CASS TOEs FOR SOFTWARE ASSESSMENT IN RELATION TO IEC 61508-3:2010 (Edition 2)

Version 3.3

Prepared for: The CASS Association (www.61508.org/cass)

By: The 61508 Association

All comment via: www.61508.org/contact

Disclaimer

While every care has been taken in developing and compiling the technical schedules and guidance to support the CASS scheme, The CASS Association, the contributors, and their parent organisations accept no liability for any loss, damage or injury caused, arising directly or indirectly in connection with reliance on its contents except to the extent that such liability may not lawfully be excluded under English Law.

The technical schedules and guidelines are intended as guidance only and any manufacturer, user, person or body using them should satisfy themselves as to the validity, appropriateness and correctness of their contents for their particular application. The CASS Association accepts no responsibility for any error or omission in the technical schedules and/or guidelines or any loss or damage suffered by or to any manufacturer, user, person, body or object arising out of the reliance or otherwise by any manufacturer, user, person or body on the guidance given by these technical schedules and guidelines in the interpretation of IEC 61508 or otherwise.

Advice, interpretations and opinions are issued for guidance only and at users risk, and The CASS Association accepts no liability for any loss or damage suffered by or to any person or object arising out of the reliance or otherwise by any person or body on such advice, interpretations or opinions issued from time to time.

©The CASS Association 2015 onwards.

This information is made freely available for public comment and use by The CASS Association, who request that appropriate acknowledgement be made when referencing the material.

Contents List

1 Acknowledgements

1 Acknowledgements

The following companies have participated and contributed to the development of the original document from which this version has been developed:

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
CSA Group (Sira Test & Certification)
The CASS Association[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 an earlier edition 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 to implement safety functions in systems. IEC 61508-3:2010 states what is required of the software used in functionally safe systems.

A clearer understanding of what is required of assessors and developers of software is needed. CASS has already developed templates for components (known as Type 1 systems), which focuses on the hardware. As software plays a major role in implementing functional safety, assessors and developers need to demonstrate compliance against IEC 61508-3. Previous experience with the templates for IEC 61508:1998 has shown their value (for example, see Ref [MLPR1]) hence, *The 61508 Association* has coordinated this updated version in response to the publication of IEC 61508:2010.

3 Document History

Version	Date	Status
1 (5129)	2009	Baseline 2009 edition of the CASS Templates edited by Graeme Parkin based on IEC 61508:1998
V0_A3	2010	Internal edition prepared by Martin Lloyd (MHL) incorporating changes for IEC 61508-3:2010.
V0_B1	20 Aug 2010	Internal Edition incorporating UKAS feedback and further work on levels of rigour based on Annex C of IEC 61508-3:2010 (MHL)
V0_C1	July 2013	Internal Edition circulated by Paul Reeve (PR) of Silmetric for further comment
V0_C2	30 Dec 2013	Internal edition incorporating feedback from Mutech and Sira (MHL)
V0_C3	2 Jan 2014	Internal edition. Feedback from Lloyd's Register acknowledged. Annex removed. (MHL)
V0_C4	11 Jan 2014	Internal edition to include book marks for each TOE so that CASS assessment tracking software can automatically refer the user to the appropriate guidance. (MHL)
V0_C5	14 Apr 2014	Internal edition to include an additional TOE concerned with the data communication requirements in IEC 61508-2 para. 7.4.11. Various cosmetic changes. (MHL)
V0_C5	14 Oct 2014	TOE 16 edited. Book marks revised for each TOE. (MHL)
V0_C5	15 Oct 2014	Note 2 added to TOE 16. (MHL)
V0_C5	16 Mar 2015	Extensive copy editing (PR)
V2_C1	18 Mar 2015	Minor changes prior to PDF and release to T6A (MHL)
V3_C1	24 Feb 2023	Flow and use improvements based on feedback from use (PB).
V3_C2	03 Nov 2023	Modified for new naming conventions (PB).
V3_C3	04 Oct 2024	Typos to moved TOEs corrected (PB).

4 Objective

This document is intended to be used as guidance in the consistent compilation of the evidence required by IEC 61508-3:2010, software requirements. This document provides guidance in assessing the conformity, to IEC 61508-3:2010, of an electronic/electrical/ programmable electronic system containing software that is intended to provide a safety function.

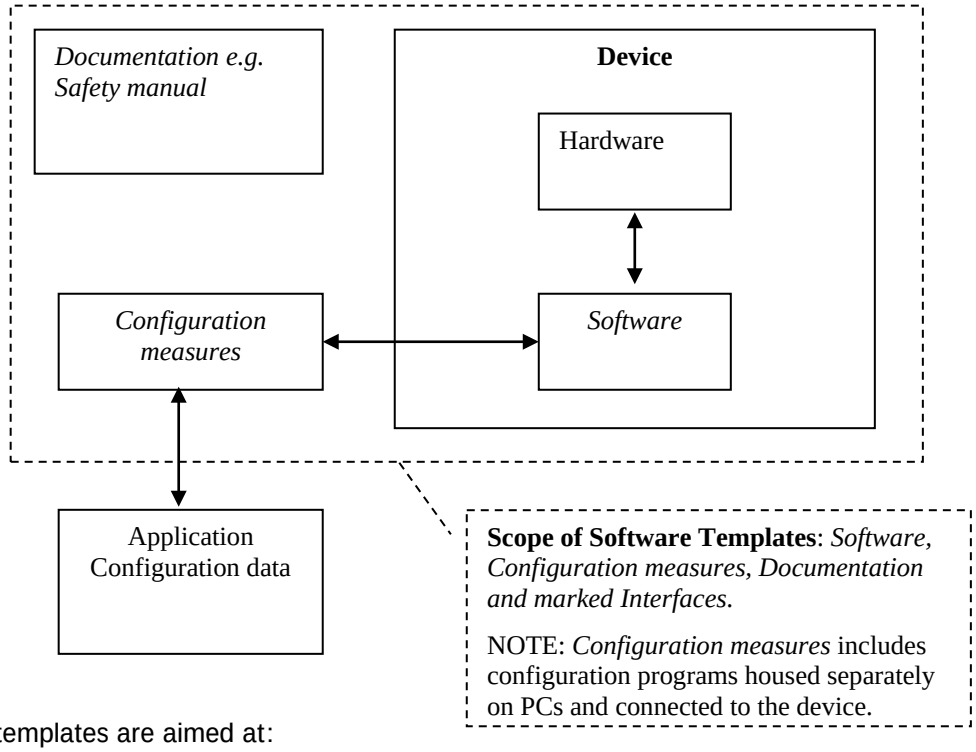
The purpose of this document is to provide a template for the assessment and acquisition of evidence for conformity to IEC 61508-3:2010 software requirements of intelligent devices. To this extent the document is also intended to provide the basis for independent verification of the software requirements for electrical / electronic / programmable electronic systems used in safety applications. The assessment template CASS-508-SW contains these TOEs in a form suitable for the assessor.

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

5 Scope

5.1 Current scope

These templates cover the assessment of device-level software, such as the embedded software and configuration measures generally found in intelligent transmitters, PLCs, and products of similar complexity to IEC 61508-3:2010. Such devices typically allow the setting of user parameters, calibration data, etc. by the integrator/end user but not application software that is developed under sector standards such as IEC 61511 and IEC 62061. 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.



These templates are aimed at:

- Developers;
- Assessors; and
- Conformity Assessment Bodies / Certifiers,

of software to IEC 61508.

This document addresses all of the activities specified in IEC 61508-3:2010 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. The assessment headings are intended to assist the certification process by introducing a collective interpretation of the standard that goes beyond the interests and preferences of any particular assessor.

This document does not replace any part or requirement of IEC 61508. The document is designed to be used with a copy of IEC 61508.

5.2 SIL 3

In the previous (IEC 61508:1998) edition of these templates SIL 3 and SIL 4 were not covered. However, experience with using them at SIL 3 has given the authors confidence that these templates can be used at SIL 3 subject to certain conditions, including:

- Independent oversight, for example UKAS accreditation of the assessment / certifying body applying the templates
- Wherever possible, deployment of confidence in use arguments
- Use of the informative guidelines on the selection of specific techniques in IEC 61508-3 Annex C "Properties for Systematic Capability" and its supporting material which is included in the TOEs that have matching tables of guidance in Annex C
- SIL 4 is not covered in full in this edition of the guidelines and further work is required on this document before it could be used in such systems.

5.3 Informal Guidance on Levels of Rigour

It should be understood that Annex C of IEC 61508-3 is informative and is couched in general terms. In this edition of the TOEs the intention has been to make the advice more specific so that developers and assessors have a clearer view of the implications of the guidance. Annex C of IEC 61508-3 introduces a simple and informal scale of rigour, which is quoted verbatim in the following table:

Level of Rigour	Definition
R1	Without objective acceptance criteria, or with limited objective acceptance criteria. For example, black-box testing based on judgement, field trials.
R2	With objective acceptance criteria that can give a high level of confidence that the required property is achieved (exceptions can be identified and justified). For example, test or analysis techniques with coverage metrics, coverage of checklists
R3	With objective systematic reasoning that the required property is achieved. For example, formal proof, demonstrated adherence to architectural constraints that guarantee the property.

