Introduction

This conformity assessment template is for safety-related software under the scope of IEC 61508-3:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements*.

The following notes should be read prior to the assessment:

General Notes

1. For general guidance on using CASS conformity assessment documents, refer to *The CASS Guide* on using the CASS Methodology available from [www.61508.org/cass](http://www.61508.org/cass) (Document: ‘*CASS-Guide-A’).*
2. Use of this template assumes acceptance of the CASS scheme liability disclaimer in ‘*CASS-Guide-A*’.
3. This conformity assessment template does not replace the standard (IEC 61508-3:2010), it is intended to be used in conjunction with a copy of the standard as a method to manage the assessment of functional safety to support the assessor. The “Purpose of TOE” is a general guide to provide context and scope, and it is the assessor’s responsibility to ensure compliance with all the relevant clauses within the standard.
4. The assessor’s comment section shall be used for positive reporting including reference to the document sections / clauses relevant to evidence compliance.

Template Specific Notes

1. This assessment template should be used in conjunction with other CASS templates for IEC 61508 (see reference documents below).
2. The software CASS Guide (‘*CASS-Guide-508-SW – IEC 61508-3 Software TOES*’) supports, guides and explains the TOEs in relation to this assessment template.
3. Software development for FVL requires much greater effort plus requires verification & validation of much greater rigor than software development for LVL.

## References

* CASS-508-FSM – Management (IEC 61508-1)
* CASS-508-SLC – Safety Lifecycle (IEC 61508-1)
* CASS-508-SYS – System (IEC 61508-2)
* CASS-508-SUB – Subsystem Element (IEC 61508-2)

Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Description of change |
| V1 | 24/05/2016 | Initial detailed version (from experience of simpler templates) |
| V2 | 24/02/2023 | Updated to new naming convention and updated from feedback |
| V2.2 | 03/11/2023 | Updated to new naming convention |
| V2.3 | 04/10/2024 | Typos corrected related to TOE references |

TOEs 1 and 2 are for Software Quality Management

TOEs 3 to 45 are for Software Safety Lifecycle Requirements, further divided as follows:

TOE 3: General Software Lifecycle

TOEs 4 – 7: Software Safety Requirements Specification

TOEs 8 – 11: Software Validation Planning

TOEs 12 – 29: Software Design and Development

TOEs 30 – 32: Programmable Electronics Integration

TOEs 33 – 34: Software Aspects of System Validation

TOEs 35 – 38: Software Modification

TOEs 39 – 45: Software Verification

| **TOE Ref.** | **Target of Evaluation (TOE)** | **Purpose of TOE / prompts & checks** | **IEC 61508 references** | **Supporting documents** | **Assessor’s comments**  **(IEC 61508-3:2010)** |
| --- | --- | --- | --- | --- | --- |
| 0 | IEC 61508 Conformance | To ensure the overall approach is in conformance with the IEC 61508 series of standards as relevant for the software.  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. | 7 (all parts) |  |  |
| 1 | Functional Safety Planning | Functional safety planning shall ensure a strategy is developed for software procurement, development, integration, verification, validation, modification. The strategy defined shall be indicative of the required SIL.  \*\*\*\*\*\*\*\*  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). The software development plan identifies the project deliverables. 8. The software development plan identifies all configuration items or refers to a configuration management plan. 9. The software development plan identifies the verification activities. 10. The software development plan identifies the persons responsible for various project roles; see [HSE1]. 11. The software development plan refers to the modification procedures. 12. 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. | 6.2.2 |  |  |
| 2 | Software Configuration Management | Software configuration management 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.  \*\*\*\*\*\*\*\*  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:  * safety analysis and requirements; * software specification and design documents; * software source code modules; * test plans and results; * verification documents; * pre-existing software elements and packages which are to be incorporated into the E/E/PE safety-related system; * all tools and development environments which are used to create or test, or carry out any action on, the software of the E/E/PE safety-related system; * reviews of documents and code (this is not explicitly stated in the standard)  1. The SCM facilitates change control of controlled entities and applies the following procedures:  * Prevents unauthorised modifications * Documents modification requests * Documents the details of, and authorisation of all approved modifications.  1. The SCM facilitates baselining of configuration items and applies procedures for:  * Establishing configuration baselines at appropriate points during the software development * Documents the (partial) integration testing of the baseline(s) * At all times, guarantees the composition of, and the building of all software baselines, including the rebuilding of earlier baselines.  1. 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:  * Guaranteeing the integrity of all media used to deliver software elements and data * 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  1. The SCM facilitates access control of controlled entities. 2. The SCM provides traceability of the changes made to configuration items to change requests and facilitates subsequent functional safety audits by documenting:  * Detailed configuration status * Release status * Justification (by reference to the impact analysis records) for and approval of all modifications * Details of every modification  1. 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:  * Software elements and data * Associated documentation * Release information | 6.2.3 |  |  |
| 3 | General Software Lifecycle Requirements | To structure the development of the software into defined phases and activities (see Table 1 and figures 3 -6 in the standard).  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. A software lifecycle 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 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:  * Consistency and complementary nature of the methods, languages and tools used throughout all the lifecycle * Each different processor used in the hardware shall have its methods, languages and tools assessed * Whether the methods, languages and tools are well-adapted to the specific problems encountered during the development and maintenance of the software  1. The phases defined for the software lifecycle shall be identified and used in the impact analysis procedure (see TOE 37) 2. The ”level of rigour” of the adopted tools and techniques shall at the very least match the informal classification:  * SILs 1 and 2 – R1 * SIL 3 – R2 * SIL 4 – highest Rn available | 7.1 |  |  |
| 4 | Specification of the Requirements for Software Safety Function and Integrity | To specify the software safety requirements in order to facilitate the design and implementation of the software to meet the E/E/PE System Safety Requirements.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. There is a software safety requirements specification. 2. The software safety requirements specification has a revision number. 3. The software safety requirements specification is held under configuration management. 4. The author(s) of the software safety requirements specification are those specified in the project plan. 5. The software safety requirements specification has been reviewed. 6. The software safety requirements specification has been approved. 7. The software safety requirements have been derived from the system requirements. A *representative sample\* of software safety requirements* shall be verified to ensure that they are derived from relevant system requirements. 8. The system requirements have been adequately detailed. 9. A *representative sample\* of the software safety requirements* shall be verified to ensure that they:  * are precisely written, * are understandable, free from ambiguity, * are free of adverse interference from non-safety functions, * provide a suitable basis for verification and validation.  1. 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. 2. 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).