Software is the primary determinant of function in many modern engineered systems, from domestic goods (such as washing machines) through mass-market products (such as cars) to civil aircraft and nuclear power plant. In a growing number of cases, the software is safety critical or safety related, i.e. failure or malfunction could give rise to, or contribute to, a fatal accident. In general, where software is a key element of a safety critical system, it is developed in accordance with a set of guidelines or standards produced by the industry, or imposed by a regulator.

The civil nuclear industry makes extensive use of software, for example in control and protection systems. Many of these systems are safety critical or safety related. Software in such systems is assessed against guidelines produced by the regulators, i.e. the Nuclear Installations Inspectorate (NII) in the UK.

Standards for safety critical software vary quite considerably between industrial sectors see, for example, [1]. The purpose of this paper is to consider the claims and dictates of standards and to compare these with what is achieved in practice, then to draw conclusions which it is hoped are relevant for the nuclear industry.

Software in Safety Critical Systems: Achievement and Prediction

by Professor John McDermid & Tim Kelly

Standards

Software failures arise as a result of systematic (design) faults that have been introduced during software development. In recognition of this, the approach taken by many of the existing software safety standards is to define requirements and constraints for the software development and assurance processes.

This approach is found in standards such as IEC 61508 [2], EN 50128 [3], DO178B [4], and the former Defence Standards 00-55 [5] and 00-56 Issue 2 [6]. By stipulating or constraining the processes to be used in the development, verification and validation of software, their intent is to reduce the number of faults *introduced* by the process (e.g. through increased rigour in specification), and to increase the number of faults *revealed* by the process (e.g. through increased rigour in verification) in order that such faults can subsequently be removed.

In addition, some standards (e.g. IEC 61508) also recommend defensive measures (e.g. architectural strategies) to mitigate faults which may remain post-development and assurance, although such (pragmatic) guidance is surprisingly uncommon.

Software standards dictate the degree of rigour required in software development and assurance, according to the criticality of the software within the system application. The degree of rigour is typically expressed in terms of Safety Integrity Levels (SILs), or Development Assurance Levels (DALs) in the case of DO178B. In IEC 61508 and EN 50128, the focus is on protection systems and the SIL required is determined according the acceptable failure rate of the protection system in question.

For example, in IEC 61508 a requirement for SIL 3 is defined as corresponding to an equivalent failure rate range of 1×10^{-7} to 1×10^{-8} failures per hour of continuous operation. In DO178B and Defence Standard 00-56 Issue 2, DAL and SIL (respectively) are determined according to the worst-case severity of the system hazard to which failure of the software can contribute, together with some consideration of the extent of possible mitigation external to the software. In the civil aerospace domain acceptable failure rates, from random causes, are also determined by hazard severity. Therefore implicitly there is a correspondence between DALs and acceptable failure rate targets. For example, the requirement of DAL A corresponds to a failure rate requirement of 1×10^{-9} per flying hour.

Having determined the overall SIL required, the standards define (typically, by lifecycle phase) the recommended techniques and processes for software development and assurance. For example, the guidance in part 2 of Defence Standard 00-56 Issue 2 states that "Informal" Requirements and Design Specification are considered acceptable for the lower integrity levels, i.e. SIL 1 and SIL 2, "Semi-formal" techniques are admissible for SIL 3, and "Formal (Mathematical)" specification techniques are expected for the highest level of integrity, i.e. SIL 4. Whilst the overall approach is common, there are often differences in the specific processes and techniques recommended by different standards. For example, Defence Standard 00-55 emphasises the use of formal verification techniques for the highest integrity level, whilst DO178B concentrates on human review and rigorous testing.

Standards differ as to whether a corresponding failure rate can be associated with software developed to a specific integrity level. Defence Standard 00-56 Issue 2 advocates the approach of determining claim limits for each Safety Integrity Level that defines the minimum (i.e. best) failure rate which can be claimed for a function or component at that level, irrespective of the calculated random failure probability for the underlying electronics. Whilst the standard expresses the desire that these claim limits should be based on actual operational experience, it provides an example set of limits that can be used in the absence of other data. For example, the corresponding claim limit for SIL 3 is defined as 1×10^{-6} failures per hour. DO178B takes a different view by

stating that, "development of software to a software level does not imply the assignment of a failure rate for the software". In the next section, we discuss the failure rates achieved in practice by industry where the requirements of the different safety standards have been followed.

For a more extensive discussion of the commonalities and differences amongst safety standards see [7].

