Keywords: 61508, 61511, functional safety, embedded software, assessment, certification, CASS

Abstract
In recent years we have conducted about 25 assessments using IEC 61508 or IEC 61511, working mainly to Safety Integrity Level (SIL) 2, but on some occasions to SIL 3. In this paper we present some of the lessons we have learned and offer advice to those seeking certification for components, systems or generic process capability. We cover the three main parts of the IEC 61508 standard: Functional Safety (FS) Management; Hardware; Software. More recently, our work has included software products whose assessment has entailed building complex arguments for their compliance. This has led us to use argument structuring techniques that we comment on at the end of this paper.

1 Introduction
Soon after IEC 61508 was introduced there was a UK government funded initiative that introduced the CASS (Conformity Assessment of Safety-related Systems) scheme [CASS] which was intended to provide an industry-wide approach to IEC 61508 assessment and certification. In conjunction with The 61508 Association [T6A] and under their agreement with CASS, we have been developing and using a streamlined set of checklists that enable us to assess lifecycle processes and E/E/PES based products in an efficient manner. The assessment process has been automated to some extent. The results presented here are therefore based on a more collective interpretation of the standard and derived from assessments made during the last five years.

Normally, a staged approach was adopted, namely gap analysis (based largely on answers elicited from structured discussions), remediation, a detailed assessment of the evidence and finally, certification. This approach ensures that major ‘gaps’ in compliance are all identified at an early stage and corrected before further effort is expended.

2 Functional Safety IEC 61508-1 (clause 6)

2.1 IEC 61508-1 Findings
All the companies we audited were operating a certified ISO 9001 quality management system (QMS). However, for those organisations new to IEC 61508, typically, the following situation was found by the gap analysis:

- There was no FS policy or responsibilities identified
- Critical documentation was missing
- There was little use of the various ‘techniques and measures’ recommended in IEC 61508
- Management review omitted FS management
- Competence records of FS staff were absent
- FS audits and assessments were not performed

Where the requirements for management of FS and the QMS overlapped the situation improved. Most organisations were performing the following activities to a greater or lesser degree of rigour:

- Corrective and preventative actions
- Modification procedures
- Supplier assessment process
- Some level of configuration management

2.2 IEC 61508-1 Recommendations
For the management of FS the Standard does not prescribe the criteria for different SILs, although there is informative guidance relating to competence. However, increased rigour and discipline with increased SIL is implicit throughout the Standard.

Requirements such as the FS policy, responsibilities, audits, management review, etc, can easily be incorporated into the existing QMS framework.

The competence of staff performing FS activities is a key element of IEC 61508 and was typically a recurring theme in gap assessments. Attention should be given to the competence criteria required for each task, assessment of individuals’ competence against these criteria and the recording of work successfully performed on FS projects. See reference [HSE].

All the successful clients had assigned an internal FS champion to drive forward the process improvements, oversee competence issues and act as a centre of knowledge for all 61508 technical and compliance matters.

3 Hardware IEC 61508-2

3.1 IEC 61508-2 Findings
In assessing the hardware development process against Part 2, typically we found the main areas of weakness to be:
• The development process was not sufficiently detailed to enable verification of each stage
• There was a general lack of supporting procedures with little or no use made of templates, forms, etc, to create and record design documentation
• A lack of an adequate specification to support all subsequent development and verification activities
• There was no FMEA procedure or report
• A general lack of design reviews, intermediate testing, verification and validation (V&V) plans or documented use of appropriate techniques and measures
• User manuals omitted proof test information and instructions to return failure data to the supplier
• Test plans and requirements specifications had no direct correspondence making traceability difficult
• The field returns analysis process was too weak

At early meetings with organisations seeking certification who were new to IEC 61508, we stressed that embracing the Standard meant a radically different way of undertaking product design. Some were already aware of this, whilst others had hoped they could simply make the product available to the certification body and wait for a certificate on completion, rather like an ATEX certificate.

On a positive note, once one product from a company was successfully certified, subsequent products achieved certification with relatively few defect reports, demonstrating the cost-benefit from a consistent development regime.

