SOFTWARE VERIFICATION RESEARCH CENTRE
SCHOOL OF INFORMATION TECHNOLOGY
THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 97-04

A systematic approach to software safety integrity levels

Peter A. Lindsay
John A. McDermid

February 1997

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

**Note:** Most SVRC technical reports are available via anonymous ftp, from `svrc.it.uq.edu.au` in the directory `/pub/techreports`. Individual abstracts and compressed postscript files are available from `http://svrc.it.uq.edu.au/cgi-bin/svrctr?yy-nn`
A systematic approach to software safety integrity levels

Peter A. Lindsay John A. Mc Dermid

Abstract

International Standards for safety-critical software typically use notions of Safety Integrity Levels (SILs) which in our experience are difficult to apply and which lack credible assessment criteria. This paper proposes risk modelling as a basis for allocation of SILs to software and illustrates its use. It also proposes software-directed evaluation criteria for SILs, to assess what level of integrity is actually achieved. We contend that the approach leads to more credible results, and more cost-effective ways of delivering software safety assurance.

1 Introduction

Software is increasingly being used in safety-critical applications. Unlike most hardware failures, software failures are “systematic” and software faults may lie hidden for a long time before being revealed.

Many standards for the development of safety-critical software and systems introduce the notion of Safety Integrity Levels (SILs). Roughly, SILs represent the “degree of freedom from flaws” that is required from the system or software. The notion implicitly recognizes the fact that 100% assurance is impossible to achieve for complex systems. In essence, the standards use SILs to dictate the nature and degree of scrutiny that is applied to safety issues during development: e.g. by recommending the kinds of techniques to be applied, the degree of independence to be used in verification and validation, and the competence of staff. This could be termed a “process-oriented” view of SILs.

Standards which apply this kind of approach include generic standards such as IEC 1508 [1], MOD 00-56 [2] and DOD 882C [3], and industry-sector specific standards for aerospace [4], railway signalling [5], and automobiles [6].

Because software does not cause harm directly, but only as part of an overall system, it is important to be able to assess the software’s contribution to, and responsibility for, overall system safety. To apply the notion of SILs to software – or other components, for that matter – it is thus necessary to have a way of apportioning integrity to system components. This is the point where standards begin to differ most, and where the process-oriented view of SILs begins to reveal its shortcomings.

We contend that current approaches to SILs lack credibility because they have concentrated almost exclusively on the process and have ignored the product. They lack credibility in theory as well as in practice [7]. A short review of current approaches is given in §2 below; for each approach it is possible to identify realistic scenarios which lead to counter-intuitive results.

Another important shortcoming of current approaches is that they give little or no guidance on how to assess whether desired levels of

\footnotesize\textsuperscript{1}SILs are called Development Assurance Levels (DALs) in DO178B.
integrity have actually been achieved, and in particular, on what forms of evidence should be required in safety cases for the different SILs. Thus, setting of SILs is often left “open loop”, and the achieved levels of integrity are not verified.

A “product-oriented” view of SILs is needed, to complement the predominantly process-oriented view. This paper sets out to address this need. In §3 below we outline requirements for a more systematic approach to SILs. §4 sketches a proposal for a new basis for allocation of SILs, which generalises the current approaches and addresses many of their limitations. Finally, §5 sketches a new basis for assessment of integrity of software components, based on the notion of “depth of analysis”.

2 Review of existing approaches

This section contains a brief review of some widely referenced standards, to set the context for our approach. The desire to resolve some of their various ambiguities is one of the motivations for the approach discussed below.

2.1 IEC 1508

Functional safety: safety-related systems [1]

This is a generic standard for safety-related computer-based systems. The standard attempts to be comprehensive, but is large and still in draft form in places, and so can be difficult to interpret and apply. Safety integrity is defined to be “the probability of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time” ([1] Part 4, p.12).

