



City Research Online

City St George's, University of London

Citation: Littlewood, B. (1996). Evaluation of software dependability. In: Wand, I. C. & Milner, R. (Eds.), Computing Tomorrow: Future Research Directions in Computer Science. (pp. 198-216). New York, USA: Cambridge University Press. ISBN 9780521460859

This is the unspecified version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://city-ref-test.eprints-hosting.org/id/eprint/1629/>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

Evaluation of Software Dependability

*Bev Littlewood
Centre for Software Reliability
City University
Northampton Square
London EC1V 0HB*

1 On disparity, difficulty, complexity, novelty - and inherent uncertainty

It has been said that the term software engineering is an aspiration not a description. We would like to be able to claim that we engineer software, in the same sense that we engineer an aero-engine, but most of us would agree that this is not currently an accurate description of our activities. My suspicion is that it never will be.

From the point of view of this essay - i.e. dependability evaluation - a major difference between software and other engineering artefacts is that the former is pure design. Its unreliability is always the result of design faults, which in turn arise as a result of human intellectual failures. The unreliability of hardware systems, on the other hand, has tended until recently to be dominated by random physical failures of components - the consequences of the 'perversity of nature'. Reliability theories have been developed over the years which have successfully allowed systems to be built to high reliability requirements, and the final system reliability to be evaluated accurately. Even for pure hardware systems, without software, however, the very success of these theories has more recently highlighted the importance of design faults in determining the overall reliability of the final product. The conventional hardware reliability theory does not address this problem at all.

In the case of software, there is no physical source of failures, and so none of the reliability theory developed for hardware is relevant. We need new theories that will allow us to achieve required dependability levels, and to evaluate the actual dependability that has been achieved, when the sources of the faults that ultimately result in failure are human intellectual failures.

The importance of human activity, and human fallibility, in the design process shows itself in several ways. One of these is the enormous disparity that is concealed under the name of 'software'. Since much of the difficulty of writing software arises from the particular nature of the application domain, it seems unlikely that software engineering will ever be a coherent discipline in the way that civil or electrical engineering are. Rather we should look to differences in the tasks that face the software designer in different applications for an explanation of the varying success - particularly varying dependability - with which systems are built.

Secondly, the question of 'difficulty' of the problems we tackle in software is not well understood. It is intuitively obvious that some problems are intrinsically harder than others: real-time problems such as aircraft flight control are probably harder to solve than, say, those involving accountancy, word-processing, etc, or those involving rigorously expressed mathematics. Models of human intellectual ability, particularly

propensity to failure, are not effective in other disciplines - we do not have good ways of characterising difficulty in mathematics, for example. We should not be surprised that we have a poor understanding of what makes software problems hard, but equally we should be aware that it is this variation in hardness that explains much of the variation in the observed dependability of the systems we produce.

The issue of difficulty, or hardness, of a problem is distinct from the notion of the complexity of its solution. Complexity has been recognised as an important factor in determining the dependability of software products. It is generally agreed that we should avoid complexity if we desire to achieve high dependability, particularly in the case of safety-critical systems. Similar arguments have also been used for dependability evaluation. Certainly it has been said that the utmost simplicity is a prerequisite for being able to have a sufficiently complete understanding of a system that one could claim that it was absolutely free of design faults. Intuitively, it seems plausible that, even when we cannot make such a strong claim as complete perfection, nevertheless keeping things as simple as possible will be 'a good thing'. Unfortunately, there are no satisfactory measures of complexity; there is not even an agreed means of deciding incontrovertibly that A is more complex than B. The few proposed 'measures' of complexity turn out in practice to be merely measures of 'size' [Fenton 1991]. More seriously, no useful relationship has been established between such measures and system attributes such as dependability.

Finally, the typically high degree of novelty of software systems distinguishes them from more conventional engineering. At the most mundane level this is manifest in a tendency to reinvent the wheel. In spite of much rhetoric about software reuse, this is still quite rare compared with the reuse of tried and tested design in other engineering disciplines. When we do not learn from previous experience, we run higher risks of failure from design faults. The most serious source of novelty, however, arises from the very success of software-based systems in delivering extensive functionality. It is clear that technological 'progress' has accelerated sharply with the widespread use of computer-based systems, and it is now commonplace to see systems that it would be unthinkable to implement purely via conventional hardware engineering: fly-by-wire and unstable aircraft control systems; railway signalling and control systems that provide greater track utilisation by allowing less physical separation of trains, etc. If we consider the evolution of civil aircraft over the past fifty years, it could be argued that the greatest discontinuity in terms of design novelty occurred with the introduction of computers in engine controllers and flight control systems. In certain industries there seems to be an almost cavalier attitude to the use of software - if it can be done it will be done. Unfortunately, knowing that certain novel functionality can in principle be delivered by software is not the same as knowing that it will be delivered with the required dependability.

If disparity, difficulty, complexity and novelty are so important, why do we not have adequate theories that account for their rôles in software engineering and allow us to control their impact? One reason lies, I think, in our aspiration to be a 'proper' engineering discipline, with a genuine scientific underpinning. This laudable aim is too often taken to mean that we must only engage with that which is objective and external to the human - preferably, in fact, with mathematics. Unfortunately, the human element in what we do seems paramount, and any theories that discount it will be doomed to failure. Thus exhortations to 'mathematise' software engineering, so as to stay in the world of the logical and deterministic, will at best only address a vanishingly small class of problems, and at worst may give a false confidence to the designers faced with those other real-world problems. My point here, in singling out (some would say caricaturing) a view of formal methods, is not to argue that it has no merit as a means of aiding system design. Rather it is to claim that for almost all real systems the problem of evaluating dependability involves an inherent uncertainty that arises from the potential fallibility of humans when they are faced with solving difficult, novel problems.

I cannot claim that the probabilistic approach to software dependability is a solution to the problems I have described. On the other hand, notwithstanding its considerable limitations, it does recognise the uncertainty present in much of what we do. A better understanding than we currently have will only come about by acknowledging the inevitability of this uncertainty.

2 The need for evaluation methods

It has become a truism that society depends more and more upon the correct functioning of software. For the most part the trust we have so far placed in computer systems has been vindicated: instances of catastrophic system failure, and consequential loss, arising from software faults are surprisingly rare. Unfortunately, system complexity continues to rise, and the extent of our dependence increases. It would be rash merely to assume that future systems can be depended upon - rather we need to be able to decide in each instance whether such dependence is justified. The subject of this essay, then, is the evaluation of dependability in the possible presence of design faults, particularly software faults.

