

# Engineering Judgement

Martyn Thomas

Martyn Thomas Associates Limited  
Holly Lawn, Prospect Place, Bath BA2 4QP, UK  
martyn@thomas-associates.co.uk

## Abstract

Engineers who develop safety-related systems are required to work to remarkably high standards: SIL 1, the lowest classification recognised by the international standard IEC 61508, requires a probability of failure/hour (pfh) of  $10^{-5}$  or lower; and many systems are required to achieve several orders of magnitude lower failure rates than this. Yet SIL 1 is already so demanding that it is impractical to test a system for long enough to provide convincing evidence that the required failure probability has been achieved, and few systems have ever been mathematically proved to implement their specifications.

Instead, safety engineers develop safety cases that argue, and seek to show, that the required pfh has been achieved.

The safety authority then exercises their *engineering judgement* that the system will have the required safety properties. In the absence of convincing evidence, engineering judgement seems our only recourse, yet the great physicist, Richard Feynman, remarked in his book about the investigation of the accident that destroyed the Challenger space shuttle: "When I hear the words 'engineering judgement', I know they are just going to make up numbers" (Feynman, 1993).

In the light of what we know, how should a professional engineer build and assure the safety of systems? Do we need different international standards and regulatory frameworks? Should there be international co-operation to develop the components and tools that could replace the proprietary, unsound and over-complex products currently available?

## 1 Introduction

There is a wide range of computer-based systems in transport applications and many of them are safety related. Most of these systems are built to meet some recognised standard for safety-related software, such as IEC 61508 or DO-178B, and most standards contain some notion of acceptable rates of unsafe failure, which may be structured as Safety Integrity Levels (SILs).

For example, IEC 61508 contains this table:

Safety integrity level	High demand or continuous mode of operation (Probability of a 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: IEC 61508 SIL

Previous papers (McDermid 2001, Thomas 2003) have considered the absurdity of assigning particular software development practices to higher SILs, when there is limited evidence that different practices lead to particular probabilities of failure. Indeed, some evidence has been published that appears to show that the defect densities in delivered software are unaffected by the expensive MCDC testing required by Level A of DO-178B (German and Mooney 2001).

I do not want to single out IEC 61508 for particular criticism because it shares its weaknesses with many other standards for developing safety-related software. In this paper, I want to focus on one particular problem: the problem that the lowest integrity level that the standard considers worth discussing in detail, SIL 1, is associated with a probability of dangerous failure per hour (pfh) that is usually, in practice, too low to be demonstrated. The problem is, of course, even stronger for higher SILs.

To show that some system met the targets for SIL 1 ( $10^{-6} < \text{pfh} < 10^{-5}$ ) would involve testing the system continuously for more than ten years, under operational conditions, with no unsafe failures and no modifications to the system (Littlewood and Strigini, 1993). This is usually impractical as a strategy to achieve the necessary assurance before releasing a safety related system into service, although there will be some circumstances where it is practical to run thirty or forty identical systems under independent operational conditions for a year or so as a demonstration that SIL 1 has been achieved. In general, multiple sources of evidence must be combined.

## 2 Safety Cases

What would constitute conclusive evidence that a system has an adequately low probability of unsafe failure?

### 2.1 Retrospective evidence

At the time that a system is finally decommissioned, it is possible (if the evidence of hours of use, modifications, and actual failures has been collected and retained) to say beyond doubt what the pfh of the system actually was.

This is not very useful to engineers, because knowledge about a system that will never be used again provides little or no insight into the behaviour of systems in the future.

After a system has been in use for some considerable time, it will be possible to assess its historic pfh (again assuming that the necessary evidence has been collected). This process has actually been used in some industries, for example it is used by the JAA in certifying aircraft engines for use on twin-engined aircraft on routes more than 120 minutes flying time from an emergency airfield (ETOPS operations). This seems to be a successful strategy, although it would seem to depend on the assumption that past performance is a powerful predictor of future performance. This would not be the case if the system was modified during the evidence period or subsequently, or if the future operational environment was significantly different from the past, as it was in the Challenger space shuttle accident, or the Ariane V explosion.