* 3. 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. 4. 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  * Accuracy * Timing and performance * Capacity * Robustness * Overload tolerance * Other characterising properties particular to the application  1. 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 2. The software safety requirements specification shall express requirements for the SIL(s) of the items in 16 above (para 7.2.2.10) and the requirements for any independence between functions. 3. Where requirements are expressed or implemented by configuration data, the data shall be:  * Consistent with the safety requirements * Expressed in terms of permitted range and authorised combinations of its operational parameters * Defined to be compatible with the underlying software  1. 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  * Need for consistency in terms of data definitions * Invalid, out of range, or untimely values * Response time and throughput, including max loading conditions * Best case and worst case execution time * Deadlock and livelock * Overflow and underflow of storage capacity  1. Operational parameters shall be protected against:  * Invalid, out of range, or untimely values * Unauthorised changes * Corruption  1. 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:   1. 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  1. 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). 2. 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:   1. 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.  1. 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; * Timing/concurrency issues; * Interface issues; * Use of mathematical algorithms; * Control mechanisms (the use of control theory should be independently validated); * Complex logic (e.g. logic based on finite state machine formalism). | 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 |  |  |
| 5 | Software Safety Requirements Presentation and Review by Software Developer | To enable the software developer to review the software safety requirements specification to ensure the requirements are adequately specified.  \*\*\*\*\*\*\*\*  For all safety integrity levels: ensure that there is evidence that the software safety requirements have been reviewed by the software developer.   1. The software developer review has considered:  * safety functions; * configuration or architecture of the system; * hardware safety integrity requirements; * software systematic capability requirements; * capacity and response time; * equipment and operator interfaces, incl. reasonable foreseeable misuse.  1. If the software developer is the author of the software safety requirements, ensure that appropriate peers have reviewed the software safety requirements. 2. Ensure that there is evidence that the review evidence identifies the version of the software safety requirements specification 3. 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 4. 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 5. Ensure that any actions arising from the review have been fulfilled according to the correct functional safety management procedures | 7.2.2.5 |  |  |
| 6 | Software Safety Requirements Traceability | To ensure the requirements are traceable to the specification of the E/E/PE safety requirements.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. Each software safety requirement is uniquely identifiable. 2. Each software safety requirement has traceability to the relevant system requirements. 3. Modified or newly introduced requirements added to a baselined software requirements specification are traceable to a change request. 4. Evidence of problems in formulating the software safety requirements which may be evidenced by (and any such shall be identified and pursued further by the assessor):  * Difficulty in agreeing particular requirements * Repeated attempts at framing particular requirements as revealed by the number of versions of the specification and changes being concentrated in particular areas | 7.2.2.2 |  |  |
| 7 | Software Safety Requirements Specification of Safety Functions and Diagnostics | To ensure there is adequate consideration of software functionality including software self-monitoring, monitoring of electronic hardware, diagnostics and periodic testing, functions that relate to EUC safe state, fault detection, annunciation and management.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. Adequate software requirements exist for:  * software self-monitoring, * monitoring of electronic hardware, * diagnostics and periodic testing (RAM check, code space checks, hardware excitation checks) * Sensors (interfaces, behaviour, monitoring, testing) * Actuators (interfaces, behaviour, monitoring, testing) * The software requirements for hardware monitoring must reflect the identified demand level of the hardware.  1. Hazard analysis has been completed and additional software safety requirements recorded. 2. Where specific software diagnostics are required to obtain the requisite safety failure fraction there shall be related software safety requirements. 3. 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  * Periodic testing of safety functions while the system is running * Enabling of safety functions to be testable while the system is running * Software functions to execute proof tests and diagnostic tests required to fulfil the safety integrity requirements  1. The software requirements specification for the product shall express requirements for the following:  * Functions that enable the EUC to achieve or maintain a safe state * Functions that detect, announce and manage faults in the PE hardware * Functions that announce and manage faults in the sensors and actuators * Functions that announce and manage faults in the software itself (software self-monitoring) * Functions that perform online periodic testing of safety functions * Functions that perform off-line testing of safety functions * Functions that allow the safe modification of the PE system * Interfaces to non-safety functions * Capacity and response time performance * Interfaces between the software and the PE system, including both off-line and programming facilities * Safety related communications  1. A suitable CCF analysis has been carried out on the requirements where relevant credible failure mechanisms identified and effective defensive measures taken. | 7.2.2.4, 7.2.2.10 |  |  |