The most dramatic issues in dependability arise in safety-critical applications when human life can be at risk in the event of failure. I shall begin by briefly listing some examples of real and proposed systems, with their required levels of dependability:

- Flight-critical avionics for civil aircraft require a failure rate better than 10^{-9} per hour [RTCA 1992]. Examples include the existing 'fly-by-wire' system in the Airbus A320 [Rouquet & Traverse 1986], and similar systems planned for aircraft such as the Boeing 777. Such requirements may be several orders of magnitude beyond what can actually be achieved but, more importantly, they pose insurmountable problems to those responsible for their validation.
- The software-driven Primary Protection System (PPS) of the Sizewell B reactor originally required a better than 10^{-4} probability of failure upon demand (pfd) [Hunns & Wainwright 1991]. As a result of concern expressed about the likelihood of being able to demonstrate such a level, an analysis of the overall reactor safety case was carried out which showed that, in fact, 10^{-3} pfd would be sufficient. There is general agreement that such modest levels are probably achievable, even for complex systems such as this. These levels can also, in principle, be validated.
- The Advanced Automation System (AAS), the new US air traffic control system, has a dependability requirement expressed as an availability: better than 3 seconds down-time per annum [Avizienis and Ball 1987]. It is curious that there does not appear to be a reliability or safety requirement: i.e. that safety-related events will only occur at an acceptably low rate.
- Computer control is becoming increasingly common in medical applications. Examples include radiation therapy machines [Jacky *et al.* 1991] (this application area is noteworthy for providing one of the few well-documented examples of computer failure resulting in loss of human life), heart pace-makers [Mojdehbakhsh *et al.* 1994], and even robotic surgeons.

These examples illustrate several things. In the first place, even for safety critical systems there is great variation in the levels of dependability that are required. Some systems seem to need levels that are probably several orders of magnitude from what we can actually achieve - and are certainly orders of magnitude beyond what it is possible to evaluate quantitatively in a scientifically convincing way. Some safety critical systems, on the other hand, have surprisingly modest requirements. Certain

types of heroic surgery upon terminally ill patients may only be possible with computer assistance; in such cases even a low probability of success may be better than the status quo. Secondly, the nature of the requirement varies from one application to another. In systems that need to be working continuously, such as an air traffic control system, availability and reliability are both important. Even in those cases where reliability is the most important issue, the precise way in which this is specified needs some care..

Consideration of these examples of safety-critical systems should not deceive us into thinking that it is only here that the validation of acceptable dependability levels is of importance. There are many other circumstances where, although human life might not be at risk, nevertheless a high dependability needs to be assured because the consequences of failure can be catastrophic: some tightly coupled financial systems come to mind in this context. And of course in more mundane applications, although ultra-high dependability is not required, users will still demand that a system is sufficiently dependable. The fallibility of the author's word-processor, together with my tendency to forget to save documents, results in frequent minor crises. It could be argued that vendors who are prepared to demonstrate that a software product achieves a particular level of reliability, even though this is relatively modest, will gain the same market advantages that accrued to Japanese manufacturers of electronic consumer goods in the past couple of decades. Certainly this approach would be preferable to some current software 'warranties' which are textbook examples of caveat emptor.

Adjudication between different competing software development methods is another area where dependability evaluation is important. If we can demonstrate that a particular approach delivers more reliable software for a given cost we would be well-disposed to use it. At present, recommendations about good practice are at best based on anecdotal evidence, at worst are dishonest special pleading with no empirical support.

The above are some of the reasons why we need to be able to measure the dependability of software. In the next section we shall look at the problem of how this can be done, briefly describing the current state of the art before going on to consider some issues that need further research.

3 Where we are now

The word dependability has come to embrace all those aspects of behaviour upon which the user of a system might need to place dependence: it thus includes reliability, safety, availability and security.

Measures of dependability are necessarily probabilistic because of an inherent uncertainty. In the case of reliability, for example, this uncertainty in the failure behaviour arises directly from two main sources. In the first place, there is uncertainty about the program itself, inasmuch as we do not know which of the inputs will, when executed, cause failure. Secondly, the operational environment is variable in a non-deterministic way: we cannot say with certainty which inputs will be presented to a program in the future. The net result is that we cannot predict with certainty when failures will occur in the future.

These remarks concern the failure behaviour of a system with constant reliability. There is a further cause of uncertainty when we debug programs and thus cause reliability growth. When we identify a fault, as a result of experiencing a failure, we cannot be certain that an attempt to fix the fault will be successful - indeed, it is common to introduce novel faults at such fix attempts. This is another aspect of the human fallibility that was discussed earlier. Even if the fix is successful, we do not know the 'size' of the fault that has been removed, i.e. there will be uncertainty about the magnitude of the improvement in reliability that will take place even in the event of a perfect fix.

A great deal of work has been carried out on the problem of estimating and predicting the reliability of a program as it is being debugged during test: the reliability growth problem. There are now many stochastic models which purport to be able to provide such predictions [Jelinski & Moranda 1972, Littlewood & Verrall 1973, Musa 1975, Goel & Okumoto 1979, Littlewood 1981, Musa & Okumoto 1984]; see [Xie 1991] for a useful survey of this work. Although none of these can be relied upon always to give reliability measures that are accurate, recent techniques allow us to check whether they are providing accurate results on a particular software system [Abdel-Ghaly *et al.* 1986, Littlewood 1988]. It is often possible to use these techniques to allow a model to 'learn' from its past errors and so recalibrate future reliability predictions [Brocklehurst *et al.* 1990]. The bottom line is that it is usually possible to obtain accurate reliability measures from software reliability growth data, and to know that confidence in such figures is justified.

