Software Safety: Where's the Evidence?

John A McDermid
Department of Computer Science
University of York
Heslington, York, YO10 5DD, UK
John.McDermid@cs.york.ac.uk

Abstract

Standards for safety critical software usually either mandate or recommend development and assessment techniques which are deemed appropriate to reduce the risk of flaws in the software contributing to accidents. These recommendations are usually broken down into a number of "levels" of rigour, with the highest levels being applied where the consequences of failure, or risk, are most severe. The paper discusses the extent to which it is possible to find evidence that there is a genuine variation in risk with level, i.e. that the principles in the standards are sound, and questions some of the assumptions underlying these standards.

The paper then goes on to discuss the potential advantages of using product-based evidence to demonstrate safety of software, as opposed to relying on process prescription. It outlines current work on developing and applying "evidence frameworks" as alternatives to the process-based approach, and identifies some of the challenges in gaining widespread acceptance of such approaches.

Finally the paper discusses the ALARP principle, and what would be necessary to show that risks associated with safety-critical software have been reduced ALARP. The paper concludes that there are some fundamental difficulties with applying the ALARP principle to software, which neither the process nor evidence-based approaches to demonstrating software safety can readily resolve.

Keywords: Safety critical software, software safety evidence, the ALARP principle.

1 Introduction

This paper presents a brief review of "accepted wisdom" in the development of software for safety critical systems then poses two questions about evidence. First, it questions whether or not there is evidence that the accepted "wisdom" is actually effective. Second, it asks what would constitute effective of evidence of software safety. The paper then goes on to question how the ALARP principle can be applied to software.

The aim of the paper is to stimulate debate on a number of key issues in dealing with the safety of software, and does not purport to present "all the answers". The paper will have served its purpose if it leads people to question the basis of existing standards and practices.

2 Evidence for Process Based Standards

There is a large range of standards for safety critical software, and the interested reader is referred to Hermann (1999) for an extensive review. These standards vary in their details, but most adopt a similar approach to dealing with safety – by recommending or prescribing development processes and methods.

2.1 Approach of the Standards

Software safety standards do not normally define a "single" software development process, but recommend processes and practices to be used to achieve different "levels" of safety. Most standards base their guidance on Safety Integrity Levels (SILs) (IEC 1999, MoD 1996, ADoD 1998), although the practice in civil aerospace is to refer to Development Assurance Levels (DALs) (RTCA 1992).

Most of the standards present, in tabular form or equivalent, lists of mandated or recommended techniques for each stage of the process. The details vary, for example, in UK Defence Standard (DS) 00-55 (MoD 1997) identifies methods as mandatory (M) or allows two different, but reasonably well-defined, forms of justification (J1 and J2) for not using particular techniques or classes of technique. In contrast, IEC 61508 uses the three primary classifications: highly recommended (HR), recommended (R) or not recommended (NR). Failure to use a HR method has to be justified – but no idea is given as to what is a valid justification. DO178B allows flexibility, although in a rather different form, by accepting alternative techniques with the justification that they are as effective as those they are replacing.

At the level of methods there are some commonalities, and some variation. DS 00-55 and DefAust 5679 place significant emphasis on formal methods. IEC 61508 identifies a large range of techniques, including formal methods. In contrast DO178B stresses human reviews and rigorous testing.

The idea of prescribing or proscribing methods based on SIL or DAL is now quite widespread, and has to be viewed as the "accepted wisdom". However there is a growing concern about this "wisdom". Several authors have queried the rules for allocating SILs, and even whether or not the concept is meaningful (Lindsay and McDermid 1997, Fowler 2000, Redmill 2000). Our aim here is not to extend this debate, but to consider whether or not the process pre-/proscriptions yield better, i.e. safer, software in practice.

2.2 Evidence for the Standards

The rationale for defining processes is complex, but there are two key assumptions:

- the processes for the higher SILs or DALs produce "better" software;
- the processes for the higher SILs or DALs are more expensive, hence it is inappropriate to use them unless the consequences of failure are severe.

As these are assumptions, and at least some of the standards have been used for a number of years, it is interesting to seek information to try to confirm these assumptions. Although these seem to be reasonable assumptions, the facts do not necessarily bear them out. Unfortunately, as cost and other data is sensitive, much of what we report is necessarily anecdotal.