| 8 | Validation Planning | To develop a plan to specify the procedural and technical steps that will be used to demonstrate that the software satisfies its safety 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 | 7.3.2.1, 7.7.2.3 |  |  |
| 9 | Validation Considerations | To ensure the software validation plan has included the full scope of considerations with sufficient detail.  \*\*\*\*\*\*\*\*  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.  * If necessary, a statement concerning the presence of an assessor during testing (IEC 61508-1 Clause 8)  1. The pass/fail criteria. The criteria for software validation shall include:  * Required input signals with their sequences and values * Anticipated output signals with their sequences and values * Other criteria such as memory usage, timing, value tolerances, etc  1. Policies and procedures relating to the validation (corporate, industrial, legislative, health and safety etc) | 7.3.2.2, 7.3.2.3, 7.3.2.4, 7.3.2.5 |  |  |
| 10 | Validation Strategy | To ensure the software validation plan contains a technical strategy with a rationale for that strategy.  \*\*\*\*\*\*\*\*  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.  * Preparation for use including setting and adjustment * Start up * Teaching/learning * Automatic operation * Manual operation * Semi-automatic operation * Steady state operation * Rest * Shut down * Maintenance * Reasonably foreseeable abnormal conditions * Reasonably foreseeable operator misuse  1. Technical strategy for the validation:  * Techniques and procedures used for confirming that each safety function conforms with both the requirements and the specified requirements for software systematic capability  1. Rationale for the validation strategy:  * Choice of manual, or automated techniques, or both * Choice of static, or dynamic techniques, or both * Choice of analytical or statistical techniques, or both * Choice of acceptance criteria based on objective factors, or expert judgement, or both | 7.3.2.3 |  |  |
| 11 | Validation Plan Review | To ensure the scope and content of the software validation plan is reviewed by an assessor with the appropriate level of independence.  \*\*\*\*\*\*\*\*  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. | 7.3.2.4 |  |  |
| 12 | Design Method | To ensure the software design method possesses features that facilitate the necessary abstraction, modularity, information flow between components, sequencing and time related information, timing constraints, concurrency, data structure and their properties, design assumptions and their dependencies, etc.  \*\*\*\*\*\*\*\*  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 has been approved. 6. The software designers are those specified in the project plan. 7. The software design is a recognized method and has features that facilitate:  * Abstraction * Modularity * Control of complexity * Ability to represent different views of the design including structural and behavioural views * Comprehensibility by developers and others who must understand the design * Verification and validation  1. The design methods has features that facilitate the expression of:  * Functionality * Information flow between elements * Sequencing and time related information * Timing constraints * Concurrency and access to shared resources * Data structures and their properties * Design assumptions and their dependencies * Exception handling * Design assumptions – pre-conditions, post-conditions and invariants * Comments  1. The software design conforms to the method used. 2. An appropriate design tool has been used.   For SIL 3 and 4, in addition to the above, there shall be evidence that:   1. A CASE tool has been used. 2. The CASE tool has been used to sufficient rigor. | 7.4.2.2 |  |  |
| 13 | Testability of Design | To ensure software testability and the capacity for safe modification is considered during the design activities.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. The design facilitates decomposition and testing. 2. At all SILs but particularly at SIL 3 and above the following techniques are to be applied in relation to testability:  * Comprehensive use of failure assertions and pre-conditions to limit the program input space and therefore the number of test cases (R2) * Where diverse monitoring is used then it covers the minimum safety requirements (R2) * Where state-based design is used then the test coverage of possible states is defined (R2) – ‘impossible’ states should be trapped by assertions. | 7.4.2.3 |  |  |
| 14 | Modification of Design | To ensure the design method chosen shall possess features that facilitate software modification such as modularity, information hiding and encapsulation.  \*\*\*\*\*\*\*\*  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 | 7.4.2.4 |  |  |
| 15 | Safety Considerations | To ensure that as far as practicable the design shall minimize the safety-related part of the software, and where both safety and non-safety related functions are handled, then all of the software is treated as safety related, unless adequate independence between the functions can be demonstrated in the design.  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.  \*\*\*\*\*\*\*\*  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, 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:  * Evidence that complete independence between the safety and non-safety elements, or that any violation of independence is controlled * The justification for independence shall be documented  1. 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) | 7.4.2.5, 7.4.2.6, 7.4.2.7, 7.4.2.8, 7.4.2.9, 7.4.2.10 |  |  |
| 16 | Data Communications Considerations | To ensure that the techniques and measures used shall be appropriate, whether the approach is ‘black channel’ or ‘white channel’.  \*\*\*\*\*\*\*\*  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:    1. Transmission errors    2. Deletions    3. Insertions    4. Resequencing    5. Corruption    6. Delay    7. Bit errors that cause addresses and other identity information to be duplicated    8. 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   1. 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. | 2/7.4.11 |  |  |
