novelty of design;
- greater degree of novelty of technology;
- lack of degree of standardization of design features.

7 IEC 61508 Part 3, Software Requirements, Targets of Evaluation

7.1 TOEs for Software Quality Management

TOE	1
Title	Functional safety planning
IEC 61508 Clauses and Tables	6.2.2
Objective	For all SILs, the functional safety planning shall ensure a strategy is developed for: 1. software procurement 2. development 3. integration 4. verification 5. validation 6. modification. The strategy defined shall be indicative of the required SIL.
Comments	The functional safety plan should take account of the varying safety integrity that is required by the device. For each phase specified in the safety plan appropriate techniques and methods relevant to that SIL must be specified.
Guideline	There should be evidence that functional safety planning has been performed. Functional safety planning should ensure that the development and realization of the software product is planned and organized. This entails identifying the quality objectives for the product and creating a plan to ensure these requirements are met. Functional safety planning involves identifying the product objectives and the work products to be produced. The functional safety plan should describe the development process (a suitable engineering paradigm should be specified) that will ensure objectives are met. It should also entail identifying roles and naming the people who will fulfil these roles [HSE1]. The functional safety plan should describe the software scope. The software scope should specify the context of the software and what constraints this imposes. Where necessary, functional safety planning should decompose the problem. The problem should be partitioned appropriately, based on the functionality to be delivered and the process that will be used to deliver it.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There is a software development plan. 2. The software development plan has a revision number. 3. The software development plan is held under configuration management. 4. The software development plan has been reviewed; see section 6.3. 5. The software development plan has been approved.

7.1 TOEs for Software Quality Management

TOE 1 - Functional safety planning

	6. The software development plan specifies the development process *(either directly or by reference).* 7. The software development plan identifies the project deliverables. 8. The software development plan identifies all configuration items. 9. The software development plan identifies the verification activities. 10. The software development plan identifies the persons responsible for various project roles; see [HSE1]. 11. The software development plan specifies the scope of the software and its context. NOTE: The software development plan may be specified in the general project plan.

7.1 TOEs for Software Quality Management

TOE 2 - Software configuration management

TOE	2
Title	Software configuration management
IEC 61508 Clauses and Tables	6.2.3
Objective	For all SILs, the software configuration management system must ensure adequate administrative and technical controls exist throughout the software safety lifecycle to ensure that the changes are adequately managed, controlled, documented and that the specified safety requirements continue to exist.
Comments	Software configuration management should ensure that appropriate change control procedures exist and are employed. Software configuration management should ensure that the required safety integrity continues to be met as change is applied, this should involve performing impact analysis and developing appropriate test plans for the proposed changes. Software configuration management should ensure that an adequate change approval mechanism exists and unauthorized change is impeded. The SCM system should ensure that all configuration items configuration state and version is clearly identified. A formal documentation of releases should be applied with unique identification of the items under configuration .
Guideline	Software configuration management (SCM) is an umbrella activity that is applied throughout the software process. It is inevitable that software will change and change can occur at any time throughout the development lifecycle. The intention of SCM is to ensure that changes are identified, controlled, properly implemented and recorded. Software configuration management is an important element of software quality assurance. Its primary responsibility is the control of change. However, SCM is also responsible for the identification of individual software configuration items that constitute elements of the final software product. When a requirement for a software change evolves the change requirement should be stated and recorded. Impact analysis should then be applied before the change is implemented to ensure that the full effect of the change is understood prior to implementation. Appropriate management procedures should exist to ensure that software changes are managed and that revision control mechanisms exist for all of the software configuration items.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. A defined software configuration management system exists; use of SCM tool is expected. 2. The SCM facilitates revision control of controlled entities. 3. The SCM facilitates baselining of configuration items. 4. The SCM facilitates access control of controlled entities. 5. The SCM facilitates modification of controlled entities. 6. The SCM provides traceability of the changes made to configuration items to change requests.

7.2 TOEs for Software safety lifecycle requirements

TOE 3 - Specification of the requirements for software safety function and integrity

7.2 TOEs for Software safety lifecycle requirements

7.2.1 Software safety requirements specification

TOE	3
Title	Specification of the requirements for software safety function and integrity
IEC 61508 Clauses and Tables	7.2.1.1, 7.2.1.2, 7.2.2.2, 7.2.2.3, 7.2.2.6, 7.2.2.7, 7.2.2.8
Objective	For all safety integrity levels: 1. The requirements for the software safety shall be derived from the safety requirements of the E/E/PE safety related system and any requirements specified in the functional safety plan. 2. The specification of the requirements shall be complete and adequately detailed. 3. The requirements shall be clear, precise, unequivocal, verifiable, testable, maintainable, feasible and commensurate with the safety integrity level. 4. Free of ambiguity. 5. The requirements shall specify all modes of operation. 6. All relevant non-functional constraints imposed by the hardware shall be specified. 7. Differentiation of safety and non-safety related functions shall be considered. 8. The required safety properties relating to external interfaces shall be specified. For SIL 3, in addition to the above: 9. The use of computer aided specification tools is HR. 10. The use of semi formal methods is HR. For SIL 4, in addition to the above: 11. The use of formal methods are HR.
Comments	If the requirements for software safety function have been specified in the requirements for the E/E/PE safety related system they need not be repeated. In addition to functional and non-functional requirements, process constraints should be specified in the software safety requirements specification if they have not been specified during functional safety planning.
Guideline	There should be evidence that the software safety requirements have been specified. There should be evidence that the software safety requirements have been derived from and are traceable to requirements specified in the safety requirements specification and the functional safety plan of the E/E/PE. The requirements should unequivocally describe the problem domain of the system and the required safety functionality and integrity. The safety function requirements should fully specify the required system functionality. The safety integrity requirements should fully specify how the safety function is to be maintained and how the software handles internal and external conditions that compromise maintaining the safety function. Functional, non-functional and process constrains should be specified in full. The software requirements for the safety functionality should be fully specified. Requirements should be specified in a consistent and cohesive manner. Each requirement of the system

7.2 TOEs for Software safety lifecycle requirements

TOE 3 - Specification of the requirements for software safety function and integrity

	identified during requirements elicitation and analysis should be precisely stated once. The higher the required system's SIL the more precisely the requirements should be specified. At SIL 1 and 2, structured language may be acceptable whereas higher SILs the use of formal methods is highly recommended. The requirements should be written in a readable manner that allows them to be processed. Typically, a word-processed document is insufficient. However, using consistent language and boilerplate statements may suffice. The language used in defining the requirement should ensure a statement of fact. Formal methods provide this autonomously however requirements written in natural language will need to be stated using clear and consistent language. In this case the use of definitive words, such as 'shall' should be used. The functional and non-functional requirements of the software safety function and integrity should be specified fully along with any process constraints. Requirements should not conflict. The requirement specified should allow for traceability throughout the development and should be individually testable.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There is a software safety requirements specification. 2. The software safety requirements specification has a revision number. 3. The software safety requirements specification is held under configuration management. 4. The author(s) of the software safety requirements specification are those specified in the project plan. 5. The software safety requirements specification has been reviewed. 6. The software safety requirements specification has been approved. 7. The software safety requirements have been derived from the system requirements. A *representative sample* of software safety requirements* shall be verified to ensure that they are derived from relevant system requirements. 8. That the system requirements have been adequately detailed. A *representative sample** of system requirements that relate to software can be traced to software safety requirements. 9. A *representative sample* of the software safety requirements* shall be verified to ensure that they are precisely written, clear from ambiguity and appropriate fitness criteria can be generated for them. 10. All modes of operation of the device have been considered and that software safety requirements exist for each of these modes, and the transitions between them. 11. Non-functional requirements have been specified. These shall include requirements for: usability, maintainability, operational, performance, security, legal, cultural and political *(where appropriate).* [SRJR1]. 12. Where a device supports safety and non-safety related functions there should be clear distinction between the safety and non-safety related functions; further, there should be justification for the partitioning of the software as safety/non-safety and the should be evidence of the non-interference of the parts. 13. External interfaces are specified. This evidence shall include security, safety and protocol requirements. 14. Changes to baselined software requirements can be traced to change requests. For SIL 3, in addition to the above, there shall be evidence that: 15. The use of a computer aided specification tools and / or semi formal methods have been used. 16. The specification tool / semi formal method has been used appropriately. This evidence must provide sufficient proof that the tool / semi formal method has been used correctly and with

7.2 TOEs for Software safety lifecycle requirements

TOE 3 - Specification of the requirements for software safety function and integrity

	sufficient rigour to ensure the benefits of the tool / semi formal method are provided. For SIL 4, in addition to the above: 17. A formal method has been employed to specify the software safety function. 18. The formal method has been used appropriately and with sufficient rigour to ensure the benefits of the formal method are provided. * A *representative sample* of the safety requirements should be chosen to focus on requirements that are inherently difficult or which experience suggests often cause problems. When areas of difficulty are found, the more requirements should be sampled. Areas of requirements that should be focussed on include, but are not limited to, the following: 19. Where there are changes in the requirements or specification; 20. Timing/concurrency issues; 21. Interface issues; 22. Use of mathematical algorithms; 23. Control mechanisms (the use of control theory should be independently validated); 24. Complex logic (e.g. logic based on finite state machine formalism).

