Bertolino, A. & Strigini, L. (1998). Assessing the Risk due to Software Faults: Estimates of Failure Rate versus Evidence of Perfection.. Software Testing, Verification and Reliability, 8(3), 155 - 166. doi: 10.1002/(SICI)1099-1689(1998090)8:3<155::AID-STVR163>3.0.CO;2-B <http://dx.doi.org/10.1002/(SICI)1099-1689(1998090)8:3<155::AID-STVR163>3.0.CO;2-B>

Assessing the risk due to software faults: estimates of failure rate *vs.* evidence of perfection

Antonia Bertolino[1] and Lorenzo Strigini[2]

1. Istituto di Elaborazione della Informazione del CNR,
Via Santa Maria, 46, 56126 Pisa, Italy.
E-mail: bertolino@iei.pi.cnr.it

2. Centre for Software Reliability, City University,
Northampton Square, London EC1V OHB, U.K.
E-mail: strigini@csr.city.ac.uk

ABSTRACT

In the debate over the assessment of software reliability (or safety), as applied to critical software, two extreme positions can be discerned: the "statistical" position, which requires that the claims of reliability be supported by statistical inference from realistic testing or operation, and the "perfectionist" position, which requires convincing indications that the software is free from defects. These two positions naturally lead to requiring different kinds of supporting evidence, and actually to stating the dependability requirements in different ways, not allowing any direct comparison. There is often confusion about the relationship between statements about software failure rates and about software correctness, and about which evidence can support either kind of statement. This note clarifies the meaning of the two kinds of statements and how they relate to the probability of failure-free operation, and discusses their practical merits, especially for high required reliability or safety.

Index terms:

Software reliability and safety, program correctness, software assessment, software safety standards, statistical testing.

1. **Introduction**

This paper discusses approaches to the assessment of software reliability before the software is put into operation. In the debate about how this assessment can be performed, especially for critical software with high required reliability, there is often confusion about how the evidence gathered to support trust in a product actually relates to the assurance that is sought. It is useful to identify two extreme positions in this debate, which can be called the "statistical" approach and the "perfectionist" approach, compare them and examine how they affect the practice of software assessment.

In the statistical approach, the software is executed in a test environment reproducing operational usage. Its failures (or their absence) are monitored, and then statistical inference is used (Parnas et al., 1990; Miller et al., 1992; Littlewood and Strigini, 1993; Littlewood and Wright, 1997) to predict its failure rate in operation, or other related measures.

The perfectionist approach, on the other hand, aims at reaching confidence that, with respect to the requirements of interest, the software is free of defects, or "perfect". This approach emphasises the use of "best development practices", document reviews and inspections, systematic testing and formal proofs. Evidence for assurance comes from the application of these methods.

Both approaches pose problems.

The intrinsic problem with the perfectionist approach is that "perfection" cannot be demonstrated with certainty. Empirically, many programs are found to contain bugs even after

This is a preprint of an article accepted for publication in the Journal of Software Testing, Verification and Reliability.

they were checked with great rigour. Theoretically (Barwise, 1989; Fetzer, 1988), even correct mathematical proofs only prove properties of abstract entities (e.g., that program code satisfies formally stated requirements), not of physical objects (e.g., that a computer where that code has been loaded will behave so as to satisfy the actual requirements of its users).

When a software product must be assessed on the basis of "process" evidence, the ideal situation is one in which the assessor has documented evidence that a certain production or verification process usually delivers fewer defects than others. This seldom happens in practice. Even if confident that a given process delivers products with few defects, an assessor would not generally know the failure rates associated with these defects. In fact, there may be wide variations between different products of this same process. Thus, the only well-developed and trustworthy approach to obtaining a *quantitative* demonstration of reliability is via statistical testing.

The main problem with the statistical approach is that, to trust the reliability estimates it produces, one must trust that the input distribution used during testing is a good approximation of the operational distribution in the environment of interest, which is difficult to judge. Claims are even often heard that the usual concepts of reliability, as used in other branches of engineering, have no meaning for software, a view which is refuted in (Littlewood and Strigini, 1993).