Hence this document attempts to define some objective acceptance criteria that may assist the developer and assessor in a particular case.

Annex C is rightly cautious about its content being applied simplistically and stresses that the properties it describes are goals to be aimed for and that their attainment may entail trade-offs between different properties, for example: simplicity vs defensive design.

Finally, Annex C makes the following general and informal recommendation for the minimum levels of rigour that should be aimed for when Annex A of IEC 61508-3's tables require the corresponding SIL performance:

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

5 Scope

The above informal classification is followed in this document wherever guidance is given in the TOEs. It should be noted that techniques mentioned in the TOEs should be taken to have a level of R1 unless they are explicitly marked with R2 or R3. Where such levels are shown they are based on IEC 61508-3 Annex C's tables.

5.4 Future scope

These templates do not currently cover:

- SIL 4 – but there is limited guidance

6 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	www.61508.org/cass
COTS	= Commercial Off-The-Shelf	[Sinclair1]
E/E/PE	= Electrical/Electronic/Programmable Electronic.	IEC 61508-4:2010, 3.2.13
E/E/PES	= Electrical/Electronic/Programmable Electronic System.	IEC 61508-4:2010, 3.3.2
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
FMEDA	Failure Modes Effects and Diagnostic Analysis	
EUC	= Equipment Under Control.	IEC 61508-4:2010, 3.2.1
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 61508-3:2010, 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: 2010, 3.8.12 See section 7.3

6 Terminology and Abbreviations

Item	Expansion / Definition / Description	Reference
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: 2010, 3.8.13 See section 7.3
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, 3.8.11 See section 7.3
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 61508-3:2010, 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 61508-1:2010, 7.4..5.4
Review	A process by which an output of the software lifecycle is examined for comment and approval.	See section 7.3
SCM	Software Configuration Management.	IEC 61508-4:2010, 3.7.3, IEC 61508-7:2010, C.5.24
SIL	Safety Integrity Level.	IEC 61508-4:2002, 3.5.8
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]

6 Terminology and Abbreviations

Item	Expansion / Definition / Description	Reference
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).	–

7 Introduction to the software templates

7.1 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 61508-3 software requirements. These tables are representative of the order of requirements specified in the standard.

A TOE, or group of TOEs 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 and, if so, what evidence is required 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 61508-1 clause 6, Management of Functional Safety.

7.2 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 IEC 61508-3. The TOEs relate to specific clauses within IEC 61508-3.

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, 2, and 3 can be achieved. There is also limited guidance for 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 IEC 61508 standard. When alternative techniques and methods are employed the assessor must ensure that they are appropriate for the TOE / SIL. IEC 61058-3 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.

7.3 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.

- 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 assessments

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;
- the degree of change with respect to both safety-related and non-safety requirements made to the application specification.

7.4 Functional Safety Assessments (FSA) against IEC 61508-3

The reviews described in 7.3 above are those conducted as part of the software development lifecycle and are therefore part of the verification process undergone by the software. On the

other hand, FSAs are conducted by both in-house and external staff, the latter typically being those assessors involved in any certification process. In house FSAs are required and in terms of specific software requirements reference should be made to Annex A Table A10 of IEC 61508-3.

In house FSAs are a necessary “dress rehearsal” for assessments conducted by certifying bodies and the use of these CASS templates has been shown to assist these pre-certification activities.

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

8.1 TOEs for Software Quality Management

TOE	0
Title	IEC 61508 Conformance
IEC 61508 Clauses/Tables	7 (general for all parts)
Objective	To ensure the overall approach is in conformance with the IEC 61508 series of standards as relevant for the software. Conformance to IEC 61508-1 is always required and conformance to IEC 61508-2 is required if hardware is also in scope for the software / equipment with software. NOTE: This TOE does not require detailed evidence for the other parts of IEC 61508 (these are covered by other CASS templates). It is here to record a transparent status. This can be simply achieved by referencing other completed CASS templates in this TOE.
Comments	One aspect of IEC 61508 may be considered / assessed before others, but the status of the others must be recorded in the assessment (in this TOE).
Guideline	There should be completed CASS templates for the other aspects as evidence of conformity or a statement is made here on the situation. A declaration of conformance or certificate shall not be issued until all relevant aspects are in conformance. A report on conformance (including the aspects that haven't been covered or don't comply) may always be issued.
Assessment Requirements	For all safety integrity levels ensure there is general evidence for: 1. Conformance with IEC 61508-1 and, where relevant, IEC 61508-2. 2. Transparency for conformance status.

TOE 1 - Functional Safety Planning

IEC 61508-3 section: 6, Management of safety-related software

TOE	1
Title	Functional Safety Planning
IEC 61508 Clauses/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 7.3. 5. The software development plan has been approved. 6. The software development plan specifies the development process *(either directly or by reference).* 7. The software development plan specifies the procurement process for pre-existing software elements and tools (either directly or by reference). 8. The software development plan identifies the project deliverables. 9. The software development plan identifies all configuration items or refers to a configuration management plan. 10. The software development plan identifies the verification activities.

	11. The software development plan identifies the persons responsible for various project roles; see [HSE1]. 12. The software development plan refers to the modification procedures. 13. 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.

TOE 2 - Software Configuration Management

TOE	2
Title	Software Configuration Management
IEC 61508 Clauses/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 management.
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 for use throughout the software lifecycle; use of SCM tool is expected. 2. As a minimum the following must be under SCM: o safety analysis and requirements; o software specification and design documents; o software source code modules; o test plans and results; o verification documents; o pre-existing software elements and packages which are to be incorporated into the E/E/PE safety-related system; o all tools and development environments which are used to create or test, or carry out any action on, the software of the E/E/PE safety-related system;

	o reviews of documents and code (this is not explicitly stated in the standard) 3. The SCM facilitates change control of controlled entities and applies the following procedures: o Prevents unauthorised modifications o Documents modification requests o Documents the details of, and authorisation of all approved modifications. 4. The SCM facilitates base lining of configuration items and applies procedures for: o Establishing configuration baselines at appropriate points during the software development o Documents the (partial) integration testing of the baseline(s) o At all times, guarantees the composition of, and the building of all software baselines, including the rebuilding of earlier baselines. 5. The SCM ensures that appropriate methods are used to load valid software elements and data correctly into the run time system and includes procedures for: o Guaranteeing the integrity of all media used to deliver software elements and data o The checking the integrity of delivered software elements and data at its installation in the target system so that the software elements and data are loaded safely and replace any earlier versions completely and correctly 6. The SCM facilitates access control of controlled entities. 7. The SCM provides traceability of the changes made to configuration items to change requests and facilitates subsequent functional safety audits by documenting. o Detailed configuration status o Release status o Justification (by reference to the impact analysis records) for and approval of all modifications o Details of every modification 8. The SCM formally documents the release of software elements and data. In order to facilitate maintenance and modification the SCM keeps master copies throughout the operational lifetime of the released software of: o Software elements and data o Associated documentation o Release information

TOE - General Software Lifecycle Requirements

8.2 TOEs for Software Safety Lifecycle Requirements

IEC 61508-3 section: 7.1, General

TOE	3
Title	General Software Lifecycle Requirements
IEC 61508 Clauses/Tables	7.1
Objective	For all SILs, the software lifecycle must be structured into defined tables and activities – see Table 1 and figures 3 -6 in the standard.
Comments	
Guideline	The software lifecycle shall be defined in a document such as a company engineering instruction. Customisation for a project may entail for small projects, amalgamating the architectural and detailed software design phases into one phase and having a single software design document The integration of the quality and functional safety procedures may entail the writing of a handbook listing the individual safety procedures that must be followed in addition to the regular quality procedures followed by non-safety related projects The early assessment of tools and techniques can be a pre-assessment where evidence is not checked but the software lifecycle user's statements are accepted for the purposes of assessing whether a proposed lifecycle is appropriate
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. A software lifecycle based on the standard has been defined with scope, inputs and outputs for each phase 2. Any customisation of the software lifecycle for a particular project shall be justified on the basis of functional safety - (this means that the V model or variations on it, including incremental development, and the Y model [code generation from a formally verified design model] can be considered. However, only limited aspects of agile methods or those based on rapid prototyping are not to be considered.) 3. Any customisation for data-driven projects shall consider use of Annex G of the standard as a guide 4. Quality and safety assurance procedures shall be integrated into safety lifecycle activities (7.1.2.6); there should be a cross reference to the QMS to ensure that activities such as reviews and approvals follow the necessary control procedures. 5. For each lifecycle the appropriate techniques and measures shall be selected, justified and used. Annexes A and B shall be followed and arguments brought forward to indicate that the choice is appropriate to the safety integrity required. 6. In cases where there is any uncertainty about the choice of software lifecycle tools and/or techniques an assessment of the tools and techniques shall be conducted at an early stage of the project. The assessment shall cover: o Consistency and complementary nature of the methods, languages and tools used throughout all the lifecycle o Each different processor used in the hardware shall have its methods, languages and tools assessed o Whether the methods, languages and tools are well-adapted to the specific problems encountered during the development and maintenance of the software