7.2 TOEs for Software safety lifecycle requirements

TOE 4 - Software safety requirements presentation and review by software developer

TOE	4
Title	Software safety requirements presentation and review by software developer
IEC 61508 Clauses and Tables	7.2.2.2, 7.2.2.4, 7.2.2.5
Objective	For all safety integrity levels: 1. The requirements shall be made available to the software developer. 2. The requirements shall be reviewed by the software developer. 3. The software developer shall resolve any disagreements over the assignment of the software safety integrity level.
Comments	When the software is reviewed by the developer particular attention must be applied to: 1. Safety functions 2. Configuration or architecture of the system 3. Hardware safety integrity levels 4. Software safety integrity requirements 5. Capacity and response time performance (non-functional requirements) 6. Equipment and operator interfaces (external interfaces)
Guideline	There should be evidence that the software safety requirements have been reviewed by the software developer(s) responsible for implementing them. There are a variety of review methods and an appropriate review technique should be employed for the project. The technique should be appropriate to the SIL level. For SIL 1 a peer review is likely to be sufficient. For higher SILs more formal review techniques should be employed e.g. Fagan inspections. The review should be planned and the reviewers specified within this plan. The results of the review should be documented along with any actions raised. The names of the developers who reviewed the safety requirements specification should correspond with those who design and develop the source code; see section 6.3. For safety products a formal review should be undertaken as a minimum.
Assessment Requirements	1. For all safety integrity levels:Ensure that there is evidence that the software safety requirements have been reviewed by the software developer. 2. If the software developer is the author of the software safety requirements, ensure that appropriate peers have reviewed the software safety requirements. 3. Ensure that there is evidence that the review evidence identifies the version of the software safety requirements specification 4. Ensure that the software developer is in agreement with the assignment of the software safety integrity level.

7.2 TOEs for Software safety lifecycle requirements

TOE 5 - Software safety requirements traceability

TOE	5
Title	Software safety requirements traceability
IEC 61508 Clauses and Tables	7.2.2.6
Objective	For all safety integrity levels: 1. The requirements shall be traceable to the specification of the E/E/PE safety requirements.
Comments	Although not explicitly mentioned in IEC61508 a formal review of the software safety requirements specification is recommended for all SILs.
Guideline	There should be evidence that the software safety requirements are traceable. There should be evidence that software safety requirements are traceable to requirements specified in earlier (system level) requirements elicitation and specification phases. Furthermore, the requirements should be traceable through later stages of analysis, design and development. An appropriate identification and traceability mechanism should be employed such that each requirement is uniquely identified and can be traced throughout the software development. If the requirements document contains safety and non-safety requirements, the safety requirements should be marked (as 'safety'). A simple numbering system may suffice whereby each requirement is given a unique number. However, this might not be possible when using automated methods. At all phases of the development lifecycle it should be possible to trace a particular item (piece of code, design or test requirement…) back to a safety requirement within the safety requirements specification.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Each software safety requirement is uniquely identifiable. 2. Each software safety requirement has traceability to the relevant system requirements. 3. Modified or newly introduced requirements added to a baselined software requirements specification are traceable to a change request. Problems in formulating the software safety requirements may be evidenced by: 4. Difficulty in agreeing particular requirements 5. Repeated attempts at framing particular requirements

7.2 TOEs for Software safety lifecycle requirements

TOE 6 - Software safety requirements specification of safety functions and diagnostics

TOE	6
Title	Software safety requirements specification of safety functions and diagnostics
IEC 61508 Clauses and Tables	7.2.29, 7.2.2.11
Objective	For all safety integrity levels: 1. Adequate consideration of software self-monitoring, monitoring of electronic hardware, diagnostics and periodic testing shall be specified. 2. The required safety properties of the product are specified, including: functions that enable the UEC to achieve or maintain a safe state, and functions that are related to fault detection, annunciation and management.
Comments	The extent to which internal diagnostic, error detection, error annunciation, recovery and graceful degradation functions should be relevant is determined by the required SIL level. Furthermore, diagnostics is a consideration in the FMEDA of the E/E/PES.
Guideline	There should be evidence that requirements for safety functions have been specified. SIL compliant safety products are required to meet a specific safety failure fraction, which is dependent on the SIL level. To achieve the required safety failure fraction appropriate diagnostics functions will need to be incorporated into the product. Furthermore diagnostic functionality will need to be implemented to ensure that the software reliably maintains the specified safety functionality. Therefore diagnostic and safety functions need to be specified. These are likely to include: walking bit RAM checks, ROM CRC checks, hardware monitoring, watchdogs, sequence of event execution analysis, IO line monitoring, configuration data checking etc. A HAZOP [HAZOP1] should be completed by the software engineers on their top-level architecture design. From this HAZOP additional software safety requirements will appear, i.e. interrupt structure, boot up sequence.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Adequate software requirements exist for software self-monitoring, monitoring of electronic hardware, diagnostics and periodic testing (RAM check, code space checks, hardware excitation checks). The software requirements for hardware monitoring must reflect the identified demand level of the hardware. 2. Hazard analysis has been completed and additional software safety requirements recorded. 3. Where specific software diagnostics are required to obtain the requisite safety failure fraction there shall be related software safety requirements. 4. Adequate software requirements exist for maintaining safe states, the handling of fault conditions and appropriate annunciation of fault conditions.

7.2 TOEs for Software safety lifecycle requirements

TOE 7 - Validation planning

7.2.2 Software safety validation planning

TOE	7
Title	Validation planning
IEC 61508 Clauses and Tables	7.3.2.1
Objective	For all safety integrity levels: 1. Planning shall be performed, specifying the procedural and technical steps that will be used to demonstrate that the software satisfies its safety requirements
Comments	
Guideline	There should be evidence that the software validation activity has been planned and documented. The software validation plan should describe the validation strategy (not the validation methods) and the validation activity considerations. The validation plan should provide evidence that the validation activity has been determined in advance of the validation process and that the planned validation is sufficient to validate the software safety functionality and integrity. There is an IEEE standard for software test documentation [IEEE829].
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There is a software validation plan. 2. The software validation plan has a revision number. 3. The software validation plan is held under configuration management. 4. The author(s) of the software validation plan are those specified in the project plan.

7.2 TOEs for Software safety lifecycle requirements

TOE 8 - Validation considerations

TOE	8
Title	Validation considerations
IEC 61508 Clauses and Tables	7.3.2.2, 7.3.2.5
Objective	For all safety integrity levels the safety validation plan shall consider the following: 1. When the validation shall take place 2. Who shall carry out the validation 3. The relevant modes of the operation of the device 4. Identification of the safety-related software for each mode of the device 5. The technical strategy 6. The required environment in which the validation activities are to be performed. 7. The pass / fail criteria for accomplishing software validation, including required inputs and expected outputs 8. The policies and procedures for evaluating the results of validation, particularly failures.
Comments	
Guideline	There should be evidence that the following consideration have been addressed in the validation plan: 1. When the validation is going to be performed 2. Who is going to perform the validation task and their competencies. 3. Where the validation is to take place 4. The validation strategy has been considered 5. Policies relating to the validation (corporate, industrial, legislative, health and safety etc) 6. A pass / fail criteria has been determined for the validation. The software validation plan should specify: who is going to perform the validation task and where the validation activity is to take place. As a minimum the persons conducting the validation should differ from those who were responsible for its development. Preferably the validation activity should be performed by a separate department or institution. The required degree of independence depends on the safety integrity level, see section 6.3. Where necessary details of the test environment should be specified. This may include specification of the test equipment or calibration standard of the test equipment. It may specify the tools used in testing the device. Validation considerations should also consider and describe the different operational modes of the device and provide a strategy for validating the device with respect to these modes. The different test techniques / strategies used to validate the device should be specified. These include techniques given in table A.7, for TOE 26. . A pass / fail criteria should be specified.
Assessment Requirements	For all safety integrity levels ensure that on examination of the software validation plan that the following are specified: 1. When the validation is going to be performed 2. Who is going to perform the validation task and their competencies. 3. Where the validation is to take place 4. Policies relating to the validation (corporate, industrial, legislative, health and safety etc)

7.2 TOEs for Software safety lifecycle requirements

TOE 8 - Validation considerations

	5. A pass / fail criterion has been determined for the validation.

7.2 TOEs for Software safety lifecycle requirements

TOE 9 - Validation strategy

TOE	9
Title	Validation strategy
IEC 61508 Clauses and Tables	7.3.2.3
Objective	For all safety integrity levels: 1. The technical strategy for the validation of safety related software shall include the following information: a. choice of manual or automated techniques or both, b. choice of static or dynamic techniques or both, c. choice of analytical or statistical techniques, or both.
Comments	
Guideline	The validation strategy should specify the validation techniques and methods used in validating the device. Furthermore there should be evidence that the validation strategy is adequate. There should also be evidence that the validation strategy is sufficient to cover all modes of operation.
Assessment Requirements	For all safety integrity levels ensure that on examination of the software validation plan that: 1. The validation strategy is adequate. As a minimum there shall be validation cases specified for all functional and non-functional software safety requirements specified in the software safety requirements specification. 2. The validation strategy covers all modes of operation. There shall be evidence that validation cases are specified for all identified modes of operation within the software safety requirements specification and transitions between those modes.