The perfectionist attitude appears to be the prevailing, though often implicit, attitude in industrial practice. For instance, standards for safety-critical software mostly prescribe methods for avoiding defects in the delivered product, while statistical evidence that the software is indeed reliable enough is either not required at all (RTCA/EUROCAE, 1993) or given a very marginal role (IEC, 1995). This form of prescriptions makes sense if one expects from their application a good chance of delivering defect-free products. Otherwise, the standard-makers should be much more concerned with the *probability* that the *remaining* defects will cause failures (or accidents) when the product is used.

The prevailing culture in reliability engineering and probabilistic safety assessment focuses on the estimation of event rates (often of upper bounds on failure or accident rates). So, software reliability assessors usually try to translate statements about best practice and stringent verification into statements about failure rates. One (unjustifiable) way of doing this is to claim that applying the prescriptions found in a IEC 1508-style standard (IEC, 1995) for a given level of (probabilistic) requirements *assures* the achievement of that level. But these arguments do not seem to match what most practitioners actually think. In the authors' (admittedly limited) personal experience, the people who are responsible for building or approving life-critical software do not think of, say, an additional verification step as a way of lowering an upper confidence bound on the software's failure rate; they think of it as a way of increasing the chance of the software being free of dangerous defects. However, these statements of "freedom from defects" or "perfection" do not fit well in arguments that centre on failure rates. Because of this heterogeneity, it is usually difficult to understand which one, between a statistical and a perfectionist argument, is stronger, or how they support each other (if at all). Surprisingly, no explicit explanation of this relationship seems to be available in the software engineering literature, although some authors (Hamlet, 1987; Hamlet, 1992; Howden and Huang, 1995) have pointed out perceived shortcomings of the statistical approach (especially the dependence on the operational usage profile) and suggested methods for quantifying confidence in software correctness.

The contribution of this paper is a basic and clear statement of the relationship between statements of confidence about a product being perfect and about it having a low failure rate, and their respective relevance for predictions about a system's lifetime behaviour. The two kinds of statements are described in common probabilistic terms. This description is a necessary basis for any argument which compares or combines the two points of view.

2. Terminology

Programs are subject to *demands* (in general, sequences of inputs). A program is *correct* if it executes correctly for *every* possible demand; otherwise the program is *faulty*. Every time a program fails to execute correctly, this event is called a *failure*; when it executes correctly, the

event is called a *success*. Clearly, only faulty programs can fail.

For the purposes of this discussion, a program is assumed to be subjected to a series of statistically independent demands. "One demand" or "one execution" of the software will mean, e.g., for an emergency shut-down system in an industrial plant, an evolution in the plant state that should initiate a shut-down sequence; for avionics software in an aircraft, a whole mission; for a batch program, a single execution. This is a restricted scenario, but it allows a rigorous description of the issues of interest, and has reasonably general applicability. In practice, the questions of interest can often be framed in terms of sequences of properly defined, independent demands: although software behaviour normally exhibits statistical dependency between failures in the short term (Strigini, 1996), independence between whole missions, or between periods of execution that are far apart in time (Galves and Gaudel, 1998) can usually be assumed.

For a given program, these three measures are defined:

- P_{perf}: the probability of perfection. P_{perf} is the relevant measure in the perfectionist view;
- ϑ: the probability of failure per execution, also called the *failure rate*. ϑ is the measure that is estimated in the statistical approach;
- $P_{surv}(T)$: the probability of surviving (operating without failures) for a stated number T of executions; this will also be referred to as the probability of "mission survival". For instance, if the product is the shut-down system for a hazardous plant, the T executions (the mission), could be all the expected demands on the shut-down system over the lifetime of the plant; if the product is a navigational aid in an aircraft, the T executions could be all its flights.

$P_{surv}(T)$ is a measure of direct interest for a safety assessor. A common requirement is that a safety-critical component (including software) has a low enough probability of failures (affecting safety) over [a certain stretch of] its operational life.

The following discussion concerns the relationship between these three probabilities.

A note is appropriate about the difference between reliability and safety concerns. For the purpose of this article, "safety" can be considered as reliability - the probability of delivering the required service - with respect to a set of requirements which differs from the set considered in reliability assessment. In the case of safety, the requirement is freedom from failures that would be dangerous in the environment where the software is used. Faults, perfection, and all the other terms can be redefined in terms of these failures only, without affecting the arguments developed here. Therefore, the formal treatment in this article uses the term "reliability", irrespective of the class of failures considered.