3.2 IEC 61508-2 Recommendations

It is important that clients realise the need for evidence of all reviews, verifications, competence, etc. We recommend the effective use of document templates to embed guidance and record decisions such as the selection of the relevant techniques and methods described in the supporting tables of IEC 61508-2. Clients should gain familiarity with these techniques and measures as the Standard imposes stricter requirements in their use with increasing SIL and these form the backbone of safety integrity.

Peer reviews are to be encouraged between development engineers (without management involvement). These need to complement higher level gate reviews.

There should be a process to collect and analyse field failure data. The results of this on-going activity should be used to validate the failure assumptions in the theoretical FMEA.

As FMEA is one of the main methods to establish the vulnerability of the design to hardware failures, it should be used to derive software requirements for diagnostic coverage.

Our overall conclusion is that instituting a compliant development lifecycle can be a difficult and time-consuming activity. The requirements should be fully planned and addressed at the earliest possible stage, preferably before any safety-related design starts.

4 Software IEC 61508-3

4.1 IEC 61511 Software

Our experience of IEC 61511 applications is that they are relatively straightforward and systems integrators requiring assessment are usually sufficiently equipped to have their in-house FS capability certified. This requires significant work on methods and procedures in some firms but is not a major difficulty. Our most recent six assessments have either been passed with a few non-compliances, or, the improvements that have been needed have been simple. In every case the systems integrator is working with special “safety” PLCs from major suppliers whose documentation and tools have greatly assisted the compliance process. Additionally, all the 61511 software assessments have entailed applications of a few thousand lines of source code written in a limited variability language (LVL) such as ladder logic. Hence, other lifecycle issues such as change control, defect tracking, software configuration management, and verification have had fewer demands placed on them compared with IEC 61508 software developments such as those reported on below. In some cases well organised manual procedures using fixed folder structures have been used to support the lifecycle and there has been less need for tool support. This is markedly different from our experience with software lifecycles based on full variability languages (FVL) such as C, which are addressed next.

4.2 IEC 61508 Embedded Software Experience Overview

For components, that is, products whose software has to be developed to comply with IEC 61508-3, we have found the situation to be very different. The following table summarises our general experience with assessing products for which an IEC 61508 certificate has been sought. Products are referred to as P1, P2, etc. The scale of the software embedded in each product is given in KLLOC - thousands of logical (comments and white space excluded) lines of C source code.

<table>
<thead>
<tr>
<th>Type</th>
<th>SIL</th>
<th>Cert</th>
<th>Remarks</th>
<th>KLLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Full</td>
<td>2</td>
<td>Yes</td>
<td>Developed by specialist consultancy</td>
</tr>
<tr>
<td>P2</td>
<td>Full</td>
<td>2</td>
<td>No</td>
<td>Failed, lack of management support</td>
</tr>
<tr>
<td>P4</td>
<td>Full</td>
<td>2</td>
<td>No</td>
<td>Discontinued, hardware non-compliant</td>
</tr>
<tr>
<td>P5</td>
<td>Full</td>
<td>2</td>
<td>Yes</td>
<td>Took two years</td>
</tr>
<tr>
<td>Type</td>
<td>SIL</td>
<td>Cert</td>
<td>Remarks</td>
<td>KLLOC</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>P6</td>
<td>Pre</td>
<td>3</td>
<td>Stalled, inadequate process, legacy issues</td>
<td>1000</td>
</tr>
<tr>
<td>P7</td>
<td>Full</td>
<td>2</td>
<td>Discontinued, technical problems</td>
<td>15</td>
</tr>
<tr>
<td>P8</td>
<td>Full</td>
<td>2</td>
<td>Discontinued, technical problems</td>
<td>15</td>
</tr>
<tr>
<td>P9</td>
<td>Pre</td>
<td>2</td>
<td>Deferred, inadequate process, legacy issues</td>
<td>10</td>
</tr>
<tr>
<td>P10</td>
<td>Pre</td>
<td>2</td>
<td>Deferred, inadequate process, legacy issues</td>
<td>2000</td>
</tr>
<tr>
<td>P11</td>
<td>Full</td>
<td>2</td>
<td>Based on certified system, took one year</td>
<td>35</td>
</tr>
<tr>
<td>P12</td>
<td>Pre</td>
<td>1</td>
<td>Certificate finally not required</td>
<td>N/A</td>
</tr>
<tr>
<td>P13</td>
<td>Pre</td>
<td>2</td>
<td>Discontinued, inadequate process, legacy issues</td>
<td>20</td>
</tr>
<tr>
<td>P14</td>
<td>Full</td>
<td>3</td>
<td>In progress Both process &amp; product need improvement</td>
<td>30</td>
</tr>
<tr>
<td>P15</td>
<td>Full</td>
<td>3</td>
<td>In progress Both process &amp; product need improvement, legacy issues</td>
<td>100</td>
</tr>
<tr>
<td>P16</td>
<td>Pre</td>
<td>2</td>
<td>In progress Inadequate process, legacy issues</td>
<td>10</td>
</tr>
<tr>
<td>P17</td>
<td>Pre</td>
<td>2</td>
<td>In progress Inadequate process, legacy issues</td>
<td>20</td>
</tr>
</tbody>
</table>