| 17 | Self Monitoring and Diagnostics | To ensure that the design shall include, commensurate with the required SIL, self-monitoring of control flow and data flow. On failure detection, appropriate actions shall be taken.  \*\*\*\*\*\*\*\*  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. | 7.4.2.7 |  |  |
| 18 | Use of Pre-existing Software Elements | To ensure 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 next TOE).  \*\*\*\*\*\*\*\*  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:  * The configuration state of the certified software element matches the state of the software element proposed for use in the safety function * 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)  1. Any IEC 61508 certificates for pre-existing software elements shall be reviewed and assessed with respect to the:  * Compatibility with the hardware used by the safety function * Compatibility with the software tools used by the build environment of the safety function * State of the documentation, including the safety manual * Scope of the certificate, including the software release number(s) and other configuration state of the software element | 7.4.2.12 |  |  |
| 19 | Safety Manual | To ensure 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.  \*\*\*\*\*\*\*\*  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 outstanding issues, statement of forwards and backwards compatibility, hardware platform (environment). 10. In cases where the element is released as source code, comprehensive compile and build instructions. 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:   1. Adequate justification for 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. | 7.4.2.12 b) |  |  |
| 20 | Proven in Use Arguments for Pre-existing Software Element(s) (Route 2) | To ensure 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.  \*\*\*\*\*\*\*\*  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 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. | 7.4.2.12 a) |  |  |
| 21 | Pre-existing Software Elements – Compliance Route 3 | To demonstrate compliance for a pre-existing software element that lacks sufficient operational hours to justify a proven in use (Route 2) argument.  \*\*\*\*\*\*\*\*  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  * Removal of unwanted functions * Disabling them * Suitable architecture – partitioning, etc * Extensive testing to show the absence of any interference  1. 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. 2. There is planning for the use of the software element: 3. The configuration of the element is identified 4. Its hardware and software environment is fully defined 5. If necessary, the tool chain needed to compile and build the element into the safety related software. 6. 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 | 7.4.2.13 |  |  |
| 22 | Data and Data Generation Languages used in Software Design and Development | To prevent the introduction of faults into applications that are based on pre-existing functionality that is configured by data to meet specific application 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. | 7.4.2.14 |  |  |
| 23 | Software Architecture Design | To define the software architecture in sufficient detail by the developer and cover aspects such as techniques and measures to be used, partitioning and structure, hardware/software interface, etc.  \*\*\*\*\*\*\*\*  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,    * emphasises static resource allocation    * 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:  * 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 * 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 * 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) * If the complete model can be used for test case generation based on formal reasoning then R2+ can be claimed * 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)  1. The architecture design description shall determine all software / hardware interactions and evaluate and detail their significance. 2. The architecture design description shall be unambiguous. 3. The architecture design description shall specify the design features used for maintaining safety integrity of all data. 4. 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:   1. The software design shall detail fault detection and diagnosis (SIL 1 and SIL 2). 2. The software design shall allow for graceful degradation. 3. Semi-formal methods shall be employed in the specification of the software architecture (SIL 1 and SIL 2), including:  * Stateless, or limited state design with well defined semantics and checking for completeness and correctness with regard to the requirements specification (R2) * Predictable scheduling architecture: * Time triggered architecture (R2, R3), or * Well defined cyclic behaviour with guaranteed max. cycle time  1. 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:   1. Error detection and correcting codes (SIL 1 and SIL 2) shall be incorporated into the design (for example, for communications, memory protection, and protection of data structures). 2. Failure assertion programming shall be employed (SIL 1 and SIL 2), in particular: 3. Pre- and post-assertions for interfaces (R2) 4. Targeted failure states and conditions (R2 and R3 depending on coverage) 5. Diverse monitoring techniques, with independence between the monitor and the monitored function in the same computer (R2) | 7.4.3.2, Annex C Table C.2 |  |  |