TOE 3 - General Software Lifecycle Requirements

	7. The phases defined for the software lifecycle shall be identified and used in the impact analysis procedure (see TOE 37) 8. The "level of rigour" of the adopted tools and techniques shall at the very least match the informal classification: o SILs 1 and 2 – R1 o SIL 3 – R2 o SIL 4 – highest Rn available

TOE 4 - Specification of the Requirements for Software Safety Function and Integrity

IEC 61508-3 section: 7.2, Software safety requirements specification

TOE	4
Title	Specification of the Requirements for Software Safety Function and Integrity
IEC 61508 Clauses/Tables	7.2.1.1, 7.2.1.2, 7.2.2.3, 7.2.2.4, 7.2.2.6, 7.2.2.7, 7.2.2.8, 7.2.2.9, 7.2.2.11, 7.2.2.12, 7.2.2.13, Annex C, Table C1
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. Forward and backward traceability between system safety requirements and software safety requirements are HR 11. The use of semi-formal methods is HR. For SIL 4, in addition to the above: 12. 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 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 at SIL 4 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 standardised terms and expressions 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. There must be evidence of close collaboration between hardware and software specifiers. For example, the hardware FMEA shall be used to ask questions as to how the software will defend against hardware failures and whether additional requirements shall be included to cover such eventualities There shall be evidence that the requirements have been verified as a suitable basis for design (see TOE 41 and 46). This evidence will vary in formality according to SIL but will typically comprise some or all of the following: o Reviews of the documents and analyses against the above criteria o Verification of algorithms specified in the mathematical specification sections of the requirements specification o Verification of any requirements models
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 [see guidance at the end of this table] 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.. 9. A representative sample [see guidance at the end of this table]of the software safety requirements shall be verified to ensure that they: o are precisely written, o are understandable, free from ambiguity, o are free of adverse interference from non-safety functions, o provide a suitable basis for verification and validation.

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 including protection from malevolent or unauthorised action, legal, cultural and political *(where appropriate).*
12. Where a device supports safety and non-safety related functions there shall be clear distinction between the safety and non-safety related functions; further, there shall be justification for the partitioning of the software as safety/non-safety and evidence of the non-interference of the parts.
13. There shall be sufficient detail in the requirements to allow the design and implementation to achieve the required SIL. Such detail shall include, where relevant, requirements for
    - o Accuracy
    - o Timing and performance
    - o Capacity
    - o Robustness
    - o Overload tolerance
    - o Other characterising properties particular to the application
14. There shall be evidence that a common cause failure analysis has been conducted, and where credible failure mechanisms have been identified, effective defensive measures taken
15. The software safety requirements specification shall express requirements for the SIL(s) of the items in IEC 61508-3 paragraph 7.2.2.10 a and the requirements for any independence between functions.
16. Where requirements are expressed or implemented by configuration data, the data shall be:
    - o Consistent with the safety requirements
    - o Expressed in terms of permitted range and authorised combinations of its operational parameters
    - o Defined to be compatible with the underlying software
17. Where data defines the interface between the software and external systems the following performance characteristics shall be considered in addition to 7.4.11 of IEC 61508-2
    - o Need for consistency in terms of data definitions
    - o Invalid, out of range, or untimely values
    - o Response time and throughput, including max loading conditions
    - o Best case and worst case execution time
    - o Deadlock and livelock
    - o Overflow and underflow of storage capacity
18. Operational parameters shall be protected against:

Invalid, out of range, or untimely values

- Unauthorised changes
- Corruption

19. In the case where mimic or other pictorial displays are used for a human interface then the notes in 7.2.2.13 shall be followed

For SIL 3, in addition to the above, there shall be evidence that:

20. The techniques and measures adopted fulfil the level of rigour R2 with respect to:
    - Completeness of coverage of the system safety requirements (for example, by a defined checklist whose points can all be justified as providing complete coverage)
    - Correctness by verifying quantitatively that the coverage is complete (for example, checking that all states are covered by exhaustive state analysis, or by a complete comparison with a functional simulation whose coverage can be shown to be sufficient and whose discrepancies can be justified)
    - Freedom from intrinsic specification faults including ambiguity by using a consistent and semantically precise notation (for example, where state machines are used define which type of state machine)
    - Testability – the requirements can be shown to be capable of being matched by a complete set of traceable verification and validation measures
21. The use of a computer aided specification tools and / or semi-formal methods have been used (the quantity of requirements is a factor, note that SIL 3 recommends specification tools, it does not highly recommend them).
22. 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 sufficient rigour to ensure the benefits of the tool / semi-formal method are provided.

For SIL 4, in addition to the above:

23. The methods used shall fulfil R2 (see above) and R3 with regard to:
    - Correctness - a formal method has been employed to specify the software safety function
    - Verification of the specification by systematic analysis (model)
    - Systematic verification - The formal method has been used appropriately and with sufficient rigour to ensure the benefits of the formal method are provided.
24. Criteria for selecting a Representative Sample of Requirements

    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:

    - Where there are changes in the requirements or specification, or any evidence of a lack of understanding between stakeholders responsible for requirements;
    - Timing/concurrency issues;

	o Interface issues; o Interaction issues arising from any power management and performance requirements o Use of mathematical algorithms; o Control mechanisms (the use of control theory should be independently validated); o Complex logic (e.g. logic based on finite state machine formalism).

TOE 5 - Software Safety Requirements Presentation and Review by Software Developer

TOE	5
Title	Software Safety Requirements Presentation and Review by Software Developer
IEC 61508 Clauses/Tables	7.2.2.5
Objective	For all safety integrity levels: The requirements shall be made available to the software developer. 1. The requirements shall be reviewed by the software developer. 2. 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 7.3. For safety products a formal review should be undertaken as a minimum.
Assessment Requirements	1. The software developer review has considered: o safety functions; o configuration or architecture of the system; o hardware safety integrity requirements; o software systematic capability requirements; o capacity and response time; o equipment and operator interfaces, incl. reasonable foreseeable misuse. 2. For all safety integrity levels: ensure that there is evidence that the software safety requirements have been reviewed by the software developer, who will not normally be the author of the software safety requirements If the software developer is the author of the software safety requirements then there is evidence to 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 sufficiently competent to understand the implications of the specification by having at least a good reading knowledge and comprehension of any formalism used to express the requirements 5. Ensure that the software developer is in agreement with the assignment of the software safety integrity level and the level of rigour implied by the SIL 6. Ensure that any actions arising from the review have been fulfilled according to the correct functional safety management procedures

TOE 6 - Software Safety Requirements Traceability

TOE	6
Title	Software Safety Requirements Traceability
IEC 61508 Clauses/Tables	7.2.2.2
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 IEC 61508 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. Attention should be given to traceability and the use of unique hyper-linked requirement tags to support it.
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 base lined software requirements specification are traceable to a change request. The assessor is required to be alert to problems in formulating the software safety requirements. Such problems may be revealed by o Difficulty in agreeing particular requirements o Repeated attempts at framing particular requirements as revealed by the number of versions of the specification and changes being concentrated in particular areas Difficulties such as these shall be identified and pursued further by the assessor.

TOE 7 - Software Safety Requirements Specification of Safety Functions and Diagnostics

TOE	7
Title	Software Safety Requirements Specification of Safety Functions and Diagnostics
IEC 61508 Clauses/Tables	7.2.2.4, 7.2.2.10
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 EUC 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, defence against hardware failures, etc.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. Adequate software requirements exist for o software self-monitoring, o monitoring of electronic hardware, o diagnostics and periodic testing (RAM check, code space checks, hardware excitation checks) o Sensors (interfaces, behaviour, monitoring, testing) o Actuators (interfaces, behaviour, monitoring, testing) o 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. To the extent required by the hardware architecture there shall be requirements for o Periodic testing of safety functions while the system is running

	o Enabling of safety functions to be testable while the system is running o Software functions to execute proof tests and diagnostic tests required to fulfil the safety integrity requirements 5. The software requirements specification for the product shall express requirements for the following: o Functions that enable the EUC to achieve or maintain a safe state o Functions that detect, announce and manage faults in the PE hardware o Functions that announce and manage faults in the sensors and actuators o Functions that announce and manage faults in the software itself (software self-monitoring) o Functions that perform online periodic testing of safety functions o Functions that perform off-line testing of safety functions o Functions that allow the safe modification of the PE system o Interfaces to non-safety functions o Capacity and response time performance o Interfaces between the software and the PE system, including both off-line and programming facilities o Safety related communications 6. A suitable CCF analysis has been carried out on the requirements where relevant credible failure mechanisms identified and effective defensive measures taken.

