THE SAFETY INTEGRITY LEVELS OF IEC 61508 AND A REVISED PROPOSAL

M. A. Hennell[1], J. C. P. Woodcock[2] and M. R. Woodward[3]

1 LDRA Ltd., Portside, Monks Ferry, Wirral CH41 5LH, U.K.
E-mail: michael.hennell@ldra.com, Tel: +44 (0)151 649 9300, Fax: +44 (0)151 649 9666

2 Department of Computer Science, University of York, Heslington, York YO1 5DD, U.K.

3 Department of Computer Science, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, U.K.

Keywords: standards, safety integrity levels, defect spanning sets, fault spanning sets.

Abstract

The paper proposes a new scheme for safety integrity levels (SILs) based on reasoned principles. The scheme provides a mechanism for selecting appropriate verification and validation techniques for a given SIL and, in particular, suggests a replacement for the existing technique selection tables of the IEC 61508 standard.

1 Introduction

Like many software standards for safety-related industries, the IEC 61508 standard for generic programmable-electronic safety-related systems [6] employs the concept of *safety integrity levels* (SILs), whereby the level of criticality of the software is used to determine aspects of the software development process, and, in particular, the extent of the verification and validation (V&V) effort deemed appropriate. The standard requires that each safety function is allocated to one of four SILs, with SIL 1 being the lowest and SIL 4 being the highest. The SILs themselves are defined in terms of ranges for the average probability of failure on demand (pfd) for a low demand mode of operation, or the probability of a dangerous failure per hour for a high demand or continuous mode of operation (see Table 1). Appropriate V&V techniques are then selected for the chosen SIL from tables of techniques having varying strengths of recommendation, namely: highly recommended (HR), recommended (R), not recommended either for or against (–), not recommended (NR). See Table 2 for the IEC 61508 recommendations for the activity of code verification. Some of the techniques (e.g. static analysis) are decomposed into sub-techniques that are listed in further similar tables. However, it is not immediately obvious how use of a particular set of techniques relates to the target failure probability.

The advantage of the new SIL scheme proposed here is that the selection of V&V techniques becomes much more obvious and, what is more, underpinning it there are reasoned principles that justify that selection. Thus, for a given application, it is possible to consider the types of faults that might be present and provide a safety case which argues that the set of validation techniques has a high probability of detecting all of those faults. Moreover, it is possible to argue that specific sets of techniques have this property over wide classes of applications. All the current techniques of static analysis, dynamic analysis and formal methods sit neatly into the concepts. A comprehensive discussion of these ideas can be found elsewhere [4]; this paper concentrates mainly on their relationship to IEC 61508.

SIL	*Low demand mode: average probability of failure on demand*	*High demand or continuous mode: probability of dangerous failure per hour*
1	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$
2	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
3	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
4	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$

Table 1: Definition of IEC 61508 SILs.

Technique	*SIL 1*	*SIL 2*	*SIL 3*	*SIL 4*
Formal proof	–	R	R	HR
Probabilistic testing	–	R	R	HR
Static analysis	R	HR	HR	HR
Dynamic analysis and testing	R	HR	HR	HR
Software complexity metrics	R	R	R	R

Table 2: IEC 61508 recommendations for code verification.

2 Terminology

First, the distinction is made between errors, faults, failures and defects. The essence of the definitions for errors, faults and failures, as standardised by the IEEE [5] will be followed, since these have been widely adopted by the software

engineering community. The definition of a defect seems less widely agreed upon. Here, the class of defects will be regarded as a broader class of problems than faults since it may also include deviations from some extra rules, such as a well-defined, safe language subset [3] as may be found in the MISRA C standard [9] for example.

- An *error* is a mistake made by a person when developing a software system. For example, it might be a misunderstanding of what is required of the software, so that the statement of requirements is incorrect, or it might be a mistake when translating a design into actual program code.

- A *fault* is a deviation in some software-related artefact, most usually the program code, from what is required of it in order to provide correct functioning behaviour.

- A *failure* is the manifestation of a fault, when faulty software is executed under conditions that cause the fault to be observed.

- A *defect* is a deviation in some software artefact from what is required either for correct behaviour or to satisfy some additional criterion that is imposed upon the representation.

3 Defect spanning sets

A *defect spanning set* (DSS) is a set of one or more V&V techniques that collectively have the potential to detect the totality of all defects in the system under test. In practice, it is impossible to demonstrate that any set is exactly a DSS, but elsewhere [4] the authors have used two approaches to overcome this problem: firstly, reasoned argument and secondly, actual experience.

The most obvious source of V&V techniques to form a DSS is to combine those techniques which generally come under the title of static analysis and formal methods. Such a collection is application dependent because a DSS is an application-dependent concept. There is, for instance, no need to include a technique to detect those defects which cannot appear in the software. In general, unit testing tends to be a defect detection technique.

4 Fault spanning sets

A *fault spanning set* (FSS) is a set of one or more V&V techniques that have the potential to detect the totality of all faults in a system under test. Given that the set of defects includes the set of faults, it should be clear that a DSS is also an FSS, but not the converse. As with the notion of a DSS, the FSS concept represents an ideal whose achievement may be impossible to demonstrate. Nonetheless, close approximations may well be achievable.