The pattern that emerges from this small sample is that even in areas like A3, which should be very familiar to software engineers, there is a significant distance between current practice and the requirements of 61508 (SIL 2). The situation worsens in other phases of the software lifecycle. In those cases where the CASS criteria are worth applying, we use the 38 “targets of evaluation” (ToEs) identified in the CASS guidelines drafted by the 61508 Association. We have refined these into 195 more detailed criteria (“subToEs”).

### 4.4 General Lessons of the unsuccessful Cases

A1 Software Requirements Specification: often missing altogether. When available, requirements were not traceable through the lifecycle, making it impossible to show coverage, particularly at the acceptance test (product validation) stage. There are also particular concerns about the mathematical specification – which is also one of the weaknesses of the standard itself in that it lacks guidance on the choice of appropriate algorithms and their error, accuracy and stability, the validation of algorithms, and the safe use of floating point. Our experience with the mathematical specification has shown that the incidence of defects is higher there than in other areas of the requirements.

A2 Software Architecture Design: too often there is a lack of linkage between the hardware and software, despite the fact that there are diagrams in the standard that illustrate the point. Hardware FMEA is a rich source of software requirements in that fault events have to be defended against by the software.

A3 Software Design and Development - Support Tools and Programming Language: there is little awareness of the value of the techniques collectively known as static analysis. Even coding standards are often missing.

A4 Software Detailed Design: many safety-related applications are amenable to the wide use of finite state machines and a good number of developers are aware of their value both at the requirements specification and design stages. However, methods of checking the completeness of...
FSMs, for example, sequence enumeration [Prow] are not well known.

A5 and A6 – Unit and integration testing are very weak – often, unit testing is omitted, and unverified (neither tested nor reviewed) software is integrated to form a system ‘build’ in the hope that product level tests will suffice. There is a lack of awareness of the value of metrics such as function complexity that can be used to focus unit testing.

A7 Software Validation – this links to the weaknesses in requirements tracing and the failure to write the acceptance test specification early in the project when the requirements are specified.

A8 and A9 Verification and modification both arise from a lack of a review culture; and weaknesses in testing, software configuration management (SCM), and change control. We have particular concerns about the absence of effective SCM, not just in software but in firmware too. For example, ASICs programmed in VHDL must be under SCM. We have discovered ASIs for which the VHDL source has been lost, rendering an entire product uncertifiable.

4.5 The Need for Tool Support

Effective and relatively inexpensive software tools are available to support many software lifecycle activities. However, we have had to introduce development teams to even the simplest and least expensive of these valuable aids to safety, quality, and productivity. Developers should automate as many repetitive and low skill operations as possible. We would encourage those seeking IEC 61508 certification to install the following tools in their software lifecycle:

Software Requirements Specification: we find that for a system with a few hundreds of software requirements a well-structured document containing uniquely tagged, comprehensible and testable requirements is manageable, provided that traceability is automated either by a specialised requirement tracking/tracing tool or via a traceability matrix (although such matrices are bulky and lend themselves to being generated automatically from a requirements database, rather than being manually built in spreadsheets).

