RIGOROUSLY ASSESSING SOFTWARE RELIABILITY AND SAFETY

Lorenzo Strigini and Norman Fenton
Centre for Software Reliability, City University, Northampton Square, London EC1V OHB, UK
Tel.: +44 171 477 8421; Fax: +44 171 477 8585; E-mail: l.strigini@csr.city.ac.uk, n.fenton@csr.city.ac.uk

ABSTRACT

This paper summarises the state of the art in the assessment of software reliability and safety ("dependability"), and describes some promising developments. A sound demonstration of very high dependability is still impossible before operation of the software; but research is finding ways to make rigorous assessment increasingly feasible. While refined mathematical techniques cannot take the place of factual knowledge, they can allow the decision-maker to draw more accurate conclusions from the knowledge that is available.

INTRODUCTION

A *quantitative*, probabilistic assessment of software dependability is usually necessary, due to: i) the need to estimate the effect of potential software defects on the reliability and safety of the system in which the software is embedded, and ii) the uncertain knowledge about the possible defects and the process that activates them, which can only be described in probabilistic terms. It may not require that precise probabilities are assigned, but at least it requires trustworthy statements about *comparisons* of probabilities between different scenarios that the decision-maker has to consider.

THE PROBLEMS WITH SOFTWARE RELIABILITY ASSESSMENT

An objection to probabilistic assessment is that it is ineffective, as it must rely on statistical testing, and for some required reliability levels infeasibly large amounts of testing would be needed. As a result, in most of the industry, dependability assessment is limited to a qualitative appraisal of the development process. Probabilities, if required, are then derived via informal "engineering judgement". Experimental research in psychology clearly shows that engineering judgement is trusted much more than it should be [1, 2]. The likely result are unfounded claims, and no knowledge of the levels of dependability actually achieved. Such judgement tends to be coded into standards and guidelines for the development of critical software: given a required level of reliability, a standard prescribes a set of techniques, which are arguably useful or necessary for achieving that level, but definitely not sufficient. Standard rules make decision-making simpler and legally safer, but are practically dangerous when not based in scientific knowledge.

The problem is in the nature of software failures, which are due to design defects. When we test a mechanical or electronic device, we usually enjoy two advantages:

- we are confident that any design defects or systematic manufacturing defects are either absent, in devices produced by a well-controlled process from a trusted design, or, e.g. when testing a new model, can be revealed by the tests we have chosen. For software, design defects are the norm, even after testing;
- since the device has continuous behaviour, we can trust test results in certain operating conditions to be representative of its behaviour over a range of similar conditions, and we can accelerate testing by making it more stressful. This applies not only to failures from wear and tear, but also to many failures caused by design defects. The discontinuous nature of software mostly precludes these advantages.

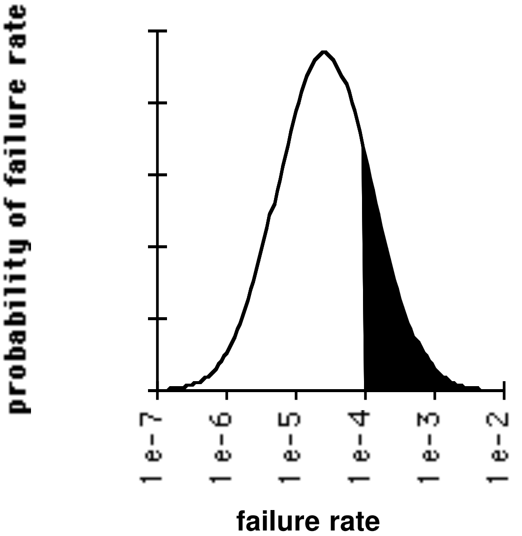
So, the main basis for trusting a physical product to be free from defects is trust in the design, manufacture and testing process. Testing is applied to each new design, and it is quite effective at detecting residual design defects. As a result, probabilistic assessments of reliability and safety concentrate on physical failures. This only becomes a problem when the design is complex and has discontinuous behaviour, e.g. in the thinking processes of human operators, or in the propagation of failures among subsystems. At the level of complexity of large industrial plants and aircraft, design defects may thus be a problem again. Current software standards tend to treat software as a simple product, where excellence of design process may ensure the absence of serious defects; yet, all available evidence indicates that software products are complex systems, and defects are always present.