Industry Data

It is difficult to obtain industrial data, partly because it is commercially sensitive, and partly because the data is not always collected systematically. In this section, we indicate what we believe is typically achieved in industrial projects, based on published data where possible, and on sanitised commercial material we have gleaned from a range of sources, where the material is not in the public domain.

Fault Density

There is a general consensus in some areas of the safety critical systems community that a fault density of about 1 per kLoC is world class. Some software, e.g. that for the Space Shuttle [8], is rather better but fault densities of lower than 0.1 per kLoC are exceptional. The UK MoD funded the retrospective static analysis of the C130J software, previously developed to DO178B, and determined that it contained about 1.4 safety critical faults per kLoC (the overall flaw density was around 23 per kLoC [9]). It is worthwhile making some observations.

First, whilst a fault density of 1 per kLoC may seem high, it is worth noting that some data suggests that commercial software is up to 10 faults per kLoC, with pre-release fault densities of up to 30 per kLoC [10, 11]. Anecdotal evidence suggests that initial fault injection rates may be much higher. Other sources, e.g. Hatton [12], confirm that the best achieved fault densities are around 0.1- 1.0 per kLoC, and suggest figures up to 4.0 faults per kLoC are typical of commercial software.

Second, not all faults are equal and with a typical safety critical development all known safety critical faults will be removed, and only those of lower importance, e.g. usability or performance issues, will remain. This is because, when changing code, there is a risk that a new fault will be introduced, so a judgement is made whether or not making a modification is likely to reduce risk.

Third, faults are generally data sensitive, e.g. the code will work correctly on most data, but certain values will give rise to problems. This might be, for example, because of a "divide by zero" or a fault in an algorithm (perhaps represented as a data table, where one of the entries is incorrect), or inappropriate initialisation when the system is restarted under unusual circumstances, e.g. in flight rather than on the ground.

Failure Rates

Failure rate data is more difficult to come by than information on fault density. First, we make some estimates of dangerous failure rates in two industries:

Ellims [13] has produced an interesting analysis of software in the automotive industry. Most automotive accidents are due to drivers. The majority of those accidents which have technical causes are due to mechanical failure. There is no data on what proportion of accidents with technical roots are caused by software.

However, some estimates can be made using recall data. Less than 0.1% of vehicle recalls are software related and some of these may be using software to rectify other faults, e.g. mechanical deficiencies. However assuming that all these were corrections of hazardous failures, Ellims has estimated that, at worst, 5 deaths and 300 injuries per annum in the UK are attributable to software in vehicles. Assuming that there are five million vehicles on the road, each being driven for 300 hours per annum on average, this amounts to about 0.2×10^{-6} failures causing injury or death per hour.

The figure considered "per system" is probably lower, as there are several computer-based systems in modern vehicles. Thus it would seem that the industry currently achieves better than 10^{-6} per hour, perhaps around 10^{-7} per hour, for those failures causing injury or death. Note that there are nuisance failures - one of the authors had his car air-conditioning system "crash"; this was fixed in a software upgrade.

Similarly, the civil aircraft industry has almost no fatal accidents which are attributed to software (although there are incidents). At present there are around 14,000 civil aircraft worldwide, with a total of about 18 million flights (the industry uses the term departures) between them per annum. The average loss rate is about 1.4 per million departures. Assuming an average flight length of 5 hours this gives a fatal accident rate of about 0.3×10^{-6} per hour.

As with the automotive industry, the majority of accidents are attributed to human error, or to mechanical failure. On the other hand, there are many software upgrades on aircraft systems which may indicate that changes were made following incidents which, in other circumstances, could have given rise to accidents. However even if one in three accidents had software as a partial cause, this still gives 10^{-7} per hour fatal accident rate from software causes. An analysis by Shooman [14] of software fault correction for avionics systems also gives a figure of about 10^{-7} per hour.

As with the automotive industry, if the data is apportioned amongst systems, the failure rate becomes even lower.

There are typically about forty computerised systems on a modern aircraft — although not all of these systems can, of themselves, cause aircraft loss.

Finally, note that the underlying software failure rate will be much higher than the accident rates quoted here, as not all software failures will give rise to accidents. Indeed, experience of flying on modern jets attests to the unreliability of some systems, especially in-flight entertainment.

Observations

This operational data suggests that, in mature industries, safety critical software has a comparatively low failure rate, and is not a major contributor to accidents. However this positive view should be tempered with the observations that there are many more nuisance failures than hazardous ones, and that software is growing in complexity and authority. Thus, although the current situation is quite positive, it cannot be assumed that it will remain this way.