7.2 TOEs for Software safety lifecycle requirements

TOE 10 - Validation plan review

TOE	10
Title	Validation plan review
IEC 61508 Clauses and Tables	7.3.2.4
Objective	For all safety integrity levels: 1. The scope and content of the validation plan shall be reviewed by an independent assessor. For SIL 3 and SIL 4: 2. The review shall be performed by an independent assessor.
Comments	In accordance with 8.2.12 of part 1 the independent assessor should be: 1. SIL 1 – an independent person 2. SIL 2 – an independent department 3. SIL 3/4 – an independent organisation
Guideline	There should be evidence that the validation plan / strategy has been reviewed.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There is documented evidence that the software validation plan has been reviewed. 2. The results of the software validation plan review are held under configuration management. 3. Changes to the software validation plan have been made when specified by the review. 4. The software validation plan has been approved.

7.2 TOEs for Software safety lifecycle requirements

TOE 11 - Design method

7.2.3 Software design and development

TOE	11
Title	Design method
IEC 61508 Clauses and Tables	7.4.2.2
Objective	For all safety integrity levels: 1. The design method chosen shall possess features that facilitate: abstraction, modularity, information flow between components, sequencing and time related information, timing constraints, concurrency, data structure and their properties, design assumptions and their dependencies.
Comments	The design should elaborate: functionality, data types and relationships and system behaviour. Where non-functional requirements impact on these the design should give clear indication.
Guideline	There should be evidence that the device has been designed using an appropriate method: structured methods, semi-formal methods, and formal methods. For higher integrity levels formal methods should be employed in preference to semi-formal methods and an appropriate CASE tool should be used. Furthermore, the design should describe the functionality of the system, the data handled and contained within the system, the relationship between the data items and system behaviour. Both structured and object orientated methods fulfil this. For example: flow charts and class collaboration diagrams can be used to describe functionality, entity relationship and class association diagrams may be used to describe data relationships and state diagrams may be used to model system behaviour.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 2. There is a software design specification. 3. The software design has a revision number. 4. The software design is held under configuration management. 5. The software design has been reviewed. 6. The software design has been approved. 7. The software designers are those specified in the project plan. 8. The software design is a recognized method. 9. The software design conforms to the method used. 10. An appropriate design tool has been used. For SIL 3 and 4, in addition to the above, there shall be evidence that: 11. A CASE tool has been used. 12. The CASE tool has been used to sufficient rigour.

7.2 TOEs for Software safety lifecycle requirements

TOE 12 - Testability of design

TOE	12
Title	Testability of design
IEC 61508 Clauses and Tables	7.4.2.3
Objective	For all safety integrity levels: 1. Testability and the capacity for safe modification shall be considered during the design activities.
Comments	A highly modular design provides for safer modification. A modular design requirement is specified later.
Guideline	The design should be testable. Ideally, the device should have been designed to be tested. The design should allow functional and data items to be identified, isolated and tested. The design should allow system behaviour to be tested. If the design is traceable to the software safety requirements then it will be easier to test.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The design facilitates decomposition and testing.

7.2 TOEs for Software safety lifecycle requirements

TOE 13 - Modification of design

TOE	13
Title	Modification of design
IEC 61508 Clauses and Tables	7.4.2.4
Objective	For all safety integrity levels: 1. The design method chosen shall possess features that facilitate software modification. Such features include modularity and encapsulation.
Comments	A design that gives high cohesion and low coupling should be strived for, to obtain high modularity. Highly modular designs are easier to maintain.
Guideline	The design method should generate a design that facilitates easy modification. The design should therefore be structured and modular. Design parts should be cohesive with minimum coupling.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The software design is modular. 2. The design is maintainable.

7.2 TOEs for Software safety lifecycle requirements

TOE 14 - Safety considerations

TOE	14
Title	Safety considerations
IEC 61508 Clauses and Tables	7.4.2.6, 7.4.2.7, 7.4.2.8
Objective	For all safety integrity levels: 1. As far as practicable the design shall minimize the safety-related part of the software. 2. Where the software is to implement both safety and non-safety related functions, then all of the software shall be treated as safety related, unless adequate independence between the functions can be demonstrated in the design. 3. Where the software is to implement safety functions of different SILs, then all of the software shall be treated as belonging to the highest SIL, unless adequate independence between the safety functions of the different SILs can be shown in the design. The justification of this shall be documented.
Comments	The design should allow identification and differentiation of the safety related functions from the non-safety related functions. Minimum coupling should exist between these entities. When coupling between safety and non-safety functions cannot be shown to be adequately minimized the non-safety functions shall be regarded as an integral part of the E/E/PEs safety function.
Guideline	The design should minimise the safety-related part of the software. The smaller the safety related part the less likely it is to contain errors and the easier it should be to test. Non-safety related software should be separate from the safety-related part of the software. Where multiple safety functions are supported the design should be implemented to the rigour of the highest SIL level. This ensures that all safety functions are designed to the required maximum level.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The software design isolates the elements that relate to the safety function from the non-safety function elements. 2. The safety function elements of the design have been minimized. 3. There is (temporal or physical) separation between the safety and non-safety function elements

7.2 TOEs for Software safety lifecycle requirements

TOE 15 - Self monitoring and diagnostics

TOE	15
Title	Self monitoring and diagnostics
IEC 61508 Clauses and Tables	7.4.2.9, 7.4.2.10
Objective	For all safety integrity levels: 1. As far as practicable the design shall include software functions to execute proof tests and all diagnostic tests in order to fulfil the SIL requirement of the E/E/PE safety related system. 2. The software design shall include, commensurate with the required SIL, self-monitoring of control flow and data flow. On failure detection, appropriate actions shall be taken.
Comments	Diagnostics functions typically include: RAM tests, CRC checks on code space, hardware monitoring. Self-monitoring of control and data flow functions might include sequence of event monitoring, configuration data monitoring.
Guideline	The design of the device should include monitoring and self-diagnostic functions. These functions should ensure correct execution of the software occurs. These functions should cover both functionality and data flow. The design of the device should include monitoring functions that ensure that the required safety failure fraction of the device is met. The inclusion of software monitoring functions may be used to reduce the number of undetected dangerous faults. These self-test functions are likely to include: RAM checks, watchdogs, ROM CRC checks, IO monitoring, processor checks, external hardware monitors etc. Additionally self-test functions should monitor the integrity of the software itself and the data handled by the software. Monitoring the code space and the sequence of execution of the program are two methods of monitoring the functionality of the software. Data checks, dual data sets and critical area RAM testing are methods of monitoring the data integrity. There are many self-test methods that can be incorporated into a design. However, the extent to which self-test functionality is incorporated should be commensurate with the SIL level required and the diagnostic functionality provided by the hardware design.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that the design: 1. Includes elements for the specified self-monitoring and diagnostics functions specified in the software safety requirements specification. 2. Includes software self-monitoring e.g. watchdogs, windowed watchdogs, sequence of event monitoring, data corruption monitoring. 3. Timeliness of monitoring of safety integrity functions and detection of faults, for high and low demand systems takes into account the process safety time.

7.2 TOEs for Software safety lifecycle requirements

TOE 16 - Reuse of design and code components

TOE	16
Title	Reuse of design and code components
IEC 61508 Clauses and Tables	7.4.2.11
Objective	For all safety integrity levels: 1. If standard or previously developed software is to be used as part of the design then it shall be clearly identified. The software's suitability in satisfying the specification of requirements for software safety shall be justified.
Comments	Suitability for the reuse of components shall be based upon evidence of satisfactory operation in similar application or having been subject to the same verification and validation procedures as would be expected for newly developed software.
Guideline	Wherever possible the design should incorporate proven components, both design and code components: 1. The re-use of design components includes frameworks and patterns that have been previously used and proven. 2. Proven code components tend to be more reliable than newly written code. Ideally domain analysis should be performed during the design process. This activity should identify which parts of the design already exist in some form of component. Where components are identified that exist, have test evidence, proven in use history and can be used without modification, the design should incorporate these. These components should be incorporated into the design architecture during the design phase. It is not recommended that component software of unknown pedigree (SOUP) be incorporated into the design, as there is very likely to be no evidence of true reliability. Justification of the suitability of software to be re-used may bepartly provided by operating hours of a particular release of the software, operating with an operational profile that provides a statistically valid comparison with the intended operation. Developers should be cautious in the re-use of software in more general context than the original use. Assessors should be cautious/sceptical in accepting field-use records as justification of suitability for re-use of software.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The designer has attempted to include re-use design components. 2. Re-used code components have a proven in use record. 3. Re-used code components are held under configuration management. 4. Re-used components have interface documentation.