We are thus compelled to use the approach of *statistical testing* to estimate the failure rate of programs. In this approach, we attempt to test the software in a way that is statistically representative of how it will actually be used in operation. This differs starkly from traditional software testing methods which focus on fault-finding. One would hope that, over time, a software production process can be made more predictable, so that acceptably low failure rates can be estimated on the basis of the process itself. This would require both a process that appears reliable and repeatable, *and* measured high reliability over its products. We are aware of only two organisations (the then IBM unit at Houston and Hitachi in Japan) who even claim that these factors are already present [3, 4].

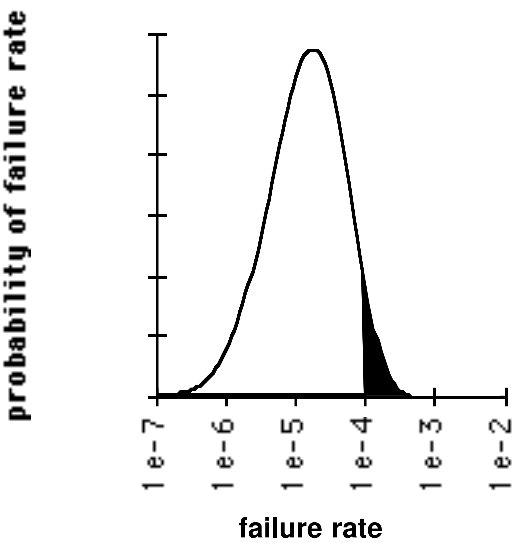
Without using any process evidence at all, a plausible inference process based on statistical testing only suggests that after N successful test cases, the expected failure rate of the software is of the order of 1/N [5, 6] .

The following four figures may help to explain the effects of non-test-based knowledge on dependability assessment. They represent probability distributions for the failure rate of a software product: a point on the *x* axis represents a value of failure rate. However, they differ in representing the effects of different process-based evidence, combined with the same amount of testing.

Presented at ESA Software Product Assurance Workshop, Nordvjik, March 96.



Product of "mediocre" process, with evidence from previous, similar products: distribution of predicted failure rate, *before* testing. Expected value of failure rate: 9 10^{-5}
Risk of failure rate greater than 10^{-4}: 21%

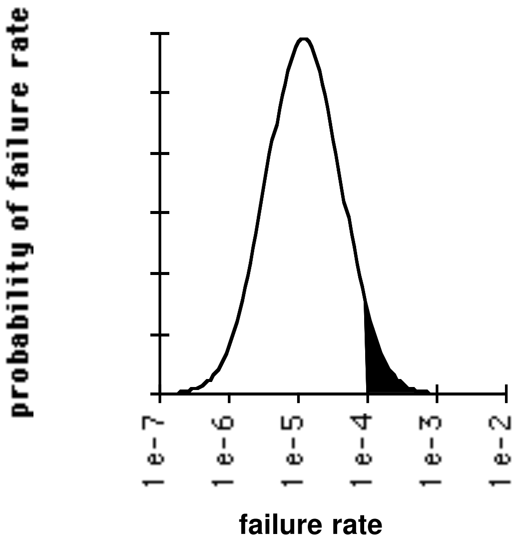


Product of "mediocre" process, with evidence from *this* product: distribution of predicted failure rate, *after* testing. Expected value of failure rate: 3 10^{-5}
Risk of failure rate greater than 10^{-4}: 5%