| 24 | Offline Support Tools | To ensure a suitable set of integrated tools, including languages, compilers, compiler run time libraries, configuration management tools, coding standards and where appropriate automatic testing tools are selected for the design method and required SIL.  \*\*\*\*\*\*\*\*  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 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  1. 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. 2. Documented results of a validation of the (T3) tools, which 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)  1. 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. 2. No errors occur when compiling or linking. 3. There are no warning or when warnings exist there are appropriate justifications for them 4. 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) 5. Tools in classes T2 and T3 that generate items in the configuration baseline are under SCM and included in the configuration baseline, in particular:  * Identification of the tool and its version * Identification of the configuration baseline items for which the tool has been used * The way the tool is used including tool parameters, options, scripts, make files for each configuration baseline * It is always possible to reconstruct any configuration baseline  1. SCM ensures that  * only qualified and validated tools are used * only tools that are compatible with each other and the safety related system (hardware and software) are used  1. 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:  * The functional differences will not affect tool compatibility with the rest of the tool set * The new version is unlikely to contain new faults (Note, this cannot be assumed with complex tools such as compilers)  1. 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) | 7.4.4.1 – 7.4.4.9, 7.4.4.15 – 7.4.4.19 |  |  |
| 25 | Programming Language Tools | To ensure the languages selected are appropriate for the design method, are strongly typed and that adequate coding standards are reviewed and used.  \*\*\*\*\*\*\*\*  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  * Validation suites may be used to assess functional and non-functional requirements of the language * The validation suites shall be capable of being run repeatedly and automatically  1. The software uses only defined language features 2. The language matches the characteristics of the application 3. The language contains features that facilitate the prevention and detection of design or programming mistakes 4. The language supports features that match the design method 5. 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. 6. All programming languages used in the development of all safety related software shall be used in accordance with a suitable programming language standard 7. Each programming language standard specifies:  * Good practice * Proscribes unsafe language features * Promotes comprehensibility, clarity and consistency by including quantitative guidelines based on suitable code metrics * Facilitates verification and validation (including features for aiding automatic static analysis) * Procedures for source code documentation (which may be generated automatically) * Where object-oriented methods are used they take account of IEC 61508-7 Annex G  1. Each programming language standard specifies information that must be included in the source code, including:  * Legal entity – company, authors, etc * Description * Inputs and outputs * 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)  1. 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:  * Be documented * The documentation shall include all the sub points under point 10 of the previous TOE which specifies the selection and validation of T3 tools * 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 * 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. | 7.4.4.10 – 7.4.4.14 |  |  |
| 26 | Detailed Design and Development | To ensure the software safety requirements specification, the description of the software architecture design and the software safety validation plan are available prior to the start of the detailed design, and that there is modularity, structure, testability, the capacity of safe modification, software module design and testing is specified and that structured methods are employed.  \*\*\*\*\*\*\*\*  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:  * Modular (decomposes into clearly distinguishable entities). * Capable of safe modification * Structured (is layered). * Highly cohesive. * Low coupling.  1. Predictable in its behaviour (rigorously defined scheduling architecture such as TTA – R2+) 2. Verifiable and testable (Test coverage defined R2+) 3. Capable of detecting faults (Fault coverage shown to be objectively complete R2+) 4. Fault tolerant (Fault coverage shown to be objectively complete R2+) 5. Free from common cause failures (Fault coverage shown to be objectively complete R2+) | 7.4.5.2, 7.4.5.3, 7.4.5.4 |  |  |
| 27 | Code Implementation | To ensure that the code is reviewed to establish that the code is: readable, understandable and testable; that it satisfies the specified requirements for software modular design; that it satisfies all relevant requirements specified during safety planning and that it satisfies the specified requirements of the coding standard(s).  \*\*\*\*\*\*\*\*  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 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:  * Peer-to-peer review * Multiple peer reviews * Software walk through with peers (author actively participates) * Formal inspection (code author restricted to introductory briefing) | 7.4.6.1 |  |  |
| 28 | Software Module Testing | To ensure that each software module is tested to ensure that it performs its intended function and does not perform unintended functions; that data recording and analysis is performed; that functional black-box testing is performed; the results of module testing are documented and that the procedures for corrective action on failure of test are specified.  \*\*\*\*\*\*\*\*  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:  * Completeness with respect to the design (objective coverage criteria such as automatic test case generation – R2) * Correctness – successful completion (formal proof of correctness R3) * Repeatability * Precisely defined testing configuration * 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.  1. The module tests verify design functionality. 2. The module tests verify data integrity. 3. The module tests have been performed by the persons specified in the project development plan. 4. There are formalized results of the module testing. 5. Module testing results are documented and held under configuration management. 6. The results of module testing indicate that the software has passed the tests 7. In structural tests all entry points must be covered 8. 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:   1. Test management and automation tools are used. 2. In structural tests all statements must be covered   For SIL 3 and 4, in addition to the above, there shall be evidence that:   1. In structural tests all branches must be covered 2. The module tests (where applicable) verify mathematical stability and accuracy. 3. The module tests (where applicable) verify performance and throughput. 4. Interface testing has been performed. 5. 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:   1. Structural test covers all conditions (100% MC/DC) 2. Probabilistic testing has been performed when appropriate to do so | 7.4.7.1 – 7.4.7.4 |  |  |