3. Modelling the event space

The first precaution when discussing probabilities is to define carefully the scenario to which the probabilities refer. This means defining the *experiment* considered, with its *sample space* (i.e., the set of all possible *outcomes*) and the *events* (subsets of the sample space) of interest. To allow $P_{surv}(T)$ to be studied, the "experiment" considered here includes T executions of the program as shown in Figure 1 below. As shown, the very first step is the development of the program, which produces the two mutually exclusive events "the software is perfect" (with probability P_{perf}), and "the software is faulty". The program then runs for T consecutive executions; each execution may result in either a success or a failure (with probability ϑ).

The possible outcomes of this experiment are thus sequences of T+1 elements: {faulty/correct, success/failure at the first execution,..., success/failure at the T-th execution}.

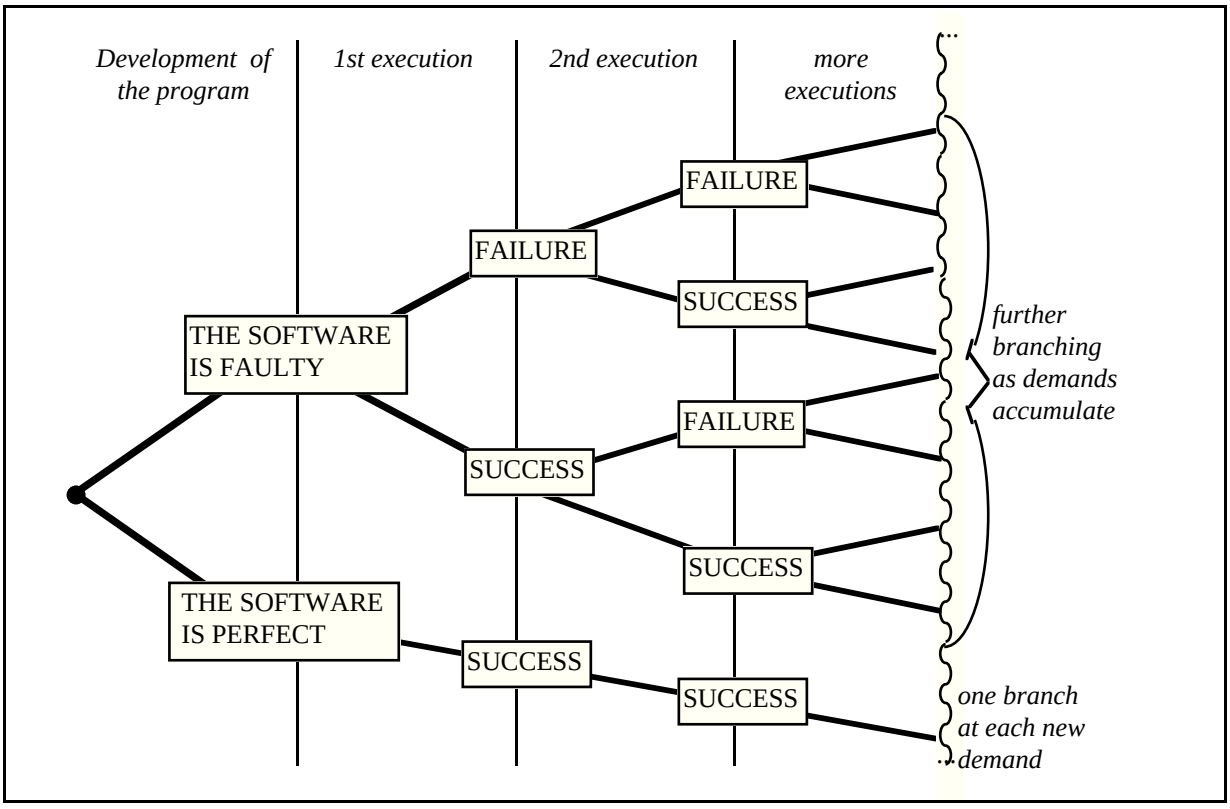

Fig. 1: The sample space for T executions of a program which may be perfect or faulty. A path from the root to a leaf represents an outcome.