There is some evidence (Shooman 1996) that airborne software developed according to DO178B and using its predecessor DO178A has a hazardous failure rate of about 10^{-7} per flying hour. This not enough for the most critical applications, i.e. level A, but is acceptable for level B. Note that this is hazardous failures – there is a rather higher level of nuisance failures. This suggests that the techniques are effective – or does it?

A more detailed analysis (some of which is contained in Shooman's paper) suggests little correlation with the "obvious" process factors. For example the programming language seems to have little bearing on the failure rate – indeed the author is aware of one system written in assembler which has had over 20,000,000 flying hours without hazardous failures. In general, there is little evidence that software of different SILs or DALs does have failures rates in "SIL/DAL order". Indeed what evidence there is (albeit mostly anecdotal) suggests that the most significant factor in achieving low hazardous failure rates is domain knowledge. This should be unsurprising as there is considerable evidence that the major factor in operational failures is requirements errors, and these are more likely to be removed by people who understand the domain.

Such a limited assessment cannot be taken as conclusive, but the evidence does not support the assumptions that the processes for the higher SILs/DALs produce software with lower failure rates. At minimum the assumption seems questionable.

There is also the issue of cost. Some recent data for Ada projects (Ada UK 2000) shows no significant cost variations between SILs 0, 2 and 3. In these cases the software was produced at around 5 LoC per person day, measured from software requirements to the end of unit test (the system independent part of the process). Also the author is aware of projects where "restrictive" practices of the higher SIL developments were migrated to lower levels, reducing the high in-process error rates being seen in the lower SIL developments. In effect the "restrictions" removed opportunities for error more than they slowed down the work. Of course there are all sorts of factors which influence such figures, e.g. were there more competent staff working on the more critical code?

Again, this assessment cannot be taken to be definitive, but is does challenge the assumption that the higher SIL processes are more costly, and thus the basis for using SILs or DALs in the first place.

This discussion has deliberately been at a high level. It would be possible to discuss the merits and demerits of particular techniques, e.g. the MC/DC testing criterion in DO178B. However considering these issues would detract from the main aim of the paper, but the interested reader is referred to McDermid and Pumfrey (2001).

2.3 Observations

Taken to its logical conclusion the above might seem to be implying that the standards are worthless, and we should have a "free for all" in software development. This is not the right conclusion to draw.

The standards do contain a lot of sensible advice and guidance (although some parts are questionable) albeit not really about safety. The standards are effectively concerned with quality and repeatability – both laudable aims. However the interested reader should try reading, say DS 00-55 and Part 3 of IEC 61508, and try to identify the techniques that focus on safety – or even the word safety! It seems to the author, and others (Leveson 2001) that there is poor correlation between standards and observed hazardous failure rates simply because the standards do not address safety issues. This, of course, raises the question of how to produce standards which are more concerned with safety.

3 Evidence based approaches

The approach proposed here is to seek explicit evidence of safety, with respect to potentially hazardous failure modes of the software, rather than make a "general appeal" to the quality of the process.

We outline the principles of such an approach, then discuss current work on development of "evidence based" software safety standards, as an alternative to process based standards.

3.1 Concept of software safety evidence

The principle of using software safety evidence is simple. First, identify the potential failure modes of software which can give rise to, or contribute to, hazards in the system context. Second, provide evidence that these failure modes:

- Cannot occur, or
- Are acceptably unlikely to occur, or
- Are detected and mitigated so that their effects are acceptable.

It other words software should be treated like any other "component" of a system, at least in principle. We briefly amplify on what this concept means and implies in principle, then consider one of the key practical issues in having an "evidence based" software safety process.

To treat software like other technologies, it is necessary to identify and allocate safety requirements and targets to software. Civil aerospace documents such as ARP 4761 (SAE 1996) require the use of fault trees to allocate requirements to sub-systems. Fault trees are constructed for each hazard, and the allocated requirements reflect the acceptable probability of sub-system (or component) failure modes as they contribute to that hazard. However ARP 4761 does not apply this principle to software (indeed it just allocates a DAL to a software system and places a failure probability of 0 in the tree!). All that is being suggested is that this principle is extended to software. However, to do this, it is necessary to determine the potential failure modes of the software.