7.2 TOEs for Software safety lifecycle requirements

TOE 17 - Software architecture design

TOE	17
Title	Software architecture design
IEC 61508 Clauses and Tables	7.4.3.2
Objective	For all safety integrity levels: 1. The proposed software architecture shall be established by the software supplier / developer and a description of the software architecture design shall be detailed. 2. The architecture description shall include design strategies for both fault tolerance (consistent with the hardware) and fault avoidance, including where appropriate redundancy and diversity. 3. The architecture description shall be based on partitioning the design into components / subcomponents. 4. The architecture design description shall determine all software / hardware interactions and evaluate and detail their significance, including both normal and failure cases. 5. The architecture design description shall be unambiguous. 6. The architecture design description shall specify the design features used for maintaining safety integrity of all data. The following techniques may be valuable at all safety integrity levels, depending on the effects of a failure occurring in that area of the design. 7. The software design shall detail fault detection and diagnosis. [Required at SIL 3 and SIL 4] 8. The software design shall allow for graceful degradation where applicable. [SIL 3 and SIL 4] 9. Semi-formal methods shall be employed in the specification of the software architecture. [SIL 3] 10. CASE tools shall be used. [SIL 3 and SIL 4] 11. Error detection and correcting codes shall be incorporated into the design. [SIL 4] 12. Failure assertion programming shall be employed. [SIL 4] 13. Diverse programming techniques shall be employed. [SIL 4] 14. The design shall facilitate retry and fault recovery mechanisms. [SIL 4] 15. Formal methods shall be employed. [SIL 4]
Comments	
Guideline	The software design architecture should capture the software developer's solution to abstracting functionality, information representation and system behaviour from the software safety requirements. An appropriate method should be used to obtain a design architecture that is structured and cohesive. The design architecture should provide an overall blueprint for the software whilst providing increasing levels of detail that eventually facilitate coding of the design. The design architecture should enable communications between all stakeholders interested in the development of the system. It should also allow early design decisions to be highlighted that have a profound impact on the software development and the required safety function.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that the design: 1. Describes the static architecture (structure / object association / object aggregation) of the system. 2. Describes the dynamic behaviour of the system. 3. Describes the information representation.

	4. Describes how information on failures will be logged and retrieved.

7.2 TOEs for Software safety lifecycle requirements

TOE 18 - Programming tools

TOE	18
Title	Programming tools
IEC 61508 Clauses and Tables	7.4.4.2, 7.4.4.3, 7.4.4.4, 7.4.4.5, 7.4.4.6
Objective	For all safety integrity levels: 1. A suitable set of integrated tools, including languages, compilers, configuration management tools and where appropriate automatic testing tools shall be selected for the required SIL. 2. To the extent required by the SIL, the programming language selected shall: have a compiler / translator which has either a certified validation to a recognized national or international standard or it shall be assessed to establish its fitness for purpose. 3. The programming language shall match the characteristics of the application. 4. The IDE/ compiler / translator shall facilitate the detection of programming mistakes. 5. The programming language shall be appropriate for the design method. 6. The programming language or subset thereof shall be strongly typed. 7. Coding standards shall be used for the development of all safety related software. 8. The coding standards shall be reviewed as fit for purpose by the assessor. 9. As a minimum the coding standards shall specify good programming practice, proscribe unsafe language features and constructs and specify procedures for source code documentation.
Comments	When certification for the selected programming language cannot be obtained, then a justification for an alternative language used, shall be documented during software architecture design description. The justification shall detail the fitness for purpose of the language, and any additional measures that address any identified shortcomings of the language.
Guideline	A suitable set of programming tools should be used to develop the code. Typically, this will entail the use of an integrated development environment supplied by a reputable manufacturer. Certified programming tools are rare and may not be available for the system being developed. When certification does not exist there should be evidence that the programming tools are suitable and have a proven in use history of reliability. Ideally, a well-known and respected manufacturer's tools should be used. Often development engineers use third party tools such as code editors and code parsers. Credit should be given to the use of code editors that support syntax highlighting and enforce compliancy to an industry standard e.g. ANSI / MISRA. Credit should also be given for the use of code analysers such as 'lint'. Ideally, there should be evidence that the manufacturers of the programming tool have listed all known issues with the programming tool or have provided evidence of proven in use history. When there are known issues with the programming tool there should be evidence that appropriate 'work-arounds'' have been adopted by the developers. When the development tools support multiple levels of warning the maximum level should be selected for the development of safety systems. Furthermore there should be evidence that no warnings or errors are produced when the code is built. Where warnings exist there should be justification for these. When the development tool supports optimization it should be disabled. Optimizers have the habit of removing or altering the operation of code that appears to have either no function or a more efficient function. This code could be a trap or defensive programming measure.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The programming tools are fit for purpose i.e. an industry recognized tool set has been used and

TOE 18 - Programming tools

	the tools are appropriate for the target. 2. The programming tools are either certified for use in the development of safety products or have a proven in use record for reliability 3. There are appropriate work a rounds for known issues with the programming tools 4. Optimization has not been used during compilation 5. No errors occur when compiling or linking. 6. There are no warning or when warnings exist there are appropriate justifications for them

7.2 TOEs for Software safety lifecycle requirements

TOE 19 - Detailed design and development

TOE	19
Title	Detailed design and development
IEC 61508 Clauses and Tables	7.4.5.2, 7.4.5.3, 7.4.5.4
Objective	For all safety integrity levels: 1. The following information should be available prior to the start of the detailed design a. the software safety requirements specification, b. the description of the software architecture design, c. the software safety validation plan. 2. The software should be produced to achieve modularity, structure, testability and the capacity of safe modification. 3. The design of each software module and the tests to be applied to each software module shall be specified. 4. Structured methods shall be employed.
Comments	
Guideline	Prior to the development of code there should be evidence that the software safety requirements, software design and safety validation plan were available to the development engineer. These documents should be under configuration management and should therefore be dated and approved by an appropriate authority. The detailed design of the software of the safety system should be a solution that is modular and structured. The design should be appropriately decomposed (functionally / objects) to an appropriate level of abstraction that readily facilitates the generation of code. The design should produce modules / objects that have minimum coupling and maximum cohesion. There should be evidence that an appropriate design paradigm has been employed e.g. object orientated / structured methods. Credit should be given whenever a CASE tool has been used to generate the design. CASE tools automatically check the design to some level. The software should be designed to facilitate testing and easy modification. The design should lead to the creation of structured code that is easy to understand by other engineers.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that the design is: 1. Modular (decomposes into clearly distinguishable entities). 2. Structured (is layered). 3. Highly cohesive. 4. Low coupling.

7.2 TOEs for Software safety lifecycle requirements

TOE 20 - Code Implementation

TOE	20
Title	Code Implementation
IEC 61508 Clauses and Tables	7.4.6.1, 7.4.6.2
Objective	For all safety integrity levels: 1. The source code shall be readable, understandable and testable. 2. The source code shall satisfy the specified requirements for software modular design. 3. The source code shall satisfy all relevant requirements specified during safety planning. 4. Each source code module shall be reviewed.
Comments	
Guideline	The code for the device should be partitioned into modules that correspond to the logical, architectural partitions specified in the software design. As a minimum the software should be written in a manner that ensures that it is readable, understandable and facilitates testing. This demands that the code is adequately partitioned, adequately commented and written in a comprehendible style. The source code within each module should be decomposed into functions / subroutines that correspond to the architectural decomposition specified in software design. As a minimum, the level of decomposition should be commensurate with that of the architectural design; however the decomposition of the code may exceed this where required. Each software module should contain a commented header that describes the contents of the module. Each function / subroutine within the module should contain a commented header that gives a description of the functionality of the function / subroutine, parameters passed to it and parameters returned by it. The source code itself should be adequately commented and commented at an appropriate level that facilitates understanding and modification of the code. Ideally, the source code should be written in accordance with a coding standard or style guide. There are a variety of industry standard coding standards for various languages. However, many organisations adopt their own standard. Where a coding standard is specified as a process constraint in the safety requirements specification there should be evidence that this constraint has been implemented. Each source code module should be reviewed. The review should ensure that the code implements the requirements specified in safety requirements specifications and that the code is written in an appropriate manner, and corresponds to the specified coding standard. There should be evidence that the code has been approved post review and that the configuration management system indicates this. This evidence could be a change in revision number / type of the modules or archiving of the reviewed modules into a controlled / managed system. For SIL 3 and SIL 4, CASE tools shall be used in the software design (TOE 17) and to support the code implementation. For SIL 4, formal methods shall be employed in the software design (TOE 17) and the formal specification shall be used as the basis for the code implementation. Defensive programming shall be employed at all level. However, for SIL 3 and SIL 4, defensive techniques for the detection of and proper response to anomalies should be part of the specification and design, and should not confined to programming/coding.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that the code implementation: 1. Is representative of the software design 2. Has the same modularity as the design 3. Has the same decomposition as the design

TOE 20 - Code Implementation

	4. Uses the same terminology (naming conventions as the design) 5. Each module is decomposed into functions / subroutines that correspond to the architectural decomposition specified in the software design. 6. Is adequately commented. 7. Is maintainable. 8. Has been written to an appropriate coding standard 9. Has evidence that it has been reviewed against the coding standard 10. Has evidence of the inclusion defensive programming techniques 11. Has been reviewed against the design and has documented evidence.