In IEC 1508, safety integrity requirements for safety functions are set directly from target (maximum) failure rates. The latter are measured in two different ways: as dangerous failures per year, for continuous control systems; and as probability of failure to perform design (sic) function on demand, for protection systems. The guidelines for determining SILs are unclear, however. For example, Part 5 talks in terms of the amount of risk reduction achieved by adding protection mechanisms: a target probability of failure is calculated as the ratio of frequency of failure with protection against frequency of failure without protection. This fails to take account of the possibility that such additions may result in new failure modes and new risks [8]; what then is the validity of using “likelihood of failure without protection mechanisms” in calculating how much risk reduction has been achieved?

Guidance is lacking at a number of other critical junctures. For example, it is commendable to acknowledge the difference between continuous control and protection, but there is a whole spectrum of applications between these two extremes. Guidance figures in terms of dangerous failure per year are provided for SILs for continuous control systems, but there is no guidance on how to determine these, and how to allocate SILs to components.

2.2 UK MoD 00-56

Safety management requirements for defence systems containing programmable electronics [2]

This is a reasonably comprehensive standard for safety management. It is generic and requires tailoring in places: e.g. it gives recommended interpretations of probability for accidents in terms of system lifetime, then requires that a project-specific Accident Risk Classification Scheme be developed which de-
fines levels of tolerability (i.e., it does not prescribe risk classification). In determining hazard probability targets, the user can take into account the "probability of events leading from a hazard to an accident".

In outline, the SIL assignment process is as follows: A "first function" is identified for each particular hazard, and assigned a SIL according to the severity of "worst credible accident" that could result from failure of the function (Part 1, Table 7). SILs for "second and subsequent functions" are set according to severity of outcome, combined with a failure probability for the first function. SILs for components are set according to the highest SIL of the functions they implement; rules are provided (Part 1, Table 8) that allow components of a lower SIL to implement a function of a higher SIL provided certain conditions are met.

In our experience, this process has a number of shortcomings. First, little guidance is provided on how to interpret worst credible accident. Next, it is not always clear what should be taken to be the first function. (What is the first function in an object-oriented software system, for example?) In our experience, it is quite often possible to identify and decompose functions in different ways, leading to different SIL-assignments for components under the above process.

More generally, application of the integrity decomposition rules is difficult and can lead to results which are hard to justify and/or counter-intuitive. For example, a "voter combinator" might guard against sensor faults but does not guard against common failure modes or faulty requirements; similarly, N-version programming does not guard against faults in the software requirements specification. There is a danger that SILs may be set lower than is appropriate. On the other hand, application of the rules often results in higher than expected SILs for components which contribute only marginally to safety.

2.3 MIL-STD-882C

System safety program requirements [3]

This US DoD standard introduces "software control categories", which are roughly a measure of the control software has over safety-critical functions. SILs are then determined directly from the control category and severity of outcome, using a risk matrix. Roughly, the more control is vested in software (rather than humans) and the worse the severity, the higher the SIL.

The software control categories are difficult to interpret – in fact, they are not even exhaustive – and little guidance is provided (however, see §2.5 below). The basis for assessing risk via such categories is highly questionable. It presumes that the risk of software fault leading to an accident is decreased by giving the human more control. This may be true for some military systems but it is certainly not true more generally: e.g. in applications calling for very short response times, or with very high operator workload.

2.4 MISRA


This is an example of a sector-specific (UK) standard which is able to take advantage of its more specialised application domain – vehicle-based software – to offer tighter guidance on SILs. The standard talks in terms of controllability – roughly, likelihood that a typical driver will recover from, or tolerate, a failure without incident – and provides useful guidelines on how to determine controllability. Software SILs are then set according to software-failure implications with respect to controllability. For a combination of reasons (not least, the perception that the driver is ultimately responsible for safety of the vehicle's
passengers), controllability categories have a degree of validity in this context, and seem much more convincing than the 882C software control categories.