Predicting Failure Rates

Even if software has a low failure rate in practice, we are still left with the difficulty of predicting failure rates before we deploy software.

There are several reasons why it is difficult to demonstrate the failure rate of software, in advance. The most basic problem arises from the low failure rates which need to be demonstrated.

First, it has long been accepted that it is not practical to experimentally quantify the failure rate of safety critical software, to show that it meets a target such as 10^{-9} per hour. Ignoring for the moment issues of statistical confidence, 10^{9} hours amounts to roughly 114,000 years - which is clearly impractical as a test period prior to deployment. Butler and Finelli [15] produced the first publication which clearly stated this difficulty but others, e.g. Littlewood and Strigini [16], reached similar conclusions.

Thus, it is accepted that direct attempts to quantify rates of occurrence of hazardous failure modes for software are infeasible. (This is true even where there are no failures. If software has executed for N hours without failure, and all we know is that it has been exercised randomly, then there is only a 50% chance that it will execute for the next N hours without failure).

In general, the safety critical software community seem to accept that a failure rate of about 10^{-3} to 10^{-4} per hour can be demonstrated prior to release to service, via statistical testing. However, lower failure rates cannot be shown in this way - although, clearly, the operational data says that they can be achieved.

There have been several attempts to circumvent this problem, still within classical statistics. Some have considered reliability growth, i.e. how reliability improves over time with fault removal, but this still suffers from the limits alluded to above.

Second, it is unclear how to relate flaw density to failure rate. There is evidence of a fairly strong correlation for systems such as programmable logic controllers (PLCs) [17]. This correlation can be used produce a stochastic model of functional failure. If this model were general it would be possible to predict failure rate from flaw density.

On the basis of the stochastic model of functional failure, the mean time to software failure can be shown to be approximately proportional to T/N, where T is the time spent testing and debugging and N is the number of systematic faults in the software. Therefore, if it is possible to halve the number of faults N this will approximately double the reliability. Similarly, doubling the time T, spent on testing, will also approximately double the reliability, provided all failures are correctly diagnosed and fixed (this is a common assumption in reliability growth modelling, albeit one not borne out by industrial experience). Note that it is still necessary to have a long testing time and a small number of faults to get a Mean Time Between Failures (MTBF) in the order of 1,000 or 10,000 hours.

In general, however, the correlation will depend on where the flaws are in the program and the typical "trajectory" through the program. Consider mass market software such as Windows. There are about 35-40 MLoC in Windows XP. If the typical figure of 10 faults per kLoC for commercial software applies, then this is just over one third of a million faults - yet Windows reliability has grown from about 300 hours MTBF with 95/98 to about 3,000 hours with the current generation of systems according to data provided by Microsoft (although the software size and thus, presumably, the number of faults has grown). Assuming that there are 100 million PCs in the world running 1,000 hours per annum (5 hours per day, 200 days per year) this gives 100 billion operating hours per annum. The above T/N formula would give an MTBF of about 100,000 hours not 3,000 hours. This suggests that faults have a non-uniform distribution, and that the T/N formula does not apply well for complex products. Other data supports this view, e.g. work by Chilarege and colleagues at IBM on the failure rates of widely distributed large-scale products such as operating systems, see for example [18].

Thus it is hard to infer operational failure rates before entry into service. On a simple statistical basis, it is hard to get beyond 10^{-3} to 10^{-4} per hour, and there seems to be no practical way of inferring failure rate from fault density. Although the T/N formula is attractive, it does not readily take us past the 10^{-3} to 10^{-4} per hour figure, and it is unclear that it applies to complex software.

Achievement and Prediction

It is instructive to relate the standards and industrial practice to compare what we can achieve, with what we can predict:

Achievement

Safety critical software, in service, has low hazardous failure rates. From the data quoted, it contributes to accidents at a rate of around 10^{-6} to 10^{-7} per hour, although it must be stressed that this data is approximate. In terms of the figures used in the standards reviewed the achieved rates are roughly:

IEC 61508 - SIL 2

DO 178B - DAL B

Defence Standard 00-56 Issue 2 - SIL 3

EN 50129 [19] - SIL 1

EN 50129 is the railway standard which sets SIL targets, and hence the context for the software requirements in EN 50128. Defence Standard 00-56 Issue 2 and DO178B are broadly in line with one another - the achieved rates fall into the second highest SIL or DAL, whereas the railway standards view the failure rate achievements as being SIL 1, and IEC 61508 is in between these extremes. In the railway interpretation of SILs, the achievement is much lower than with the other standards, but they are placing much more stringent requirements on themselves!