Evidence from extensive statistical testing should be no different from evidence from real operational use. It is subject to the same restriction on modification and to the same requirement that the test environment should accurately reflect the future operational environment for the system. The latter requirement may be very difficult to achieve unless the real environment is available for use as a test-bed.

The number of hours of operational evidence that is required depends on a number of factors, but will not be less than the reciprocal of the target pfh (Littlewood and Strigini 1993).

## 2.2 Proof of correctness

Under some circumstances, it may be possible to prove that the system meets its specification. This may be highly desirable for some classes of system and it has occasionally been done for some subsystems, but it provides very limited information, if any, about the pfh of the system.

Proof is not about the probability of unsafe behaviour, unless “unsafe” can be defined completely and with mathematical formality. Proof is an extremely powerful way of checking the consistency between two definitions of how the system should behave—usually between the design or the high-level language source and some specification of the system’s required properties. In general, we can only talk about the safety of software in the context of the system of which it is part, and it can be very difficult to have confidence that we have considered all possible ways in which a system could be unsafe, and all ways in which the software could contribute to the unsafe behaviour of the system. For these reasons, it is unlikely that an engineer can specify the required safety properties of a software component in a way that can be shown to be complete, so even a complete proof that the software has the specified properties will still not amount to a proof that it cannot lead the system into unsafe states.

## 2.3 Testing

What do you know about a system that has been tested, that you did not know before the testing started?

1. How the system behaves on the tests.
2. If the test-set is statistically representative of the operational profile for the system, then you can infer the failure probability in further use in that operational profile.
3. If you have a mathematically sound analysis that shows that passing one of your tests proves, by induction, that the system will pass other tests, then you can deduce that behaviour.
4. ... nothing else.

Is this knowledge useful? In my opinion, only in cases 2 or 3, or where the tests fail. Most testing creates unjustifiable confidence in the future behaviour of the tested system.

## 2.4 Process-based evidence

Most standards require or recommend (which can amount to the same thing) that particular processes are followed in developing safety-related software. Such mandatory process might include, for example, hazard analysis, risk mitigation, version control, regression testing, or the production of design documentation in a particular form.

It is easy to see that good processes do not guarantee products with the desired properties: system properties depend more on what you do than on how you do it; a perfectly manufactured chocolate kettle is still useless for making tea. What makes good processes essential is the confidence that they provide in their outputs: without well-defined testing processes you cannot interpret the results from testing; without strong version control you cannot have confidence that the product that was tested is the same one that was inspected, or integrated into the system. So strong, auditable processes are necessary if we are to have confidence in the product, but the fact that a process has been followed in developing a product does not tell you anything about the properties of the product itself.

The *outputs* of a process may tell you something about the product. If you have confidence in a testing process (and its assumptions) then the test results may tell you how this particular product will behave under these test conditions. If you have confidence in a proof process (and its assumptions) then the proof may tell you that this particular product always has some particular properties. In a typical development, we will have a variety of evidence of the partial properties of the product, with differing levels of confidence in the evidence and possibly differing assumptions. Somehow, that disparate evidence needs to be assembled into a coherent picture of the safety of the product.

## 2.5 Safety Arguments

Given the impracticality of collecting evidence from sufficient and representative testing, and the impracticality of achieving a system-wide proof of safety, what does the responsible engineer do to convince themselves (and any independent assessor or regulator) that their system has an adequately low probability of unsafe failure?

The traditional solution is to assemble a safety argument that combines evidence from a variety of sources and to rely on the judgement of experienced engineers to assess whether this argument compels belief that the system is adequately safe. But what evidence is sufficient? Traditionally, developers turn to standards to tell them what they should do.