The relationship between the three probabilities considered can now be stated formally. Figure 2 shows the relevant events: the outcomes enclosed within the bubble represent the event "survival for T executions", with probability $P_{surv}(T)$. This probability can easily be written as:

(1) $P_{surv}(T) = P_{perf} + P$(program is faulty AND does not fail in T executions),

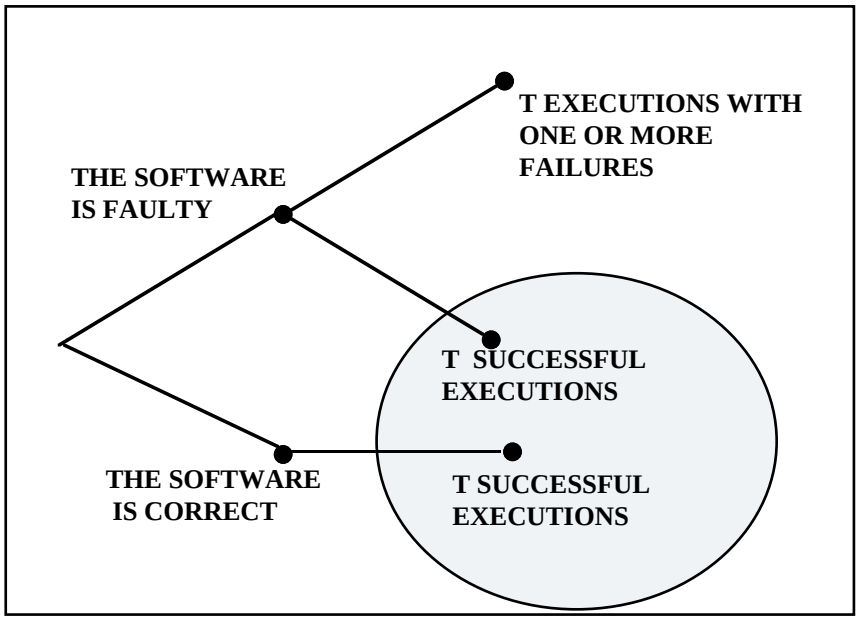

Fig. 2: The event "the program actually produced survives without failures through a 'mission' of T executions".

4. The relationship between the perfectionist and the statistical views

Equation (1) above can also be written (using self-explanatory abbreviations to designate the various events) as:

(1') $P_{surv}(T) = P_{perf} + P(no\_fail\_in\_T \mid faulty)\, P(faulty) =$
$P_{perf} + P(no\_fail\_in\_T \mid faulty)\, (1-P_{perf}) =$
$P_{perf}\, (1- P(no\_fail\_in\_T \mid faulty)) + P(no\_fail\_in\_T \mid faulty)$

All terms in (1') are probabilities, with possible values in the interval [0,1]. Hence the following inequalities follow from (1) and (1'):

(2) $Psurv(T) \geq Pperf$

(3) $Psurv(T) \geq P(no\_fail\_in\_T \mid faulty)$

In the statistical approach, it is generally assumed that $P_{perf}=0$ (presumably a good approximation for most software). So, given a failure rate ϑ^* ($\vartheta^*>0$), (1') becomes:

(4) $P_{surv}(T) = P(no\_fail\_in\_T \mid faulty) = (1-\vartheta^*)^T$

If P_{perf} is believed to be non-zero, but no estimate is available for it, (3) shows that simply assuming $P_{perf}=0$ and using (4) for predictions yields a conservative estimate of P_{surv}, given any value of ϑ^*.

The problem with the expression in (4) is then that as T, the number of demands (or the time period of interest), increases, the probability of surviving without failure tends to 0. This behaviour of the exponential reliability function is shown dramatically in Table I below, for a few hypothetical scenarios of interest in the safety area.

If one only allows statistical evidence about ϑ to be brought as evidence of reliability, predicting a long period of failure-free operation may be infeasible. The deciding factor is the ratio between the duration of this period and the total testing time over which the product has been or can reasonably be observed to support the prediction. When this ratio is high, the estimated bound on ϑ is likely to be too high for supporting the desired conclusion.

On the other hand, if one could estimate P_{perf}, as in the perfectionist approach, one could use this estimate as the lower bound for $P_{surv}(T)$, via inequality (2). Indeed, the probability of failure over an *arbitrary* length of time is less than $1-P_{perf}$. An upper bound on the probability that the program is faulty is thus an upper bound on the probability of any number of failures over any period of time.