TOE 8 - Validation Planning

IEC 61508-3 section: 7.3, Software safety validation planning

TOE	8
Title	Validation Planning
IEC 61508 Clauses/Tables	7.3.2.1, 7.7.2.3
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 necessarily the validation methods, which can be described separately and referred to from the validation strategy) 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. 5. Any division of responsibility for validation shall be documented

TOE 9 - Validation Considerations

TOE	9
Title	Validation Considerations
IEC 61508 Clauses/Tables	7.3.2.2, 7.3.2.3, 7.3.2.4, 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. Pass / fail criteria have 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 7.3. 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.
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. For each of the relevant modes of EUC operation, identification of the software which

	needs to be validated before commissioning begins 5. Required environment in which the validation activities are to take place (this could include calibrated tools and equipment) 6. Evidence that the scope and contents of the validation plan for software aspects of the system have been agreed with the assessor, or a party representing the assessor. o If necessary, a statement concerning the presence of an assessor during testing (IEC 61508-1 Clause 8) 7. The pass/fail criteria. The criteria for software validation shall include: o Required input signals with their sequences and values o Anticipated output signals with their sequences and values o Other criteria such as memory usage, timing, value tolerances, etc. 8. Policies and procedures relating to the validation (corporate, industrial, legislative, health and safety etc.)

TOE 10 - Validation Strategy

TOE	10
Title	Validation Strategy
IEC 61508 Clauses/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. o Preparation for use including setting and adjustment o Start up o Teaching/learning o Automatic operation o Manual operation o Semi-automatic operation o Steady state operation o Rest o Shut down o Maintenance o Reasonably foreseeable abnormal conditions o Reasonably foreseeable operator misuse 3. Technical strategy for the validation: o Techniques and procedures used for confirming that each safety function conforms with both the requirements and the specified requirements for software systematic capability 4. Rationale for the validation strategy: o Choice of manual , or automated techniques, or both

	o Choice of static, or dynamic techniques, or both o Choice of analytical or statistical techniques, or both o Choice of acceptance criteria based on objective factors, or expert judgement, or both

TOE 11 - Validation Plan Review

TOE	11
Title	Validation Plan Review
IEC 61508 Clauses/Tables	7.3.2.4
Objective	For all safety integrity levels: 1. The scope and content of the validation plan shall be reviewed and agreed 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 with the required level of independence for the SIL / SC. 2. There is a statement about assessor (FSA) presence at validation testing. 3. The results of the software validation plan review are held under configuration management. 4. Changes to the software validation plan have been made when specified by the review. 5. The software validation plan has been approved.

TOE 12 - Design Method

IEC 61508-3 section: 7.4, Software design and development

TOE	12
Title	Design Method
IEC 61508 Clauses/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 and object-oriented 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: 1. There is a software design specification. 2. The software design has a revision number. 3. The software design is held under configuration management. 4. The software design has been reviewed. 5. The software design reviews have concluded with a successful approval process 6. The software designers are those specified in the project plan. 7. The software design is a recognized method and has features that facilitate: o Abstraction o Modularity o Control of complexity o Ability to represent different views of the design including structural and behavioural views o Comprehensibility by developers and others who must understand the design o Verification and validation 8. The design method has features that facilitate the expression of: o Functionality o Information flow between elements o Sequencing and time related information o Timing constraints

	o Concurrency and access to shared resources o Data structures and their properties o Design assumptions and their dependencies o Exception handling o Design assumptions – pre-conditions, post-conditions and invariants o Comments 9. The software design conforms to the method used. 10. An appropriate design tool has been used. 11. For SIL 3 and 4, in addition to the above, there shall be evidence of the use of a CASE tool that enables verification of the design, for example, by the running of a design model on a model checker and checking it automatically for properties such as completeness with respect to a requirements model, absence of race hazards, etc. 12. The CASE tool has been used to sufficient rigor.

TOE 13 - Testability of Design

TOE	13
Title	Testability of Design
IEC 61508 Clauses/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. (It should be noted that 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. 2. At all SILs but increasingly at SIL 3 and above the following techniques are to be applied in relation to testability: o Comprehensive use of failure assertions and pre-conditions to limit the program input space and therefore the number of test cases (R2) (Note that assertions are required in production builds, not just in test versions.) o Where diverse monitoring is used then it covers the minimum safety requirements (R2) o Where state-based design is used then the test coverage of possible states is defined (R2) – 'impossible' states should be trapped by assertions. (Again, this is a general technique that applies outside the context of testing)

TOE 14 - Modification of Design

TOE	14
Title	Modification of Design
IEC 61508 Clauses/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, information hiding 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 by virtue of its control of complexity and its information hiding features

TOE 15 - Safety Considerations

TOE	15
Title	Safety Considerations
IEC 61508 Clauses/Tables	7.4.2.5, 7.4.2.6, 7.4.2.7, 7.4.2.8, 7.4.2.9, 7.4.2.10
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. Note that data communications requirements are addressed in a separate TOE (16).
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 design representations shall be based on a notation which is unambiguously defined or which is restricted to unambiguously defined features. 3. The safety function elements (that is, the SIL rated parts) of the design have been kept simple as far as practicable. 4. The design shall include, commensurate with the SIL, self monitoring of control flow and data flow. On failure detection appropriate action shall be taken. 5. Where the software is to implement both safety and non-safety functions then all the software shall be treated as safety related unless adequate design measures ensure that failures in the non-safety functions cannot adversely affect the safety functions. 6. Where the software is to implement safety functions of different SILs then all software shall be treated as belonging to the highest SIL 7. There is (temporal or physical) separation of all resources (including memory, processor time, IO, pre-allocated operating system data structures, etc.) between the safety and non-safety function elements so that safety elements access to resources is never compromised by competing non-safety elements. There shall be: o Evidence of complete independence between the safety and non-safety elements, or that any violation of independence is controlled

	o The justification for independence shall be documented 8. Where the systematic capability of a software element is lower than the SIL of the safety function which the software element supports, the element shall be combined with other elements such that the systematic capability of the combination equals the SIL of the safety function. (See IEC 61508-2, 7.4.3)

TOE 16 - Data Communications Considerations

TOE	16
Title	Data Communications Considerations
IEC 61508 Clauses/Tables	IEC 61508-2 para 7.4.11
Objective	For all safety integrity levels: 1. Although data communications requirements are handled in IEC 61508-2 they are usually implemented in software. Hence they must be considered as a software issue as well. 2. Both internal and external data communications shall be considered when failures in the communication process are taken into account.
Comments	
Guideline	1. The following shall be taken into account for all communications: a. The results of the FMEDA (Failure Mode, Effects and Diagnostic Analysis) applied to all components in data communications interfaces. See note below b. At the unit level, the risk of addresses being confused with another c. At the message level the risk of a non-safety-related message being confused with a safety-related message d. At the message and higher levels, methods including sequence numbers, time outs, node identities and other techniques implemented by the particular chosen protocol 2. Internal data communications are typically between processors and peripherals, or in multi- processor (which includes ASICs) designs between processors as well. The design of any such internal communications shall take the following into account: a. The methods by which the design handles failures that impact internal communications (for example, a stuck chip select output, or a stuck data line). b. For inter-processor communications, the effects of data corruption and delay 3. For external data communications, the use of “safe” variants of industry standard communications protocols shall be considered. Note 1: The FMEDA may be based on diagnostic functions implemented in software. Therefore a list of software functions must be provided to support the claim that the required diagnostic coverage is achieved. Note 2: This requirement includes all communication based on serial interfaces which operate under a protocol to handle errors, e.g. field bus devices and local networks. Consideration of the FMEDA may reveal additional software requirements such as the need for: "safe” variants of communications protocols, or, additional protective measures such as CRC protection for critical data fields within message packets as well as the usual CRCs on whole messages.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The FMEDA has been completed and includes a full treatment of the data communications functions. 2. The results of the FMEDA has been fed into the software design 3. The FMEDA is complete with respect to each stage of the communication between sender and receiver and that each layer of communication has been covered with respect to: a. Transmission errors b. Deletions c. Insertions d. Resequencing

	e. Corruption f. Delay g. Bit errors that cause addresses and other identity information to be duplicated h. And, where relevant, protection against the risk of masquerade and external attack (for example, by instructions in the safety manual on the correct installation of protection for the external communications) 4. Consideration has been given to data communications both internal to and external to the safety related system. 5. Where layered protocols are used there is an explicit distinction between the responsibilities of each layer so that the design is well structured 6. The functions needed to provide the necessary diagnostic coverage have been included in the design At SIL 3 and above 7. If a communication protocol has been specified formally in the requirements specification then the software design will be based on that formal representation and not on any natural language interpretation of the formal design. Note: If the IEC 61508-2 and IEC 61508-3 assessments are conducted by different persons or teams then the assessors are required to agree that this TOE has been met.

TOE 17 - Self Monitoring and Diagnostics

TOE	17
Title	Self Monitoring and Diagnostics
IEC 61508 Clauses/Tables	7.4.2.7
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 and the mean time to repair.