*"The nice thing about standards is that you have so many to choose from; furthermore, if you do not like any of them, you can just wait for next year's model."* attributed to Professor Andrew Tanenbaum.

Furthermore, international standards are the products of negotiation in committees, which is rarely the way to achieve the ideal solution to a problem, because the process terminates not when the right answer is found, but when an acceptable compromise is reached. Unfortunately, conforming to international standards can be the trump card in satisfying a regulator that your system is adequately safe, (or in convincing a law court that the accident was not your fault). The result of the widespread reliance on standards is that considerable effort is focused in meeting the requirements of a standard, rather than on addressing the risks that are specific to this development of this system for this application. Where the standard has weaknesses (as most do), this guarantees that the product is either less safe or more expensive than it should be, and perhaps both.

## 3 SIL 0

What about the many systems that could be considered safety related but which have tolerable rates of failure that are higher than the higher bound for SIL 1? These systems do not attract the attention of standards writers; it is assumed that the normal standards of industrial software development (whatever they are) will suffice. This is unrealistic: few software-based systems have a mean time to failure of more than twelve months, unless unusual care has been taken in their development. Yet IEC 61508, for example, considers these systems to be outside its scope:

This standard does not cover E/E/PE systems where (inter alia):

- a single E/E/PE system is capable of providing the necessary risk reduction; and the required safety integrity of the E/E/PE system is less than that specified for safety integrity level 1 (the lowest safety integrity level in this standard).

How many systems are deployed in safety applications every year relying on this exemption?

Normal industrial software practices do not include safety analysis, so where this exemption is used, the system is unlikely to be adequately safe, except by chance.

## 4 Maintenance

What do you know about a system that has just been modified?

You may have considerable knowledge about the original system, from millions of hours of operational use before the modification, but unless you are able to analyse the system rigorously to show the worst-case impact of the modification, the value of all that knowledge has been destroyed by the change. Will you therefore repeat all the assurance activities that preceded putting the system into service originally? History suggests not.

The rate of introduction of new, safety-related systems has accelerated in recent years, so it seems likely that the majority of safety-related systems are less than five years old. Anecdotally, it appears that problems from maintenance changes in commercial and industrial systems are cumulative, as the intrinsic knowledge of the system is lost to the maintenance team as staff move on or retire, and as the complexity of the system increases because changes are introduced in less than ideal ways. This analysis (though admittedly superficial) suggests that we shall see a growing rate of failures attributable to defects introduced during maintenance.

## 5 A possible future

What could be done to greatly increase the probability that new applications can be implemented safely and cost-effectively? I believe we need to tackle the problem on several fronts:

### 5.1 Engineering Competence

*We are like the barber-surgeons of earlier ages, who prided themselves on the sharpness of their knives and the speed with which they dispatched their duty -- either shaving a beard or amputating a limb.*

*Imagine the dismay with which they greeted some ivory-towered academic who told them that the practice of surgery should be based on a long and detailed study of human anatomy, on familiarity with surgical procedures pioneered by great doctors of the past, and that it should be carried out only in a strictly controlled bug-free environment, far removed from the hair and dust of the normal barber's shop. (Professor Sir Tony Hoare 1984)*

Modern computer-based systems are some of the most complex artefacts that have ever been built, and most of the complexity is in the software, for good systems engineering reasons. Yet most software developers lack both the depth of engineering knowledge and the humility of most of their contemporaries in other engineering disciplines. This is not surprising, because software engineering is at most 55 years old and when civil

engineering was 55 years old they hadn't even invented the brick.

Over centuries, as a result of many accidents and many deaths, other engineering disciplines have discovered that it is essential to build intellectually manageable models of systems and to reason about them before building the real thing. These models usually start as something informal then progress to a mathematically rigorous basis that will support strong investigation of properties. Computer scientists have provided the foundations for such professional engineering and some pioneer companies have used this science with remarkable success (Praxis Critical Systems, 2004). It seems to me that the software industry needs to recognise that mathematics is fundamental to all engineering and that software engineering, above all, needs the rigour that mathematics alone can bring.