Of course, in this data, the failure rate of the software itself could be higher than quoted as the accidents relate to those unsafe failures which are not mitigated. On the other hand, assuming that all software recalls for cars or aircraft are to fix a hazardous software defect seems rather pessimistic. Thus, although these figures cannot be taken as firm, there is evidence that high SIL/DAL levels are achieved, in practice, at least according to some interpretations of the concept. Perhaps what is most striking, however, is the degree of variation in interpretation of the concept between these standards.

It is, of course, accepted that there are significant levels of nuisance failures in the industries surveyed, but we should not let this stop us from recognising that these figures show that the number of unsafe failures, caused by software, is very low.

Note that the development processes from which we have drawn the failure rate data do not generally employ static analysis, so these failure rates are achieved without complying with the requirements of the highest SILs in Defence Standard 00-56 Issue 2 and in IEC 61508. There is some evidence that formal techniques such as static analysis [9], do find faults, although there is no easy way of quantifying the benefits (the return on investment).

Prediction

In some senses, the most difficult problem is one of prediction. Statistically, it is not reasonable to claim better than about 10^{-3} to 10^{-4} per hour, on the basis of pre-operational testing. Thus we cannot predict with confidence the failure rates we can achieve - indeed the gap is significant, at around three orders of magnitude.

Some attempts have been made to overcome this "prediction gap", e.g. by use of Bayesian Belief Networks (BBNs) [20]. However, whilst the BBN models which are produced often seem compelling qualitatively, or structurally, quantitatively they depend on expert judgement on how the factors in the models combine (technically these are conditional probability tables at the nodes in the BBNs). It is difficult to see how to validate this expert judgement, in numerical terms, although generally the qualitative relationships, e.g. that the use of static analysis helps to reduce fault density, are much easier to validate.

Software in Nuclear Plant

We understand that the NII is reluctant to accept claims beyond 10^{-2} per annum or (1.14 x) 10^{-6} per hour for systems containing software, without formal analysis. It is instructive to review this position, in the light of the above data:

A Dilemma

Put simply, the above analysis seems to show we can achieve the sorts of failure rates we require for safety critical software - at least for SIL 3 systems - without the use of static analysis, but we simply cannot show it in advance. Further, the operational times needed to show that SIL 4 systems meet their targets are very long - and probably only attainable in mass market products such as cars. The best avionics system we are aware of has operated for about 25 million hours without unsafe failure, i.e. with a failure rate of about 4 x 10^{-8} per hour at 50% confidence, but it is now getting towards the end of its life!

Given the above analysis, we seem to have two options:

- Do not accept failure rate claims for software better than 10^{-4} per hour, the lowest rate which can realistically be shown by statistical testing, unless and until the software is proven in service;

- Accept failure rate claims as low as 10^{-7} per hour, so long as good processes have been used, and there is no contrary evidence, i.e. no failures in testing, when that testing was sufficient to reveal faults if the software didn't meet at least 10^{-3} per hour.

The first option is perhaps the most "scientifically defensible". However, the second option would allow operators to gain benefit from using software at the "cost" of using an argument which extrapolates beyond available data. Arguably, this is currently what is done, albeit implicitly, with process-based arguments.

Thus, there is a dilemma. If we stick to the "scientifically defensible" approach then the risk of failure of the software which is deployed will be low - but this may mean that desirable functions or capabilities are not provided, and may therefore increase risk at the whole plant level. Alternatively, if extrapolation beyond demonstrated failure rates is allowed there is a greater risk that the deployed software will fail in service, possibly in an unsafe manner, and the safety argument is intrinsically weaker as we are making arguments about a specific system, based on general observations about the class of safety critical software. Both approaches have drawbacks, and neither is really attractive.

A Third Way?

It can be argued that the "core" of the above dilemma is the approach to the whole problem, SILs are a "blunt instrument", and several analyses, e.g. by Fowler [21] and Redmill [22], have shown that the "logic of SILs" does not always stand up to scrutiny when applied to particular systems. There is, however, a possible "third way" - provide safety arguments based on the failure modes of concern, not the "blunt instrument" of SILs.