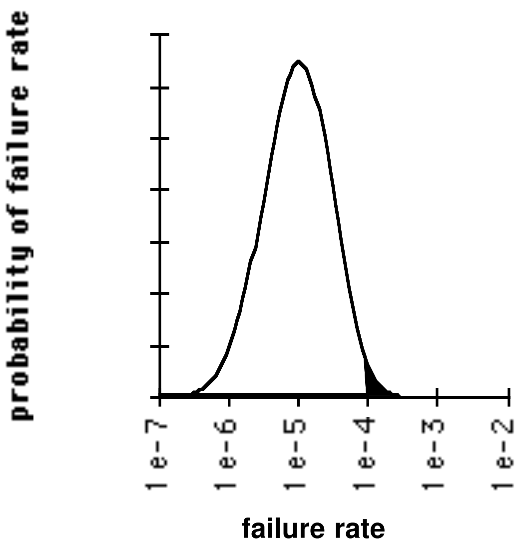
The first figure represents such a distribution (here just a plausible, arbitrary distribution, chosen for the sake of illustration), assuming it is predicted from process information only, *before* testing the software. This prediction could be based on how reliably previous, similar products of the same process have been observed to behave. We have assumed a reliability requirement that the failure rate be no greater than 10^{-4} per demand. The darkened part of the graph corresponds to failure rates that do not satisfy the requirement: its area represents the probability that the software is too unreliable. This figure represents a case of somewhat weak process evidence: for instance, the average failure rate is barely within the required bound. More importantly, the probability that the actual failure rate is unsatisfactory, i.e., the project risk, is 21%. This was clearly a risky project.

The second figure takes into account the fact that the software passed 10,000 tests, i.e., evidence specific to this individual product, in addition to the generic, process evidence on which the previous distribution was based. The new distribution shown has been obtained by a standard Bayesian procedure (which we discuss in the next section). The risk of a bad product has decreased to 5 %, which may be of some reassurance to a customer.

The next two figures describe another distribution prior to testing. This is clearly more favourable than the first one, and the risk is "only" 5 %. After testing, the risk has decreased to 1 %.



Product of "improved" process: distribution of predicted failure rate, *before* testing. Expected failure rate: 3 10^{-5}.Risk of failure rate greater than 10^{-4}: 5 %



Product of "improved" process: distribution of predicted failure rate, *after* testing. Expected failure rate: 2 10^{-5}. Risk of failure rate greater than 10^{-4}: 1 %

We thus see the role of testing in reducing the probability that a dangerously unreliable product is deployed, together with the effect that we could expect from a well-founded knowledge that the process usually delivers good software.

The demand for assessment methods that are not limited to statistical testing is clearly justified. Software often proves much more reliable than it could be claimed to be on the basis of testing. Qualitative, process-based assessment attempts to overcome this apparent paradox, using knowledge that is available to the assessor: the use of good development methods, the past record of a development organisation, and so on. The problem is in the *practice* of assessment: whatever knowledge is available is not used in a rigorous manner; and the available knowledge may not be enough to support the desired conclusions. An example is given by "maturity models", like e.g. CMM [7]. It is plausible that applying the CMM scheme benefits the software industry and its customers, but not that the benefits include being able to specify a high CMM level as a contractual requirement *and thus assure* a required level of reliability of the software product. There is no evidence that CMM levels correlate (across developers) with product reliability. One of the expected advantages of higher CMM levels, i.e., repeatability of the process, would improve the assurance that can be drawn from process evidence; **but** this advantage cannot be realised without statistics of the reliability actually delivered by the process.

The problem of limited knowledge is actually two-fold: first, *general* knowledge about software engineering methods is very limited [[8-10]; second, reliability requirements are often so stringent that no amount of *specific* knowledge that it is feasible to collect about an individual product can demonstrate their attainment [6].