| 29 | Software Integration Testing | To ensure that software integration tests are specified during the design and development phases; that the specified tests state the test cases and test data, types of test to be performed, the test environment, tools, configuration and programs; that the test criteria on which the completion of the test will be judged; that the procedures for corrective action on failure of test; that any modification resulting from software integration testing is subject to an impact analysis that will determine the software modules impacted and the necessary re-verification and re-design activities.  \*\*\*\*\*\*\*\*  Where software integration testing is applicable, for all safety integrity levels ensure that there is evidence that:   1. A software integration test specification exists. 2. The software integration test specification is held under configuration management. 3. The software integration test plan has been reviewed. 4. The specified integration tests specify appropriate test cases and test data 5. The software integration test specification specifies the test environment, test tools, software configuration and requisite test programs. 6. The software integration test specification specifies a pass / fail 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. | 7.4.5.5, 7.4.8.1 – 7.4.8.5 |  |  |
| 30 | Specification of Integration Tests | To ensure that integration tests are specified during the design and development phase to ensure the compatibility of the hardware and the software in the device with consideration of the split of the system into integration levels; test cases and test data; types of test to be performed; test environment including tools, support software and configuration description; test criteria on which the completion of the test will be judged; distinction between test activities carried out by the developer on his premises and those that access the user’s site; the specified integration tests for programmable electronics shall distinguish between merging of software system on the target programmable electronic hardware, the E/E/PE integration and the total integration of the device and E/E/PE safety related system.  \*\*\*\*\*\*\*\*  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. 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:  * Merging the software with the target programmable electronic hardware * Integrating the E/PE/PE interfaces such as those to sensors, actuators, communications equipment * Applying the integrated system to the EUC or a simulation of its interfaces  1. The software integration test specification is complete i.e. contains :  * An adequate set of test cases for the complete validation of the software and its interface with the hardware. * The expected results for each test case, which can be used for subsequent analysis of test results * Objective measures of coverage are available (R2)  1. On larger systems an incremental strategy to testing has been adopted | 7.5.2.1, 7.5.2.2, 7.5.2.3, 7.5.2.4, 7.5.2.7 |  |  |
| 31 | Not Used | Now merged into TOE 37. |  | Not Used. | Not Used. |
| 32 | Documentation of Results | To ensure the documenting of the test cases and their expected results, the objective and pass/fail criteria of the test cases have been met, and any failure to meet test criteria.  \*\*\*\*\*\*\*\*  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. | 7.5.2.7, 7.5.2.8 |  |  |
| 33 | Execution of Software Aspects of System Safety Validation | To ensure the validation activities are carried out as specified during software safety validation planning and that testing is the main validation method for software; animation and modelling may be used to supplement the validation activities.  \*\*\*\*\*\*\*\*  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+) | 7.7.2.2 |  |  |
| 34 | Recording of Results of the Software Safety Validation | To ensure the software validation activities are documented in order to record that the specified functions for software safety are correctly performed, that the software system does not perform unintended functions, the test cases and their results and if failed the reason for failure.  \*\*\*\*\*\*\*\*  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 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. | 7.7.2.4, 7.7.2.5, 7.7.2.6, 7.7.2.7, 7.7.2.8, 7.7.2.9 |  |  |
| 35 | Modification Procedures | To ensure that prior to carrying out any software modification, software modification procedures shall be made available.  \*\*\*\*\*\*\*\*  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:  * Hazards which may be affected * Scope of the modification * Reasons for making the modification  1. A formalized process exists for verification of the implemented modification:  * Reverification of the changed modules * Reverification of affected module(s) * Revalidation of the complete system * Regression testing * In each of these aspects R2 can be claimed if there are objective and quantifiable verification targets such as coverage  1. 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 2. The modification process involves the update of associated documentation.   Note: the planning of the modification shall meet the demands of TOE 31 scope of revalidation and testing of the modification to the extent required by the SIL | 7.8.2.1, 7.8.2.2, 7.8.2.3 |  |  |