Other engineers avoid novelty wherever possible, especially in safety-related systems.

*A good scientist is a person with original ideas. A good engineer is a person who makes a design that works with as few original ideas as possible. There are no prima donnas in engineering.*  
Freeman Dyson (2001: 114)

Software engineers frequently violate this rule.

## 5.2 Minimal defect construction

Edsger Dijkstra recognised three decades ago that the main cost in developing software is the cost of removing the defects that were introduced earlier in the process and that, perhaps uniquely among engineers, software engineers had the advantage that there was no conflict between high quality and cost effectiveness—to save development costs we had to avoid introducing defects or to discover them almost immediately, and that this would inevitably lead to higher quality software as well.

Today, Correct-by-Construction (CbC) development incorporates the best practices from the past decades: planning that identifies and focuses on the key risks (“are we confident that the users know how they want to use the system? No? then let’s build prototypes”; “will the interprocess communications be fast and robust enough? Let’s build it first, to make sure”); formal (partial) specification to reduce ambiguity and to allow key system properties to be explored early; design by refinement; Fagan reviews of almost everything; implementation in a safe language with powerful static analysis tools (such as SPARK) so that errors are found very early and run-time exceptions (and buffer overflows) cannot occur; frequent integration and regression testing; root-cause analysis when faults are found, for process improvement ... The result can be very high productivity (25+ LoC/person day) and very low defect rates (<0.04 defects/KLoC reported by users after delivery in the first 12 months of use) (Praxis Critical Systems, 2004).

## 5.3 Independent Safety Assessment

It is much easier to build a safe system than it is to show that it is safe, but professional engineers should be

modest, particularly about their own abilities, so their greatest sceptic should be themselves. If they have created enough evidence during development to satisfy themselves, the system should pass scrutiny by an independent safety assessor or regulator.

Safety assessment will ultimately be a matter of expert judgement based on sound (but necessarily incomplete) evidence. The CbC processes described above would leave useful evidence for the safety assessor: a hazard analysis and risk-based plan; investigation of risk areas with their resolution; requirements analysis and formal specifications; rigorous analysis of system properties; refinements of the specification into source code with assertions; automated and deep static analysis results; test results; logs of versions and modifications; review reports; root cause analyses of detected faults, and so on. These documents would be produced as a necessary part of the development, with no additional overhead.

The safety assessment will take account of this evidence and form an expert conclusion—but whom should society trust to make a decision that may, literally, be one of life or death?

There are precedents. Where particular expertise is required to ensure safety, many industries use licensed staff: for example, for fitting avionics or welding aircraft structures, for piloting large vessels into port, or for driving vehicles on a public highway. Why should we demand less of those who allow some of the most complex safety-related systems to be put into service? Safety assessors should be adequately trained, experienced, and licensed; nothing less makes sense. Then we have a mechanism for determining the basis on which expert engineering judgement is exercised, and for assigning responsibility for any abuse and reducing the risk that it will happen again.

## 5.4 Re-assessment after modification

A system will normally be assessed as safe to operate in a defined environment, because the evidence that would be required to assess a system as safe in *all* environments would be prohibitively expensive. A modified system, or a system in a modified environment, therefore requires reassessment. What should this entail?

In principle, the risks are the same as for a newly developed system, and so the assessment should be the same. However, much of the required evidence will already exist and can be reused if the engineers can show that the modification has not invalidated it, which may be fairly straightforward with a development that includes a formal specification that can be analysed to prove the maximum impact of any change. Without a formal specification of the system components that have been modified, any re-assessment is likely to be unnecessarily expensive, incomplete, or both.