TOE 18 - Use of Pre-existing Software Elements

TOE	18
Title	Use of Pre-existing Software Elements
IEC 61508 Clauses/Tables	7.4.2.12
Objective	For all safety integrity levels to show that the re-use of existing software elements to implement all or part of the safety function meets the requirements for safety integrity: by following one of the compliance routes and providing a suitable safety Manual (see TOE 19)
Comments	This TOE states the compliance requirements
Guideline	There are three possible compliance routes for pre-existing software elements: 1. Route 1, developed in conformance with IEC 61508 at the appropriate SIL and fulfils the standard's requirements for the avoidance and control of systematic faults in the software 2. Route 2, Proven in use where evidence is provided to as if the software element was a hardware element being assessed under IEC 61508-2 7.4.10. (see TOE 19) 3. Route 3, via an assessment of a non-compliant development process as stated in 7.4.2.13 (see TOE 20) 4. Additionally, it may be necessary to use a combination of routes. For example: o A proven in use argument could be applied to an earlier release of a software element thanks to its full operational history and satisfactory defect records (Route 2). This release could then be the baseline. o Route 3 applied between the baseline and the present o Route 1 applied hereafter
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The choice of compliance route(s) is justified using quantitative and qualitative arguments 2. A suitable safety manual exists (see TOE 19) for the software element 3. If the pre-existing software element has an IEC 61508 certificate then the scope of the certificate has been checked to verify that: o The configuration state of the certified software element matches the state of the software element proposed for use in the safety function o The certified software element is compatible with the safety function hardware and any other system resources such as networks (for example, the scope of the certificate may exclude protocol stacks) 4. Any IEC 61508 certificates for pre-existing software elements shall be reviewed and assessed with respect to the: o Compatibility with the hardware used by the safety function o Compatibility with the software tools used by the build environment of the safety function o State of the documentation, including the safety manual o Scope of the certificate , including the software release number(s) and other configuration state of the software element

TOE 19 - Safety Manual

TOE	19
Title	Safety Manual
IEC 61508 Clauses/Tables	7.4.2.12 b)
Objective	To show there is a safety manual for each pre-existing software element in the safety function and that each is comprehensive and compliant with Annex D of IEC 61508-3
Comments	In some cases the hardware and software aspects of an element will be combined into a single safety manual.
Guideline	
Assessment Requirements	The safety manual shall include: 1. Comprehensive instructions for using the software element available to the integrator (this can be in separate documentation) 2. Definition of the scope and configuration status of the element 3. The configuration (in the sense of setting up) of the element in terms of its software and hardware environment and, if relevant, its cyber security environment shall be fully defined 4. A definition of the configuration used in the safety function 5. The assumptions used to justify the safe use of the element 6. Definition of the competence required for the integrator to use the element 7. Details of certification of the element including the scope, hardware and software environment, software tools to be used with the element, 8. Installation instructions 9. Configuration status of the released version including a. outstanding issues, b. Details of unimplemented requirements c. statement of forwards and backwards compatibility, d. hardware platform (environment) e. software platform details (for example, RTOS version number) 10. In cases where the element is released as source code, comprehensive compile and build instructions together with detailed documentation of the version numbers of all T3 tools used in the tool chain. 11. Definition of fault handling and the safe state(s) for the element 12. Support information including method for submitting change/enhancement requests, fault reports, support contacts, etc. 13. Definition of mandatory constraints and rules that must be observed by the integrator using the element at design time and at run time. These include the residual risks passed on to the software designer / integrator. 14. Definition of highly recommended and recommended instructions that are to be applied at design time and at run time 15. A statement on the strategy for security and / or cyber security including constraints and rules that are relevant for, or must be observed by, the integrator. If security or cyber security has not been considered, this shall be stated in the safety manual. Note: This does not mean confidential (security) information must be openly available

	in the market, but that there is supporting information for the integrator. The safety manual must be supported by: 16. Adequate justification of all claims in the safety manual. This supporting evidence can be from the element supplier's own records or be created or supplemented by the integrator Note that where there is no evidence available to support the functional safety assessment of the software element it cannot be used in a safety related system.

TOE 20 - Proven in Use Arguments for Pre-existing Software Element(s) (Route 2)

TOE	20
Title	Proven in Use Arguments for Pre-existing Software Element(s) (Route 2)
IEC 61508 Clauses/Tables	7.4.2.12 a)
Objective	For all safety integrity levels to show that all pre-existing software elements used to implement all or part of the safety function justified by Route 2 have sufficient evidence for the claim of proven in use.
Comments	This TOE must be applied to each pre-existing software element for which a Route 1 claim is made.
Guideline	1. The scope of the element must be defined and documented 2. There must be comprehensive software configuration management records covering at least the period of the lifetime of the software element for which the claims are made 3. There must be comprehensive records of the numbers of live copies of the software elements. Reasonable and verifiable assumptions must be made about the duty cycle of the element. 4. The records of live copies used as a basis for the proven in use argument shall be auditable, for example actual licence sales or downloads – sales of blocks of untraceable licences do not count because there is no evidence that they are ever actually used 5. There must be a database containing internally and externally reported software faults. It shall be possible to justify the estimates for the true fault rate as opposed to the actual rate, which may be skewed by under reporting from the field. 6. Using the sales and fault information it must be possible to show that the fault density of the software element is declining overall 7. In cases where the software element has gone through a number of versions and in order to achieve the necessary number of operational hours a number of these have to be conflated into one effective release, there shall be a quantitative justification for treating a number of releases as effectively one.
Assessment Requirements	There is evidence that for each relevant software element there is: 1. Documentation defining its scope and use, and history of releases 2. All constituent parts of the element are under comprehensive software configuration management and have been so throughout the period on which the proven in use argument is based 3. There are comprehensive and auditable records of the number of copies of the software element running in the field 4. The operational profile expected of the software element when it has been integrated into the safety function shall be similar to that for the release(s) of the element used in the calculation of operational hours 5. The calculations showing the claimed number of operational hours shall be presented and justified 6. The fault history shall be available to the assessor and the calculated fault density shall be presented and justified 7. If multiple versions of the element (that is, an unbroken series of releases) are being used as the basis of the proven in use argument then there must be a quantitative

TOE 20 - Proven in Use Arguments for Pre-existing Software Element(s) (Route 2)

	justification for this approach. 8. The proven in use argument shall be quantified and compared with the appropriate SIL figures for probability of failure and justified together with a confidence estimate.

TOE 21 - Pre-existing Software Elements – Compliance Route 3

TOE	21
Title	Pre-existing Software Elements – Compliance Route 3
IEC 61508 Clauses/Tables	7.4.2.13
Objective	To demonstrate compliance for a pre-existing software elements that lacks sufficient operational hours to justify a proven in use (Route 2) argument
Comments	
Guideline	The evidence demanded by this TOE can, to some extent, be assembled by using these CASS Guidelines recursively by applying them to each software element being considered for Route 3 compliance assessment.
Assessment Requirements	For all safety integrity levels ensure that there is evidence for Route 3s that: 1. There is a fully documented specification of requirements for the software element in its new application. The level of precision and rigour shall be the same as any safety element of the same systematic capability. (TOE 3) 2. The use of the element shall be justified by evidence of the desirable safety properties 3. There is a software design specification which is sufficiently precise to provide evidence of compliance with the requirements specification and the required systematic capability. 4. The integration of the software element with the hardware is specified in line with TOE 22 5. There is systematic verification and validation of the software element with documented reviews and testing and reviews of all parts of the code 6. If the software element contains functions which are not required in the safety related system then the non-required functions must be shown not to prevent the safety related system from meeting its safety requirements, for example by o Removal of unwanted functions o Disabling them o Suitable architecture – partitioning, etc. o Extensive testing to show the absence of any interference 7. All credible failure mechanisms of the software element have been identified and that appropriate fault mitigation measures have been implemented, including using a suitable architecture, and exception handling. 8. There is planning for the use of the software element: a. The configuration of the element is identified b. Its hardware and software environment is fully defined c. If necessary, the configuration of the tool chain (including make files and so on) needed to compile and build the element into the safety related software. d. The use of the element in the application respects the assumptions in the safety manual for the element and there is a valid justification for that use

TOE 22 - Data and Data Generation Languages used in Software Design and Development

TOE	22
Title	Data and Data Generation Languages used in Software Design and Development
IEC 61508 Clauses/Tables	7.4.2.14
Objective	Prevention of the introduction of faults into applications that are based on pre-existing functionality that is configured by data to meet specific application requirements
Comments	This TOE is concerned with applications that are configured in the way that PLCs and other control equipment are integrated: there is an underlying platform of a certified systematic capability and the integrator adds the application software, which is usually expressed in a LVL (limited variability language) or an FPL (Fixed Programming Language). These applications are often certified by sector standards of IEC 61508, for example, IEC 61511, IEC 62061. Note that if a full functionality programming language is used (for example, Ada or C) to build the application then the full rigour of IEC 61508 applies and the sector standards cannot be used.
Guideline	System Integrators who build these application systems should have a systematic functional capability that is their functional safety management system; including the software lifecycle should be assessed against the appropriate sector standards. Once certified, the integrator will then use an IEC 61508 certified PE system (such as a 'safety' PLC) as a platform on which the application can be programmed in an appropriate LVL. The application can then be assessed against the relevant sector standard without their being any need to invoke IEC 61508 directly. Hence this TOE provides general guidance and more detail should be followed in the sector standard.
Assessment Requirements	There are no assessment requirements for this TOE at the present time. Note: the requirements of this TOE may be fulfilled by appropriate training and competence for the relevant E/E/PE equipment.