Another area where modelling has had some success is in the incorporation of structural information about the program into the reliability estimation process. These models aim to emulate the classical hardware procedures which allow the reliability of a system to be computed from knowledge of the reliabilities of its constituent components, together with information about the organising structure [Barlow & Proschan 1975]. Whilst these hardware theories are essentially static, the software approach must be dynamic and emulate the way in which software components (modules) are successively exercised in time. This is done by assuming Markovian [Littlewood 1976, Siegrist 1988a, Siegrist 1988b] or semi-Markovian [Littlewood 1976, Littlewood 1979] exchanges of control between modules. Each module can itself fail with its own unique failure rate, and the exchanges of control between modules are failure-prone in the most general formulation. The potential advantage of this approach is that it allows the reliability of a system to be predicted before it is built, in the event that it is to be built of modules whose failure history in previous use is known.

Other structural models have been studied in order to model the failure behaviour of fault-tolerant software based on the idea of design diversity. The great difficulty here is that we know from experimental studies [Knight & Leveson 1986, Eckhardt *et al.* 1991] that it would be too optimistic to assume that diverse software versions fail independently. This precludes the simple modelling assumptions that are sometimes made in the case of hardware redundancy. Estimating the actual degree of dependence in failure behaviour between two 'diverse' software versions seems very difficult; indeed, it seems as hard as simply treating the whole fault-tolerant system as a black box and estimating its reliability directly by observing its failure behaviour [Miller 1989].

These are some of the areas where there have been advances recently in our ability to measure and predict the reliability of software. It must be admitted, however, that this success story relates only to those cases where the reliability being measured is quite modest. It is easy to demonstrate that reliability growth techniques are not plausible ways of acquiring confidence that a program is ultra-reliable [Littlewood 1991; Littlewood and Strigini 1991; Parnas, Schowan *et al.* 1990]: the testing times needed become astronomically large as a result of a law of diminishing returns, and the issue of whether the test inputs are truly representative of those the system will meet in operational use becomes a serious one. Similarly, as we have seen, the effectiveness of fault tolerance is limited by the degree of dependence in the failure processes of the different versions, and experiments [Knight & Leveson 1986, Eckhardt *et al.* 1991] suggest that this will be significant. Arguments about the reliability of a software system based upon the efficacy of the development methods used will probably remain weak, even when we have good evidence for such efficacy - and this is not the case at present.

If we really need an assurance of ultra-high system reliability, and this seems inescapable in some safety-critical applications, it seems that this will have to be achieved without depending upon software to be ultra-reliable. In fact, of course, the problem is even worse, since everything we have said about software applies to design faults in general. Any claims that particular systems are safe because their control systems are ultra-reliable must take note of these unpalatable facts. The only possible exceptions are those systems that are so simple that it can be argued that they are completely correct (and are a complete and accurate embodiment of their high level requirements, which must also be extremely simple)¹. This observation may give us some leeway to build computer-based systems, with the extra non-safety functionality that these can deliver, that are nevertheless measurably safe, by confining the safety-critical functionality in a tightly controlled kernel.

4 Future work: needs and practicalities

Dependability modelling is an area of research which is largely driven by the problems of system validation that people meet in real life. Rather than pretend that there is a coherent organising framework, then, this section will be presented in terms of the different problems that need to be addressed.

It seems inevitable that social and political concerns about safety critical systems will continue to play a large rôle in deciding which problems are important. However, it may be the case that some of this work will find its widest application in more modest applications. The success of manufacturers of fault-tolerant hardware in selling systems for applications such as financial services, for example, might suggest that software fault-tolerance techniques might also move out of their present ghetto of safety-critical applications. Although we shall begin with some problems of dependability evaluation that presently concern safety-critical systems, it should be born in mind that solutions to these problems may have much wider applicability.

4.1 *Safety critical systems and the problem of assuring very high dependability*

It seems clear that computers will play more and more critical rôles in systems upon which human lives depend. Already, systems are being built that require extremely high dependability - the figure of 10^{-9} probability of failure per hour of flight that has been stated as the requirement for recent fly-by-wire systems in civil aircraft is not exceptional. There are clear limitations to the dependability levels that can be achieved and assured when we are building systems of a complexity that precludes us from making claims that they are free of design faults.

Although a complete solution to the problem of assessing ultra-high dependability is beyond us, there is certainly room for improvement on what we can do presently. Probabilistic and statistical problems abound in this area, where it is necessary to squeeze as much as we can from relatively small amounts of often

¹ It may at first seem contradictory that I would believe your claim that the failure rate is zero, but regard as untenable your assertion that this was a system with a failure rate of 10^{-9} per hour - after all, the former is the stronger claim. In fact, the reasoning in the two cases will be very different. In the first case it will be completely logical and deterministic: you will be asserting that the system is sufficiently simple that you have convinced yourself, and are trying to convince me, that it contains no design faults and thus cannot fail (at least as a result of design faults). In the second case, you are acknowledging the possible presence of design faults, and thus are in the realm of probability, but you are claiming that their impact upon the reliability is, literally, incredibly small. I might accept the first argument, but I would not accept the second.

disparate evidence. The following are some of the areas which could benefit from investigation.

Design diversity, fault tolerance and general issues of dependence

Clearly, one promising approach to the problem of achieving high dependability (here reliability and/or safety) is design diversity: building two or more versions of the required program and allowing an adjudication mechanism (e.g. a voter) to operate at run-time. Although such systems have been built and are in operation in safety-critical contexts, there is little theoretical understanding of their behaviour in operation. In particular, the reliability and safety models are quite poor.

For example, there is ample evidence that, in the presence of design faults, we cannot simply assume that different versions will fail independently of one another. Thus the simple hardware reliability models that involve mere redundancy, and assume independence of component failures, cannot be used. It is only quite recently that probability modelling has started to address this problem seriously [Eckhardt & Lee 1985, Littlewood & Miller 1989]. These new models provide a formal conceptual framework within which it is possible to reason about the subtle issues of conditional independence involved in the failure processes of design diverse systems. They provide a link between informal notions such as ‘common fault’, and precise formulations of the probabilistic dependence between the failure behaviours of different versions. The key novel idea here is that of variation of ‘difficulty’ over the input space of a particular problem: those inputs that are most ‘difficult’ are ones that will tend to have highest chance of failure when executed by all versions. The notion of difficulty in these models is an abstract one, but it is an attempt to represent the intuitive concept of difficulty discussed earlier. In the Eckhardt and Lee work it is assumed that something that is difficult for one team will also be difficult for another; our own work, on the other hand, allows there to be differences between the teams in what they find difficult (perhaps because they are using development tools that have different strengths and weaknesses).