7.2 TOEs for Software safety lifecycle requirements

TOE 21 - Software module testing

TOE	21
Title	Software module testing
IEC 61508 Clauses and Tables	7.4.7
Objective	For all safety integrity levels: 1. Each software module shall be tested as specified during software design to ensure that it performs its intended function and does not perform unintended functions. 2. Data recording and analysis shall be performed 3. Functional black-box testing shall be performed. 4. The results of module testing shall be documented. 5. The procedures for corrective action on failure of test shall be specified. For SIL 3 and SIL 4: 6. Performance testing shall be performed. 7. Interface testing shall be performed. For SIL 4: 8. Probabilistic testing shall be performed.
Comments	
Guideline	There should be evidence that each software module has been tested prior to integration of that module to the software system. Module (unit) testing focuses on a small unit of code. It is intended to ensure that the piece of code being tested functions correctly (as per the software design) before it is integrated into the complete system. Unit testing provides assurance of the operation of a piece of code prior to integration. This assurance may be difficult to attain through system testing due to the effect of the interfacing code or because of the complexity in devising appropriate system tests that provide adequate test coverage. Unit testing involves stimulation of the module's interface (inputs) and examination of the outputs with the intention of ensuring that the module functions correctly and maintains data integrity. The module should be invoked with known input values and the output values produces validated to ensure their correctness. Ideally, boundary value analysis techniques should be employed. Input values should also be selected that ensure that all independent execution paths are excited. There should be evidence that the module interface has been tested to ensure correct data flow in and out of the module. The tests implemented should provide assurance that the module functions as per the software design and that no unintended functionality occurs. The unit tests should focus on data integrity and execution path excitation. The test cases applied should be designed to uncover errors due to erroneous computations and improper control flow. An appropriate system should be in place to ensure that whenever defects are found the developers are instructed to take appropriate corrective action. This system could be an in-house procedure or the use of a bug-tracking tool. Probabilistic testing shall be performed at SIL 4 but, in practice, it is difficult to demonstrate ultra-high levels of reliability using this technique.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Module tests have been written for each software module.

7.2 TOEs for Software safety lifecycle requirements

TOE 21 - Software module testing

	2. The module test specification is held under configuration management. 3. The module test specification has been reviewed. 4. The module test specification has been approved. 5. The module tests provide adequate test coverage. 6. The module tests verify design functionality. 7. The module tests verify data integrity. 8. The module tests have been performed by the persons specified in the project development plan. 9. There are formalized results of the module testing. 10. Module testing results are held under configuration management. 11. The results of module testing indicate that the software has passed the tests. For SIL 3 and 4, in addition to the above, there shall be evidence that: 12. The module tests (where applicable) verify mathematical stability and accuracy. 13. The module tests (where applicable) verify performance and throughput. 14. Interface testing has been performed. For SIL 4 in addition to the above, there shall be evidence that: 15. Probabilistic testing has been performed.

7.2 TOEs for Software safety lifecycle requirements

TOE 22 - Software integration testing

TOE	22
Title	Software integration testing
IEC 61508 Clauses and Tables	7.4.8.1
Objective	For all safety integrity levels: 1. Software integration tests shall be specified concurrently during the design and development phases. 2. The specified integration tests shall specify the test cases and test data, types of test to be performed, the test environment, tools, configuration and programs; the test criteria on which the completion of the test will be judged; and procedures for corrective action on failure of test. 3. During software integration any modification or change to the software shall be subject to an impact analysis that shall determine a. the software modules impacted and b. the necessary re-verification and re-design activities.
Comments	Integration testing in this instance refers to the integration / unification of separate software components / modules to form the complete software solution for the safety system.
Guideline	When the software in the device comprises separate software modules that have been developed independently there should be evidence that the unification process (integration) of the modules has been tested in accordance with a software integration test specification that was written during the design and development phase. The software integration test specification should provide adequate confidence that when the tested software modules are integrated into the software system of the device they perform as per the intent of the design. Integration testing assures that when tested modules are put together they continue to function correctly and that their interfaces operate correctly .
Assessment Requirements	Where software integration testing is applicable, for all safety integrity levels ensure that there is evidence that: 1. A software integration test specification exists. 2. The software integration test specification is held under configuration management. 3. The software integration test plan has been reviewed. 4. The specified integration tests specify appropriate test cases and test data 5. The software integration test specification specifies the test environment, test tools, software configuration and requisite test programs. 6. The software integration test specification specifies a pass / fail criteria. 7. The software has passed integration testing. 8. For recorded failure cases requiring software modification an impact analysis has been performed and recorded.

7.2 TOEs for Software safety lifecycle requirements

TOE 23 - Specification of integration tests

7.2.4 Programmable electronics integration

TOE	23
Title	Specification of integration tests
IEC 61508 Clauses and Tables	7.5.2.1, 7.5.2.2, 7.5.2.3, 7.5.2.4
Objective	For all safety integrity levels: 1. Integration tests shall be specified during the design and development phase to ensure the compatibility of the hardware and the software in the device 2. The integration tests for the device shall specify the following: a. The split of the system into integration levels; b. Test cases and test data; c. Types of test to be performed; d. Test environment including tools, support software and configuration description; e. Test criteria on which the completion of the test will be judged. 3. The integration tests specified shall distinguish between those activities that can be carried out by the developer on his premises and those that access the user's site. 4. The specified integration tests for programmable electronics shall distinguish between the following activities: a. merging of software system on the target programmable electronic hardware; b. E/E/PE integration c. total integration of the device and E/E/PE safety related system
Comments	
Guideline	There should be evidence that a test specification has been written, during the design and development phase of the project, to validate that the software works correctly when integrated with the hardware. The intention of the hardware / software integration test specification is to ensure that when the hardware is married to the software any compatibility issues are exposed. There should be evidence that the integration of the software with the target hardware has been planned. The hardware / software integration test specification should specify the test strategy i.e. when and where the software is integrated with the hardware. The integration test plan should also specify the strategy of how the software is integrated with the hardware i.e. in full or in part, and if in part the order of integration. Larger more complex systems may consist of discrete components that might be best tested by incrementally integrating individual components. In such a case the integration test specification / plan should specify the strategy for incrementing and testing the components. The integration test specification should specify appropriate test cases and test data that provides adequate assurance that the software is compatible with the hardware. This will involve specifying a variety of test types that exercise the integrated software and hardware in differing fashions. The intention is to ensure that different facets of the integrated software and hardware are exercised and are shown to perform as required. This should involve the specification of tests that ensure that the integrated software and hardware meet the non-functional requirements specified in the safety requirements specification of the system. However, it is not possible to specify tests for some non-functional requirements (e.g. maintainability): these requirements must be verified by (code) review.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that:

7.2 TOEs for Software safety lifecycle requirements

TOE 23 - Specification of integration tests

	1. There is a software integration test specification. 2. The software integration test specification has a revision number. 3. The software integration test specification is held under configuration management. 4. The software integration test specification has been reviewed. 5. The software integration test specification was written during the design and development phase of the project. 6. The software integration test specification contains evidence of a planned approach to testing the integrated software. 7. The software integration test specification is complete i.e. contains an adequate set of test cases for the complete validation of the software and its interface with the hardware. 8. On larger systems an incremental strategy to testing has been adopted.

7.2 TOEs for Software safety lifecycle requirements

TOE 24 - Impact analysis on change

TOE	24
Title	Impact analysis on change
IEC 61508 Clauses and Tables	7.5.2.4, 7.5.2.6
Objective	For all safety integrity levels: 1. During the integration testing of the safety related programmable electronics, any modification or change to the integrated system shall be subject to an impact analysis that shall determine a. all the software modules impacted and b. the necessary re-verification activities.
Comments	
Guideline	During the integration testing of the software with the hardware defects may be exposed or the need for modification might be revealed. In such cases the software is likely to require modification and an impact analysis for each change should be performed. The impact analysis should identify areas of the software that will be affected by the change and what tests need to be performed to verify that the change is implemented correctly and affected areas function correctly.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. An impact analysis has been performed on the software when a need for modification has arisen 2. The impact analysis information is held under configuration management. 3. The impact analysis information is uniquely identifiable and traceable to a bug report / change request. 4. An approval process exists for authorising modifications on completion of the impact analysis. 5. Modification requests are recorded, dated, uniquely identified and reference impact analysis (the use of a bug tracking system is evidence of this e.g. Bugzilla). 6. The impact analysis clearly identifies the area of the software to be changed. 7. The impact analysis clearly identifies associated documentation to be changed.

7.2 TOEs for Software safety lifecycle requirements

TOE 25 - Documentation of results

TOE	25
Title	Documentation of results
IEC 61508 Clauses and Tables	7.5.2.7, 7.5.2.8
Objective	For all safety integrity levels: 1. Test cases and their resulted shall be documented for subsequent analysis. 2. Whether the objective and criteria of the test criteria have been met shall be documented. 3. Failure to meet test criteria shall be documented.
Comments	
Guideline	The results of the software and hardware integration tests should be documented each time the tests are performed regardless of the results of the tests. This documentation should include the following: 1. The version of the software and hardware tested 2. The overall result of the tests 3. The results of the individual test cases performed 4. The name(s) of the tester 5. The date of the test
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There are recorded software integration test results. 2. The software integration test results have a revision number. 3. The software integration test results are dated. 4. The software integration test results identify the individuals who performed the testing. 5. The software integration test results uniquely identify the software revision being tested. 6. The software integration test results uniquely identify the hardware revision being tested. 7. The software integration test results specify the overall result of the tests. 8. The software integration test results specify the overall result of the individual test cases performed. 9. The software integration test results are held under configuration management.