TOE 23 - Software Architecture Design

TOE	23
Title	Software Architecture Design
IEC 61508 Clauses/Tables	7.4.3.2, Annex C Table C.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. For example, error detection and correcting codes are recommended to be used at all SILs even if the standard does only recommend them highly at SIL 4. At SIL 1 and SIL 2 some of the following are of great value as well. They are highlighted in bold font. 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. Where Object oriented techniques are applied close attention shall be paid to the informative information in IEC 61508-7 Annex G “Guidance for the development of safety-related object oriented software”
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 , o emphasises static resource allocation o restricts dynamic resource allocation (for example, to the start up sequence only) 2. Describes the dynamic behaviour of the system. 3. Describes the information representation. 4. Describes how information on failures will be logged and retrieved. 5. The architecture description shall include design strategies for both fault tolerance (consistent with the hardware) and fault avoidance, including where appropriate redundancy and diversity. 6. The architecture description shall be based on partitioning the design into components / subcomponents. This modular approach shall be applied with additional rigour with respect to: o Correctness where modularity targets for simplicity, completeness, predictability of behaviour, verifiability are supported by using methods such as metrics, interface complexity limits, pre-and post-assertions, generation of interface test cases, etc. to achieve R2 o Where automatic model based testing (MBT) is used and the model can be shown to be complete with respect to the design specification then R2 can be claimed o If certain intrinsic design faults can be systematically eliminated from a design model (for example, deadlock is demonstrated as being absent via a suitable (automatic) formal model checker then R3 can be claimed) o If the complete model can be used for test case generation based on formal reasoning then R2+ can be claimed o Using trusted/verified software modules and elements (Rigour level depends on their systematic capability, for example, a SIL 3 systematic capability certified RTOS kernel would typically be R2) 7. The architecture design description shall determine all software / hardware interactions and evaluate and detail their significance. 8. The architecture design description shall be unambiguous. 9. The architecture design description shall specify the design features used for maintaining safety integrity of all data. 10. The architecture design shall facilitate appropriate integration testing. 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. Especially for SIL 3 and SIL 4: 11. The software design shall detail fault detection and diagnosis (SIL 1 and SIL 2). 12. The software design shall allow for graceful degradation. 13. Semi-formal methods shall be employed in the specification of the software

	architecture (SIL 1 and SIL 2), including: o Stateless, or limited state design with well defined semantics and checking for completeness and correctness with regard to the requirements specification (R2) o Predictable scheduling architecture: • Time triggered architecture (R2, R3), or • Well defined cyclic behaviour with guaranteed max. cycle time 14. CASE tools shall be used. The following techniques are valuable at all safety integrity levels, depending on the effects of a failure occurring in that area of the design, particularly at SIL 3 and SIL 4: 15. Error detection and correcting codes (SIL 1 and SIL 2) shall be incorporated into the design depending on the effects of a failure occurring in specific areas of a design (for example, for communications, memory protection, and protection of data structures). 16. Failure assertion programming shall be employed (SIL 1 and SIL 2), in particular: o Pre- and post-assertions for interfaces (R2) o Targeted failure states and conditions (R2 and R3 depending on coverage) 17. Diverse monitoring techniques, with independence between the monitor and the monitored function in the same computer (R2)

TOE 24 - Offline Support Tools

TOE	24
Title	Offline Support Tools
IEC 61508 Clauses/Tables	7.4.4.1 – 7.4.4.9, 7.4.4.15 – 7.4.4.19
Objective	For all safety integrity levels: 1. A suitable set of integrated tools, including languages, compilers, compiler run time libraries, 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 compiler 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	The requirements for the selection and validation of tools to support the development, verification and testing of safety related software have been greatly extended in ed2.0, to reflect the importance of this topic and the wide variety of support tools which are now available. Tools are classified as T1, T2, or T3 depending on their purpose in the software safety lifecycle and the possible effects of incorrect outputs from the tool (note that these tool classes are defined in IEC 61508-4 rather than in Part 3). For example, a software design tool which provides automatic source code generation is more critical than a simple text editor, and a compilation system more critical than a source code generator. For tools with higher criticality (T2 and T3), a specification of the tool is required so that its behaviour and the implications of using it can be fully understood. If a support tool includes a run-time component which is included in the safety related application, then that run-time component must be treated as a safety related element in accordance with the requirements of the Standard. A common case is the run-time library provided by a programming language compiler, but other support tools such as those which generate code from a design representation could also provide run-time elements. The 2nd edition of the standard introduces a classification of tools (see IEC 61508-4), it is as follows: Software off-line support tool Definition Software tool that supports a phase of the software development lifecycle and that cannot directly influence the safety-related system during its run time. Software off-line tools may be divided into the following classes:

	– T1 generates no outputs which can directly or indirectly contribute to the executable code (including data) of the safety related system; NOTE 1 T1 examples include: a text editor or a requirements or design support tool with no automatic code generation capabilities; configuration control tools. – T2 supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software; NOTE 2 T2 examples include: a test harness generator; a test coverage measurement tool; a static analysis tool. – T3 generates outputs which can directly or indirectly contribute to the executable code of the safety related system. NOTE 3 T3 examples include: an optimising compiler where the relationship between the source code program and the generated object code is not obvious; a compiler that incorporates an executable run-time package into the executable code. 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 optimisation it should either be disabled or used consistently and with great care to use the lowest feasible level. Optimisers 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. Any change in optimisation level must be treated as a significant modification to the safety related software. The choice of optimisation level must be justified.
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: The software offline support tools are fit for purpose i.e. an industry recognized tool set has been used and the tools are appropriate for the target, 1. The language chosen is strongly typed and in the case of all full variability languages such as Ada, C/C++ a safer subset is used. The rigour of the subset shall reflect the SIL 2. The selection of software offline support tools is justified by the developer in a tools

selection and definition document or in a section of the software development plan

3. The users of the tools are competent to do so
4. To minimise human error, tools are integrated so that the output from one tool can act as automatic input to another.
5. The software offline support tools are either certified for use in the development of safety products or have a proven in use record for reliability
6. All off line support tools in categories T2 and T3 has documentation that clearly defines the behaviour of the tool and instructions or constraints on its use.
7. All tools in classes T2 and T3 have been assessed to determine the level of reliance placed on the tools and the potential failure mechanisms of the tools that may affect the executable software.
8. The potential failure mechanisms identified in the tools assessment are identified and appropriate mitigation / prevention measures taken. The mitigation measures may include:
    - List of known bugs and means used to avoid them
    - Restricted use of tool functionality
    - Checking of the tool outputs
    - Use of diverse tools for the same purpose
9. Each tool in class T3 conforms to its specification or documentation. This evidence may be based on a suitable combination of history of successful use in similar environments and for similar applications. Where such evidence only exists for earlier versions of a tool then greater reliance shall be placed on tool validation.
10. Documented results of a validation of the (T3) tools must include:
    - Chronological record of validation activities
    - Version of the tool and its manual being used
    - Tool functions being validated
    - Results including passes, failures and reasons for failures
    - Test cases and their results
    - Test cases shall be selected to reflect the pattern of use of the tool (for example, if an application makes extensive use of pointers then the compiler could be tested using an automatic test program generator set up to produce pointer intensive test code)
    - Test report showing results and subsequent analysis and discrepancies between expected and actual results (for example, a new version of a compiler might be checked by using it to build a previous version of the application that was built originally using the earlier compiler. Running the suite of unit, integration and validation tests on the recompiled previous application version and comparing the test results from the two versions of the application built with the respective compilers will yield a useful comparison)
11. Either optimisation has not been used or is used consistently in a strictly controlled manner during compilation such that any change in optimisation is treated as a significant software modification and is subject to the full rigour of the modification procedures.
12. No errors occur when compiling or linking.
13. There are no warning or when warnings exist there are appropriate justifications for

	them 14. The compatibility of the tools in an integrated toolset has been verified so that the tools cooperate automatically and minimise scope for human error (for example, evidence that make files have been verified) 15. Tools in classes T2 and T3 that generate items in the configuration baseline are under SCM and included in the configuration baseline, in particular: o Identification of the tool and its version o Identification of the configuration baseline items for which the tool has been used o The way the tool is used including tool parameters, options, scripts, make files for each configuration baseline o It is always possible to reconstruct any configuration baseline 16. SCM ensures that o only qualified and validated tools are used o only tools that are compatible with each other and the safety related system (hardware and software) are used 17. SCM ensures that every new version of an offline support tool is qualified, this may rely on evidence from a previous version if sufficient evidence is recorded to the effect: o The functional differences will not affect tool compatibility with the rest of the tool set o The new version is unlikely to contain new faults (Note, this cannot be assumed with complex tools such as compilers) 18. Conformance with this TOE may well devolve on a number of responsible parties, the division of responsibilities shall be documented in the safety planning document (System Development Plan)