Design Tools. The standard encourages simplicity: a fixed number of tasks, minimal interrupts, and restricted concurrency, deterministic scheduling, where possible. In the systems we have assessed it is possible to make wide use of finite state machines (FSMs) as a semi-formal methods of design, at both architectural and detailed levels. Tools are needed for checking FSMs for completeness and consistency. Code Development and Support Tools: static analysis is essential yet some development teams are unaware of its benefits. A range of analysis techniques is needed [Henn]:

<table>
<thead>
<tr>
<th>Analysis Technique</th>
<th>Shallow</th>
<th>Deep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Data Flow</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Range Checking and Unsafe code detection (indexing faults, pointer faults, etc)</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Stack Usage Analysis</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Additionally, there has to be greater awareness of the need for validation of all tools that affect the integrity of a software build, either from confidence in use arguments, validation testing or both. Nearly every one of our clients has had issues with compiler or library defects at some time or the other, which points to the need for caution in selecting code optimisation options, using libraries, and upgrading a tool from a well used trouble-free version.

Testing tools should support generation of test cases as demanded by the standard. A key capability is the automatic running of regression tests both at the unit (module) level and at the integration test level.

Integration testing: in a number of cases we have found that strict unit (module) testing can be eased if the following measures are taken:

- Static analysis of code is always performed
- Code is peer reviewed in detail prior to unit test, particularly the assumptions concerning pre- and post conditions

The difficulties arise at the integration stage where module interfaces are found to be defective, or the wrong assumptions have been made about them. This experience has been useful in guiding the selection of tools.

At the product validation level some practitioners use automated acceptance test rigs to execute traceable acceptance tests and provide semi-automatic generation of test reports and archiving of the results under SCM.

Lifecycle infrastructure tools: SCM, defect and issue tracking, change note and impact analysis tracking can all be facilitated by tools that are widely available. Work flow tools (of which some are open source) are used to support reviews as well as issue and defect tracking. In some cases reviews are captured on wikis implemented in the company intranet, which prove more visible and effective than paper based review minutes.

4.6 Legacy Code Issues

This has been the most difficult aspect of gaining certification. Leaving aside ASIC source code and other SCM related issues, we find the following situation typical:

- Many thousands of lines of source written over many years unaccompanied by design information
- Mixtures of in house developed and purchased components
- Lack of a compliant software lifecycle: in some cases SCM is not adequate with inadequate version control of the compiler and other tools
- Safety-related and non-SIL programs sharing the same address space

Here, it is not cost-effective to retrospectively bring the product up to standard. We find that in such cases there is a choice:
• For small bodies of code, to completely develop the product from scratch using a 61508 compliant product lifecycle
• For large volumes of legacy code, particularly in distributed, networked products, to consider developing a small, self-contained SIL-rated product that in effect acts as a safety-monitoring and emergency shut down device for the main legacy product.
• A successful certification of legacy code can only be achieved when that code has been developed within a rigorous quality control system. In such cases it is necessary to study the code, documentation, and recorded defect history accumulated over the millions of operating hours required. One must establish whether it is possible to construct a sound safety argument on the basis of valid operational use and validated defect reports for at least an older, well-established version of the software. We have had success with this approach in some products.

4.7 Lessons from Successful Assessments

In the successful cases an incremental approach was used, to put it informally: a sequence of “mini-waterfalls” was followed during which software releases of increasing capability were developed. The standard was applied throughout because early in the project, at the pre-assessment stage, we had identified non-compliances in the software lifecycle and had monitored their remediation. Project P5 is an example of one that started with an inadequate lifecycle unsupported by tools. It took two years to bring the lifecycle, its tool support, and the product up to the necessary standard.

As for the general lessons learned, it is noticeable that:
• Great effort was made to understand the requirements and to share that understanding through prototyping certain aspects of the software and gaining feedback from the stakeholders in the product. Whether the prototype was simulated, or based on actual hardware, it was well understood that it was just a prototype and was not a suitable basis for a safety related product.
• Investment was made in investigating appropriate tools and techniques prior to taking purchase decisions, particularly their trial use, to establish usability and compatibility with the needs of the development team.
• A well scrutinised approach was taken to all stages of the software lifecycle with intense use of reviews
• Management understood the issues and were willing and able to commit funds for tools, training, and the development of good practice in the company.