SOUND METHODS FOR COMBINING EVIDENCE

Mathematical techniques for dealing with uncertain knowledge have existed for a long time. We illustrate here some of their applications. For instance, the four figures above demonstrate the use of Bayesian inference, in updating the beliefs that can be held prior to testing with the results of testing. Bayesian probabilistic reasoning is the main tool available for combining diverse evidence into a reliability assessment. It offers a language and calculus (not the only one proposed, but the most mature and well-developed, and arguably the most convincing) for reasoning about the beliefs that can be reasonably held, in the presence of uncertainty, about future events, on the basis of available evidence. *Prior* probabilities are thus updated, after new events are observed, to produce *posterior* probabilities. By repeating this process, the implications of multiple sources of evidence can be calculated in a consistent way. Questions in software reliability assessment that can be answered with these techniques are, for instance:

- given that statistical testing is used, with reliability requirements for which it is practical, what are the consequences of finding a fault (decreasing confidence in the product)? Specifically, how should one change the number of tests required on the software after the faults is fixed? [11]
- given an estimate of the fault-revealing power of tests, how can one estimate the reliability of software which did not fail during testing? [12, 13]

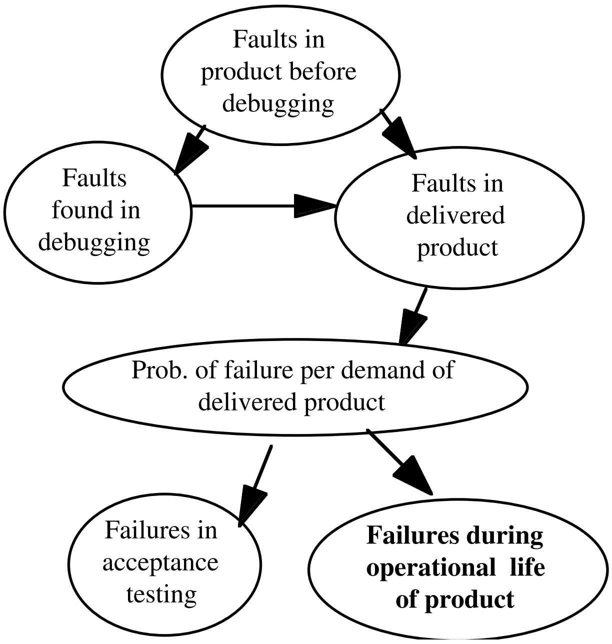
The "prior beliefs" are clearly the most difficult input to these methods of reasoning. Yet they are necessary to model sound reasoning. One can avoid them by using "classical" statistical inference, deriving statements like "after N successful tests, I have confidence C that the software has failure rate lower than q". But classical inference does not model an important part of judgement: after N successful tests on two programs, the confidence bounds derived are identical, even if one program is the professional product of the best developers on the market and the other program is known to be low-quality work of unskilled amateurs. It is these additional factors that must be captured in the choice of prior probabilities.

The most favourable condition is that in which the evidence from testing is so overwhelming that the prior beliefs have little weight in comparison. However, as we have stated, for stringent reliability requirements this is seldom the case. The prior beliefs must be based on process-related evidence. As a minimum, one could try to exploit the available track record of an organisation, ideally represented by the reliability demonstrated by its previous products. The paper [14] illustrates a procedure for this derivation.

It seems desirable to take into account explicitly all the factors that affect the reliability of a software product: proficiency of developers, effectiveness of tools, effectiveness of inspections and debug testing, effects of specification and programming languages, specific difficulties of an application or a specific project, etc. The usual impediment to using this multiple evidence has been that the reasoning needed is overly complex. However, computer tools for manipulating so-called *Bayesian belief networks* (BBNs for brevity) have pushed back the boundary of the problems that can be attacked. BBNs offer a visually intuitive language for representing probabilistic relationships between events. Their use will be illustrated by a couple of examples. The importance of this language lies in the fact that it is intuitive enough to help in manipulating and communicating complex webs of inference, and yet it has a rigorous, mathematical meaning so that software tools can interpret it and perform the complex calculations needed in its use.

