Certifying industrial systems using IEC 61508

Shrikant Satyanarayan, LDRA, Inc., www.ldra.com

With recent advances in automation, software is no longer a small part of electro-mechanical systems, but instead forms the underlying technology providing functional safety for many products. As a result, the requirement for software functional safety has become critical in industrial automation, transportation, nuclear energy generation and other markets. To ensure functional safety, many have adopted IEC 61508 as the basic standard, on which sector-specific values are built.

The IEC 61508 standard is a risk-based approach for determining the SIL (Safety Integrity Level) of safety instrumented functions. If computer system technology is to be effectively and safely exploited, it is essential that the available guidance on these safety-related aspects is adequate to make correct decisions.

Notably, implementing a process standard for software development involves much more than simply understanding the rules and knowing how to apply them. To implement a standard effectively, it is essential to integrate the standard into the entire development lifecycle from requirements through test. This article will demonstrate how an automated process can close the loop, assuring developers and the IEC 61508 regulators that the code—from requirements through to test—is traceable and has been verified as compliant.

IEC 61508 basics

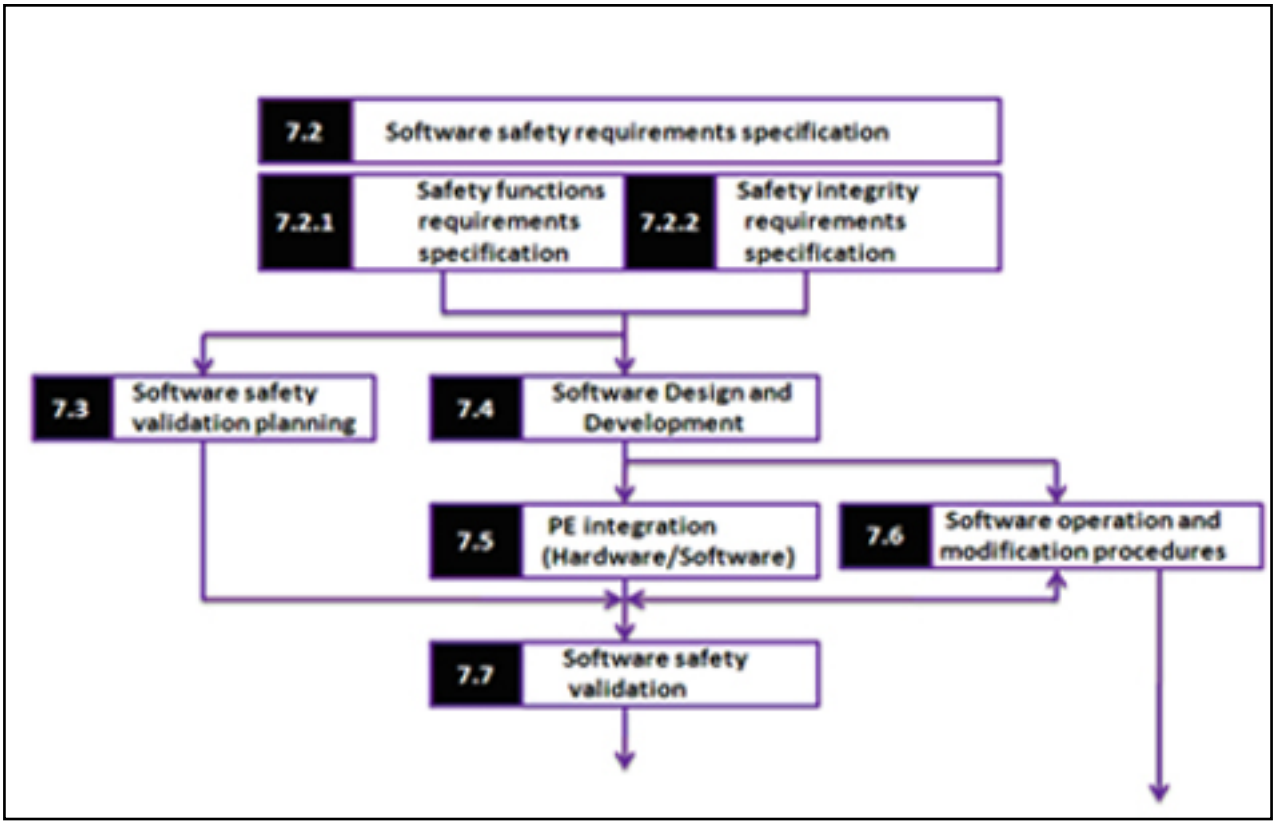
A number of applications use Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related systems in a variety of sectors that involve a wide range of complex, hazard and risk potentials. The required safety measures for each application depend on many factors. The generic nature of IEC 61508 makes it an ideal “blank canvas” for seamless integration of these application-dependent factors.