TOE 25 - Programming Language Tools

TOE	25
Title	Programming Language Tools
IEC 61508 Clauses/Tables	7.4.4.10 – 7.4.4.14
Objective	For all safety integrity levels: 1. The programming language shall be appropriate for the design method. 2. The programming language or subset thereof shall be strongly typed. 3. Coding standards shall be used for the development of all safety related software. 4. The coding standards shall be reviewed as fit for purpose by the assessor. 5. 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	
Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. The language compiler has been assessed for fitness for purpose and validated against appropriate test suites o Validation suites may be used to assess functional and non-functional requirements of the language o The validation suites shall be capable of being run repeatedly and automatically 2. The software uses only defined language features 3. The language matches the characteristics of the application 4. The language contains features that facilitate the prevention and detection of design or programming mistakes 5. The language supports features that match the design method 6. Where 1- 5 above cannot be fully satisfied the fitness for purpose of the language and any additional measures taken to address its shortcomings are justified and documented. 7. All programming languages used in the development of all safety related software shall be used in accordance with a suitable programming language standard 8. Each coding standard specifies: o Good practice o Proscribes unsafe language features o Promotes comprehensibility, clarity and consistency by including quantitative guidelines based on suitable code metrics o Facilitates verification and validation (including features for aiding automatic static analysis) o Procedures for source code documentation (which may be generated

	automatically) o Where object oriented methods are used they take account of IEC 61508-7 Annex G 9. Each coding standard specifies information that must be included in the source code, including: o Legal entity – company, authors, etc. o Description o Inputs and outputs defined at the function definitions either as comments or as verifiable annotations o Configuration Management history (this depends on the SCM system and the history need not be kept in the source code if this is inconvenient, but it must be readily available) 10. Where automatic code generation or similar automatic translation is performed, the suitability of the automatic code generator and translator shall be assessed as part of the selection of the development support tools. This assessment shall: o Be documented o The documentation shall include all the sub points under point 10 of the previous TOE which specifies the selection and validation of T3 tools o In addition to the above, if the code generator takes design model outputs as its input there shall be a complete validation of the design models used to validate the code generator, typically using a model checker to demonstrate the validity of the models before code is generated from it. The demonstration of validity shall include properties such as freedom from exceptions (whether from file access, arithmetic over- and underflows, etc.). This model validation shall be documented o Any model used in validation must have a scope, profile, and scale that is equal to or greater than the models that are to be used in the safety related system. Any restrictions to the model state space that are introduced to limit the execution time of the model shall be justified.

TOE 26 - Detailed Design and Development

TOE	26
Title	Detailed Design and Development
IEC 61508 Clauses/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. The necessary information (specification of requirements, architecture design and validation plan for software aspects of system safety) was available prior to the start of software design 2. The design of each module is specified 3. Capable of verification which is expressed in the software module test specification 4. Complete with respect to the requirements specification (objective completeness criteria applied – R2) 5. Correct with respect to the requirements specification (Formal correctness arguments applied – R2+) 6. Free from intrinsic design faults specification (Formal correctness arguments applied – R2+) 7. Simple and understandable and is: o Modular (decomposes into clearly distinguishable entities).

	o Capable of safe modification o Structured (is layered). o Highly cohesive. o Low coupling. 8. Predictable in its behaviour (rigorously defined scheduling architecture such as TTA – R2+) 9. Verifiable and testable (Test coverage defined R2+) 10. Capable of detecting faults (Fault coverage shown to be objectively complete R2+) 11. Fault tolerant (Fault coverage shown to be objectively complete R2+) 12. Free from common cause failures (Fault coverage shown to be objectively complete R2+)

TOE 27 - Code Implementation

TOE	27
Title	Code Implementation
IEC 61508 Clauses/Tables	7.4.6.1
Objective	For all safety integrity levels code review is mandatory and shall establish that the code has the following properties: 1. Readable, understandable and testable. 2. It satisfies the specified requirements for software modular design. 3. It satisfies all relevant requirements specified during safety planning. 4. It satisfies the specified requirements of the coding standard(s)
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 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 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, and the reviews have been conducted after the code has been statically analysed so that trivial non-compliances have been removed, and metrics are available to the reviewer so that objective measures of complexity and other parameters are also available 10. Has evidence of the inclusion of defensive programming techniques (objective evidence of complete defensive coverage shows R2+) 11. Has been completely reviewed against the design and has documented evidence. 12. Has undergone effective static analysis (shallow free-tool style R1, or deeper R2+ for SIL 3+)) 13. The level of review is appropriate to the intended Safety capability and the requirements fulfilled by the code. The level of review shall be justifiable to the assessor and shall be selected in the following order of rigour: o Peer-to-peer review o Multiple peer reviews o Software walk through with peers (author actively participates) o Formal inspection (code author restricted to introductory briefing)

TOE 28 - Software Module Testing

TOE	28
Title	Software Module Testing
IEC 61508 Clauses/Tables	7.4.7.1 – 7.4.7.4
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 verified (that is, reviewed, analysed and 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.

TOE 28 - Software Module Testing

Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 1. There is a module test procedure and it includes a definition of the actions to be taken when tests are not passed 2. Module tests have been specified for each software module in accordance with the test procedure 3. The module test specification is held under configuration management. 4. The module test specification has been reviewed. 5. The module test specification has been approved. 6. The module tests provide adequate test coverage in terms of: o Completeness with respect to the design (objective coverage criteria such as automatic test case generation – R2) o Correctness – successful completion (formal proof of correctness R3) o Repeatability o Precisely defined testing configuration o In cases where the development uses formal methods, formal proofs or assertions the module tests may be reduced in scope to the extent that the formal proofs assertions used can be shown to equate to test cases for specific functions in the module. Tests for module properties such as performance must of course still be performed. 7. The module tests verify design functionality. 8. The module tests verify data integrity. 9. The module tests have been performed by the persons specified in the project development plan. 10. There are formalized results of the module testing. 11. Module testing results are documented and held under configuration management. 12. The results of module testing indicate that the software has passed the tests 13. In structural tests all entry points must be covered 14. In the case of all tests, if the necessary coverage is not achieved then an appropriate and justifiable reason must be recorded For SIL 2 and above there is evidence that: 15. Test management and automation tools are used. 16. In structural tests all statements must be covered For SIL 3 and 4, in addition to the above, there shall be evidence that: 17. In structural tests all branches must be covered 18. The module tests (where applicable) verify mathematical stability and accuracy. 19. The module tests (where applicable) verify performance and throughput. 20. Interface testing has been performed. 21. Module tests are forward traceable from the software design specification to the module test specification (HR at SIL 3 and 4) For SIL 4 in addition to the above, there shall be evidence that: 22. Structural test covers all conditions (100% MC/DC)

TOE 28 - Software Module Testing

	23. Probabilistic testing has been performed when appropriate to do so.

TOE 29 - Software Integration Testing

TOE	29
Title	Software Integration Testing
IEC 61508 Clauses/Tables	7.4.5.5, 7.4.8.1 – 7.4.8.5
Objective	For all safety integrity levels: 1. Software integration tests shall be specified during the design and development phases. 2. The specified integration tests shall state 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 (elements/subsystems)/ 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 shall 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 shall 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 and approved. 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 criterion (or criteria) for each test. 7. The software has passed integration testing and objective measures of coverage can be shown (R2). 8. For recorded failure cases requiring software modification an impact analysis has been performed and recorded.

TOE 30 - Specification of Integration Tests

IEC 61508-3 section: 7.5, Programmable electronics integration

TOE	30
Title	Specification of Integration Tests
IEC 61508 Clauses/Tables	7.5.2.1, 7.5.2.2, 7.5.2.3, 7.5.2.4, 7.5.2.7
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.

TOE 30 - Specification of Integration Tests

Assessment Requirements	For all safety integrity levels ensure that there is evidence that: 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 and approved. 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. It distinguishes between: o Merging the software with the target programmable electronic hardware o Integrating the E/PE/PE interfaces such as those to sensors, actuators, communications equipment o Applying the integrated system to the EUC or a simulation of its interfaces 7. The software integration test specification is complete i.e. contains : o An adequate set of test cases for the complete validation of the software and its interface with the hardware. o The expected results for each test case, which can be used for subsequent analysis of test results o Objective measures of coverage are available (R2) 8. On larger systems an incremental strategy to testing has been adopted.

TOE 31 - Not Used

TOE	31
Title	Not Used
IEC 61508 Clauses/Tables	
Objective	Not Used.
Comments	Was “Impact analysis on change” (now merged with TOE 37).
Guideline	Not Used.
Assessment Requirements	

TOE 32 - Documentation of Results

TOE	32
Title	Documentation of Results
IEC 61508 Clauses/Tables	7.5.2.7, 7.5.2.8
Objective	For all safety integrity levels: 1. Test cases and their expected results shall be documented for subsequent analysis. 2. Whether the objective and pass/fail criteria of the test cases 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 test environment(s) used to support and perform the tests 3. The overall result of the tests 4. The results of the individual test cases performed 5. The name(s) of the tester 6. 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 identify the test environment(s) used 8. The software integration test results specify the overall result of the tests and their coverage 9. The software integration test results specify the overall result of the individual test cases performed. 10. The software integration test results are held under configuration management.