Our preliminary investigations on Bayesian belief networks show great promise of making dependability assessment easier to describe, and thus to check, communicate and audit [15, 16].

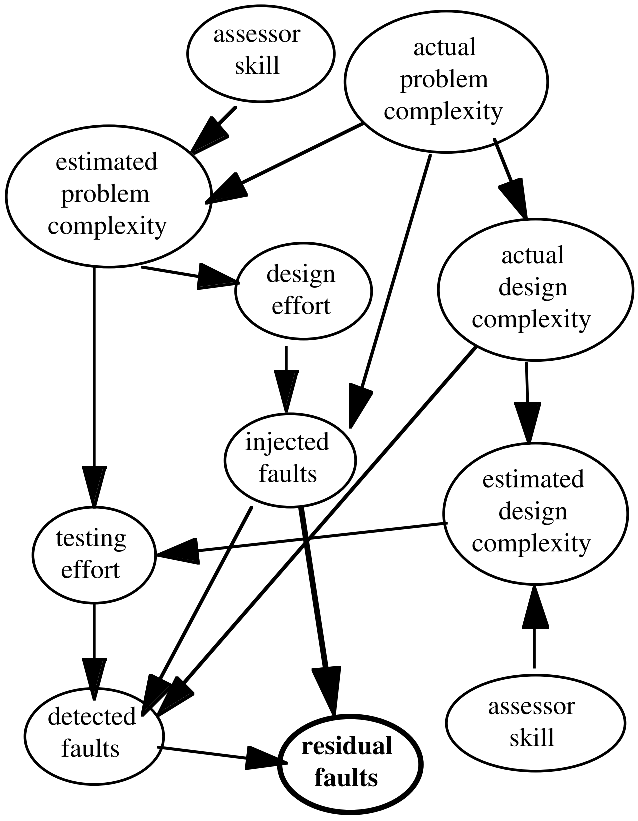
Two examples of BBNs are illustrated below. The first one (from [15]) uses comparatively little evidence, depending only on the observed reliabilities and defect counts of previous products of the same process, and on the defects discovered in the current product during debugging. The topology of the graph is used to indicate probabilistic relationships among the variables described in the nodes. For instance, knowing the number of faults present in the product before debugging (top node in the graph) would allow one to state the probabilities of finding one, two, etc., faults during debugging. This conditional probability distribution (representing one's knowledge about the fault-finding effectiveness of the debug process), as well as the probability distribution for the number of faults present, and conditional probability distributions for all the other nodes, are represented in data structures associated with the graph: via the graph plus these distributions, one represents previous knowledge about the product to be assessed (including the process that produced it).



This previous knowledge thus implies prior probabilities for the values of each variable (i.e., each node) in the belief network. When an event (value of a variable) is actually observed, all these can be updated, using Bayes' theorem, obtaining posterior probabilities that take into account the events observed. Both calculations can be performed automatically by software tools.

Bayesian nets also allow an injection of scientific rigour when these probability distributions are simply "expert opinions". A BBN will derive all the implications of the beliefs input to it, and some of these implications are statements of fact that can be checked against the observed reality of a software project, or simply against the experience of the experts and decision makers themselves. The second BBN shown, from [17], includes some more subjective indicators, like problem complexity. Thus, this network is meant to be populated with probabilities that are not all derived from statistical inference, but at least in part from "expert opinion". the advantage of using belief networks is then that of checking the *consistency* of one's beliefs and one's decisions.

The ability, through BBNs, to make one's assumptions and detailed reasoning explicit and check their consistency (internal as well as with other known facts and experience of the experts), may substantially improve the way the available information is used in decision making.