There is possibly a subtle and important distinction to be made here between identical mistakes and common faults. In experiments like that of Knight and Leveson, the different teams sometimes made exactly the same mistakes (for example in misunderstanding the specification). In the Eckhardt and Lee model, there is a propensity for different teams to make a mistake in similar circumstances, but not a requirement that the nature of the mistakes be the same. It is an open question worthy of investigation as to whether this distinction is important. For example, what are the implications for the shapes of individual fault regions in the input space? (An interesting experiment by Amman and Knight explored the shapes of ‘faults’ in subsets of the input space [Amman & Knight 1988])

Further probabilistic modelling is needed to elucidate some of the other complex issues here. For example, there has been little attention paid to modelling the full fault tolerant system, with diversity and adjudication. In particular, we do not understand the properties of the stochastic process of failures of such systems. If, as seems likely, individual program versions in a real-time control system exhibit clusters of failures in time, how does the cluster process of the system relate to the cluster processes of the individual versions? Answering questions of this kind requires information about the shapes of the fault regions, discussed above, and about the nature of execution trajectories in the input space. Although such issues seem narrowly technical, they are of vital importance in the design of real systems, whose physical integrity may be sufficient to survive one or two failed input cycles, but not many.

Judgement and decision-making framework

Although probability seems to be the most appropriate mechanism for representing the uncertainty that we have about system dependability, there are other candidates such as Dempster-Shafer [Shafer 1976] and possibility theories [Dubois & Prade 1988]. These latter might be plausible alternatives in those safety-critical contexts where we require quantitative measures in the absence of data - for example, when we are forced to rely upon the engineering judgement of an expert. Further work is needed to elucidate the relative advantages and disadvantages of the different approaches for this specific application.

There is evidence that human judgement, even in 'hard' sciences such as physics, can be seriously in error [Henrion & Fischhoff 1986]: people seem to make consistent errors, and to be too optimistic in their own judgement of their likely error. It is likely software engineering judgements are similarly fallible, and this is an area where some empirical investigation is called for. In addition, we need formal means of assessing whether judgements are well-calibrated, as well as means of recalibrating those judgement and prediction schemes (humans or models) which have been shown to be ill-calibrated. This problem has some similarity to the problems of validation of the predictions emanating from software reliability models, in which the prequential ideas of Dawid have proved very useful [Dawid 1984, Brocklehurst *et al.* 1990].

It seems inevitable that when we are reasoning about the fitness for purpose of safety-critical systems, the evidence upon which we shall have to make our judgements will be disparate in nature. It could be failure data, as in the reliability growth models; human expert judgement; evidence of efficacy of development processes; information about the architecture of the system; evidence from formal verification. If the required judgement depends upon a numerical assessment of the dependability of the system, there are clearly important issues concerning the composition of evidence from different sources and of such different kinds. These issues may, indeed, be overriding when we come to choose between the different ways of representing uncertainty - Bayes, for example, may be an easier way of combining information from different sources of uncertainty than possibility theory.

A particularly important problem concerns the way in which we can incorporate deterministic reasoning into our final assessment and judgement of a system. Formal methods of achieving dependability are becoming increasingly important, ranging from formal notations to assist in the elicitation and expression of requirements, through to full mathematical verification of the correspondence between a formal specification and an implementation. One view would be that these approaches to system development remove a certain type of uncertainty, leaving others untouched (uncertainty about the completeness of the formal specification, the possibility of incorrect proof, etc). In which case we need to factor into our final assessment of the dependability of a system the contribution that comes from such deterministic, logical evidence, keeping in mind, though, that there is an irreducible uncertainty about the failure behaviour of a system arising from different sources.

Systems issues

Designers need help in making decisions throughout the design process, but none more so than those at the very highest level. For example, the allocation of dependence between computers, hardware and humans often seems to be carried out rather informally. In addition, real systems often pose difficult problems of assessment because of these early trade-offs

In the Airbus A320, for example, an early decision was to take certain responsibilities out of the hands of the pilot, and place a great deal of trust in the computerised fly-by-wire system. Furthermore, most of the hardware was removed

that would have allowed a pilot, in extremis, to fly the aircraft manually: in the event of complete loss of the computer control system only rudder and tail trim can be activated manually.

In the Sizewell B nuclear reactor, the software-based Primary Protection System (PPS) is backed up by a very much simpler hard-wired secondary system (SPS), which in the event of failure of the PPS can handle most but not all demands. This decision was taken early in the development. Quite late in the day it became clear that it was going to be very difficult to justify having met the original requirement for the PPS of 10^{-4} probability of failure upon demand (or better) - by which time it was too late to increase the coverage of the SPS without considerable delay to the project. A recalculation of the overall reactor safety case then showed that in fact a PPS requirement of only 10^{-3} pfd would satisfy the overall plant safety requirement - at the time of writing this seems to have saved the day, but somewhat fortuitously.

In taking these very early decisions, designers need help in two ways. Firstly, they need to be able to set realistic targets for the dependability of the various system components in order that there will be a good chance of the overall system meeting its reliability and safety targets. In particular, this means that they need to be able to set such targets for software. The example of the A320 illustrates the problem: here the 10^{-9} comes not from a reasoned view of what is achievable with software, but from a crude allocation of responsibilities among many critical subsystems in order to arrive at an overall figure of 10^{-7} for the aircraft as a whole. Most of us would agree that 10^{-9} is far beyond what is actually achievable in software for a system as complex as this. The 10^{-4} goal for the PPS, on the other hand, is a more considered judgement: it was devised as a general claim limit for critical software in UK nuclear power stations [CEGB 1982a, CEGB 1982b].

The second area where designers need help is in designing for eventual validation. It may be that here formal methods and probability modelling can work together. For example, in certain cases it may be possible to argue deterministically about safety - essentially prove that certain classes of safety-related events cannot occur - and leave only reliability to be treated probabilistically. This may have been possible in the Sizewell example, where a software-based, highly functional PPS could have been completely backed up by a simple SPS, guaranteed free of design faults. Such a solution provides the advantages of the software functionality most of the time, but in extremis provides the required system safety without unreasonable dependence upon the software.

4.2 The rôle of evaluation at modest levels of dependability

In this section I will examine a few of the more important general issues concerning dependability evaluation and its relationship to software engineering. There has been no attempt here to be exhaustive, but rather to pick out some areas where a probabilistic and statistical approach could be helpful.