2.5 STANAG 4404

_Safety design requirements and guidelines for munition related safety critical computing systems [9]_

This generic NATO software-safety standard defines safety design requirements for software, and gives guidance on how and when the requirements should be applied, verified and tested. It also has tailoring guidelines for a number of different kinds of munition-related systems. It gives “computing system control categories” for many examples, for use with sister-standard STANAG 4452 (Safety assessment requirements for munition related computing systems).

We mention STANAG 4404 here because it is one of the few examples of a combined product/process-oriented approach, whereas all of the preceding are process-oriented.

3 Requirements for an improved approach

Each of the above approaches has its problems. This section discusses requirements for a more systematic approach to SILs.

3.1 Severity

The definition of risk as a combination of severity and likelihood [10] is widely accepted and is not in dispute. When predicting or assessing system and software risk, however, the severity of a given hazard is not always clear. There may be factors outside the system’s control which affect the likelihood that a system hazard leads to a mishap. Different outcomes may be possible: the “worst case” may be much rarer than the “most usual outcome” but have far greater severity. The severity of an accident might be different depending on when it occurs: e.g. 00–56 allows that “distinct safety targets may be agreed for different groups of people who may be harmed” (trained operators, MOD employees, general public). Most standards are overly simplistic in their approach to assessing severity, or give little useful guidance. An approach is required which takes into account all the outcomes in a balanced way.

3.2 Likelihood

Since software failures are “systematic” it is very difficult to predict the likelihood of a hazard occurring. In fault-tolerant design, one of the aims is to identify possible “random” failures and try to mitigate their results; in safety-critical design, systematic failures should be treated similarly. Defence-in-depth means that different layers of protection should be designed in. As a result, relatively sophisticated models of cause and effect are needed, as a basis for predicting likelihood of system failure from likelihood of component failure can be very complicated. Again, standards are overly simplistic on this point. The IEC 1508 categorisation of systems as continuous control or protection systems, and measurement of risk reduction through protection, is an example of over-simplification. The 00–56 use of first functions and subsequent functions is another example. A more sophisticated framework for likelihood prediction is required.

For example, for an air-traffic control system, the worst outcome for a “loss of separation” hazard is many of hundreds of fatalities in a mid-air collision, but the more likely outcome is a near-miss, without loss of life or injury.
3.3 Hazards vs benefits

Safety analysis tends to concentrate on hazards, which means that the benefits offered by a system may be overlooked. Quantities such as “higher throughput of patients” and “faster dispatch response times” do not usually enter the safety equation since they do not directly relate to hazards of the system, although they can reduce “global” risk. Similarly, defensive systems may add to safety even if they are not highly reliable.

The standards’ approaches to risk assessment lack a coherent basis for judging cost-benefits in such cases. The ALARP principle (“as low as reasonably practicable”) is a good general principle for assessing cost-effectiveness of hazard mitigations, but the standards give little practical guidance in its application in determining SILs. For example, for defensive systems the “worst credible accident” might be associated with action under hostile circumstances and may have catastrophic consequences. It would clearly be unreasonable to rule out consideration of low-reliability systems where there are no alternatives. Clearly the “failure to defend” hazard is not the one to use in SIL assignment in this case, but the standards’ SIL assignment processes do not cover this well.

There is thus a requirement that the framework for risk assessment be able to take into account arguments about benefits as well as about hazards.

3.4 Low integrity components

As computers become more widely used, it is increasingly difficult to argue that high integrity development techniques need to be applied to software in its entirety. For example, if a high integrity (S4) component includes a word processor for preparing reports and that word processor does not interfere with normal operation of the component, then is it really necessary to develop the word processor to level S3, as ISO-56 suggests?

This argument extends to software developed to earlier, less stringent, standards and commercial off-the-shelf (COTS) software. In these cases, the SIL assignment process should enable the designer to look at the risk in the application context, rather than apply “blanket” rules. Similarly, an application developed under an S3-level process for one set of requirements should not automatically be considered to have S3-integrity for a different set of requirements, as the critical failure modes may be different.