THE NEED FOR MEASUREMENT

Methods like Bayesian calculus, assisted by software tools, allow us to make the best use of the complex evidence available - our factual knowledge about a software product - in predicting software dependability. Our ability to predict very high dependability would thus no longer be limited by the complexity of reasoning about such complex knowledge, but by the extent of the knowledge itself. Apart from the *direct* knowledge that comes from statistical testing, our knowledge on the impact of process factors on dependability consists of data about their relationship with observed dependability in classes of products. Such data are now scarce and inconclusive. Some progress can be expected from the increasing popularity of measurement programmes within the industry. If sound, such programmes will allow companies to document the effectiveness of the methods they use: confidence in claims derived from process-related evidence will increase. The degree and importance of this increase can be calculated through formalised, computer-aided reasoning techniques like BBNs.

A well managed software development project could have available a wealth of potentially important quantitative information to support safety assessment: data from testing; fault and change reports from various project phases; test results including coverage measures; outputs from static analysis or metrics tools; various internal system quality indicators such as modularity measures; measures of effort associated with various project phases; last but not least, historical evidence of efficacy of tools and techniques. In particular, good data about software defects and changes, with adequate records of previous projects, can provide important quantitative information on which to base a reliability assessment or safety case.

Measurement of appropriate indicators during projects has been shown to help in project management, in particular early detection of problems [18, 19]. However, before it can be used for usefully strong predictions of reliability, it must have been applied to enough projects to collect some useful experience. In other words, introducing measurement practices can produce an immediate payoff in *achieving* quality (including reliability) but only a delayed payoff in the ability to predict markedly higher levels of operational dependability. In the short term, direct evidence (from statistical testing) will still be the only basis for strong predictions, and these will be limited by the amount of testing that is feasible.

DECISIONS WITH LIMITED DATA

As we pointed out earlier, Bayesian methods allow rigorous reasoning with uncertain knowledge, irrespective of whether abundant statistical data are available. However, they require the problem to be fully described in probabilistic terms, which may be difficult for an untrained user. Rigorous methods for structuring one's decision are also available for situations in which the available knowledge is more difficult to express in terms of probabilities, as those studied in the field of Multi-Criteria Decision Aid (MCDA) [20, 21].

There are three classes of methods which come under the umbrella of MCDA. The most well known is *multiple attribute utility theory* (MAUT). All the methods in this class attempt to optimise some utility function, defining a strict order over the set of all possible decisions. One such method is the Analytical Hierarchy Process [22]. This was used in the SHIP project to assess dependability of PLC’s [23, 24]. It was also used [25] to assess (theoretically) the ‘best’ way to improve safety on the Space Shuttle. The problem with MAUT is that, in forcing a strict order over the set of decisions, it makes very strong assumptions about the underlying criteria; in particular, there is an assumption that the criteria are measurable on a *ratio* (rather than simply an *ordinal*) scale. This is generally unrealistic for software dependability assessment. Thus we have experimented with a second class of MCDA methods called *outranking methods*, which depend on much less stringent assumptions. The result is that you get a partial (as opposed to a strict) order over the set of decisions. This then means that your choice is narrowed down to the set of decisions which are optimal in the partial ordering. The third class of MCDA methods are the interactive methods whereby the set of decisions is incrementally narrowed by interactive techniques (after each ‘round’ the decision maker is asked to input additional information).

These methods allow the decision-maker to articulate requirements and decision criteria, and choose among available alternatives in a sensible way: while they do not offer "optimality" in the same sense that Bayesian decision theory does, they do prevent most of the inconsistencies commonly associated with such decision-making.