Experimentation and data collection, general statistical techniques

Software engineering is an empirical subject. Questions concerning issues such as the efficacy of software development methods ought to be resolved by observation of real practice or by experiment. Instead, a dearth of data has been a problem in much of this area since its inception. There are still only a handful of published data sets even for the software reliability growth problem, and this is by far the most extensively developed part of dependability modelling. Sometimes the problem arises because there is no statistical expertise on hand to advise on ways in which data can be collected cost effectively. It may be worthwhile attempting to produce general guidelines for data

collection that address the specific difficulties of the software engineering problem domain. Confidentiality issues are a problem here - industrial companies are reluctant to allow access to failure data because it is thought that this will cause people to think less highly of their products. More use could be made of anonymous reporting methods to overcome such difficulties: even in the case of safety critical system failures it might be possible to adopt confidential reporting as has been done successfully for air miss incidents. An alternative would be to make reporting of safety-related incidents mandatory.

Experimentation, with notable exceptions, has so far played a low-key rôle in software engineering research. The most extensive research involving experiments has been, somewhat surprisingly in view of its difficulty and cost, in investigation of the efficacy of design diversity [Anderson *et al.* 1985, Knight & Leveson 1986, Eckhardt *et al.* 1991]. There are other areas where experimental approaches look feasible and should be encouraged. The most obvious question to address would be the general one of which software development methods are the most cost effective in producing software products with desirable attributes such as dependability. Statistical advice on the design of such experiments would be essential, and it may be that innovation in design of experiments could make feasible some investigations here that presently seem too expensive to contemplate.

The main problem in this kind of statistical approach to evidence arises from the need for replication over many software products. On the other hand, there are some areas where experiments can be conducted without the replication problem being overwhelming. These involve the investigation of quite restricted hypotheses about the effectiveness of specific techniques. An example concerns questions related to software testing: are the techniques that are claimed to be effective for achieving reliability (i.e. effectiveness of debugging) significantly better than those, such as operational testing, that will allow reliability to be measured? Such evidence that we have suggests that testing in a way that allows reliability measurement can also be efficient at debugging [Duran & Ntafos 1984]: if this can be shown to be generally true, it would be an important factor in encouraging practitioners to obtain numerical estimates of product reliability.

The influence of the software development process, and other factors, on dependability

Some of the most important and contentious issues in software engineering concern the efficacy of different practices and methods. There is surprisingly little real evidence to support even the most widely accepted claims, for example about the effectiveness of structured programming. In more contentious areas, such as formal methods, the picture is even more bleak. But some tools for such investigations are largely in place. Issues of comparative effectiveness are largely ones of costs and benefits; costs are relatively easy to track, and product reliability is one benefit that is now generally measurable with reasonable accuracy.

The main difficulty in such investigations, when they are based upon real product development, is to identify and control those confounding factors which can interfere with the simple hypothesis under test. Thus if we want to examine whether process A can deliver a given level of reliability more cost-effectively than B, we need to be able to exclude the influence of factors such as quality of personnel, difficulty of problem being tackled, and so on. The simplest way of doing this involves comparisons where all these factors are kept constant, with only the process factors under examination being allowed to vary. Unfortunately, this requires replication of the entire development, which is usually prohibitively expensive for realistic software products (even where this has been done, at great expense, in investigations of the effectiveness of design diversity, the problems tackled have been somewhat artificial).

There are other statistical approaches to these problems. If we could identify all the confounding factors, and measure them, we could build larger regression models. The problem here is that we would require quite large samples of different software product developments, which brings us back to the data collection difficulties of the previous section.

Because of all these difficulties, most investigations of the efficacy of software development process have involved case studies: essentially just single developments in which measurement of interesting factors takes place. Clearly there are strong limitations to what can be concluded from such studies, particularly the extent to which the conclusions generalise outside the immediate context. Nevertheless, realism suggests that this may be the best we can hope for. Even here, it is vital that investigation proceeds within a proper measurement framework [Fenton 1991], and is not reduced to mere anecdote.

The influence of the operational environment on dependability

It can be misleading to talk of ‘the’ reliability of a program: just as is the case in hardware reliability, the reliability of a program will depend on the nature of its use. For software, however, we do not have the simple notions of stress that are sometimes plausible in the hardware context. It is thus not possible to infer the reliability of a program in one environment from evidence of its failure behaviour in another. This is a serious difficulty for several reasons.

In the first place, we would like to be able to predict the operational reliability of a program from test data. The only way that this can be done at present is to be certain that the test environment - i.e. the type of usage - is exactly similar to the operational environment. Real software testing regimes are often deliberately made different from operational ones, since it is claimed that in this way reliability can be achieved more efficiently: this argument is similar to hardware stress testing, but is much less convincing in the software context.

A further reason to be interested in this problem is that most software goes out into the world and is used very differently by different users: there is great disparity in the population of user environments. Vendors would like to be able to predict how different users will perceive the reliability of the product, but it is clearly impractical to replicate every different possible operational environment in test. Vendors would also like to be able to predict the properties of the population of users. Thus it might be expected that a less disparate population of users would be preferable to a more disparate one: in the former case, for example, the problem reports from the different sites might be similar and thus involve less costs in fault fixing.

The data requirements for studies of operational environments are much less severe than for those investigating the relationship between process and product attributes. Manufacturers of, say, operating systems should have considerable quantities of information about the different environments in which a particular product operates. If we could identify interesting explanatory variables - a problem for software engineers rather than statistical modellers - it should be possible to use standard statistical techniques to predict reliability in a novel environment, and other things of interest. There may be other ways of forming stochastic characterisations of operational environments. Markov models of the successive activations of modules, or of functions, have been proposed [Littlewood 1979, Siegrist 1988a, Siegrist 1988b], but have not been widely used. Further work on such approaches, and on the problems of statistical inference associated with them, seems promising.

Stochastic models for security evaluation

Ideally, a measure of the security of a system should capture quantitatively the intuitive notion of ‘the ability of the system to resist attack’. That is, it should be operational, reflecting the degree to which the system can be expected to remain free of security breaches under particular conditions of operation (including attack). Instead, current security levels [NCSC 1985] at best merely reflect the extensiveness of safeguards introduced during the design and development of a system. Whilst we might expect a system developed to a higher level than another to exhibit ‘more secure behaviour’ in operation, this cannot be guaranteed; more particularly, we cannot infer what the actual security behaviour will be from knowledge of such a level.