There are various ways of identifying potential failure modes of software including application of functional failure analysis (FFA) (SAE 1996) and adaptations of HAZOP, e.g. SHARD (McDermid and Pumfrey 1994). These techniques identify potential failure modes and required failure mitigation mechanisms some of which will be implemented in software, i.e. the give rise to derived software safety requirements. Acceptable rates of occurrence of such failure events can be calculated from the fault trees (Lindsay et al 2000). This process defines the software safety requirements – but what of the evidence that the software meets these requirements?

Techniques such as SHARD define classes of potentially hazardous failure modes, e.g. omission, late. Evidence can be provided in terms of these classes – for example, evidence that there is no omission of a function in some schedulable item of software would comprise:

- The results of control flow analysis to show that the function is always called (under the relevant conditions);
- The results of schedulability analysis to show that the scheduable item is always run called (under the relevant conditions);
- The results of integrity analysis to show that the software in questions, and the scheduler data structures cannot be corrupted by other software.

In practice the details of the evidence would vary with the system design, e.g. the third point might be addressed by static analysis of the other application code, or analysis of the hardware and operating system if there is hardware supported memory partitioning.

This idea can be expanded to all the evidence types, although there is not sufficient space to do that here (a Doctoral student in York is working on a complete characterisation of such evidence types).

There is one practical point which takes us right back to the SIL/DAL debate – how much evidence, of what type, is needed. Again there is a simple answer in principle – for safety related code, a single item of evidence is required for each failure mode of concern. For safety critical code, two diverse items of evidence are needed, e.g. from test and from static analysis.

The concepts are quite straightforward, but it becomes rather more complex when one tries to put the ideas into practice. It is thus instructive to consider current attempts to produce evidence-based software safety standards, and to validate the approach.

3.2 Current work

The author's initial ideas on an evidence-based approach were developed whilst trying to develop a safety case for a legacy air traffic control system. At one level this work was satisfactory, i.e. a workable safety case resulted, but it became clear that a more detailed consideration of the principles was required. Fortunately the opportunity to do this arose in two different areas.

The UK Civil Aviation Authority (CAA) contributed to the development of DO178B, but does not believe it appropriate to use that standard for ground based systems. Instead they built on the work mentioned above to develop new guidelines for assessing software safety. These guidelines, known as SW01, are part of CAP 670 (CAA 1998) which deals with the design and assessment of ground based services including air traffic control systems. The first serious trial of this material is on the New en route Centre (NERC) at Swanwick, although it is too early to say whether or not the ideas are effective.

More recently, the author has been working for the European Space Agency (ESA) trying to find systematic ways of combining techniques to show the dependability of space systems (McDermid, Wills and Henry 2000). This work has enabled some of the ideas to be refined, and there is the possibility of being able to conduct some experiments to compare the cost and effectiveness of a number of different combinations of technique, in the foreseeable future (although this depends on ESA being prepared to fund the next stage of work).

Thus the ideas of an evidence based approach are still in their infancy, but there is some substantive work being undertaken and there is a reasonable change of getting some feedback on the viability of the approach in the near future.

3.3 Observations

The concept of an evidence based approach seems to be sound – after all, it is only saying that we should work out requirements on the software and provide evidence that we have met the requirements. However there are a lot of "implementation details" to address to establish a method of assessment based on these ideas. Also, we cannot completely escape process-based evidence.

The evidence-based approach relies on some (a minimum amount) of process evidence. For example, it is necessary to know what the installed system configuration is, that the test results correspond to that configuration, and so on. More technically, there are issues of requirements validity which require judgements about the rigour of the process. However the process requirements are simpler and more defensible than in standards such as IEC 61508.

There is inevitably still some judgement about what constitutes an acceptable set of evidence for different types of software requirement. However the approach of linking evidence to software failure mode classes makes this much less contentious. For example, static control flow analysis tells us about omission failures, but not about value domain failures – so it is useful evidence in the former case but not the latter. By focussing on failure modes we are able to determine much more clearly what the evidence is for, which is a welcome change to the long lists of techniques in standards such as IEC 61508.