A framework is required which allows components’ contribution to safety to be assessed and SILs allocated accordingly.

3.5 Evaluation criteria

In existing approaches there is typically no “closing of the loop” with respect to integrity: once SILs are set they are not normally analysed, verified or tested anywhere in the rest of the development life-cycle. Knowing what process was followed in developing a system is not assurance enough on its own: evaluation criteria should be defined, and related back to integrity requirements allocation, to assess how thoroughly the process was followed and how thoroughly the product was checked.

Similarly, there are many cases where information on the development process is not available or not applicable: e.g., customization; maintenance; use of COTS: safety integrity of data (e.g. for rule-based or table-driven applications). Ideally, we would like to have ways of analysing the product and evaluation criteria which can be applied independently of knowledge about the development process.
3.6 Summary

There are a number of unsatisfactory aspects of current approaches to setting SILs, and the absence of effective means for assessing achievement of SILs. Taken individually, none of the difficulties are sufficient to suggest that current approaches should be abandoned, but taken together they suggest a more systematic and defensible approach is needed.

4 Risk modelling and SILs

This section proposes what we believe to be a more effective approach to setting and assessing SILs, together with some theoretical justification. The approach is based on a framework for modelling risk and integrity allocation which generalizes existing approaches and addresses the requirements above.

Risk models are explained in §4.1. §4.2 indicates how SILs would be allocated, and §4.3 gives a small example. We hope to be able to provide practical evidence of the utility of the approach in the foreseeable future.

4.1 Risk modelling using CCA

The framework adapts and refines Cause-Consequence Analysis (CCA) [11] as a technique for modelling risk in computer-based systems. CCA combines the inductive reasoning features of Event Tree Analysis (ETA) with the deductive features of Fault Tree Analysis (FTA).

In brief, construction of a CCA risk model starts with the choice of a critical event – typically, failure of a major component. The consequences of the event are traced through to outcomes in a manner similar to ETA, by considering the different chains of events that may arise. Branches in chains correspond, for example, to the success or failure of protective features in the system’s design. As in ETA, probabilities are associated with branching nodes; however, as an extension of ETA, fault trees may also be attached to nodes to explain how the probabilities were calculated. Possible common causes are noted and taken into account in probability calculation. Finally, another fault tree is used to record all possible causes of the critical event in question, and to explain its probability. A simple example of a CCA model is given in Fig. 2 below.

We have chosen CCA because it supports both causal and consequence analysis in a single framework and it has a quantitative basis. It generalises ETA and FTA, two of the most commonly applied quantitative approaches,\(^3\) and is a natural extension of CHAZOP [13].

4.2 Proposal for allocation of SILs

It is evident that, because of the huge variety of possible applications of software, a rule-based approach to allocation of SILs is inevitably of limited credibility, at least for generic standards. Rather than give rules for how SILs are to be assigned to systems or components, we propose that system designers be allowed to use their experience to determine SILs and then justify their choice using risk models. The risk models would describe the system’s safety features and explain the allocation of integrity requirements. For the reasons given above, we believe CCA to be an appropriately general framework for risk modelling.

Following IEC 1508, we think of a SIL as a measure of “the likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time” [1].

\(^3\)FTA has been successfully applied at software level [12].
The integrity requirements allocation process would then proceed as follows:

1. Define risk models for the system at an appropriate level of detail.
2. Determine tolerable failure rates for each of the different possible classes of outcome.
3. Choose targets for component failure rate limits, and confirm that they lead to tolerable system failure rates using the risk models.
4. Assign SILs to components to meet all identified target failure rates, using some agreed interpretation of SILs.

A simple example is sketched in §4.3 below.

This process could be tailored to give more specific rules for setting SILs for specific application domains where, for example, the context and tolerabilities of risk are well understood. For example, the above approach would allow sector-specific regulatory authorities to define rules based on, say, worst case hazard severity, most likely hazard severity, or “average” severity as they deemed appropriate, and to have the risk analysis as the context for explaining their rules.

