TOWARDS A CALCULUS FOR SOFTWARE SIL

R.C. Short*

* Atkins Rail Limited
Euston Tower
286 Euston Road
London NW1

E-mail:roger.short@atkinsglobal.com

Keywords: Software, safety, complexity

Abstract

Safety Integrity Level (SIL) is very widely used to define the safety properties of software, but because it does not take account of the scale of software in terms of size and complexity it can give a misleading impression of the potential unsafe failure rate. This paper proposes an approach to assessing the SIL achieved by software so as to include the effects of scale.

1 Introduction

In many industrial applications assurance that software is fit for the safety functions which it is required to perform is provided by showing that it complies with the requirements for an appropriate Safety Integrity Level (SIL) as defined by IEC 61508 [6] or one of its industry specific equivalents such as EN 50128 for railway applications [2].

There have been a number of criticisms of the SIL concept, including that it is over dependent on engineering judgement, it is related more to software quality than to safety, and that the term SIL is often used loosely [7,8,9]. These criticisms are soundly based, but the standards compliance/SIL approach is now so firmly embedded and so convenient to suppliers and users that there is little prospect of it being abandoned or fundamentally changed in the foreseeable future.

This paper is concerned with another problem with the way that SIL is interpreted: the absence of a concept of scale when assessing the SIL of software. In effect, the various SIL-based software standards all say that if any of the sets of techniques or measures which they recommend for a particular SIL are applied software will achieve that SIL. Since the standards make no mention of the scale of the software, in terms of size, complexity or any other dimension, this is tantamount to saying that, regardless of scale, the same result will achieved whether interpreted as a failure rate, probability or level of confidence.

Not only is this at variance with the entire science of software metrics but it is also contrary to the engineering judgement and experience on which the technical content of the standards is based.

This paper proposes an approach to estimating the SIL achieved by software taking account of factors related to scale so that SIL can be used to reason about the expected safety performance of the software.

2 The Meanings of SIL

IEC 61508 defines SIL as a discrete level (one out of a possible four) where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 has the lowest, and it defines safety integrity as the probability of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time.

IEC 61508 also includes a famous table relating SIL to dangerous failure rate, reproduced below as Table 1. A careful reading of IEC 61508 shows that the standard does not actually claim that by following its recommendations for each SIL for software the corresponding probabilities of dangerous failure will actually be achieved, but it is widely[1], if incorrectly, interpreted as saying this.

Safety Integrity Level	Probability of dangerous failure per hour
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

Table 1: SIL and Failure Probability in IEC 61508

The railway industry standards EN 50128 and EN 50129 [3] are careful to avoid mentioning probability in the definition of SIL, which they define as a number which indicates the required degree of confidence that a system will meet its specified safety functions with respect to systematic failures,

[1] For example, the author has encountered this interpretation on numerous occasions when reviewing safety cases associated with major railway projects.

and the table in EN 50128 which is the equivalent of Table 1 above uses descriptive terms ranging from "low" to "very high" in place of the probability values. Confidence is used by these standards in its plain language sense and not with a mathematical meaning which can be evaluated or used in calculations, but it is true in practice that people are confident to use software which they are assured has achieved an appropriate SIL.

Despite the differences in their approaches to quantification, the IEC and EN standards both use SIL in two ways:

a) As a measure of the *desired* probability or degree of confidence that software *should* achieve in satisfactorily performing the required safety functions (shown as SIL(R) below).
b) As a *prediction* of the probability or degree of confidence that software *will* to achieve in satisfactorily performing the required safety functions (shown as SIL(A) below)

A number of techniques, such as the use of risk graphs, are available for (a). They are all based on assessment of the risks of the application and, subject to all the uncertainties inherent in reliance on human judgement and fuzzy and incomplete knowledge of risk, are consistent with the probability calculus.

Prediction of the expected performance as in (b) is customarily based on showing that the software has been developed in conformity with one of the sets of techniques and measures recommended for the required SIL by the respective standard. This is summarised by Figure 1 below, where the achieved SIL is equal to the required SIL irrespective of the scale of the software.