It may be felt that the evidence based approach has lost much of what is "good practice". This needn't be so. The requirement to demonstrate safety should constrain the development process as little as possible. This should therefore allow project managers to use best practice, and to develop software cost-effectively – so long as they do so in such a way that safety evidence can be obtained in a trustworthy manner. Indeed there is much in the existing standards which would be of value in a good practice guide for high integrity software – it just has little to do with demonstrating safety.

It might also be thought that the evidence based approach might weaken standards. This needn't be so. The use of static analysis, high levels of test coverage, and so on are still relevant – where they provide safety evidence. If anything the approach might strengthen standards as the reason for using the technique is more obvious. It may also shed light on some of the COTS issues. If the evidence can be found then it is reasonable to use COTS software; if not, then there is a judgement to be made whether or not the risk is acceptable – which brings us to the ALARP principle.

4 Software and the ALARP principle

The principle of reducing risks as low as reasonably practicable (ALARP) is enshrined in UK Law, and has been adopted as a risk acceptance principle by various other groups, e.g. the Swedish Defence Materiel Agency. The question is how does this apply to software and relate to evidence based approaches to software safety.

4.1 The ALARP principle

The ALARP principle arose out of a judgement made by Lord Asquith in 1949. It is worth quoting the judgement as it makes the concept clear:

> "Reasonably practicable is a narrower term than physically possible because it seems to imply that a computation must be made in which the quantum of risk is placed on one scale and the sacrifice involved in the measures necessary for averting the risk (whether in money, time or trouble) is placed on the other. If it be shown that there is a gross disproportion between them – the risk be insignificant in relation to the sacrifice – the defendants discharge the onus upon them"

This is simply a cost-benefit argument – if the cost of an action greatly outweighs the benefit, then there is no need to undertake it. This seems obvious and unarguable.

In practice the concept is refined, and distinctions are made between scenarios where the cost merely has to be disproportionate, rather than grossly disproportionate, for the risk reduction action to be unnecessary. The concept of ALARP has been formalised using patterns in the goal structuring notation (GSN) and the interested reader is referred to Kelly (1999) for a more complete exposition of the ideas, and a discussion of ways of articulating ALARP safety arguments in GSN patterns.

4.2 ALARP and software

At one level the ALARP principle seems like common sense, and would be expected to be broadly applicable. However people have found difficulty in applying it to software (and in some other circumstances, e.g. ordnance and explosives). Why should this be?

There seem to be three related issues which make it difficult to apply the ALARP principle to software:

1. Most of the techniques we are interested in, e.g. rigorous testing, provide information about risk, they do not reduce risk (in this sense ALARP simply doesn't apply);
2. Even if we assume we will remove faults we find by carrying out some analysis we cannot predict what these faults will be in advance – so we cannot know the benefit of applying the technique in advance so there is no prior basis to make the judgement whether or not application of the technique complies with ALARP;
3. Less obviously, there is an implicit assumption behind the ALARP principle that determining risk is cheap, but that reducing risk is expensive. This is not the case for software – finding the problems through testing, etc. is the expensive part of the process, and writing the code is only 5-10% of the cost.

The above suggests that not only is it hard to apply ALARP to software, it may not be sensible. This is awkward, as (at least in the UK) there is a legal obligation to do so!

Perhaps more realistically, the above says that we cannot apply ALARP in a quantified manner and we should instead look for qualitative arguments to decide when risk has been reduced ALARP. In other areas which have similar problems the issue is usually resolved by appeal to standards, e.g. for design of pressure vessels whose probability of rupture cannot be predicted. Note that, in this case, we are appealing to a product standard, not the sort of standard which we discussed in section 2!

4.3 Observations

It is not obvious how to resolve the "ALARP dilemma" for software. It is unlikely that we could produce product-specific standards as is done in other disciplines, as there are too many products, and technology moves to fast.

The evidence based approach *may* provide a suitable way out of the dilemma but there will still be judgements about what is an appropriate level (depth) of evidence.

However, there may be a simpler resolution – which is to forget about ALARP! Lord Asquith's judgement, and ALARP, applies in criminal law. Civil law will normally impose more stringent requirements, e.g. to employ "all due diligence", which will typically mean doing that which is possible, not "merely" that which is reasonably practicable. This would suggest a much more "absolutist" view, and again the evidence based approach might be a way of reaching a consensus on such standards.