Clearly there are similarities between reliability and security and it would be desirable to have measures of ‘operational security’ similar to those that we have for reliability of systems. Very informally, these measures could involve expressions such as the rate of occurrence of security breaches (cf rate of occurrence of failures in reliability), or the probability that a specified ‘mission’ can be accomplished without a security breach (cf reliability function). Recent work [Littlewood, Brocklehurst et al. 1994 (to appear)] has started to investigate this analogy between reliability and security and a number of important open questions have been identified that need to be answered before the quantitative approach can be taken further.

Empirical investigation of these issues is difficult: problems of data collection here are even more acute than in the rest of software engineering. Partly this is because security events, by their very nature, are incompletely observed. Another reason is that the security community has tended to concentrate on the problems of ultra-high security, since much of the research funding here comes from military and other government sources. Evaluating operational security at these very high levels is likely to be even more difficult than the problems of evaluating ultra-high reliability - and we have seen that these are essentially insurmountable. Work on stochastic models of security should therefore concentrate on systems which have more modest requirements - analogous to the work that has been successful in reliability modelling.

One approach to the data problem that might be worth pursuing involves experimenting with instrumented systems and sanctioned attackers. Some tentative steps have been taken in this direction [Brocklehurst *et al.* 1994] with qualified success.

5 Conclusion

In this essay I have tried to make clear what I believe to be certain unpalatable truths, and to sugar this pill with some suggestions as to where modest progress might be made.

The first point I wanted to make was that there is an inherent uncertainty about the behaviour of the systems we build, which forces us to talk about their dependability in the language of probability and statistics. This is unpalatable to many computer scientists, who conclude from the deterministic predictability of the machine itself that we can have similar determinism at the macroscopic level of the behaviour observed by a user. Whilst I will concede that arguments of fault-freeness are *possible*, at the price of only building systems of great simplicity, I think that in practice such simplicity will never be achieved. Certainly, I have never seen a computer system used in a context where it would be plausible to claim that the software could never fail.

If readers concede this first point, it seems to me that questions about whether a software-based system is fit for its purpose become questions about whether a particular probabilistic dependability level has been achieved - a problem of numerical evaluation. It is easy to show that we can only answer such questions when the level

required is quite modest: we *cannot* gain confidence in ultra-high dependability without obtaining (literally) incredible amounts of evidence.

I think these difficulties have serious implications for the builders of safety-critical systems, and for society at large. It is easy to be seduced by the extensive functionality that can be provided by software, without the constraints of ensuing hardware unreliability. Some of the benefits from this functionality may indeed be claimed to bring enhanced safety. But at the end of the day we have a right to demand that the system is sufficiently safe, and this cannot be demonstrated for some of the systems that we are building even now. Perhaps now is a time to take stock and consider some retrenchment - for example, deciding that an unstable civil airliner is not a good thing.

All is not gloom. Systems with these dramatic requirements are not that common. For more modest systems, with modest dependability requirements, evaluation is possible. Further research will extend this achievement - but only relatively modestly.

Acknowledgements

My work in this area has been funded over the years by several agencies, too numerous to name here. Most recently, my thinking about the limitations to the evaluation of software dependability has been supported by the CEC ESPRIT programme under project number 6362, PDCS2.

References

- [Abdel-Ghaly *et al.* 1986] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, "Evaluation of Competing Software Reliability Predictions", *IEEE Trans. on Software Engineering*, 12 (9), pp.950-67, 1986.
- [Amman & Knight 1988] P. E. Amman and J. C. Knight, "Data diversity: an approach to software fault tolerance", *IEEE Trans on Computers*, 37 (4), pp.418-25, 1988.
- [Anderson *et al.* 1985] T. Anderson, P. A. Barrett, D. N. Halliwell and M. R. Moulding, "An Evaluation of Software Fault Tolerance in a Practical System", in *Proc. 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, (Ann Arbor, Mich.), pp.140-5, 1985.
- [Barlow & Proschan 1975] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*, 290p., Holt, Rinehart and Winston, New York, 1975.
- [Brocklehurst *et al.* 1990] S. Brocklehurst, P. Y. Chan, B. Littlewood and J. Snell, "Recalibrating software reliability models", *IEEE Trans Software Engineering*, 16 (4), pp.458-70, 1990.
- [Brocklehurst *et al.* 1994] S. Brocklehurst, B. Littlewood, T. Olovsson and E. Jonsson, "On Measurement of Operational Security", in *COMPASS 94 (9th Annual IEEE Conference on Computer Assurance)*, (Gaithersburg), pp.257-66, IEEE Computer Society, 1994.
- [CEGB 1982a] CEGB, *Design Safety Criteria for CEGB Nuclear Power Stations*, Central Electricity Generating Board, N°HS/R167/81, 1982a.
- [CEGB 1982b] CEGB, *Pressurised Water Reactor Design Safety Guidelines*, Central Electricity Generating Board, N°DSG2 (Issue A), 1982b.
- [Dawid 1984] A. P. Dawid, "Statistical theory: the prequential approach", *J Royal Statist Soc, A*, 147, pp.278-92, 1984.
- [Dubois & Prade 1988] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerised Processing of Uncertainty*, Plenum Press, New York, 1988.
- [Duran & Ntafos 1984] J. T. Duran and S. Ntafos, "An evaluation of random testing", *IEEE Trans Software Engineering*, 10 (4), pp.438-44, 1984.