7.2 TOEs for Software safety lifecycle requirements

TOE 26 - Execution of software safety validation

7.2.5 Software safety validation

TOE	26
Title	Execution of software safety validation
IEC 61508 Clauses and Tables	7.7.2.2
Objective	For all safety integrity levels: 1. The validation activities shall be carried out as specified during software safety validation planning. 2. Testing shall be the main validation method for software; animation and modelling may be used to supplement the validation activities.
Comments	In the event of formal design models being used and validated by automatic proof techniques then testing may be used to complement such validation.
Guideline	There should be evidence, in the form of documentation, that the software has been validated (tested) in accordance with the software safety validation plan. The recording of the results of the software safety validation provides evidence that the activity has occurred.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There are recorded software validation test results.

7.2 TOEs for Software safety lifecycle requirements

TOE 27 - Recording of results of the software safety validation

TOE	27
Title	Recording of results of the software safety validation
IEC 61508 Clauses and Tables	7.7.2.4
Objective	For all safety integrity levels: 1. The results of software safety validation shall be documented. 2. For each safety function, the following shall be documented: a. a chronological record of the validation activities; b. the version of the software safety validation plan being used; c. the safety function being validated; d. tools and equipment used together with calibration data e. discrepancies between expected and actual results. 3. The tests shall show that all of the specified requirements for software safety are correctly performed and the software system does not perform unintended functions. 4. Test cases and their results shall be documented for subsequent analysis and independent assessment as required by the SIL 5. The test documentation shall indicate whether the test has passed or failed, and if failed the reason for its failure.
Comments	
Guideline	There should be documented evidence that the results of the software safety validation have been recorded. 1. The safety validation activity documentation should, for each safety function specified include: 2. A date and time (chronological record) indicating when the test was performed 3. The version of the safety validation plan used 4. Identification of the safety function being tested 5. A list of the tools and equipment used for the test, this may include serial numbers / asset numbers and calibration dates for the equipment 6. The results of the validation activity 7. Any discrepancies between the expected and actual result, this shall include justifications.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The software validation test results have a revision number. 2. The software validation test results are dated. 3. The software validation test results identify the individuals who performed the testing. 4. The software validation test results identify any test equipment used. 5. The software validation test results uniquely identify the software revision being tested. 6. The software validation test results specify the overall result of the tests. 7. The software validation test results specify the overall result of the individual test cases performed.

7.2 TOEs for Software safety lifecycle requirements

TOE 27 - Recording of results of the software safety validation

	8. The software validation test results are held under configuration management. 9. The software validation test results are complete i.e. all software test cases specified in the software test specification have been executed and a result recorded.

7.2 TOEs for Software safety lifecycle requirements

TOE 28 - Modification procedures

7.2.6 Software modification

TOE	28
Title	Modification procedures
IEC 61508 Clauses and Tables	7.8.2.1
Objective	For all safety integrity levels: 1. Prior to carrying out any software modification, software modification procedures shall be made available.
Comments	
Guideline	A procedural system for the modification of released (operational) software should exist. This system should allow a person to raise a request for modification. The modification procedure should also ensure that a procedure exists for the implementation, validation and verification of the modification. Furthermore the modification procedure should ensure that an impact analysis is performed, associated documentation is updated and that versioning of affected entities is performed. The modification procedure may be part of the standard quality assurance system, part of the standard configuration management system or a system bespoke to the project. Typically, the modification procedures will include procedures for: 1. raising a modification request (sometimes called a change request or change order) 2. approval of the modification request 3. obtaining the source code under revision control 4. applying impact analysis 5. revision control of the software changes 6. verification of the changes applied to the software and impacted areas. The modification procedures might specify the use of standardized forms and/or the use of software tools for bug logging and tracking.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. A formalized process exists for requesting modification of released (operational) software. 2. A formalized process exists for modification of released (operational) software. 3. A formalized process exists for verification of the implemented modification 4. The modification process involves performing impact analysis. 5. The modification process involves the update of associated documentation.

7.2 TOEs for Software safety lifecycle requirements

TOE 29 - Authorisation of software modification

TOE	29
Title	Authorisation of software modification
IEC 61508 Clauses and Tables	7.8.2.3
Objective	For all safety integrity levels: 1. A modification shall be initiated only on the issue of an authorised software modification request under the procedures specified during safety planning.
Comments	When authorisation is made the following should be considered: 1. The existing hazards that would be removed or mitigated and any new hazards that would be introduced. 2. The proposed change 3. The reason for the change
Guideline	No change to the operational software should be applied unless it has been authorized. If a modification request has been raised there should be evidence that it has been approved by appropriate personnel prior to commencement of the change. The approval method should be specified within the quality assurance procedures or procedures specified in the project plan for the product.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. All software modifications have been authorized. 2. The approval process is formalized in the QA procedures.

7.2 TOEs for Software safety lifecycle requirements

TOE 30 - Impact analysis

TOE	30
Title	Impact analysis
IEC 61508 Clauses and Tables	7.8.2.3, 7.8.2.4, 7.8.2.5
Objective	For all safety integrity levels: 1. 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. 2. The impact analysis results obtained shall be documented. 3. All modifications that 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.
Comments	
Guideline	There should be evidence, if modification has been applied to the software, that an impact analysis has been performed. The impact analysis should clearly identify which areas of the software have been affected by implementation of the modification. Furthermore, the impact analysis should clearly specify the tests that are required to verify that the changes are implemented correctly and areas of the software affected continue to maintain their requisite functionality. The impact analysis must ensure that a verification procedure is specified to ensure that safety functionality of the device is maintained.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. An impact analysis has been performed on the software when a need for modification has arisen 2. The impact analysis information is held under configuration management. 3. The impact analysis information is uniquely identifiable and traceable to a bug report / change request. 4. The impact analysis clearly identifies the area of the software to be changed. 5. The impact analysis clearly identifies associated documentation to be changed. 6. Modification requests are recorded, dated, uniquely identified and reference impact analysis (the use of a bug tracking system is evidence of this e.g. Bugzilla).

7.2 TOEs for Software safety lifecycle requirements

TOE 31 - Modification of the software

TOE	31
Title	Modification of the software
IEC 61508 Clauses and Tables	7.8.2.6, 7.8.2.7, 7.8.2.8, 7.8.2.9
Objective	For all safety integrity levels: 1. The safety planning for the modification of the software shall include: identification of staff and specification of their required competency; a detailed specification for the modification; verification planning; scope of re-validation and testing of the modification. 2. Modification shall be carried out as planned 3. Details of all modifications shall be documented, including references to: the modification request; the result of the impact analysis; software configuration management history; deviation from normal operations and conditions; all documented information affected by the modification activity; re-verification and revalidation of data results.
Comments	
Guideline	There should be evidence that when a modification has been applied to an operational software that the modification was planned. The plan should identify the staff responsible for implementing, authorizing / approving and validating the modification. If the competencies of the identified have not already been assessed as part of the development activity then the software modification plan should give evidence of their competencies. There should be evidence that the modification of the software complies with the modification plan and that the re-validation and testing of the modification complies with the impact analysis. There should be evidence that the modification of the software has been documented. This evidence should exist in the source code, the development documentation and configuration management of the software. The documentation should contain a revision history detailing the changes. Detail of the modification, date of the modification and who implemented the modification should be evident in the revision history of the source code. The revision history of the source code may be contained within a commented region of the source code module or may be recorded by the source code control system.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Modifications to the software have been planned. 2. The staff responsible for implementing, authorizing / approving and validating the modification are named. 3. Modifications to the software have been identified i.e. the revision control system contains information of the change, what was changed and when it was changed.

7.2 TOEs for Software safety lifecycle requirements

TOE 32 - Verification planning

7.2.7 Software verification

TOE	32
Title	Verification planning
IEC 61508 Clauses and Tables	7.9.2.1, 7.9.2.2
Objective	For all safety integrity levels: 1. The verification of software shall be planned concurrently with the development for each phase of the software safety lifecycle and this information shall be documented. 2. The software verification planning shall refer to the criteria, techniques and tools to be used in the verification activities, and shall address: a. the evaluation of the safety integrity requirements; b. the selection and documentation of verification strategies, activities and techniques; c. the selection and utilisation of verification tools (test harness, special test software, input/output simulators etc.); d. the evaluation of verification results; e. the corrective actions to be taken.
Comments	
Guideline	There should be evidence that for each phase of development (design and coding) verification activities have been planned to ensure that the output of the development activity meets input requirements for that phase. The verification activity is intended to ensure that the software correctly implements the functionality specified in the software safety requirements specification. Verification is an activity that is intended to reduce systematic errors in software by ensuring that the specified requirements are correctly implemented and therefore addresses traceability and testing to ensure compliance to requirements. There should be documented evidence that verification has been planned concurrently with the development phase. This evidence may include identification of revisions of the development phase and chronological evidence.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. For each phase of development (design and coding) that verification activities have been planned. Evidence of planned reviews for these phases is sufficient.