5. Recommendations on Software Issues

We are struck by the gap, on the one hand, between what can be achieved, for example by practitioners using the “Correctness by Construction” approach [Amey] or Analytical Software Design [Broy], where in each case demonstrably defect free software can be produced with high rates of productivity, and, on the other hand, by what we have found in industry in recent years. We are also aware of the powerful criticisms from within our industry by workers such as Martyn Thomas [Jack] of relying on software lifecycle arguments alone. Hence it seems to us that more has to be done to raise the quality of our industrial software practice in the realisation of IEC 61508 (rather than IEC 61511) software. Before making any specific points we would also like to stress that, for example, the automotive industry is also confronting these issues, and, with the advent of ISO 26262 (an adaptation of IEC 61508 to automotive systems) we should be particularly aware of their experience [Broy].

Our specific recommendations for future action are:
• Industry, perhaps through an organisation such as The 61508 Association, should develop and make available template documents that constitute the main elements of a generic compliant embedded software lifecycle that can be adapted to the needs of newcomers to IEC 61508
• Greater awareness of the benefits of the “Correctness by Construction” approaches. [Amey], [Broy]
• For companies whose products entail many software variants that are updated during the life of the product, a greater awareness of the value of model based design, as the basis of software reuse. A new product often differs in functionality by less than 20% compared with its predecessor, yet its software reuses but a fraction of the previous product software, even where there is an awareness of reuse
• The elicitation and specification of requirements is still one of the key issues. The means for expressing complex requirements such as interfaces that have to be fulfilled are not adequate. For example, although UML has useful diagramming notations it is not based on sufficiently formal semantics that we can use to reason about the interface. In this respect more needs to be done to support the rigorous use of well known methods like FSMs, for example as in [Broy] or to use semantically more precise subsets of UML
• We need a means by which developers of safety-related applications can use improved techniques to implement their products. At present the market is largely locked into using C as the main implementation language for these applications and it is clear to us that such an approach is becoming more questionable as the size of these applications continues to increase. We recommend the adoption of techniques and tools more suitable than ANSI C (and its associated tools). The IEC 61508 standard highly recommends using a safe subset of a programming language, and MISRA C is often used in that role. However, we suggest going beyond that by using an annotated subset of C to achieve verifiable proof conditions as outlined in [Croc]. This approach would mirror the successful “Correctness by Construction” approach using Spark Ada [Barn]; we encourage funding bodies to support the development of these tools
• The previous point relates to the need for wider and deeper use of static analysis tools
• The adoption of suitable testing tools presents a major hurdle to many clients. Assistance is needed with
selecting the right techniques, generating test cases (preferably by automatic means rather than writing test drivers), and producing test reports as evidence of testing, and further regression testing after modifications.

6. Assessment of Complex Software Products

Recently, we have been assessing products for which the compliance arguments are based on a blend of confidence-in-use data, process criteria (CASS targets of evaluation), and verification results. In particular where earlier versions of a product have not complied with IEC 61508, but an equivalent level of safety has been reached. The strategy followed typically has three stages:

1. Confidence in use argument for historical releases of the product supported by strong evidence of licences sold, actual use, field reports, and previous third party assessments

2. The bringing of the product to a compliant baseline state using improved methods, retrospective V&V, retrospective use of the product safety manual with respect to defects, and whatever historical data and field reports are available

3. The development of future releases of the product from its compliant baseline using methods that comply with the standard

The framing of these arguments has been complex and we have been greatly helped by using goal oriented techniques as an aid to structuring them. We considered goal structuring notation, see, for example, [Kell] but in the end opted for Bayesian networks, in for example, [Fent] primarily because our clients happened to be more familiar with the technique. This has led us to building several networks to express and analyse the arguments. What has struck us is the way that we have had to nest them to two or three levels in some cases, as in this fragment that forms part of a stage 2 argument that the design of the baseline is compliant:

Memory use analysis breaks down into further levels as of course does the CPU time and scheduling analysis.

We hope to report in more detail on the uses of these techniques in later publications.

Acknowledgements

We thank the many engineers with whom we have raised and discussed the issues presented in this paper.

References