STANDARDS AND GUIDELINES

These considerations also have implications for standards and guidelines applied to the development and procurement of critical software. It is now impossible to prescribe sets of techniques that will *guarantee* a certain level of reliability. Standards are useful for setting minimum requirements (although standard-makers should be wary of prescribing detailed techniques and thus possibly impeding the adoption of better alternatives), and must certainly be continuously updated as new findings recommend the use of a specific technique over another, or increase the confidence that can be derived from a specific technique. But we should not expect some methodological breakthrough to eliminate the uncertainty in software dependability assessment. We should, rather, expect our uncertainty to be slowly reduced. There is a danger for customers and regulators in a "prescriptive" approach, which effectively allows a software developer to state "I complied with all the prescriptions in the standard, therefore my software must be considered acceptable". They may be better advised to adopt an approach more similar to the "safety case" now typical of, e.g., the offshore industry. The developer would then have to build a convincing argument explicitly linking the relevant evidence with the claim that the software is acceptable (i.e.., presents a sufficiently low risk) for its use. These arguments would be organisation- and application-specific, thus taking into account factors, like quality of personnel, that are extremely important but difficult to treat in a prescriptive fashion. They would depend on data collected in comparable circumstances, typically within the same organisation. Tools like BBNs would make the argument easier to check and negotiate between the parties.

A useful addition to current standards would be directives for collecting and analysing reliability data, and thus fostering both continuous improvement and better assessment. In addition to project history data, the monitoring of actual operation is highly desirable: for instance, online recording of failures (e.g., discrepancies in voted systems); logging of periods of operation of a system, of its modes of operation and of the failures observed, and analysis of their causes. These activities benefit the industry in the long term, but may be difficult to justify among the costs of an individual project, unless there is a consensus (a standard) on their desirability. Over time, the collected data would produce useful knowledge like, for instance, the spread in reliability to be expected for a given subsystem in different uses, or the likelihood that a "formal verification" method actually guarantees a product free from design faults of the pertinent class, as a function of the complexity of the product.

CONCLUSIONS

We have recalled the reasons of the problems which afflict software reliability assessment, and why only direct evidence of observed reliability can produce strong predictions about future reliability. The source of all problems is the simple fact that the strength of these predictions is actually commensurate to the effort spent in observation (testing effort, or time in actual operation).

We have unavoidably been led to point at the paucity of the knowledge on which predictions, as well as project decisions, have now to be based, and hence to the need for better measurement.

We have described two rigorous approaches (BBNs and MCDA) which can be used to combine evidence to support a safety assessment, irrespective of what particular measurement data are known. In both approaches the very act of modelling reaps an immediate dividend in terms of visibility of assumptions and arguments. This makes for sounder reasoning, after which the second advantage of these methods, the availability of computer support, can be used for the complex calculations that derive the consequences of such reasoning.

What are direct implications of these considerations for the space industry in particular? They are not very different from those for any other industry that sees its dependence on software increase together with the size and complexity of the software it uses:

- need for more measurement. Both development organisations and client organisations need to chart where they stand and where they are steering from there;
- need for realistic requirements. If a system design imposes software reliability requirements which, upon examination, can be satisfied with a probability of 99 %, or even 80 % or 50 %, the software development organisation may be stimulated to well reasoned steps to improve quality and reduce risk. If instead the reliability required is so high that the chance of obtaining it is small and uncertain, the software developers can certainly do their best to achieve it, but any attempt to rational decisions and assessment will be discouraged in favour of defensive hand waving, invoking the protection of standards, and "playing the number game" . A customer or regulator that accepts such a system design is asking to be deceived.

In summary, more emphasis on collecting hard evidence and using it well would allow possibly more modest claims, but sounder decisions.

ACKNOWLEDGEMENTS

This work was funded in part by ESPRIT Long Term Research Project 20072 "DeVa", and by the U.K. EPSRC and DTI through the DATUM project (grant GR/H89944, project number IED4/19314).

REFERENCES

[1] L. Strigini, “Limiting the Dangers of Intuitive Judgment in Decision Making,” *IEEE Software, "Quality Time"*, January, pp. 101-103, 1996.

[2] P. Ayton, “On the Competence and Incompetence of Experts", in *Expertise and Decision Support*, G. Wright and F. Bolger, Eds.: Plenum Press, 1993, pp. 77-105.