## 5.5 COTS

The use of Commercial off the Shelf (COTS) components creates difficult problems for safety assessors, as these components usually lack enough evidence that they are

safe to use in the intended application. Often, commercial pressures are allowed to compromise the standards that should be applied. The result is that systems are put into service that should never have been built: the SS Yorktown for example, which apparently could not start its engines after a run-time failure in Windows NT had immobilised the entire ship, or any system that has been compromised by buffer overflows in components implemented in the C programming language.

There is a growing need for a suite of components that have been developed to high standards and that are available with their certification evidence, full fault history, modification history and so on. This would be a worthwhile focus for international investment in open-source components with open-source V&V and other evidence of security and quality, so that the next generation of safety-related and other critical systems can be built on sound foundations, in the public interest.

## 6 Conclusions

The safety critical software industry has come a long way in half a century but it falls far short of the standards expected from a mature industry developing very complex and highly critical systems. We need to find a way to bring about rapidly a transition that has taken generations in some other engineering disciplines.

At a minimum, we need to achieve:

- widespread competence in the mathematics that we need if our work is to be under control: discrete mathematics for specification and reasoning, continuous mathematics so that we can communicate with other engineers and understand our application domains, and statistics so that we can make sensible judgements about low-probability events;
- acceptance that mathematically formal specification should be the norm for all safety-related systems, wherever practical, so that unnecessary ambiguities, omissions and contradictions are reduced;
- design and implementation using notations and programming languages that have a well-defined meaning and that can be analysed by software tools; in particular, we should acknowledge that C and C++ are not suitable implementation languages for safety-related software;
- agreement that it is impractical to have sound evidence that a system has achieved a pfh of  $10^{-5}$  or lower and that the safety assurance of safety-related systems is therefore inevitably a matter of judgement; agreement that this judgement must be based on the best evidence that can be practically achieved, and that this evidence must include the results of deep (SPARK-like) static analysis and statistically valid testing, and that the judgement must be exercised by someone who can demonstrate appropriate education, training and experience and who has agreed to accept professional responsibility for the consequences of their judgement (for example, the loss of their licence to practice);
- new standards that incorporate these principles;
- investment in a new generation of real-time operating systems, compilers, development tools, user-interface generators and related products, so that it becomes practical to implement the applications that are now envisaged, cost-effectively and without needing to incorporate COTS software that has not been developed to safety-related standards.

Evolutionary change is likely to be far too slow to achieve the necessary improvements in time to build the very complex transport and other systems that have been proposed, or to secure the critical systems in the national and international infrastructures. Accelerating the necessary changes will require investment and regulation, through some form of licensing. The investment will be small and easily affordable compared with the egregious waste of money caused by cancelled, delayed, over-budget and failing systems today; the negative effects of regulation will be outweighed by the benefits.

If our industry does not design its own future in a form that addresses these issues, there is a risk that a sequence of accidents or near-misses will lead to public pressure and ill-considered political action.

## 7 References

- Dyson, F. (2001): *Disturbing the Universe* Harper Collins.
- Feynman, R. (1988): *What do you care what other people think?* New York, W.W. Norton and Company.
- German, A., Mooney, G. (2001): Air Vehicle Software Static Code Analysis Lessons Learnt. In *Aspects of Safety Management*, REDMILL, F. and ANDERSON, T. (eds). Springer.
- Hoare, C.A.R. (1984): *IEE Software* 1(2): 15.
- Littlewood, B. and Strigini, L. (1993): Validation of Ultra-High Dependability for Software-based Systems, *Communications of the ACM* 36(11): 69-80.
- McDermid, J. (2001): Software Safety: Where's the Evidence? *Proc. 6<sup>th</sup> Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, Brisbane.
- Praxis Critical Systems: Papers and Articles <http://www.praxis-cs.co.uk/publications/>. Accessed 2 June 2004.
- Thomas, M. (2003): Issues in Safety Assurance. In *Computer Safety, Reliability and Security*. ANDERSON, S., FELICI, M. and LITTLEWOOD, B. (eds). LNCS 2788: 1-7. Springer.