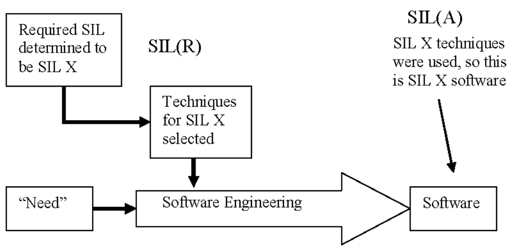

Required SIL determined to be SIL X
SIL(R)
Techniques for SIL X selected
SIL(A)
SIL X techniques were used, so this is SIL X software
"Need"
Software Engineering
Software

Figure 1: SIL(A) = SIL(R) by definition

It is the proposal of this paper that a distinction should be made between the required and achieved SILs, and that the achieved SIL should be estimated by taking the SIL rating as determined by compliance with the recommendations of the relevant standard and adjusting it by means of factors which take account of size, complexity and the nature of potential failures, as summarised in Figure 2 below.

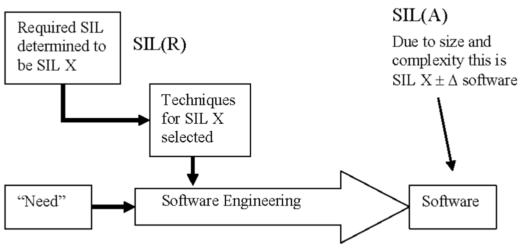

Required SIL determined to be SIL X
SIL(R)
Techniques for SIL X selected
SIL(A)
Due to size and complexity this is SIL X ± Δ software
"Need"
Software Engineering
Software

Figure 2: SIL(A) ≠ SIL(R)

3 Defects and Failures

The various ways of defining SIL are concerned with the way in which software behaves in relation to safety functions rather than with the number of defects in the software. For the purposes of this paper a defect is taken to be any feature of software which will result in failure, while for failure the common sense definition used by the American Society of Civil Engineers, "failure is an unacceptable difference between expected and observed performance" is eminently suitable. Viewed in this way, "defect" includes not only lines of code which cause the software to behave in a way not intended by the designer, but also omissions and wrong intentions in the design or specification.

A defect will only result in a failure if conditions occur which cause the defective code to be executed (or in the case of omission require the presence of code which is not there). The distribution of failures which is exhibited by the software will thus depend on both the distribution of defects within the software and the distribution of demands made on the software. The distribution of demands is likely to be imperfectly understood, while the distribution of defects will almost certainly be unknown, so the prospects for accurately predicting the distribution of failures are likely to be slim.

For software to be acceptably safe it is the probability of unsafe failures rather than the number of defects which must be made acceptably low. The extent to which failures can be classified as safe or unsafe is very dependent on the application. For example in railway signalling systems, whose primary aim is to prevent collisions between trains, any failures which tend to stop or prevent train movements are regarded as safe: an unsafe failure is one which permits a train to move when it should not. In applications in other industries different aspects of software behaviour may be important and it may be less easy to designate any failures as safe.

Although in many cases not all failures will be unsafe, the software engineering measures and techniques recommended by the standards are almost universally concerned with reducing the number of defects, regardless of whether the failures which they cause are likely to have an adverse effect on safety. However, whilst the majority of generic software engineering techniques may not inherently differentiate between defects leading to safe or unsafe failures it may be possible to focus some techniques, especially those concerned with analysis or testing, on the avoidance or removal of defects which cause unsafe failures in a specific application.