We believe the above to be a rational way of handling SILs which is compatible with the technically defensible aspects of the approaches reviewed above, and which avoids some of the more questionable aspects of those approaches.

### 4.3 A simple example

Fig. 1 shows a simple ‘recovery block’ strategy for checking the results of a calculation before passing them on, and substituting default values if an error is revealed. The rationale for such an architecture is that it is often simpler to check a solution than to calculate it, and default values can often be found which may be sub-optimal but which at least are safe (e.g. when calculating the setting for a control signal). We assume two independent means are used for accessing input data.

Fig. 2 shows a risk model for one of the main system failure modes: the critical event ‘error in response calculation’. Note that the model makes some strong assumptions about independence of components which won’t be discussed here.

Suppose the tolerable occurrence rates for the various possible outcomes are as follows:

\[ c_1 = 10^{-5}, \quad c_2 = 10^{-3}, \quad c_2 = 10^{-2} \]

(given as probability per operation of the system). Assuming the risk model in Fig. 2 is a reasonable model of cause and effect for the given architecture, then the designer would be justified in setting the following target component failure rates:

\[ p_1 = 10^{-2}, \quad p_2 = 10^{-4}, \quad p_3 = 10^{-3}, \quad p_4 = 10^{-2} \]

Using an interpretation of SILs as a measure of failure rates such as

<table>
<thead>
<tr>
<th>SIL</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure rate (\uparrow)</td>
<td>(10^{-1})</td>
<td>(10^{-2})</td>
<td>(10^{-3})</td>
<td>(10^{-4})</td>
</tr>
</tbody>
</table>

\(\uparrow\) probability of failure upon demand

then the following assignment of SILs to software components would be consistent with the risk model: response logic (S4); check response (S4); get data-2 (S4); default values (S3); get data-1 (S2); calculate response (S2).

### 4.4 Observations on the example

We can briefly review the example against the requirements of §3:

1. The severity assessment takes into account all the outcomes in a balanced way, although other strategies (e.g. worst case outcome) would have been possible.
2. Likelihood is evaluated using classical, well accepted analysis techniques. Note that the possibility of new hazards (e.g. inappropriate default values substituted) is also covered.

3. Hazards and benefits are shown in one diagram, enabling benefits to be taken into account, if so desired. The CCA model includes 'appropriate response' as an outcome.

4. Software components have been isolated, enabling various integrity levels to be assigned (although this must be tempered by the ability to ensure partitioning in implementation).

5. The framework spans different technologies.

6. With each SIL is associated a range of failure probabilities for which statistical evaluation support is available (see §5 below). Note also that an integrity level has been assigned to data; in our experience this is often both necessary and helpful – although it is not fully encompassed within existing standards.

5 Software-directed evaluation criteria

As noted in §3.5 above, it is important that SIL achievement can be verified independent of assignment. As we have argued, the current process – which could be caricatured as roughly "we need this SIL so we use this process; we've used this process so we've achieved this SIL" – does not provide sufficient assurance. Ideally, standards should define what evidence is required in a safety case to show that the target SIL has been achieved.

Space does not permit a full exposition of the proposed approach here, but we give an outline, starting with three key principles:

1. As far as possible, SIL probabilities (claim limits) should be statistically verifiable.\(^4\)

2. The predominant source of problems is requirements analysis and definition, so even the lowest SIL should be based on a requirements assessment. Successively higher SILs should require deeper assessment – of detailed design, then source

\(^4\)See [14] for more discussion of these and related points.

\(^5\)Ideally, analytic evidence of safety would be presented, in the form of software safety analysis or formal verification, for example. In many cases, however, statistical approaches may be the only alternative (e.g. with COTS).
Figure 2: CCA model for failure of the ‘calculate response’ software component.

code, and finally object code.