- [Eckhardt *et al.* 1991] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk and J. P. J. Kelly, "An experimental evaluation of software redundancy as a strategy for improving reliability", *IEEE Trans Software Eng*, 17 (7), pp.692-702, 1991.
- [Eckhardt & Lee 1985] D. E. Eckhardt and L. D. Lee, "A Theoretical Basis of Multiversion Software Subject to Coincident Errors", *IEEE Trans. on Software Engineering*, 11, pp.1511-7, 1985.
- [Fenton 1991] N. E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman and Hall, London, 1991.
- [Goel & Okumoto 1979] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures", *IEEE Trans. on Reliability*, 28 (3), pp.206-11, 1979.
- [Henrion & Fischhoff 1986] M. Henrion and B. Fischhoff, "Assessing uncertainty in physical constants", *Americal J. of Physics*, 54 (9), pp.791-8, 1986.
- [Hunns & Wainwright 1991] D. M. Hunns and N. Wainwright, "Software-based protection for Sizewell B: the regulator's perspective", *Nuclear Engineering International*, September, pp.38-40, 1991.
- [Jacky *et al.* 1991] J. Jacky, R. Risler, I. Kalet, P. Wootton, A. Barke, S. Brossard and R. Jackson, "Control system specification for a cyclotron and neutron therapy facility", in *IEEE Particle Accelerator Conference*, (San Francisco), IEEE, 1991.
- [Jelinski & Moranda 1972] Z. Jelinski and P. B. Moranda, "Software Reliability Research", in *Statistical Computer Performance Evaluation* (W. Freiberger, Ed.), pp.465-84, Academic Press, New York, 1972.
- [Knight & Leveson 1986] J. C. Knight and N. G. Leveson, "Experimental evaluation of the assumption of independence in multiversion software", *IEEE Trans Software Engineering*, 12 (1), pp.96-109, 1986.
- [Littlewood 1976] B. Littlewood, "A Semi-Markov Model for Software Reliability with Failure Costs", in *MRI Symp. Computer Software Engineering* pp.281-300, Polytechnic Press, Polytechnic of New York, New York, 1976.
- [Littlewood 1979] B. Littlewood, "Software reliability model for modular program structure", *IEEE Trans Reliability*, 28 (3), pp.241-6, 1979.
- [Littlewood 1981] B. Littlewood, "Stochastic Reliability Growth: A model for fault removal in computer programs and hardware designs", *IEEE Trans. on Reliability*, 30, pp.313-20, 1981.
- [Littlewood 1988] B. Littlewood, "Forecasting software reliability", in *Software Reliability Modelling and Identification* (S. Bittanti, Ed.), Lecture Notes in Computer Science 341, pp.141-209, Springer, Heidelberg, 1988.
- [Littlewood & Miller 1989] B. Littlewood and D. R. Miller, "Conceptual Modelling of Coincident Failures in Multi-Version Software", *IEEE Trans on Software Engineering*, 15 (12), pp.1596-614, 1989.
- [Littlewood & Verrall 1973] B. Littlewood and J. L. Verrall, "A Bayesian Reliability Growth Model for Computer Software", *J. Royal Statist. Soc. C*, 22, pp.332-46, 1973.
- [Miller 1989] D. Miller, "The role of statistical modelling and inference in software quality assurance", in *Software Certification* (B. d. Neumann, Ed.), Elsevier Applied Science, Barking, 1989.
- [Mojdehbakhsh *et al.* 1994] R. Mojdehbakhsh, W.-T. Tsai, S. Kirani and L. Elliott, "Retrofitting software safety in an implantable medical device", *IEEE Software*, 11 (1), pp.41-50, 1994.
- [Musa 1975] J. D. Musa, "A Theory of Software Reliability and its Application", *IEEE Trans. on Software Engineering*, 1, pp.312-27, 1975.
- [Musa & Okumoto 1984] J. D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement", in *Proc. Compsac 84*, (Chicago), pp.230-8, 1984.
- [NCSC 1985] NCSC, *Department of Defense Trusted Computer System Evaluation*, National Computer Security Center, Department of Defense, N°DOD 5200.28.STD, 1985.

- [Rouquet & Traverse 1986] J. C. Rouquet and P. J. Traverse, "Safe and reliable computing on board the Airbus and ATR aircraft", in *Safecom: 5th IFAC Workshop on Safety of Computer Control Systems*, (W. J. Quirk, Ed.), Pergamon Press, 1986.
- [RTCA 1992] RTCA, *Software considerations in airborne systems and equipment certification*, Requirements and Technical Concepts for Aeronautics, N°DO-178B, July 1992 1992.
- [Shafer 1976] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, 1976.
- [Siegrist 1988a] K. Siegrist, "Reliability of systems with Markov transfers of control", *IEEE Trans Software Engineering*, 14 (7), pp.1049-53, 1988a.
- [Siegrist 1988b] K. Siegrist, "Reliability of systems with Markov transfers of control, II", *IEEE Trans Software Engineering*, 14 (10), pp.1478-80, 1988b.
- [Xie 1991] M. Xie, *Software Reliability Modelling*, World Scientific, Singapore, 1991.
- [Abdel-Ghaly, Chan et al. 1986] A.A. Abdel-Ghaly, P.Y. Chan and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. on Software Engineering*, vol. SE-12, no. 9, pp.950-967, 1986.
- [Amman and Knight 1988] P.E. Amman and J.C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Trans on Computers*, vol. 37, no. 4, pp.418-425, 1988.
- [Anderson, Barrett et al. 1985] T. Anderson, P.A. Barrett, D.N. Halliwell and M.R. Moulding. "An Evaluation of Software Fault Tolerance in a Practical System," in *Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, pp. 140-145, Ann Arbor, Mich., 1985.
- [Avizienis and Ball 1987] A. Avizienis and D.E. Ball, "On the achievement of a highly dependable and fault-tolerant air traffic control system," *IEEE Computer*, vol. 20, no. 2, pp.84-90, 1987.
- [Barlow and Proschan 1975] R.E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing*, New York, Holt, Rinehart and Winston, 1975, 290 p.
- [Brocklehurst, Chan et al. 1990] S. Brocklehurst, P.Y. Chan, B. Littlewood and J. Snell, "Recalibrating software reliability models," *IEEE Trans Software Engineering*, vol. SE-16, no. 4, pp.458-470, 1990.
- [Brocklehurst, Littlewood et al. 1994] S. Brocklehurst, B. Littlewood, T. Olovsson and E. Jonsson. "On measurement of operational security," in *COMPASS*, 1994.
- [CEGB 1982a] CEGB. *Design Safety Criteria for CEGB Nuclear Power Stations*, HS/R167/81, Central Electricity Generating Board, 1982a.
- [CEGB 1982b] CEGB. *Pressurised Water Reactor Design Safety Guidelines*, DSG2 (Issue A), Central Electricity Generating Board, 1982b.
- [Dawid 1984] A.P. Dawid, "Statistical theory: the prequential approach," *J Royal Statist Soc, A*, vol. 147, pp.278-292, 1984.
- [Dubois and Prade 1988] D. Dubois and H. Prade. *Possibility Theory: An Approach to Computerised Processing of Uncertainty*, New York, Plenum Press, 1988.