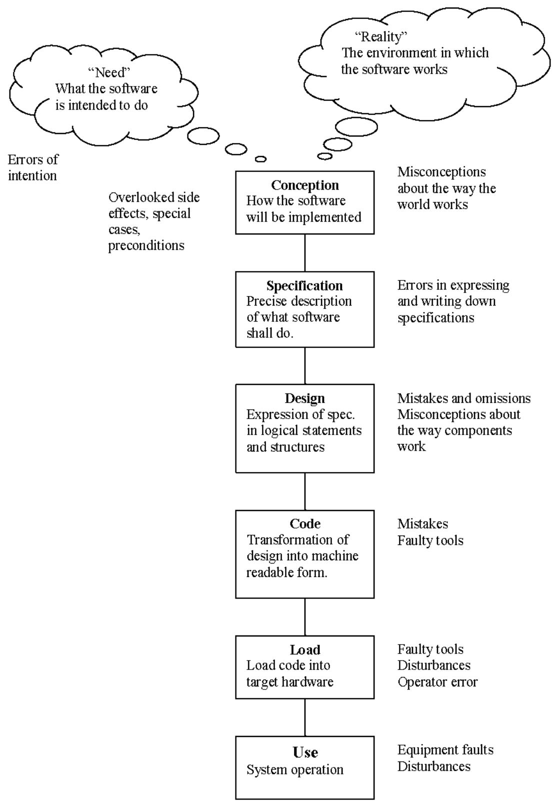
4 Size, Complexity and Software Defects

4.1 The Origin of Defects

Some of the principal causes of software defects are summarised in Figure 3. Most defects are the result of human error or misunderstanding in the various activities which transform the need into the product which is brought into use, although it is also possible for defects to be caused by the behaviour of software tools or by physical disturbance leading to corruption of the software.

It can be seen that software is likely to contain defects due to a number of causes: human error in performing software engineering activities, misunderstanding or ignorance of the application and its environment, and technical causes.

The number of defects due to human error in software engineering activities could be estimated as the product of the error rate per action and the number of actions to be performed. The error rate will depend on the competence of the people, the techniques used and the complexity of the software, while the number of actions will depend on both the size and the complexity of the software.



"Need"
What the software is intended to do
"Reality"
The environment in which the software works
Errors of intention
Overlooked side effects, special cases, preconditions
Conception
How the software will be implemented
Misconceptions about the way the world works
Specification
Precise description of what software shall do.
Errors in expressing and writing down specifications
Design
Expression of spec. in logical statements and structures
Mistakes and omissions
Misconceptions about the way components work
Code
Transformation of design into machine readable form.
Mistakes
Faulty tools
Load
Load code into target hardware
Faulty tools
Disturbances
Operator error
Use
System operation
Equipment faults
Disturbances

Figure 3: The Origin of Defects

4.2 Complexity

A small digression on the nature of software complexity will make clear how it affects both the error rate and the number of actions. In plain English "complexity" is defined as "the state or quality of being intricate or complex". In turn, the dictionary [1] includes among its definitions of "complex": "**1** made up of various interconnected parts; composite. **2** (of thoughts, writing, etc.) intricate or involved." The first definition given by the same dictionary for "intricate" is "difficult to understand".

Both definitions of complexity are relevant to the existence of software defects. Structural complexity ("made up of various interconnected parts"), e.g. in the form of many branches and paths, will involve more design decisions, and also more tests and more analyses, than for simpler software of the same size measured in lines of code. Cognitive complexity ("hard to understand") will increase the probability of error in each action.

A significant factor in cognitive complexity is the number of separate items that a person has to hold in their short-term memory in order to understand a particular operation. Software which makes greater demands on short-term memory for its understanding is less likely to be understood. It has been estimated that the human short-term memory is generally capable of holding no more than seven items, and perhaps less, at one time [10]. Information which would need to be held in short-term memory in order to understand a piece of software would include not only the variables used by the software but also the operations which were performed on them and the path or paths which were being taken through the program.

It is not only structurally complex software which is likely to have a high cognitive complexity. Size alone can lead to cognitive complexity even in software with a simple structure heavy demands are made on short-term memory in understanding a long sequence of operations.

A number of metrics exist for evaluating structural complexity, such as McCabe's Cyclomatic Complexity or Halstead Complexity Measures, but the current state of the art seems to lack a metric for cognitive complexity.