Starting at the system level, it may be possible to show that we can afford a fairly high rate, say 10^{-3} per hour for a smart sensor, because the system uses high levels of redundancy (indeed it is likely that availability will be a greater driver than safety). In this case, the most difficult issue is likely to be common mode failures - and SILs don't seem to help with that at all. If we can produce system arguments which are defensible given such failure rates, then black box testing or in-service data should provide sufficient evidence.

If such arguments are not tenable, and we need to show that much more stringent failure rates are met, then there may be benefit from analysing specific failure modes. Consider again the case of a smart sensor. The critical failure modes may be plausible but wrong data, perhaps caused by a slow drift. Evidence that these failure modes do not arise would come from analysis of the design or software itself, in this case considering numerical accuracy of algorithms, including possible drift in numerical integrators. An argument focused on such issues would give direct evidence of safety, rather than the indirect argument based on SILs. Of course, some "backing evidence" is needed to show that the results of the analysis apply to the software as delivered, that it is properly scheduled, and so on, but these arguments and the supporting evidence focus directly on the safety properties of concern, rather than being a very indirect argument about "SIL". In other words, the "third way" is to extend the safety arguments and safety case down to the level of software, and specific software failure modes [23]. This should allow scientific arguments about the "absence" of unsafe failure modes, due to systematic error, using analytical techniques, such as static analysis to be produced; these can avoid the problems of predicting failure rates by focusing on deterministic analyses.

Some further observations about SILs are in order. We have previously argued [24] that there is little correlation in practice between observed failure rate and SIL; further, there is little correlation between SIL and cost. This led us to question the utility of SILs. We have recently come across an earlier paper [25] which analyses the predicted costs of developing software at different SILs using Boehm's COCOMO II model [26]. Benediktsson shows that the initial development costs of higher SIL software are greater than those of lower SILs - but once support costs are considered the cost of higher SIL software is less than that of the lower SIL software. As the use of SILs seems to be predicated on the basis that higher SILs cost more, hence should only be used for more critical software [24] this, together with our rather more empirical data, tends to undermine the motivation for introducing SILs!

Conclusions

Software is currently an important element of many safety critical systems, and the trend is towards greater dependence on software. In some cases, this greater dependency reflects a desire to increase capability; in other cases it is simply due to the infeasibility of avoiding software, e.g. where so-called "smart sensors" have replaced "dumb" ones.

There has been some concern over the use of software in safety critical applications, but the available evidence suggests that software has not been a major contributor to accidents, in the aerospace and automotive sectors, and unsafe failure rates of about 10^{-6} to 10^{-7} per hour have been achieved in such applications. This failure rate corresponds to SIL 2 in IEC 61508, or DAL B in DO 178B. These low failure rates have been achieved without the use of static analysis, although there is empirical evidence that static analysis techniques do reveal faults in software. This suggests that static analysis has a role from the point of view of achieved integrity at the SIL 4/DAL A level, but it is hard to justify on these grounds at lower integrity levels. Static analysis may be cost-effective at lower integrity levels, but there are limited data points to support such an assertion (see [27] for an example).

A problem however exists with prediction; statistical techniques are insufficient to "prove" a failure rate of better than 10^{-3} to 10^{-4} per hour prior to deployment of the system. If pre-operational claims were limited to what can be shown by testing, then this would severely limit the classes of system which could be developed and deployed, as operational achievement is around three orders of magnitude better than can be shown via testing. In some regards, this seems to be the view the NII takes of software systems; this is, of course, conservative and "safe", but it may limit the deployment of perfectly acceptable technology. On the other hand, just appealing to the process and saying that "SIL X achieves a failure rate of better than Y" is not compelling.

One possible alternative approach is to move away from simple reliance on SILs and to analyse systems and software to show that particular failure modes of concern, e.g. plausible but erroneous data values, cannot arise. In other words, it would be possible to extend the safety argument and safety case to deal explicitly with software. There is some move in this direction, e.g. for ground-based systems in the civil aerospace sector, and in Issue 3 of Defence Standard 00-56 which was released at the end of 2004.

In summary, despite around thirty years of experience in using software in safety related and safety critical systems, we do not have consensus on what can be achieved with software, how best to achieve it, nor on how to "prove" what has been achieved. It is therefore likely that software in safety critical systems will remain a contentious issue for some time to come and it may be that there is a major shift in approach, e.g. towards goal-based standards, and reduced reliance on prescriptive standards. There certainly seems to be merit in keeping an open mind, and being prepared to accept a range of arguments for demonstrating system and software safety, rather than sticking dogmatically to the prescriptions of any one standard.

References

[1] Hermann, D., *Software Safety and Reliability*. 1999: IEEE Computer Society Press.