In most situations, safety is achieved by a number of systems that rely on many technologies (e.g., mechanical, hydraulic, pneumatic, electrical, electronic, programmable electronic). Any safety strategy must therefore consider not only all the elements within an individual system (i.e. sensors, controlling devices and actuators), but also all subsystems which make up the safety-related system as a whole.

As industrial systems depend more and more on software, the primary focus in certifying software to IEC 61508 lies with IEC 61508-Part 3, which discusses software requirements, and IEC 61508-Part7, which describes different techniques and measures required to achieve the relevant SIL for the application.

Creating a software safety lifecycle

IEC 61508 describes a software safety lifecycle which involves the systematic development process, requirement traceability, software validation and modifications. As you can see in Figure1, the structure of the software safety lifecycle divides the software development lifecycle into defined phases and activities.



In addition to this outline of the various phases, the following sections define the

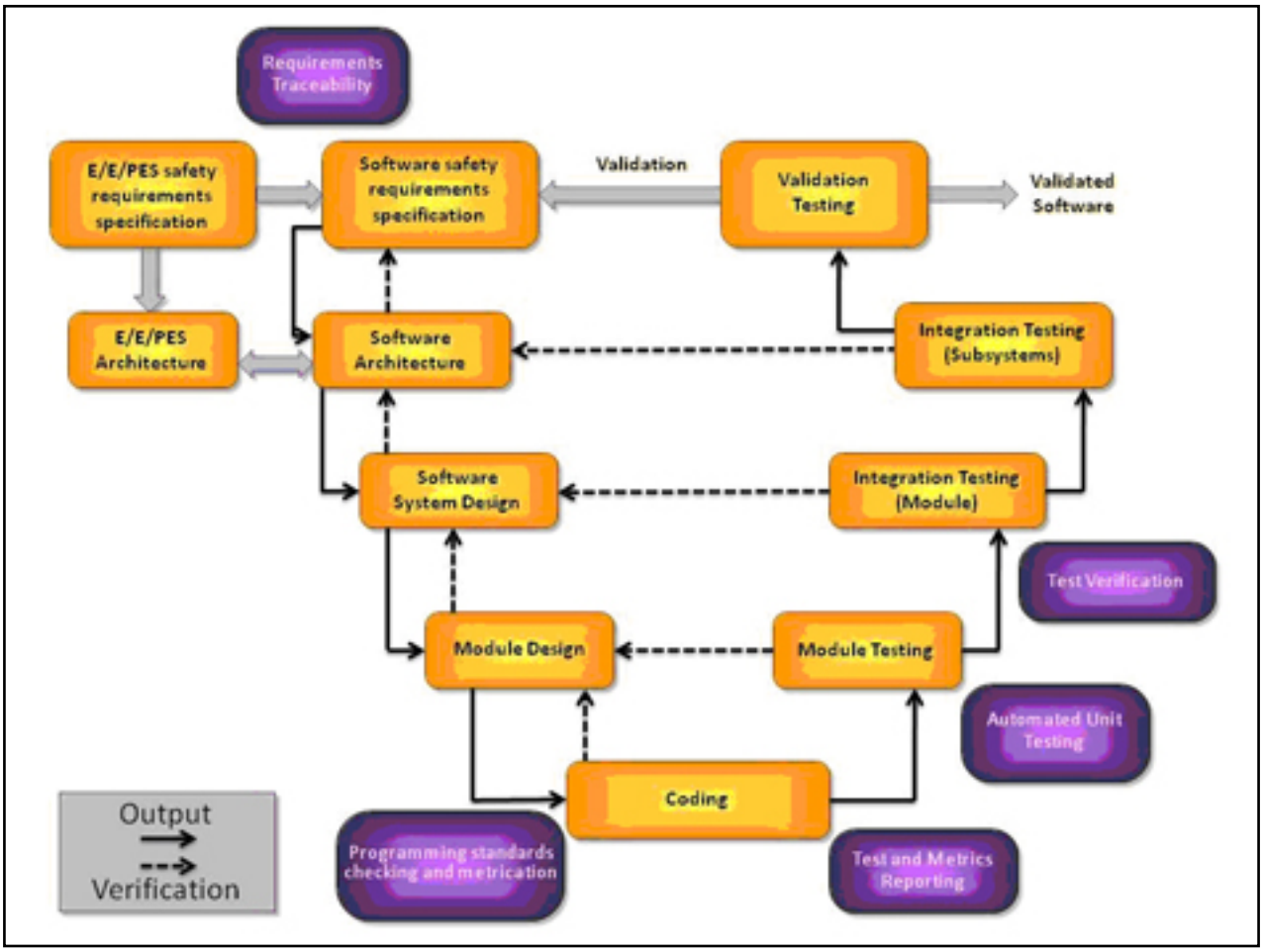
software process necessary to achieve IEC 61508 compliance:

* 7.2 Software safety requirements specification
* 7.3 Validation plan for software aspects of system safety
* 7.4 Software design and development
o 7.4.3 Requirements for software architecture design
o 7.4.4 Requirements for support tools, including programming languages
o 7.4.5 Requirements for detailed design and development – software system design
o 7.4.6 Requirements for code implementation
o 7.4.7 Requirements for software module testing
o 7.4.8 Requirements for software integration testing
* 7.5 Programmable electronics integration (hardware and software)
* 7.6 Software operation and modification procedures
* 7.7 Software aspects of system safety validation
* 7.8 Software modification
* 7.9 Software verification

Starting with requirements specification, developers must specify the requirements for safety-related software of each E/E/PE safety-related system and achieve the SIL specified for each safety function allocated to that system. IEC 61508 specifies four SIL levels with SIL1 demanding the lowest level of safety and SIL4 the most rigorous. Each technique or measure has an associated level of recommendation used to select a safety integrity level. The higher the SIL, the more highly recommended a practice is.

The software safety validation plan details how the software design and development, hardware/software integration and any modifications required achieve standard compliance. Fundamental to IEC 61508’s validation process is bidirectional traceability, a process of linking all aspects of the software development lifecycle together. Bidirectional traceability ensures that each system requirement links to the relating code, tests, verification and documentation and that any change to any of the linked processes transfers information forward and downward through all phases of development. Specifications, and all plans for software safety validation, software modification, software design specification and software verification (including data verification) as information is added or deleted.

The development process starts with detailing the safety requirements as shown in Figure 2.


Requirements Traceability
E/E/PES safety requirements specification
Software safety requirements specification
Validation
Validation Testing
Validated Software
E/E/PES Architecture
Software Architecture
Integration Testing (Subsystems)
Software System Design
Integration Testing (Module)
Test Verification
Module Design
Module Testing
Automated Unit Testing
Coding
Programming standards checking and metrication
Test and Metrics Reporting
Output
Verification

Software safety processes specify safety function and safety integrity requirements. The safety function requirements influence the input/output sequences that perform safety-critical operation such as detection of faults in sensors, actuators, programmable electronics hardware, etc.

The safety integrity requirements of a system are composed of diagnostics and other fail-safe mechanisms which ensure that failures of the system are detected and that the system goes into a safe- state mode if it's unable to perform a safety function.

As with safety-critical system software development, the design is derived from safety requirements—both for the safety critical and non-safety-critical components—to meet the required levels of safety and integrity. During design, if any safety functionality is overlooked at the requirements level, it would potentially compromise the criticality of each software module developed. In order to avoid this, IEC 61508 requires that traceability be established between requirements and software architecture, and the software design specification.

Software architecture further considers selection of the tools including languages, compilers, user interfaces, run-time interfaces, etc., all of which contribute to the safety of the system as per the requirements. The toolset includes verification and validation tools such as static analyzers, test coverage monitors, and functionality testers.

Moving further into the development process, we get into the phase of software implementation. Implemented software should fulfill all the safety functionality described in software architecture and software design specification including complete traceability. The software should be compatible with the programmable electronic target as well.

IEC 61508 enforces static analysis as the first step in comprehensive software quality control. During static analysis, the source code is reviewed against programming standards like MISRA and CERT to detect latent errors and vulnerabilities like array bounds, divide by zero, and uninitialized pointers, which in turn can be exploited during the execution of the software.

In this phase, consistency of the source code is checked to detect the presence of any dead code or uncalled procedures. It determines the quality of the software by measuring metrics like clarity, maintainability, testability and complexity.

Data flow analysis generates a series of analytical verifications of variable usage, procedure interaction and interrupts present in the source code. A control-flow diagram consists of a subdivision to show sequential steps, with if-then-else conditions, repetition, and/or case conditions. Suitably annotated geometrical figures are used to represent operations, data, or equipment, and arrows are used to indicate the sequential flow from one to another. The extent of rigor in enforcing the standard depends on the safety integrity level needed for the safety-related systems.

Table 1 explains about the different techniques/measures for static analysis and its recommendation as per the safety integrity level.

The definitions are:

HR	Highly Recommended
R	Recommended
NR	Not Recommended
—	No Recommendation

Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4
Boundary value analysis	R	R	HR	HR
Checklists	R	R	R	R
Control flow analysis	R	HR	HR	HR
Data flow analysis	R	HR	HR	HR
Error guessing	R	R	R	R
Formal inspections, including specific criteria	R	R	HR	HR
Walk-through (software)	R	R	R	R
Symbolic execution	--	--	R	R
Design review	HR	HR	HR	HR
Static analysis of run time error behavior	R	R	R	HR
Worst-case execution time analysis	R	R	R	R

Table1: List of techniques and measures which are carried out as part of static analysis and its recommendations for each level of SIL.