Example application	T	ϑ=	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}
Protection system in nuclear power plant, 30 year lifetime, 2 demands/year	60	P_{surv}=	$1-6\times10^{-8}$	$1-6\times10^{-7}$	$1-6\times10^{-6}$	$1-6\times10^{-5}$	$1-6\times10^{-4}$	$1-6\times10^{-3}$	0.94
Flight-critical system in airliner, type lifetime 30 years, 400 aircraft, 300 flights/year	3.6×10^{6}	P_{surv}=	0.9964	0.965	0.698	0.027	~0	~0	~0
Critical system in car, 1 million sold, 10 year lifetime, 300 trips/year	3×10^{9}	P_{surv}=	0.05	~0	~0	~0	~0	~0	~0

Table I. Probability of surviving T executions, as a function of the probability of failure per execution, ϑ.

What is of interest in Table I is the difference between the examples. For a seldom-required protection system, assurance that the probability of it failing in operation is less than 10% can be obtained by demonstrating an upper bound on ϑ on the order of 10^{-3}.For equipment on an aircraft (second example in the table), "catastrophic" equipment failures are required to be "not anticipated to occur over the entire operational life of all airplanes of one type" (FAA, 1985). If by "not anticipated to occur" the licensing authorities meant a probability lower than, for instance, 10%, even demonstrating an upper bound on ϑ of the order of 10^{-7}, if feasible, would not be enough to give the required assurance. Yet, the same assurance would be given by knowing that the probability of the equipment containing defects capable of causing such failures is lower than 10%. If one demanded similar assurance on P_{surv} for the whole career of a car type, arguments based on demonstrating upper bounds on the failure rate would be even more hopeless.

So, if one wants to predict "ultra-high" reliability over many executions (as in the latter two examples), it may be worthwhile to reason in terms of the probability of absence of faults, P_{perf}.

Last, it must be noted that even when using a "perfectionist" argument, statistical evidence about ϑ is not useless. Even trusting that, for instance, P_{perf}=0.99 implies no indication of how likely the software would be to fail *if* it were faulty, i.e., if the software at hand were the unlucky "one in one hundred" case. Even if statistical testing cannot be performed long enough to demonstrate that the software is as good as required, one may well be interested in knowing at least that it is not *too* bad. The merits of this position are illustrated by these examples of systems developed for very high reliability:

- the Space Shuttle software was first delivered with a bug which would cause a synchronisation error upon initialisation with a probability greater than 10^{-2} (Garman, 1981);
- the Ariane 5 guidance system was delivered with a defect with probability 1 of causing loss of mission (Lions, 1996). For systems with a non-negligible risk of being so "perfectly unreliable", realistic testing, as required by the statistical approach, is sure to be useful. A *single* test will either show that this worst-case scenario is not verified, or point to the presence of defects.

5. Can claims of perfection be plausible?

These considerations naturally lead to the issue (also raised by the reviewers of this paper) of whether expecting reasonable claims of "perfection" is realistic at all. There are actually three related questions that need to be briefly addressed:

1. in which cases, if any, could P_{perf} reasonably be expected to be non-negligible?
2. how can one estimate P_{perf}?
3. can estimates of P_{perf} be stated in terms that would be accepted by the pertinent sector of the safety community?

Regarding the first question, most practitioners would accept that most well-known software products, from large telephone switching applications to popular word processors, have a negligible chance of being defect-free, due to complexity or lack of commitment to quality or combinations of both. However, in programs for safety-critical applications at least the following scenarios may occur that make claims of perfection (say, $P_{perf}>90\%$) plausible:

- the program performs a very simple function (e.g., comparing a sensor reading against a threshold and producing a single-bit output), is extremely simple in its implementation (few lines of code with very few branches), and much effort has been spent in checking it;
- a claim of perfection is only made with regard to specific failure behaviours, and a formal proof exists that the program is exempt from that kind of behaviour (e.g., deadlock); a similar case is that in which the claim of perfection is only made concerning safety-relevant failures, the ways the program can affect safety in the system of which it is part are a small and clearly understood set, and rigorous checks or proofs have gone into excluding the possibility of that behaviour.