[3] K. Yasuda and K. Koga, “Product development and quality in the software factory", in *Software Quality Assurance and metrics: A Worldwide perspective*, N. E. Fenton, R. W. Whitty, and Y. Iizuka, Eds.: International Thomson Press, 1995, pp. 195-205.

[4] T. Keller, “Measurements role in providing "error-free" onboard shuttle software", 3rd Intl Applications of Software Metrics Conference, La Jolla, California, 1992.

[5] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas, “Estimating the Probability of Failure When Testing Reveals No Failures", *IEEE Transactions on Software Engineering*, vol. 18, pp. 33-43, 1992.

[6] B. Littlewood and L. Strigini, “Validation of Ultra-High Dependability for Software-based Systems", *Communications of the ACM*, vol. 36, pp. 69-80, 1993.

[7] M. Paulk, W. C. V., and B. Curtis, *The Capability Maturity Model for Software: Guidelines for Improving the Software Process,*: Addison Wesley, 1994.

[8] N. Fenton, “How effective are software engineering methods?", *Journal of Systems and Software*, vol. 20, pp. 93-100, 1993.

[9] N. Fenton, “Software Measurement: a Necessary Scientific Basis", *IEEE TSE*, vol. SE-20, pp. 199-206, 1994.

[10] N. Fenton, S. Pfleeger, and R. Glass, “Science and Substance: A Challenge to Software Engineers", *IEEE Software*, July, pp. 86-95, 1994.

[11] B. Littlewood and D. Wright, “On a Stopping Rule for the Operational Testing of Safety Critical Software", 25th Annual International Symposium on Fault -Tolerant Computing), Pasadena, 1995.

[12] A. Bertolino and L. Strigini, “On the use of testability measures for dependability assessment", *IEEE TSE*, vol. 22, pp. 97-108, 1996.

[13] A. Bertolino and L. Strigini, “Predicting Software Reliability from Testing Taking into Account Other Knowledge about a Program", to be presented at Proc. Quality Week '96, San Francisco, 1996.

[14] B. Littlewood and D. Wright, “A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability", presented at SafeComp95, Belgirate, Italy, 1995.

[15] K. A. Delic, F. Mazzanti, and L. Strigini, “Formalising a software safety case via belief networks", SHIP Project Technical Report T046, July 1995.

[16] M. Neil, B. Littlewood, and N. Fenton, “Applying Bayesian Belief Networks to Systems Dependability Assessment", Proc. Fourth Safety-Critical Systems symposium, Leeds, U.K., 1996.

[17] N. E. Fenton, “The role of measurement in software safety assessment", , Report DATUM/CSR/11, 1995.

[18] S. Henry, C. Selig, “Design metrics which predict source code quality", *IEEE Software*, March, 1990.

[19] O. Johansson and C. Nord, “Using predictions to improve software reliability", *Ericsson Review*, vol. 1, pp. 30-35, 1995.

[20] N. E. Fenton, “Multi-criteria Decision Aid; with emphasis on its relevance in dependability assessment", Report CSR DATUM/CSR/02, 1995.

[21] P. Vincke, *Multicriteria Decision Aid*. New York: J. Wiley, 1992.

[22] T. Saaty, *The Analytic Hierarchy Process*. New York: McGraw Hill, 1980.

[23] A. Auer, “A judgement and decision making framework", SHIP Project Report SHIP/T/013, 1994.

[24] A. Vaisanen, A. Auer, and J. Korhonen, “Assessment of the safety of PLCs: Janiksenlinna water plant study", SHIP Project Report SHIP/T/033, 1994.

[25] M. V. Frank, “Choosing among safety improvement strategies: a discussion with example of risk assessment and multi-criteria decision approaches for NASA", *Reliability Engineering and System Safety*, vol. 49, pp. 311-324, 1995.