After static analysis has been done, dynamic analysis is performed in an effort to uncover subtle defects or vulnerabilities. Dynamic analysis is the testing and evaluation of a program by executing data in real-time. The objective is to find errors in a program while it is running, rather than by repeatedly examining the code offline. It is performed at unit, module and system level to achieve all safety functionality at the required level of safety integrity.

The validation test plan is developed by designing test cases for unit, module or system levels to provide complete coverage of the software's functionality. The test cases need to be designed for all input combinations, boundary conditions, ranged values, timing mechanisms, etc. and checked against the expected output to validate the safety functionality of the system. The validation plan and test cases need to be traced back to the requirements to make sure the desired level of integrity and safety functionality is achieved and that complete traceability between requirements and module integration (hardware/software) test specifications, software safety validation plan is in place.

Dynamic analysis needs to be done in two stages:

a) Functionality analysis (Black box testing)

b) Structural code analysis (White box testing)

In black box testing, the test data (inputs and expected outputs) for the test cases are derived from specified safety functional requirements. This phase offers complete requirement-based testing because in this approach only the functionality of the software module is affected by executing the software with the desired level inputs and subsequently checking the results with the expected outputs. Boundary value analysis uses a functional testing technique where extreme boundary values (maximum, minimum, just inside/outside boundaries, typical values and error values) are used in the system to ensure no defects exist.
In white box testing, the structure of the source code is tested by having the inputs exercise all paths though code and determine the appropriate outputs. It tests paths within a unit, paths between units during integration, and between subsystems during a system level test. The design of the test cases is intended to:

a) Exercise independent paths in unit or module
b) Exercise the logical decisions on both true and false.
c) Execute loops at their boundaries and operational bonds.
d) Exercise the internal data structure to ensure their validity.

White box testing includes coverage metrics like entry points, statement, branch, conditions, Modified Condition/Decision Coverage (MC/DC) to make sure each part of the software has been covered and tested against the requirements to complies with the required level of integrity. These coverage metrics are outlined in the safety integrity level of the system.

Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4
Test case execution from boundary value analysis	R	HR	HR	HR
Test case execution from error guessing	R	R	R	R
Test case execution from error seeding	--	R	R	R
Test case execution from model-based test case generation	R	R	HR	HR
Performance modeling	R	R	R	HR
Equivalence classes and input partition testing	R	R	R	HR
Structural test coverage (entry points) 100 %	HR	HR	HR	HR
Structural test coverage (statements) 100 %	R	HR	HR	HR
Structural test coverage (branches) 100 %	R	R	HR	HR
Structural test coverage (conditions, MC/DC) 100 %	R	R	R	HR

Table2: List of techniques and measures which are carried out as part of dynamic analysis and its recommendations as defined by each level of SIL.

During the testing phase, software is tested on programmable electronic hardware to meet all safety functionality. After validating the software's safety, some corrections and/or enhancements may be necessary to ensure compliance with requirements. The enhancements or corrections in the software must also have complete traceability to requirements. All modifications need to be documented along with an impact analysis that determines which software modules and verification activities are affected. Impact analysis determines what modules or functionalities are affected by the change, how many new test cases must be created to cover the new functionality and whether any other system requirements are involved in testing this new change.

After doing the modification in the source code, the complete source need to be regressed by performing regression analysis to make sure earlier safety functionality is not affected by the modification. Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements

Conclusion

The advent of IEC 61508 has brought isolated companies involved in either systems or software development for industrial systems together into the same process just as their counterparts in industries such as aerospace and defense. All disciplines now face the same quality assurance efforts required to achieve compliance with a demanding standard.

The need for such compliance has mandated business evolution in which processes and project plans are documented, requirements captured, implementation and verification carried out with respect to the requirements, and all artifacts fully controlled in a configuration management system.

Adopting IEC 61508 as a process for industrial systems software development requires conformance to the processes, activities and tasks defined by the standard. Fundamentally, IEC 61508 demands that requirement traceability be met at all stages of the software development process.
Qualified, well-integrated tools ensure that developers can automate the process more easily and efficiently.

While moving to automated compliance involves upfront costs and potential change to current practices, companies can achieve higher quality software and compliance to IEC 61508 more easily, reducing costly manual efforts. The higher-quality, safe product avoids expensive recalls and ensures that the same development process can underpin the maintenance and upgrade process. Not only do these factors contribute to the manufacturer's bottom line, but the company achieves significant

ROI in improved credibility and reputation.

Author:

Shrikant Satyanarayan is a Technical Consultant with LDRA, in India, specialising in the development, integration and certification of mission- and safety-critical systems in avionics, nuclear, industrial safety and automotive domains. With a solid background in software development, software testing and real-time operating systems, Shrikant guides organisations in selecting, integrating and supporting their embedded systems from development through to certification.