[2] *61508 - Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems,* International Electrotechnical Commission (IEC), 1999

[3] *EN 50128 - Railway applications: software for railway control and protection systems*, CENELEC, 2001

[4] *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics RTCA DO-178B/EUROCAE ED-12B, 1993

[5] *Safety Management Requirements for Defence Systems*, Def Stan 00-56, UK Ministry of Defence, Issue 2, 1996

[6] *Requirements for Safety Related Software in Defence Equipment,* Def Stan 00-55, UK Ministry of Defence, Issue 2,1997

[7] Papadopoulos, Y., McDermid, J. A., *The Potential for a Generic Approach to Certification of Safety Critical Systems in the Transportation Sector*, Reliability Engineering and System Safety, 63(1), 47-66, Elsevier Science, 1999

[8] Barnard, J., *The Value of a Mature Software Process,* United Space Alliance, presentation to UK Mission on Space Software, 10th May 1999.

[9] German, A., Mooney, G., *Air Vehicle Static Code Analysis - Lesson Learnt,* in Aspects of Safety Management, F Redmill, T Anderson (Eds), Springer Verlag, 2001.

[10] Fenton, N.E., Ohlsson, N., *Quantitative Analysis of Faults and Failures in a Complex Software System*, IEEE Transactions on Software Engineering, 26(8), 797-814, 2000.

[11] Reifer, D.J., *Industry Software Cost, Quality and Productivity Benchmarks,* at: *www.softwaretechnews.com/stn7-2/reifer.html*

[12] Hatton, L., *Estimating Source Lines of Code from Object Code: Windows and Embedded Control Systems,* 2005, at: *www.leshatton.org/Documents/LOC2005.pdf*

[13] Ellims, M., *On Wheels, Nuts and Software,* to appear in proceedings of the 9th Australian Workshop on Safety Critical Systems, Australian Computer Society, August 2004.

[14] Shooman, M.L., *Avionics Software Problem Occurrence Rates,* in proceedings of the 7th International Symposium on Software Reliability Engineering, White Plains, NY, 1996. p. 53-64.

[15] Butler, R.W., Finelli, G.B., *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software,* IEEE Transactions on Software Engineering, 19(1), pp.3-12, January 1993

[16] Littlewood, B., Strigini, L., *Assessment of Ultra-high Dependability of Software-Based Systems,* Communications of the ACM, 36(11), pp 69-80, November 1993.

[17] Bishop, P. G., Bloomfield, R. E., *A Conservative Theory for Long-Term Reliability Growth Prediction* , in proceedings of the Seventh International Symposium on Software Reliability Engineering, IEEE, White Plains, NY, IEEE Computer Society, Nov. 1996.

[18] Chillarege, R., Biyani, S., Rosenthal, J., *Measurement of Failure rate in Widely Distributed Software,* 25th International Symposium on Fault-tolerant Computing, June 1995.

[19] *EN 50129, Safety-related electronic systems for signalling*, CENELEC, 1998

20 Littlewood B., Wright D., *A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability*, in proceedings of the 14th International Conference on Computer Safety, pp. 173-188, Springer, 1995.

[21] Fowler, D., *Application of IEC 61508 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, Anderson, T., Redmill, F. (ed.s), pp226-245, Southampton, UK, Springer Verlag.

[22] Redmill, F., *Safety Integrity Levels - Theory and Problems, Lessons in Systems Safety, in Lessons in System Safety*, Proceedings of the Eighth Safety-Critical Systems Symposium, Anderson, T., Redmill, F. (ed.s), pp1-20, Southampton, UK, Springer Verlag.

[23] Weaver, R. A., McDermid, J. A., Kelly, T. P., *Software Safety Arguments: Towards a Systematic Categorisation of Evidence*, in proceedings of the 20th International System Safety Conference (1SSC 2002), Denver, Colorado, USA, System Safety Society, 2002.

[24] McDermid, J.A., Pumfrey, D.J., *Software Safety: Why is there no Consensus,* 22nd International System Safety Conference, Providence Rhode Island, System Safety Society, 2004.

[25] Bendiektsson, O., *Safety critical software and development productivity,* The Second World Congress on Software Quality, Yokohama, 2000.

[26] Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D., Steece, B., *Software Cost Estimation with COCOMOII*, Prentice Hall, 2000.

[27] Hall, A., Chapman, R., *Correctness by Construction: Developing a Commercial Secure System,* IEEE Software, January/February, 2002.