5 Conclusions

There is growing concern that the current set of software safety standards are inadequate or inappropriate. This paper has highlighted some fundamental issues, but it could also have considered problems of dealing with legacy and COTS code, and many more.

It has been suggested that an evidence based approach may be better than the use of the current "accepted wisdom" in process based standards. Whilst, in this paper, it has only been possible to sketch the basic ideas, there are several more complete expositions of the ideas being developed and experimentally evaluated. It is hoped that this might, in time, become the basis for a consensus on software safety (McDermid and Pumfrey 2001).

There are legal requirements in some countries to apply the ALARP principle. However there is considerable difficulty in seeing how the principle applies to software. Almost undoubtedly any application of the principle will involve judgements about what constitutes "good practice" rather than quantified assessments of risk and benefit. However it may be simpler to avoid the difficulties and to think in terms of "best practice" as typically required by civil law. Put another way, it may be helpful to take a legal standpoint, rather than a technical risk perspective, when trying to determine what are appropriate methods for designing or assessing safety-critical software. There is some reason for believing that the evidence based approach might be a better basis for gaining consensus on such techniques, but it will be some time before this hypothesis can be validated.

6 References

ADA UK (2000): The Contribution of the Ada Language to System Development A Market Survey, available from www.adauk.org.uk.

ADOD (1998):, *Australian Defence Standard Def(Aust) 5679: Procurement of Computer-based Safety Critical Systems,*. Australian Department of Defence.

CAA (1998): *CAP670: Air Traffic Services Safety Requirements*, Civil Aviation Authority.

FOWLER, D. (2000): Application of IEC61508 to Air Traffic Management and Similar Complex Critical Systems - Methods and Mythology. in Lessons in System Safety: Proceedings of the Eighth Safety-Critical Systems Symposium, 226-245 REDMILL F. and ANDERSON A. (eds) Southampton, UK, Springer Verlag

HERMANN D.S. (1999): Software Safety and Reliability, IEEE Computer Society Press.

IEC (1999): *IEC 61508: Fundamental Safety of Electrical /Electronic/Programmable Electronic Safety Related Systems*. International Electrotechnical Commission.

KELLY T.P. (1999): Arguing Safety – A Systematic Approach to Managing Safety Cases, DPhil Thesis, Department of Computer Science, University of York.

LEVESON N.G. (2001): Private Communication.

LINDSAY, P.A. and MCDERMID, J.A. (1997): A Systematic Approach to Software Safety Integrity Levels. in Proceedings of the 16th International Conference on Computer Safety (SAFECOMP 97), York, 70-82, DANEIL P (ed), Springer Verlag.

LINDSAY, P., McDERMID, J. A. and TOMBS, D. (2000) Deriving quantified safety requirements in complex systems. In Proc 19th Int Conf on Computer Safety, Reliability and Security (SAFECOMP 2000), Rotterdam. 117-130. KOORNNEEF, F. (ed). LNCS 1943. Springer Verlag.

MCDERMID, J.A and PUMFREY D.J. (1994): *A* Development of Hazard Analysis to aid Software Design. in COMPASS '94: Proceedings of the Ninth Annual Conference on Computer Assurance, NIST Gaithersburg MD, IEEE Computer Society Press.

MCDERMID, J.A and PUMFREY D.J. (2001): Software Safety: Why is there no Consensus?, In Proc. of ISSC 2001, Huntsville, System Safety Society.

MCDERMID J.A WILLS S.C.B. and HENRY G.K. (2000): *Combining Various Approaches for the Development of Critical Software in Space Systems*, European Space Agency.

MOD (1996) *Defence Standard 00-56 Issue 2: Safety Management Requirements for Defence Systems*. UK Ministry of Defence.

MOD (1997): *Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment*, UK Ministry of Defence.

REDMILL, F. (2000): Safety Integrity Levels - Theory and Problems. in Lessons in System Safety: Proceedings of the Eighth Safety-Critical Systems Symposium, Southampton, 1-20 REDMILL F. and ANDERSON A. (eds) Southampton, UK, Springer Verlag

RTCA (1992): *Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B*, Radio Technical Commission for Aeronautics.

SAE (1996): *ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, Society of Automotive Engineers, Inc.

SHOOMAN, M.L. (1996): Avionics Software Problem Occurrence Rates. 53-64, IEEE Computer Society Press.