4.3 Understanding

In a review of 34 incidents involving control system failures the Health and Safety Executive (HSE) found that 44% had inadequate specification as their primary cause [5]. Inadequate specification of software is likely to be a reflection of poor understanding of the need which the software is intended to fulfil and lack of knowledge of the environment in which it will operate. Poor understanding and lack of knowledge may of course relate to the competence of the people concerned, but even if it is assumed that the people have an average level of competence in the relevant field their knowledge and understanding may be taxed by the novelty and complexity of the application.

There do not seem to be any metrics for the complexity of applications, and the very idea of a metric for ignorance is contradictory: how can we measure what we do not know? The nearest approach to a measure for these factors may well be judgement based on the degree of similarity to past experience.

5 Relating SIL to scale

The review in sections 3 and 4 above of the main factors which determine the probability that software will exhibit unsafe failures shows that they can be evaluated only to a very rough approximation, if at all, and that for some factors no relevant metrics exist.

If the relationship between the factors were well enough understood, and if suitable metrics existed for all the factors, and if sufficient data were available to assign values to them, the unsafe failure rate of software could be estimated from the following formula.

$$\lambda_{usf} = \chi\ S_C\ A\ F\ P_D \qquad (1)$$

Where

λ_{usf} = Rate of unsafe failures per hour

χ = Number of defects per unit of software engineering effort

S_C = Effort in producing software as a function of size and complexity

A = Multiplier depending on size and complexity of application

F = Proportion of defects which result in unsafe failures

P_D = Probability per hour that a defect will be activated.

The term χ can be regarded as representing the combined effect of the competences, methods and techniques applied to the engineering of the software. For software developed in accordance with IEC 61508 it would represent the effect of the combination of techniques that had been chosen: a set of techniques recommended for SIL 1 could be represented by χ_1, a set of techniques recommended for SIL 2 could be represented by χ_2, and so on.

If we accept that the relative effectiveness of the combinations of techniques for different SILs is as stated in IEC 61508, i.e. each SIL corresponds to an unsafe failure rate an order of magnitude lower than that of the next lower SIL, then the following relationship will apply to the values of χ.

$$\chi_1 = 10\chi_2 = 100\chi_3 = 1000\chi_4 \qquad (2)$$

Let us suppose that there are nominal values of S_C, A, F and P_D, (written as $\mathit{S_C}$, A, F and $\mathit{P_D}$) for which unsafe failure rates or confidence predicted by IEC 61508 are justified. For a SIL 3 system this could be represented by writing equation (1) as:

$$\lambda_{usf} = \chi_3\ \mathit{S_C}\ \mathit{A}\ \mathit{F}\ \mathit{P_D} \qquad (3)$$

If it was judged that a software system being developed to comply with the recommendations of SIL 3 was exceptionally large and complex, say 5 times as large and complex as "normal", and if the application was judged to be about twice as complex as usually experienced, then (3) would be modified as follows;

$$\lambda_{usf} = \chi_3\ (\mathit{S_C} \text{ x } 5)\ (\mathit{A} \text{ x } 2)\ \mathit{F}\ \mathit{P_D} \qquad (4)$$

which can be rearranged as

$$\lambda_{usf} = 10\chi_3\ \mathit{S_C}\ \mathit{A}\ \mathit{F}\ \mathit{P_D} \qquad (5)$$

From (2) it can be seen that $10\chi_3 = \chi_2$, so that SIL 3 methods are estimated in this case to give a level of confidence equivalent only to SIL 2.

Another example can be provided by the case of a protection system for which the required SIL was determined to be SIL 2, but which was assessed to have been developed only to SIL 1 because not all of the recommendations of IEC 61508 for SIL 2 had been followed. However, the software was an adaptation of that used in about 10 very similar previous applications, and because the application was not complex, being defined by a fairly simple control table, it was judged that the application complexity could be treated as an order of magnitude lower than "normal", so that equation (3) became:

$$\lambda_{usf} = \chi_1\ \mathit{S_C}\ (\mathit{A} \text{ x } 0.1)\ \mathit{F}\ \mathit{P_D} \qquad (6)$$

From (2) it can be seen that $0.1\chi_1 = \chi_2$, so for this simple application a SIL 1 process was able to give a level of confidence equivalent to SIL 2.