The authors have identified two potential FSSs in addition to the one DSS mentioned in the previous section. They are dynamic analysis with high coverage levels and statistical testing. Dynamic analysis has a good track record as a fault-finding mechanism, but statistical testing is normally used to demonstrate the absence of faults or at least to show that they have a low probability of occurring.

That either of these two possibilities is independent of the DSS is probably not contentious, but it is not clear to what extent they can be considered to be mutually independent. The basis underlying each is, in principle, quite different. In dynamic analysis one attempts to construct test data which explores the requirements to the extent that certain structural coverage metrics are met whereas the statistical testing strategy samples from a model of use to produce test data which is representative of the actual inputs to the system, i.e. an operational profile [10, 11]. Both techniques need knowledge of the requirements of the system but with a different emphasis. That there is some independence, particularly if they are performed by different groups, is again probably uncontentious; it is only the extent to which they might be considered to be independent that is an issue.

5 SIL scheme proposal

This section outlines the proposed scheme for safety integrity levels; it is comprised of five levels with SIL 0 being the lowest and SIL 4 being the highest. The proposal is based on the concept of fault spanning sets, as just introduced, together with the familiar notion of redundancy. Triple redundancy has always been seen as the ideal for safety-critical hardware, and so the same notion has provided the rationale for these definitions of SILs for safety-critical software.

SIL 0: This level is for software systems having no criticality at all, so there are no dependability requirements.

SIL 1: This level is for software systems that should possess high dependability, but are not critical. Such systems require a high degree of checking. A demonstration of conformance to requirements is required.

SIL 2: This level is for software systems that are, in some sense, critical and must have a demonstration that all fault types have been checked. At least *one* FSS must be applied.

SIL 3: This level is for software systems that are mission-critical or safety-related. Here, a high level of certainty is required, so the principle of redundancy is applied. In other words, at least *two* independent FSSs must be used to check for the presence of each type of fault.

SIL 4: This level is for software systems that are safety-critical. Here, at least *three* independent FSSs must be used to check for the presence of each type of fault.

High SILs demand that the requirements are checked thoroughly, preferably by several mechanisms. The basis of the proposal is that three of the levels (namely SILs 2, 3 and 4) should be satisfied by exploiting from one to three independent FSSs. In principle, any one of the FSSs should be appropriate for SIL 2, but there is the additional constraint that there should be a demonstration that the software performs according to its requirements. This constraint is normally discharged by executing the software with high quality test data that may well constitute an FSS itself. That being the case, it is therefore sensible practice to utilise the one FSS for both tasks. With regard to SIL 1, there is again a need to demonstrate conformity with requirements, although the achievement of a high level of dependability is not paramount.

For SIL 3 the requirement is for two FSSs. This, therefore, suggests that two out of the three primary FSSs should be used. A common sense strategy is to use static analysis as one and dynamic analysis as the other, as required, for example, in Def-Stan 00-55 [8] that mandates use of both approaches. The use of dynamic analysis and statistical testing is possible, though this does place total reliance exclusively on dynamic approaches. The combination of static analysis and statistical testing tends to leave more uncertainty with respect to the possibility of residual faults in the binaries: this is because statistical testing may result in poor coverage of infrequently-used portions of code, thereby putting more reliance on static analysis to detect any faults in those regions. Finally, for SIL 4, all three FSSs must be used, to achieve the best possible likelihood of discovering all faults.

The approach outlined above is to be contrasted with other SIL schemes. In the proposal here, the reasoning is that all faults should be detectable at each of the three critical SIL levels. Then, because of uncertainties in the completeness of any given FSS, diversity is exploited by using more than one FSS. Other SIL schemes drop techniques, and hence the ability to detect their associated types of faults, as one goes down through the levels.

6 Relationship to reliability

Consider the use of three FSSs as proposed for SIL 4 in the new scheme. Let p_i be the probability of failure achieved by use of the *i*th FSS. Then, assuming the three FSSs are independent, the resultant probability of failure will be $p_1 \times p_2 \times p_3$. If, for example, each p_i individually is 10^{-2} the resultant combined probability has the improved value of 10^{-6}.

Of course, estimating the p_i that results from using each of the three FSSs is likely to be a very difficult task. For dynamic analysis, tentative steps have been taken to provide a formula relating reliability to the value of given coverage metrics [13]. For static analysis and formal methods, analysis of the evidence from a number of substantial case studies [14, 12] may be the route to suitable estimates. Statistical testing by definition can be used to determine reliability [7].

A further possible approach results from consideration of the DO-178B standard for civil avionics software [2]. An argument can be presented [4] that the V&V techniques inherent in DO-178B must be reasonably close to forming two FSSs, one being dynamic analysis and the other basically being static analysis. It is evident from experience that the DO-178B standard is very effective and possibly the achieved reliability is close to the level required by the equivalent SILs in IEC 61508.