As for the second question, the feasibility of estimating P_{perf} depends on the availability of relevant evidence. Statistical reasoning would be needed, as a rule. For instance, how many programs has the same company produced that were similar in general characteristics, and how many of these can be considered to have been defect-free? From such statistics one can infer the probabilities on a new, similar program. In most cases, there is too small a population, and too weak a basis for claiming similarity between new and old programs, for supporting claims of a high P_{perf}; yet, the possibility of such a statistical argument cannot be excluded.

More refined reasoning is possible in some cases. If the claim for freedom from a certain class of defects is based on the use of formal proof, one would want statistics about applications of similar formal methods to similar problems, and the relative frequency with which they failed to discover defects. There is no special difficulty with statistical inference of this kind. The only problem is that even in those cases in which it is plausible that P_{perf} is high, appropriate statistics are not being collected.

Other authors (Hamlet, 1987; Voas et al., 1995; Howden and Huang, 1995) have proposed ways of establishing confidence in perfection using the results of testing. Hamlet and Voas have only studied "classical" confidence levels (related to the probability of a product being accepted as perfect *if* faulty), which are inadequate for prediction and acceptance decisions, as they ignore the base probability of programs being faulty in the first place. (Howden and Huang, 1995) instead consider the probability of a program being faulty *and* passing tests as though it were perfect, which is a sound basis for decisions, and study its application in detail, e.g. looking for optimal allocation of effort to various verification strategies. All these methods use assumptions on the probabilities of defects being missed by testing, which must be estimated statistically. Bayesian procedures which yield proper probabilities of perfection as part of complete distributions of Θ are possible and are described for instance in (Bertolino and Strigini, 1996b; Bertolino and Strigini, 1996a).

The likelihood that the safety community will accept claims of perfection is a separate issue. The authors believe that it is desirable for such claims to be considered acceptable in principle, and their supporting evidence to be carefully scrutinised in practice. Some of the existing safety communities routinely use unsound formulations of the safety claims about products and especially software. For instance, formal proof of "correctness" is often equated to certainty of correct behaviour. Another example is given by the civil aviation community, which demands assurance that certain failures are unlikely to happen during the whole lifetime of an aircraft type, but requires no estimate of the probability that software will cause such failures (RTCA/EUROCAE, 1993). In these circumstances, the certification or licensing process may

have merits as a challenge which motivates the developers to improve product quality, but is not a sound decision-making process. Allowing those who present evidence of perfection to label it as such would be a step in the right direction.

If an estimate of a non-negligible probability of perfection is obtained, it is also possible to account for it in inference about failure rates, and in predictions about P_{surv} based on failure rate. To this end, equation 1 is rewritten as:

$$(5)\quad P_{surv}(T) = \int_0^1 P(\text{program does not fail in T executions} \mid \Theta=x)\, f_\Theta(x)\, dx$$

where Θ represents the failure rate, seen as a random variable, $f_\Theta(x)$ is its probability density function, and the case of perfection is represented by $\Theta=0$. A non-zero value of P_{perf} is represented by setting $f_\Theta(x) = P_{perf}\, \delta(x)$, where $\delta(x)$ is Dirac's "delta" function, defined so that $\int_{0-}^{0+} \delta(x)=1$. (Bertolino and Strigini, 1996a) shows the Bayesian inference procedure for inferring a posterior distribution of Θ from successful operational testing. The distribution of Θ thus obtained can then be plugged into (5) for predicting $P_{surv}(T)$ (if there are uncertainties about the assumptions used in the inference, it is possible to calculate the errors that they may generate and verify which assumptions in a given range produce the most pessimistic predictions). Discussing these methods is outside the scope of this paper. The interested reader may also consider, for instance, (Delic et al., 1997), showing how a prediction procedure can exploit statistics about the faults found in the programs produced by a given process; in this procedure, "perfection" is represented by a zero value for a random variable representing the number of residual faults in a program.

6. Conclusions

A rigorous statement has been provided for the probabilistic arguments that would be allowed, in support of deeming a software product acceptable for a certain mission, by knowing either the probability of failure per execution (the statistical position), or the probability that the software is fault-free (the perfectionist position).