| 36 | Authorisation of Software Modification | To ensure that a modification is initiated only on the issue of an authorised software modification request under the procedures specified during safety planning.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. All software modifications have been authorized. 2. The approval process is formalized in the QA procedures. | 7.8.2.2 |  |  |
| 37 | Impact Analysis | To ensure that an analysis is carried out on the impact of the proposed software modification on the functional safety of the E/E/PE safety related system, that the impact analysis results obtained shall be documented and that 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.  To ensure the use of impact analysis during integration testing of the software.  \*\*\*\*\*\*\*\*  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 e.g. Bugzilla). | 7.5.2.6, 7.8.2.3, 7.8.2.4, 7.8.2.5 |  |  |
| 38 | Modification of the Software | To ensure that 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. Also, that modification shall be carried out as planned, and that 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.  \*\*\*\*\*\*\*\*  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:  * Identify staff and their level of competency * Specification of the modification * Planning of the verification of the modification * Scope of revalidation and testing of the modification to the extent required by the SIL  1. 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. 2. All modifications are recorded | 7.8.2.6, 7.8.2.7, 7.8.2.8, 7.8.2.9 |  |  |
| 39 | Verification Planning | To ensure that the verification of software is planned concurrently with the development for each phase of the software safety lifecycle and this information shall be documented, and that the planning refers to the criteria, techniques and tools to be used in the verification activities.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. For each phase of development (design and coding) that verification activities have been planned. Evidence of planned reviews for these phases is sufficient. | 7.9.2.1, 7.9.2.2 |  |  |
| 40 | Not Used | General software verification has now moved to the end of the template (new TOE 46). |  | Not Used. | Not Used. |
| 41 | Software Safety Requirements Verification | To ensure the software safety requirements have been adequately verified before commencing design and development.  \*\*\*\*\*\*\*\*  For all safety integrity levels ensure that there is evidence that:   1. safety requirements specification has been verified 2. the verification activities ensure that all of the safety requirements specified in E/E/PES safety requirements, relating to the software, are covered by requirements in the safety requirements specification. | 7.9.2.8 |  |  |
| 42 | Software Architecture Verification | To ensure the description of the software architecture design and the corresponding tests adequately fulfill the specified software safety 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. | 7.9.2.9 |  |  |
| 43 | Software System Design Verification | To ensure the specified software system design and the corresponding tests adequately fulfill the software architecture design.  \*\*\*\*\*\*\*\*  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. If the software architecture design is combined with the detailed software design, then this can be justified 6. the design can be readily interpreted by the development and verification team 7. the design is sufficiently malleable for safe modification. | 7.9.2.10 |  |  |
| 44 | Software Module Design Verification | To ensure the specified software module design and the corresponding tests adequately fulfill the specified software system design.  \*\*\*\*\*\*\*\*  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. | 7.9.2.11 |  |  |
| 45 | Code and Data Verification | To ensure the source code is verified by suitable methods to ensure conformance to the specified module design, the required coding standards and the requirements of safety planning.  \*\*\*\*\*\*\*\*  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. The verification activities ensure that all of the safety requirements specified in module design are covered by the source code. 2. The data structures specified during design shall be verified for: 3. Completeness against the application requirements; 4. self-consistency; 5. protection against alteration or corruption; 6. consistency with the functional requirements of the data-driven system. 7. The application data shall be verified for 8. consistency with the data structures; 9. completeness; 10. compatibility with the underlying system software (for example sequence of execution, run-time, etc.); 11. correctness of the data values. 12. All modifiable parameters shall be verified for protection against 13. invalid or undefined initial values; 14. erroneous, inconsistent or unreasonable values; 15. unauthorised changes; 16. data corruption. 17. All plant interfaces and associated software (i.e. sensors and actuators, and off-line interfaces shall be verified for: 18. detection of anticipated interface failures; 19. tolerance to anticipated interface failures. 20. All communications interfaces and associated software shall be verified for an adequate level of 21. failure detection; 22. protection against corruption; 23. data validation. 24. Verification of timing performance and predictability of behaviour in the time domain. | 7.9.2.12, 7.9.2.13,  7.9.2.14 |  |  |
| 46 | Verification of the Software (General) | To ensure that verification of the software is performed at each stage according to the plan and is completed using the correct techniques and methods.  \*\*\*\*\*\*\*\*  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: 2. software safety requirements specification 3. software architecture design 4. software system design 5. software module design 6. coding 7. data formation 8. software module testing 9. software integration testing 10. software validation 11. programmable electronics integration testing 12. software safety requirements testing (software validation) 13. The review activities specified include ensuring the input requirements to the phase have been met by the output activities of the phase. | 7.9.2.3, 7.9.2.4, 7.9.2.5, 7.9.2.6, 7.9.2.7 |  |  |

\*\*\* END OF TEMPLATE \*\*\*