7 Comparison with IEC 61508

The principle problem with IEC 61508 [6], and indeed also with DO-178B [2], is that as the criticality of the software rises, new techniques are required to be applied. Yet these new techniques discover specific classes of defects and there is no reason advanced as to why these faults are less significant for the lower SIL software. The underlying philosophy is more that as the criticality rises, the reasons for avoiding the more onerous techniques become less significant.

The current proposal is that the new SILs are assumed to be equivalent to the levels in IEC 61508 but equally they could be considered to equivalent to the levels of DO-178B with its level *A* being comparable to the SIL 4 of Section 5 and its level *E* being comparable to SIL 0.

The proposal advocated in this paper is based on the need to reduce faults in all software and still have some assurance that the number of residual faults in higher-criticality software is reducing.

8 Future developments

The principal problem with the new proposal is that there is still no clear relationship between the proposed SILs and the requirement for specific levels of reliability or dependability. However, Section 6 has provided some indication of avenues for further exploration.

Other problems are related to the claim that a given set of techniques is, or is not, a close approximation to a defect spanning set or a fault spanning set. Proving that a set of techniques is *not* an FSS is considerably easier, but obviously, as more techniques are included, this gets more difficult. However, a failure to find a fault does not necessarily imply that the techniques used constitute an FSS.

As an example, suppose one wished to assess how close the dynamic analysis requirement of DO-178B was to being an FSS. At the highest SIL, modified condition/decision coverage (MC/DC) [1] is required. It would be possible to conduct an empirical study to determine how effective MC/DC-adequate test data was at detecting mutation-generated faults. As far as the current authors are aware, no such study has yet been performed.

9 Conclusions

The authors have presented a reasoned case for revising the basis on which safety integrity levels for safety-critical, safety-related and mission-critical software are apportioned to fault and defect detection techniques.

The proposal has the basis of some scientific underpinning and looks capable of further development. The three proposed FSSs are practicable, in that they have all been used in some form over many years, and they have the merit that the only issue to be decided for a given project is whether they have the capability of finding the faults which might be present in the software. It is also possible to conceive that a generic set of V&V techniques exist which will detect the faults in most classes of software. These can then be augmented by specific techniques which will expose those faults which might be present in special classes of software.

Acknowledgements

This work has been financially supported by British Energy Generation Ltd, British Energy Generation (UK) Ltd, British Nuclear Group (Magnox) and British Nuclear Group, Management Services, Sellafield. It has been managed by the Control & Instrumentation Nuclear Industry Forum (CINIF).

References

[1] J.J. Chilenski, S.P. Miller. "Applicability of modified condition/decision coverage to software testing", *Software Engineering Journal*, **9**(5), pp. 193-200, (1994).

[2] European Organisation for Civil Aviation Equipment. *Software Considerations in Airborne Systems and Equipment Certification (ED-12B/DO-178B)*, EUROCAE, 17 rue Hamelin, F-75783 Paris Cedex 16, France, (1992). Available at http://www.eurocae.org

[3] L. Hatton. "Safer language subsets: an overview and a case history, MISRA C", *Information and Software Technology*, **46**(7), pp. 465-472, (2004).

[4] M.A. Hennell, J.C.P. Woodcock, M.R. Woodward. "A proposal for software safety integrity levels based on fault spanning sets and principles of redundancy", Submitted for publication, (2006).

[5] Institute of Electrical and Electronic Engineers, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, Corrected Edition, IEEE, New York, (February 1991).

[6] International Electrotechnical Commission, *Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems (IEC 61508)*, International Electrotechnical Commission, 3 rue de Varembé, Geneva, Switzerland, (1998). Available at http://www.iec.org.ch

[7] S. Kuball, J. May. "Test-adequacy and statistical testing: Combining different properties of a test-set", *Proceedings of the 15th International Symposium on Software Reliability Engineering*, St. Malo, France, November 2004. IEEE Computer Society Press: Los Alamitos, California; pp. 25-34.

[8] U.K. Ministry of Defence. *Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment*, Issue 2, (August 1997). Available at http://www.dstan.mod.uk/

[9] Motor Industry Software Reliability Association, *Guidelines for the Use of the C Language in Vehicle Based Software*, Motor Industry Research Association, Watling Street, Nuneaton, Warwickshire CV10 0TU, U.K., (1998). Available at http://www.misra.org.uk

[10] J.D. Musa. "Operational profiles in software-reliability engineering", *IEEE Software*, **10**(2), pp. 14-32, (1993).

[11] J. Musa, G. Fuoco, N. Irving, D. Kropfl, B. Juhlin. "The operational profile", Chapter 5 in *Handbook of Software Reliability Engineering*, M.R. Lyu (Ed.), McGraw-Hill: New York; pp. 167-216, (1996).

[12] S.L. Pfleeger, L. Hatton. "Investigating the influence of formal methods", *IEEE Computer*, **30**(2), pp. 33-43, (1997).

[13] A. Veevers, A.C. Marshall. "A relationship between software coverage metrics and reliability", *Software Testing, Verification and Reliability*, **4**(1), pp. 3-8, (1994).

[14] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J.P. Hudepohl, M.A. Vouk. "On the value of static analysis for fault detection in software", *IEEE Transactions on Software Engineering*, **32**(4), pp. 240-253, (2006).