In the case of a subsystem or function for which it is possible to be confident that only a small proportion of potential defects would result in an unsafe failure the F term can be scaled down so that a level of confidence equivalent to a high SIL can be achieved by software that has been developed according to the recommendations for a lower SIL. For example, a unit such as a modem or multiplexer forming part of a data transmission system might have software developed according to the recommendations for SIL 1 but owing to the coding and addressing of its input and output data have very few potential defects capable of generating incorrect output data which complied with the coding and addressing schemes of the data transmission system. If it were judged that no more than 1 in 1000 potential defects could produce such an unsafe output the version of equation (3) for the software of this device would be:

$$\lambda_{usf} = \chi_1\ \mathit{S_C}\ \mathit{A}\ (\mathit{F} \text{ x } 0.001)\ \mathit{P_D} \qquad (6)$$

Again it can be seen from (2) that $0.001\chi_1 = \chi_4$, so for the software of this device a SIL 1 process was able to give a level of confidence equivalent to SIL 4.

It seems unlikely that the distribution of demands on the software relative to the distribution of defects would be known well enough to draw any conclusions about its effect on the rate of unsafe failures.

6 Conclusion

The approach outlined in section 5 above provides a means of evaluating the level of safety which software can be expected to achieve, taking account of the methods used, the size and complexity of the software, the complexity and novelty of the application, and the proportion of defects which are likely to cause unsafe failure. The result is expressed in the familiar form of a SIL, but this is now a SIL that is related to scale and can be used, perhaps as a fuzzy quantity, in reasoning about safety.

There are many imprecisions and uncertainties involved in estimating the effects of the factors in the equations above, and it would probably be flattering to this approach to even call its results fuzzy. It is completely dependent on engineering judgement, which is notoriously fallible. Richard Feynman remarked in the context of the investigation into the Challenger space shuttle disaster "When I hear the words 'engineering judgement', I know they are just going to make up numbers" (quoted in [9]). However, the risk assessments on which the allocation of SIL and other aspects of system safety management are based are also heavily dependent on the judgement of engineers and other experts who are required to make judgements about the frequency and consequences of hazards.

The use of engineering judgement to estimate factors such as the size and complexity of software in order to evaluate the effectiveness of achieving a SIL can be regarded as commensurate with the use of engineering judgement to estimate the frequency and consequences of hazards so as to determine the required SIL.

This approach is not claimed to equal the rigour or theoretical soundness of methods of evaluating evidence about software such as those based on Bayesian Belief Networks [??]. Its sole claim to merit is that it may open a way to a more rational use of the SIL concept in areas where the reliance on compliance with standards is deeply ingrained, often to the exclusion of other methods.

References

[1] Collins English Dictionary, Millennium Edition

[2] BS EN 50128 Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems, British Standards Institute

[3] BS EN 50129 Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling, British Standards Institute

[4] N. Fenton and M. Neil, Software Metrics and Risk, FESMA 99, 2nd European Software Measurement Conference, October 1999

[5] HSE, Out of Control: Why control systems go wrong and how to prevent failure (2nd Edition), HSE Books, 2004

[6] IEC 61508 – Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems. International Electrotechnical Commission.

[7] J. McDermid, Software Safety: Where's the Evidence? *6th Australian Workshop on Industrial Experience with Safety Critical Systems and Software (SCS '01)*, Brisbane. Conferences in Research and Practice in Information Technology, Vol. 3 P Lindsay, Ed.

[8] F. Redmill, Understanding the Use, Misuse and Abuse of Safety Integrity Levels, Proceedings of the Eighth Safety-critical Systems Symposium, Southampton, UK. Springer, 8-10 February 2000

[9] M. Thomas, Engineering Judgement, *9th Australian Workshop on Safety Related Programmable Systems (SCS'04)*, Brisbane 2004

[10] H. A. Simon, The Sciences of the Artificial, 2nd ed. The MIT Press, 1981.