7.2 TOEs for Software safety lifecycle requirements

TOE 33 - Verification of the software (general)

TOE	33
Title	Verification of the software (general)
IEC 61508 Clauses and Tables	7.9.2.3, 7.9.2.4, 7.9.2.5, 7.9.2.6, 7.9.2.7
Objective	For all safety integrity levels: 2. The software verification shall be performed as planned. 3. Evidence shall be documented to show that the phase being verified, has in all respects, been satisfactorily completed. 4. 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. 5. 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, ▪ 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. 6. The following verification activities shall be performed: d. verification of software safety requirements e. verification of software architecture f. verification of software system design g. verification of software module design h. verification of code i. data verification j. software module testing k. software integration testing l. programmable electronics integration testing

7.2 TOEs for Software safety lifecycle requirements

TOE 33 - Verification of the software (general)

	m. j) software safety requirements testing (software validation).
Comments	
Guideline	Verification should be performed after completion of each phase of development listed below: 1. software safety requirements specification 2. software architecture design 3. software system design 4. software module design 5. coding 6. data formation 7. software module testing 8. software integration testing 9. software validation 10. programmable electronics integration testing 11. software safety requirements testing (software validation) There should be evidence that the verification plan for each development phase has been executed and satisfactorily completed. The verification activity should ensure that the completed development phase has been satisfactorily accomplished. This activity entails ensuring that software safety requirements are fully developed into a solution that has been tested and integrated. For each requirement specified in the software safety requirements specification there should be a clearly traceable implementation of that requirement and evidence that the requirement has been validated. There should be evidence that functional requirements have been implemented, non-functional (performance) requirements met and constraints adhered to. Typically, safety requirements should have a unique identification. At each verification phase, it should be possible to check that the uniquely identified requirement has been addressed i.e. that is has been designed into the system / module, has been coded and has been tested. The verification activity should ensure that the inputs to the development phase have been satisfactorily addressed and that corresponding outputs exist that allow the next development phase to be accomplished.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Verification activities have been performed and completed for: a. software safety requirements specification b. software architecture design c. software system design d. software module design e. coding f. data formation g. software module testing h. software integration testing i. software validation j. programmable electronics integration testing k. software safety requirements testing (software validation) 2. The review activities specified include ensuring the input requirements to the phase have been

	met by the output activities of the phase.

7.2 TOEs for Software safety lifecycle requirements

TOE 34 - Software safety requirements verification

TOE	34
Title	Software safety requirements verification
IEC 61508 Clauses and Tables	7.9.2.8
Objective	For all safety integrity levels: 1. Once the software safety requirements have been specified and before the next phase, software design and development begins, verification shall a. consider whether the specified software safety requirements (adequately fulfill the specified E/E/PES safety requirements (see IEC 61508-2) for functionality, safety integrity, performance, and any other requirements of safety planning; b. consider whether the software safety validation planning adequately fulfils the specified software safety requirements; c. check for incompatibilities between ▪ the specified software safety requirements and the specified E/E/PES safety requirements (see IEC 61508-2), ▪ the specified software safety requirements and the software safety validation planning
Comments	
Guideline	There should be evidence that the safety requirements specification has been verified. Verification of the software safety specification should ensure that each of the requirements specified in the E/E/PES safety requirements (see IEC 61508-2) for functionality, safety integrity, performance etc, relating to software have a corresponding requirement specified in the software safety requirements specification. The requirements specified in the software safety requirements specification should be easily traced to higher level requirements specified in the E/E/PES safety / system requirements. The verification process should show evidence that the requirements specified in the software safety requirements specification are complete, adequate and accurate.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. safety requirements specification has been verified 2. the verification activities ensure that all of the safety requirements specified in E/E/PES safety requirements, relating to the software, are covered by requirements in the safety requirements specification.

7.2 TOEs for Software safety lifecycle requirements

TOE 35 - Software architecture verification

TOE	35
Title	Software architecture verification
IEC 61508 Clauses and Tables	7.9.2.9
Objective	For all safety integrity levels: 1. after the software architecture design has been established, verification shall a. consider whether the description of the software architecture design adequately fulfils the specified software safety requirements; b. consider whether the specified tests of the software architecture integration are adequate for the description of the software architecture design; 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; ▪ safe modification to permit further evolution; ▪ check for incompatibilities between the following ▪ the description of the software architecture design and the specified software safety requirements, ▪ the description of the software architecture design and the specified tests of the software architecture integration, ▪ the specified tests of the software architecture integration and the software safety validation planning.
Comments	
Guideline	There should be evidence that the software architecture design has been verified to ensure that it adequately meets the requirements specified in the software safety requirements specification. There should be evidence of traceability of the architecture to requirements contained in the software safety requirements specification. The verification evidence should show that tests specified for the architectural design in the software safety test specification are adequate. The verification evidence should show that the software architecture has been examined to ensure that: 1. the safety performance required is feasible 2. the architecture lends itself to test 3. the architecture can be readily interpreted by the development and verification team 4. the architecture is sufficiently malleable for safe modification The verification evidence should show that checks have been made to look for inconsistencies between the software architectural design and the software safety requirements specification and software test specification.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. the software architecture design has been verified 2. the verification activities ensure that all of the safety requirements specified in software

TOE 35 - Software architecture verification

	requirements specification are covered by the software architecture design. 3. software architecture design provides the required safety performance 4. software architecture lends itself to test 5. the architecture can be readily interpreted by the development and verification team 6. the architecture is sufficiently malleable for safe modification.

7.2 TOEs for Software safety lifecycle requirements

TOE 36 - Software

TOE	36
Title	Software
IEC 61508 Clauses and Tables	7.9.2.10
Objective	For all safety integrity levels: 1. after the software system design has been specified, verification shall: a. consider whether the specified software system design adequately fulfils the software architecture design; b. consider whether the specified tests of the software system integration adequately fulfil the specified software system design; c. consider whether the attributes of each major component of the specified software system design are adequate with reference to ▪ feasibility of the safety performance required; ▪ testability for further verification; ▪ readability by the development and verification team; ▪ safe modification to permit further evolution; d. check for incompatibilities between ▪ the specified software system design and the description of the software ▪ architecture design, ▪ the description of the software system design and the specified tests of the ▪ software system integration, ▪ the specified tests of the software system integration and the specified tests of the architecture integration.
Comments	NOTE – The software system integration tests may be specified as part of the software architecture integration tests.
Guideline	There should be evidence that the software system design has been verified to ensure that it adequately meets the requirements specified in the software safety requirements specification and software architecture specification. There should be evidence of traceability of the design to the software architecture. The verification evidence should show that tests specified for the system design in the software safety test specification are adequate. The verification evidence should show that the software design has been examined to ensure that: 1. the safety performance required is feasible 2. the design lends itself to test 3. the design can be readily interpreted by the development and verification team 4. the design is sufficiently malleable for safe modification The verification evidence should show that checks have been made to look for inconsistencies between the software design and the software safety requirements specification and software test specification.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. the software design has been verified 2. the verification activities ensure that all of the safety requirements specified in software

	requirements specification are covered by the software design. 3. software design provides the required safety performance 4. software lends itself to test 5. the design can be readily interpreted by the development and verification team 6. the design is sufficiently malleable for safe modification.

7.2 TOEs for Software safety lifecycle requirements

TOE 37 - Software module design verification

TOE	37
Title	Software module design verification
IEC 61508 Clauses and Tables	7.9.2.11
Objective	For all safety integrity levels: 1. after the design of each software module has been specified, verification shall: a. consider whether the specified software module design adequately fulfils the specified software system design; b. consider whether the specified tests for each software module are adequate for the specified software module design; c. consider whether the attributes of each software module are adequate with reference to ▪ feasibility of the safety performance required; ▪ testability for further verification; ▪ readability by the development and verification team; ▪ safe modification to permit further evolution; d. check for incompatibilities between ▪ the specified software module design and the specified software system design ▪ (for each software module) the specified software module design and the specified software module tests ▪ the specified software module tests and the specified tests of the software system integration.
Comments	
Guideline	There should be evidence that the software module design has been verified to ensure that it adequately meets the requirements specified in the software system design. There should be evidence of traceability of the module design to the system design. The verification evidence should show that tests specified for the module design in the software safety test specification are adequate. The verification evidence should show that the software module design has been examined to ensure that: 1. the safety performance required is feasible 2. the module design lends itself to test 3. the module design can be readily interpreted by the development and verification team 4. the module design is sufficiently malleable for safe modification The verification evidence should show that checks have been made to look for inconsistencies between the software module design and the software system design and software test specification.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. the software module design has been verified 2. the verification activities ensure that all of the safety requirements specified in software system design are covered by the software design. 3. software module design provides the required safety performance

7.2 TOEs for Software safety lifecycle requirements

TOE 37 - Software module design verification

	4. software modules lend themselves to test 5. the software module designs can be readily interpreted by the development and verification team 6. the software module designs are sufficiently malleable for safe modification.