3. Diverse means of assessment should be used wherever possible: e.g. static analysis and dynamic testing.

These principles underlie the choice of failure rates ($10^{-1} \cdots 10^{-4}$ for S1 \cdots S4) in §4.3, since failure probabilities of $10^{-4}$ per demand are about the limit of what can be evaluated statistically under normal circumstances [15]. Note that by applying such stringent claim limits to SILs, we are not explicitly ruling out the use of software in higher reliability categories (i.e., with failure rates below $10^{-3}$), but instead are requiring that software architectures for such applications be modelled in sufficient detail, and with sufficient justification of software component independence and fault-propagation behaviour, that the above figures can be applied.

Turning to the individual SILs, we propose the following measures to demonstrate attainment of S1:

- Preliminary Hazard Analysis (PHA) and traceability from requirements to PHA results and hazard log (where appropriate) to show validity of requirements;
- requirements review by an independent team to show validity (diverse check);
- design documented and reviewed
- full traceability from requirements to code, to show all functions are implemented and no spurious functions are accessible;

\footnote{This allows for the possibility of spurious, but inaccessible, functions in COTS software, for example.}
- black-box testing with a “life-testing” reliability probability target of $10^{-1}$ [15];

- testing in the integrated system, as a diverse check;

- version and configuration control [16], without which the other results are meaningless.

These measures follow the above principles and give a base level of assurance that would be required for any safety-related system: namely, that safety requirements have been identified and implemented.

For SIL S2 we would add:

- safety analysis of the architecture, e.g., by CHAZOP and identification of derived safety requirements, to ensure the design introduces no new hazards or intolerable risks;

- traceability extended to include derived safety requirements;

- validation of architecture by review, formal analysis or animation, as a diverse form of evidence;

- black-box testing, with reliability probability target increased to $10^{-2}$.

This gives additional confidence regarding safety of the design, and thus the use of an S2 component has lower risk than use of S1 software – which is what we’d expect.

For SIL S3 we would extend hazard/safety analysis down to source code and require coverage analysis (e.g., via white-box testing). At SIL S4, these principles would be extended to object code. At each stage, the black-box testing target would be increased by factor of 10.

Space does not allow full justification of the above, but some points are worth noting here:

1. For most software, failure probability targets of less than $10^{-4}$ are beyond the range of what can be established by positive testing [15]. Testing targets would still be useful, however, for negative assessment of SILs: e.g., to reject software for S3 or S4 when bugs are found when testing against a target of $10^{-4}$.

2. No special case has been made for formal methods in the above. We note however that such methods become increasingly cost-effective for higher SILs: e.g., for path function analysis [17] to show source code coverage at level S3.

3. The degree of complexity of the software should be taken into account: the more complex the solution, the more thorough the checks applied. Consideration should also be given to the amount and nature of operating experience.

Finally, it is also worth noting that an advantage of the risk-model based approach in §4 is that it identifies the need for independence, for example in the software/hardware mapping or with respect to common-mode design faults. The analysis at each SIL can be extended to consider independence, but space does not allow discussion here.

6 Conclusions

In summary, this paper has drawn attention to some of the problems with standard approaches to software safety integrity levels and has proposed an improved approach. The proposal centres around the use of risk assessment as a technique for assigning and assessing safety integrity requirements. The paper proposes Cause-Effect Analysis as an appropriate basis for risk assessment, and outlines an approach to evaluation criteria for assessing achievement of SILs.
We believe CCA is easier to understand and apply than current approaches to SIL allocation, and leads to more credible results. While CCA may not currently be familiar in the software safety community, and needs some adaptation and refinement for software analysis, the technique is a straightforward extension of concepts (ETA, FTA) which are currently widely understood and applied. We note that CHAZOP is essentially a predictive form of CCA, as it considers both causes and effects of deviations (critical events). Thus, a final benefit of our proposed approach is that it can be applied initially in association with CHAZOP and then refined in the detailed-design stages, when enough information is available to construct the CCA.

In further work we plan to validate the proposed approach by (1) showing how existing approaches map onto it, and (2) applying it to industrial case studies (albeit retrospectively).

References