TOE 33 - Execution of Software Aspects of System Safety Validation

IEC 61508-3 section: 7.7, Software aspects of system safety validation

TOE	33
Title	Execution of Software Aspects of System Safety Validation
IEC 61508 Clauses/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 is a (set of) precisely defined validation configuration(s) 2. There are recorded software validation test results. 3. There are operational profile coverage targets defined, justified and met (R2 for SIL 3+)

TOE 34 - Recording of Results of the Software Safety Validation

TOE	34
Title	Recording of Results of the Software Safety Validation
IEC 61508 Clauses/Tables	7.7.2.4, 7.7.2.5, 7.7.2.6, 7.7.2.7, 7.7.2.8, 7.7.2.9
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 and include a chronological record of the validation activities that will permit the sequence of activities to be retraced. 3. The software validation test results identify the individuals who performed the testing. 4. The software validation test results identify any test environment/equipment used. 5. Only validated/approved test tools that satisfy TOE 18 (para 7.4.4) shall be used 6. In cases where more than one party is responsible for system validation then the developer responsible for these software aspects of the system validation shall make

TOE 34 - Recording of Results of the Software Safety Validation

	the results available to the system integrator responsible for compliance with IEC 61508-1 and IEC 61508-2 7. The software validation test results uniquely identify the software revision being tested. 8. The software validation test results specify the overall result of the tests. 9. The software validation test results specify the overall result of the individual test cases performed. 10. The software validation test results are held under configuration management. 11. 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, in each case Pass, or, reasons for not passing.

TOE 35 - Modification Procedures

IEC 61508-3 section: 7.8, Software modification

TOE	35
Title	Modification Procedures
IEC 61508 Clauses/Tables	7.8.2.1, 7.8.2.2, 7.8.2.3
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, it includes stating: o Hazards which may be affected o Scope of the modification o Reasons for making the modification 2. A formalized process exists for verification of the implemented modification: o Reverification of the changed modules o Reverification of affected module(s) o Revalidation of the complete system o Regression testing o In each of these aspects R2 can be claimed if there are objective and quantifiable verification targets such as coverage 3. The modification process involves performing and documenting an impact analysis (see TOE 37), which must indicate the phase of the software lifecycle that must be returned to in order to perform the modification

TOE 35 - Modification Procedures

	4. The modification process involves the update of associated documentation. o Note that the planning of the modification shall meet the demands of TOE 30 and the scope of revalidation and testing of the modification to the extent required by the SIL

TOE 36 - Authorisation of Software Modification

TOE	36
Title	Authorisation of Software Modification
IEC 61508 Clauses/Tables	7.8.2.2
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.

TOE 37 - Impact Analysis

TOE	37
Title	Impact Analysis
IEC 61508 Clauses/Tables	7.5.2.6, 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. 4. 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	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. 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. A software lifecycle change procedure exists which includes the criteria to be applied by an impact analysis 2. An impact analysis has been performed on the software when a need for modification has arisen 3. The impact analysis information is held under configuration management. 4. The impact analysis information is uniquely identifiable and traceable to a bug report / change request. 5. The impact analysis clearly identifies the area of the software to be changed. 6. An approval process exists for authorising modifications on completion of the impact analysis 7. The impact analysis clearly identifies associated documentation to be changed. 8. The impact analysis identifies all phases of the software lifecycle affected by the proposed modification 9. Modification requests are recorded, dated, uniquely identified and reference impact

	analysis (the use of a bug tracking system is evidence of this).

TOE 38 - Modification of the Software

TOE	38
Title	Modification of the Software
IEC 61508 Clauses/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 for all instances of the operational software, which may have been deployed across numerous sites. 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, and, the planning of the modification shall meet the demands of IEC 61508-1: o Identify staff and their level of competency o Specification of the modification o Planning of the verification of the modification o Scope of revalidation and testing of the modification to the extent required by the SIL 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. 4. All modifications are recorded.

TOE 39 - Verification Planning

IEC 61508-3 section: 7.9, Software verification

TOE	39
Title	Verification Planning
IEC 61508 Clauses/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 (not just 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 that verification activities have been planned. Evidence of planned reviews for these phases is sufficient.

TOE 40 - Not Used.

TOE	40
Title	Not Used.
IEC 61508 Clauses/Tables	
Objective	Not Used.
Comments	General software verification has now moved to the end of the template (new TOE 46).
Guideline	
Assessment Requirements	Not Used.

TOE 41 - Software Safety Requirements Verification

TOE	41
Title	Software Safety Requirements Verification
IEC 61508 Clauses/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 o the specified software safety requirements and the specified E/E/PES safety requirements (see IEC 61508-2), o 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 corresponding requirement(s) 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.

TOE 42 - Software Architecture Verification

TOE	42
Title	Software Architecture Verification
IEC 61508 Clauses/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 o feasibility of the safety performance required; o testability for further verification; o readability by the development and verification team; o safe modification to permit further evolution; o check for incompatibilities between the following o the description of the software architecture design and the specified software safety requirements, o the description of the software architecture design and the specified tests of the software architecture integration, o 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 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. If the software architecture design is combined with the detailed software design, then this can be justified. 6. the architecture can be readily interpreted by the development and verification team. 7. the architecture is sufficiently malleable for safe modification.

TOE 43 - Software System Design Verification

TOE	43
Title	Software System Design Verification
IEC 61508 Clauses/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 o feasibility of the safety performance required; o testability for further verification; o readability by the development and verification team; o safe modification to permit further evolution; d. Check for incompatibilities between o the specified software system design and the description of the software o architecture design, o the description of the software system design and the specified tests of the o software system integration, o 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.

TOE 43 - Software System Design Verification

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.

TOE 44 - Software Module Design Verification

TOE	44
Title	Software Module Design Verification
IEC 61508 Clauses/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 o feasibility of the safety performance required; o testability for further verification; o readability by the development and verification team; o safe modification to permit further evolution; d. check for incompatibilities between o the specified software module design and the specified software system design o (for each software module) the specified software module design and the specified software module tests o 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. 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.

TOE 45 - Code and Data Verification

TOE	45
Title	Code and Data Verification
IEC 61508 Clauses/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 against the application requirements; 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	

TOE 45 - 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 (static analysis) tools. Where a tool is used to verify conformance there should be evidence of the tools suitability. Static analysis tools shall be used; their rigour and depth of analysis should reflect the SIL. 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. The source code has been verified to ensure timing performance and predictability of the behaviour in the time domain.

ToE 46 - Verification of the Software (General)

TOE	46
Title	Verification of the Software (General)
IEC 61508 Clauses/Tables	7.9.2.3, 7.9.2.4, 7.9.2.5, 7.9.2.6, 7.9.2.7
Objective	Considering the verification aspects considered for TOE's 41 to 45, for all safety integrity levels ensure that there is evidence that: 1. The software verification shall be performed as planned. 2. Evidence shall be documented to show that the phase being verified, has in all respects, been satisfactorily completed. 3. 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. 4. 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: o functionality, o safety integrity, o performance and other requirements of safety planning (see clause 6), o readability by the development team, o testability for further verification, o 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 o the tests specified in phase N, and the tests specified in the previous phase N–1, o the outputs within phase N. 5. The following verification activities shall be performed: a. verification of software safety requirements b. verification of software architecture c. verification of software system design d. verification of software module design e. verification of code f. data verification g. software module testing h. software integration testing i. programmable electronics integration testing

ToE 46 - Verification of the Software (General)

	j. j) software safety requirements testing (software validation).
Comments	
Guideline	Verification should be performed after completion of each development activity listed below: 1. software safety requirements specification 2. software architecture design 3. software system design 4. software module design 5. coding 6. configuration of functionality by means of data 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 activity has been executed and satisfactorily completed. The verification activity should ensure that the completed development activity 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 activity, 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 activity have been satisfactorily addressed and that corresponding outputs exist that allow the next development activity to be accomplished.
Assessment Requirements	Considering the verification aspects considered for TOE's 41 to 45, 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

ToE 46 - Verification of the Software (General)

	j. programmable electronics integration testing k. software safety requirements testing (software validation) 2. The review activities specified include ensuring the input requirements to the activity have been met by the outputs from the preceding activity.

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

[MLPR1] M H Lloyd, P J Reeve. *IEC 61508 and IEC 61511 Assessments – some Lessons Learned* . IEE Systems Safety Conference, 2009. http://farsideresearch.co.uk/images/docs/safety2009_0022_finalpaper.pdf

[BLLS] B. Littlewood & Lorenzo Strigini *Validation of Ultra High Dependability for Software-based Systems*. CACM 1993