7.2 TOEs for Software safety lifecycle requirements

TOE 38 - Code and data verification

TOE	38
Title	Code and data verification
IEC 61508 Clauses and Tables	7.9.2.12, 7.9.2.13
Objective	For all safety integrity levels: 1. The source code shall be verified by static methods to ensure conformance to the specified design of the software module the required coding standards and the requirements of safety planning. 2. The data structures specified during design shall be verified for: a. completeness; b. self-consistency; c. protection against alteration or corruption; d. consistency with the functional requirements of the data-driven system. 3. The application data shall be verified for a. consistency with the data structures; b. completeness; c. compatibility with the underlying system software (for example sequence of execution, run-time, etc.); d. correctness of the data values. 4. All modifiable parameters shall be verified for protection against a. invalid or undefined initial values; b. erroneous, inconsistent or unreasonable values; c. unauthorised changes; d. data corruption. 5. All plant interfaces and associated software (i.e. sensors and actuators, and off-line interfaces shall be verified for: a. detection of anticipated interface failures; b. tolerance to anticipated interface failures. 6. All communications interfaces and associated software shall be verified for an adequate level of a. failure detection; b. protection against corruption; c. data validation.
Comments	

7.2 TOEs for Software safety lifecycle requirements

TOE 38 - Code and data verification

Guideline	There should be evidence that the source code has been verified to ensure that it adequately meets the requirements specified in the software module design. There should also be evidence that the source code meets the requirements of the specified coding standard for the project and any additional requirements specified in the safety plan. Typically conformance to module design is likely to be performed by peer review. However, there should also be evidence of the use of code checking tools that perform static analysis where such tools have been stipulated in the safety plan. Evidence for source code conformance to coding standards could be manually generated via reviews or automatically generated using appropriate tools. Where a tool is used to verify conformance there should be evidence of the tools suitability. There should be evidence that the source code has been verified to ensure that it is complete, correct and meets the functional requirements of the system. There should be evidence that the source code has been verified to ensure that defensive programming techniques have been incorporated. Defensive programming techniques should ensure that the software is intrinsically safe and protected against external influences. Intrinsic defensive programming measures ensure that software operates correctly and the internal data is valid. External defensive programming measures cover both security and mal configuration. See IEC 61508-7 C.2.5. There should be evidence that the source code's handling of data has been verified. There should be evidence that ensures that the source code's representation of data correctly addresses the data type, format and size appropriately. Data structures should be verified to ensure they are consistent with the design. There should be evidence that the source has been verified to ensure that it correctly interfaces to other software components / systems e.g. operating systems, protocol stacks. There should be evidence that the source code has been verified to ensure that it adequately performs detection and handling of failures caused by interface failures to external entities such as sensors, actuators etc. There should be evidence that the source code has been verified to ensure that it adequately handles the detection and failures of communication interfaces and that the communications interfaces provide adequate protection to ensure against failure of the device.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The source code has been verified 2. The verification activities ensure that all of the safety requirements specified in module design are covered by the source code. 3. Software modules lend themselves to test 4. The source code has been verified to ensure that data is handled correct 5. The source code has been verified to ensure that data representation and type is correct 6. The source code has been verified to ensure that it correctly interfaces to other software components / systems e.g. operating systems, protocol stacks. 7. The source code has been verified to ensure that it adequately performs detection and handling of failures caused by interface failures to external entities such as sensors, actuators etc.

8 TOEs and IEC 61508-3, Annex A Guide to the selection of techniques and measures

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

Table A.1 Software safety requirements specification

Technique / Measure	**Ref**	**SIL1**	**SIL2**	**SIL3**	**SIL4**	**TOEs**
Computer aided specification tools	B.2.4	R	R	R	HR	3
Semi formal methods	Table B.7	R	R	HR	HR	3
Formal methods	C.2.4	–	R	R	HR	3

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

Technique / Measure	**Ref**	**SIL1**	**SIL2**	**SIL3**	**SIL4**	**TOEs**
Fault detection and diagnosis	C.3.1	–	R	HR	HR	15
Error detecting and correcting codes	C.3.2	R	R	R	HR	15
Failure assertion programming	C.3.3	R	R	R	HR	17
Safety bag techniques	C.3.4	–	R	R	R	17
Diverse programming	C.3.5	R	R	R	HR	17
Recovery block	C.3.6	R	R	R	R	17
Backward recovery	C.3.7	R	R	R	R	17
Forward recovery	C.3.8	R	R	R	R	17
Retry fault recovery mechanism	C.3.9	R	R	R	HR	17
Memorising executed cases	C.3.10	–	R	R	HR	17
Graceful degradation	C.3.11	R	R	HR	HR	17
Artificial intelligence – fault correction	C.3.12	–	NR	NR	NR	17
Dynamic reconfiguration	C.3.13	–	NR	NR	NR	17
Structured methods, for example: JSD, MASCOT, SADT, Yourdon.	C.2.1	HR	HR	HR	HR	11
Semi-formal methods	Table B.7	R	R	HR	HR	11
Formal methods, for example: CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	–	R	R	HR	11
Computer aided specification tool	B.2.4	R	R	HR	HR	11

8 TOEs and IEC 61508-3, Annex A
Guide to the selection of techniques and measures

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

Technique / Measure	Ref	SIL1	SIL2	SIL3	SIL4	TOEs
Suitable programming language	C.4.6	HR	HR	HR	HR	18
Strongly typed programming language	C.4.1	HR	HR	HR	HR	18
Language subset	C.4.2	–	–	HR	HR	18
Certified tools	C.4.3	R	HR	HR	HR	18
Tools: Increased confidence from use	C.4.4	HR	HR	HR	HR	18
Certified translator	C.4.3	R	HR	HR	HR	18
Translator: Increased confidence from use	C.4.4	HR	HR	HR	HR	18
Library of trusted / verified software modules and components.	C.4.5	R	HR	HR	HR	16

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

Technique / Measure	Ref	SIL1	SIL2	SIL3	SIL4	TOEs
Structured methods, for example: JSD, MASCOT, SADT, Youdon	C.2.1	HR	HR	HR	HR	19
Semi formal methods	Table B.7	R	HR	HR	HR	19
Formal methods, for example: CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z.	C.2.4	–	R	R	HR	19
Computer-aided design tools	B.3.5	R	R	HR	HR	19
Defensive programming	C.2.5	–	R	HR	HR	20
Modular approach	Table B.9	HR	HR	HR	HR	20
Design and coding standards	Table B.1	R	HR	HR	HR	20
Structured programming	C.2.7	HR	HR	HR	HR	20
Use of trusted / verified software modules and components	C.2.10 C.4.5	R	HR	HR	HR	20

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

Technique / Measure	Ref	SIL1	SIL2	SIL3	SIL4	TOEs
Probabilistic testing	C.5.1	–	R	R	R	21
Dynamic testing and analysis	B.6.5 Table B.2	R	HR	HR	HR	21
Data recording and analysis	C.5.2	HR	HR	HR	HR	21
Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR	21
Performance testing	C.5.20 Table B.6	R	R	HR	HR	21
Interface testing	C.5.3	R	R	HR	HR	22

8 TOEs and IEC 61508-3, Annex A
Guide to the selection of techniques and measures

Table A.6. Programmable electronics integration

Technique / Measure	**Ref**	**SIL1**	**SIL2**	**SIL3**	**SIL4**	**TOEs**
Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR	23
Performance testing	C.5.20 Table B.6	R	R	HR	HR	23

Table A.7. Software safety validation

Technique / Measure	**Ref**	**SIL1**	**SIL2**	**SIL3**	**SIL4**	**TOEs**
Probabilistic testing	C.5.1	–	R	R	HR	7, 26
Simulation / modelling	Table B.5	R	R	HR	HR	7, 26
Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR	7, 26

Table A.8. Modification

Technique / Measure	**Ref**	**SIL1**	**SIL2**	**SIL3**	**SIL4**	**TOEs**
Impact analysis	C.5.23	HR	HR	HR	HR	24, 30
Re-verify changed software modules	C.5.23	HR	HR	HR	HR	30
Re-verify affected software modules	C.5.23	R	HR	HR	HR	30
Revalidate complete system	C.5.23	–	R	HR	HR	30
Software configuration management	C.5.24	HR	HR	HR	HR	30
Data recording and analysis	C.5.2	HR	HR	HR	HR	30

9 References.

[BBF1] PG Bishop, RE Bloomfield and PKD Froome. *Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications*. Report No: CRR336 HSE Books 2001 ISBN 0 7176 2010 7, http://www.hse.gov.uk/research/crr_pdf/2001/crr01336.pdf.

[HAZOP1] IEC 61882 Ed. 1.0 *Hazard and operability studies (HAZOP studies) - Application guide*, 2001-05-28.

[HSE1] HSE. *Human factors: Competence.* http://www.hse.gov.uk/humanfactors/comah/competence.htm

[IEEE829] IEEE 829-2008 *IEEE Standard for software and system test documentation (Revision of IEEE 829-1998)* ISBN: 978-0-7381-5747-4, http://ieeexplore.ieee.org/servlet/opac?punumber=4578271

[Sinclair1] I J Sinclair, *The use of Commercial Off The Shelf (COTS) Software in Safety-related Applications*, HSE Contract Research Report 80/1995 http://www.hse.gov.uk/research/crr_pdf/1995/crr95080.pdf

[SRJR1] Suzanne Robertson, James Robertson. *Mastering the Requirements Process*. Addison-Wesley, ACM-Press, ISBN 0-201-36046-2