- [Duran and Ntafos 1984] J.T. Duran and S. Ntafos, "An evaluation of random testing," IEEE Trans Software Engineering, vol. SE-10, no. 4, pp.438-444, 1984.
- [Eckhardt, Caglayan et al. 1991] D.E. Eckhardt, A.K. Caglayan, J.C. Knight, L.D. Lee, D.F. McAllister, M.A. Vouk and J.P.J. Kelly, "An experimental evaluation of software redundancy as a strategy for improving reliability," IEEE Trans Software Eng, vol. SE-17, no. 7, pp.692-702, 1991.
- [Eckhardt and Lee 1985] D.E. Eckhardt and L.D. Lee, "A Theoretical Basis of Multiversion Software Subject to Coincident Errors," IEEE Trans. on Software Engineering, vol. SE-11, pp.1511-1517, 1985.
- [Fenton 1991] N.E. Fenton. Software Metrics: A Rigorous Approach, London, Chapman and Hall, 1991.
- [Goel and Okumoto 1979] A.L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures," IEEE Trans. on Reliability, vol. R-28, no. 3, pp.206-211, 1979.
- [Henrion and Fischhoff 1986] M. Henrion and B. Fischhoff, "Assessing uncertainty in physical constants," American Journal of Physics, vol. 54, no. 9, pp.791-798, 1986.
- [Hunns and Wainwright 1991] D.M. Hunns and N. Wainwright, "Software-based protection for Sizewell B: the regulator's perspective," Nuclear Engineering International, vol. September, pp.38-40, 1991.
- [Jacky, Risler et al. 1991] J. Jacky, R. Risler, I. Kalet, P. Wootton, A. Barke, S. Brossard and R. Jackson. "Control system specification for a cyclotron and neutron therapy facility," in IEEE Particle Accelerator Conference, San Francisco, IEEE, 1991.
- [Jelinski and Moranda 1972] Z. Jelinski and P.B. Moranda. "Software Reliability Research," in Statistical Computer Performance Evaluation, pp. 465-484, New York, Academic Press, 1972.
- [Knight and Leveson 1986] J.C. Knight and N.G. Leveson, "Experimental evaluation of the assumption of independence in multiversion software," IEEE Trans Software Engineering, vol. SE-12, no. 1, pp.96-109, 1986.
- [Littlewood 1976] B. Littlewood. "A Semi-Markov Model for Software Reliability with Failure Costs," in MRI Symp. Computer Software Engineering, pp. 281-300, Polytechnic of New York, New York, Polytechnic Press, 1976.
- [Littlewood 1979] B. Littlewood, "Software reliability model for modular program structure," IEEE Trans Reliability, vol. R-28, no. 3, pp.241-246, 1979.
- [Littlewood 1981] B. Littlewood, "Stochastic Reliability Growth: A model for fault removal in computer programs and hardware designs," IEEE Trans. on Reliability, vol. R-30, pp.313-320, 1981.
- [Littlewood 1988] B. Littlewood. "Forecasting software reliability," in Software Reliability Modelling and Identification, pp. 141-209, Heidelberg, Springer, 1988.
- [Littlewood, Brocklehurst et al. 1994 (to appear)] B. Littlewood, S. Brocklehurst, N.E. Fenton, P. Mellor, S. Page, D. Wright, J.E. Dobson, J.A. McDermid

- and D. Gollmann, "Towards operational measures of computer security," *Journal of Computer Security*, 1994 (to appear).
- [Littlewood and Miller 1989] B. Littlewood and D.R. Miller, "Conceptual modelling of coincident failures in multi-version software," *IEEE Trans on Software Engineering*, vol. SE-15, no. 12, pp.1596-1614, 1989.
- [Littlewood and Verrall 1973] B. Littlewood and J.L. Verrall, "A Bayesian Reliability Growth Model for Computer Software," *J. Roy. Statist. Soc. C*, vol. 22, pp.332-346, 1973.
- [Miller 1989] D. Miller. "The role of statistical modelling and inference in software quality assurance," in *Software Certification*, Barking, Elsevier Applied Science, 1989.
- [Mojdehbakhsh, Tsai et al. 1994] R. Mojdehbakhsh, W.-T. Tsai, S. Kirani and L. Elliott, "Retrofitting software safety in an implantable medical device," *IEEE Software*, vol. 11, no. 1, pp.41-50, 1994.
- [Musa 1975] J.D. Musa, "A Theory of Software Reliability and its Application," *IEEE Trans. on Software Engineering*, vol. SE-1, pp.312-327, 1975.
- [Musa and Okumoto 1984] J.D. Musa and K. Okumoto. "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," in *Proc. CompSAC 84*, pp. 230-238, Chicago, 1984.
- [NCSC 1985] NCSC. Department of Defense Trusted Computer System Evaluation, DOD 5200.28.STD, National Computer Security Center, Department of Defense, 1985.
- [Rouquet and Traverse 1986] J.C. Rouquet and P.J. Traverse. "Safe and reliable computing on board the Airbus and ATR aircraft," in *Safecomp: 5th IFAC Workshop on Safety of Computer Control Systems*, Pergamon Press, 1986.
- [RTCA 1992] RTCA. Software considerations in airborne systems and equipment certification, DO-178B (Draft), Radio-Technical Commission for Aeronautics, 1992.
- [Shafer 1976] G. Shafer. *A Mathematical Theory of Evidence*, Princeton University Press, 1976.
- [Siegrist 1988a] K. Siegrist, "Reliability of systems with Markov transfers of control," *IEEE Trans Software Engineering*, vol. SE-14, no. 7, pp.1049-1053, 1988a.
- [Siegrist 1988b] K. Siegrist, "Reliability of systems with Markov transfers of control, II," *IEEE Trans Software Engineering*, vol. SE-14, no. 10, pp.1478-1480, 1988b.
- [Xie 1991] M. Xie. *Software Reliability Modelling*, Singapore, World Scientific, 1991.