The conclusion is that requirements for survival over many demands or long operational periods tend to tilt the balance in favour of "perfectionist" arguments, if available. A moderate trust that a program is fault-free allows stronger belief in the chance of surviving many executions than the belief warranted by even very low estimates of its failure rate. How long the period of operation has to be to make the probability of perfection the crucial consideration depends on the details of each specific context, and typically on the amount of statistical testing that is economically feasible vs the values of P_{perf} that can be reasonably claimed.

The purpose of this note has been to clarify an aspect of software assessment that is often confusing, and to argue that:

- arguments of "perfection" may well have a role in probabilistic assessment of reliability and safety, but
- using them creates an obligation to provide supporting evidence of a scientific, usually statistical nature.

Even "moderate trust" in perfection is very difficult to obtain, except perhaps for extremely simple programs (and "perfection" here means not simply "correctness" according to a formal mathematical specification, but correctness with respect to the actual requirements). Almost no information is commonly available to actually bound the probability of faults in well-developed, well-verified software. For instance, the use of "formal methods" may guarantee that faults *of certain kinds* will be *almost certainly* absent, but this statement is useless without quantification (of the "almost certainty", and of the probability of other kinds of faults). So, even when a "perfectionist" argument fares better than a "statistical" one, it is not usually a basis for any

great degree of confidence in reliable performance. Recognising that even claims of perfection must be probabilistic in nature indicates the way towards better confidence: collecting scientific evidence about the effectiveness of the methods used. Practitioners need information, for instance, on the effectiveness of specific verification methods and of their combinations in eliminating *all* defects of concern, rather than simple lists of "recommended" or "highly recommended" methods.

The formal description given in Section 4 makes it easier to compare the benefits of a "statistical" *vs* a "perfectionist" approach to collecting evidence about software reliability, and thus supports better decision-making. Neither kind of evidence is generally superfluous, however. When formulating a conjecture ("this product is satisfactory") on uncertain grounds, the only scientific way of building trust in the conjecture is to submit it to multiple challenges: looking for faults via techniques that attempt to demonstrate perfection, and looking for excessive failure rates via statistical testing. Evidence of both kinds is naturally produced by any well-managed process of development, verification and validation: putting such evidence to good use for quantitative assessment does not imply much additional cost. The considerations developed here concern this latter step.

Evidence collected following a "statistical" approach and following a "perfectionist" approach can be rigorously combined via probabilistic reasoning, as mentioned in Section 5. Bayesian techniques lend themselves to this use (Littlewood and Wright, 1995; Littlewood and Strigini, 1993; Bertolino and Strigini, 1996c), and there is encouraging research on the use of convenient formalisms like "Bayesian belief networks" for representing complex probabilistic arguments (Strigini and Fenton, 1996; Neil et al., 1996; Delic et al., 1997). These allow one to consider all kinds of evidence - good development methods, program structure, favourable "metrics", statistical testing, etc. - together, taking account of their known interrelationships (if any), in one, complex argument. These techniques are *not* a substitute for factual, statistical knowledge, but they may exploit such knowledge better than a purely "perfectionist" or purely "statistical" argument, as outlined here, would. By applying Bayesian reasoning to the results of testing together with knowledge about the effectiveness of the development and verification methods used, one can derive a distribution for the failure rate of the program or, better, a probability of failure over the mission time of interest. In particular, one can explicitly combine the effects of evidence for perfection and for a low failure rate (Bertolino and Strigini, 1996a), rather than neglecting one of the two.

The considerations developed here apply, with minor changes, to all systems subject to design faults, but for the sake of concreteness this paper has used the terminology of software, which is commonly considered the main example of this kind of problem.

Acknowledgements

This work was partially funded by the European Commission via the "OLOS" research network (Contract No. CHRX-CT94-0577) and the ESPRIT LTR Project 20072 "DeVa". Bev Littlewood, Martin Woodward and the anonymous reviewers provided helpful comments on the previous versions of this paper.

References

Barwise, J. (1989). ‘Mathematical proofs of computer system correctness’. *Notices of the American Mathematical Society*. **36,** 844-851.

Bertolino, A. and Strigini, L. (1996a). ‘Acceptance Criteria for Critical Software Based on Testability Estimates and Test Results’. *SAFECOMP 96, 15th International Conference on Computer Safety, Reliability and Security*. Vienna, Austria. Springer. 83-94.

Bertolino, A. and Strigini, L. (1996b). ‘On the use of testability measures for dependability assessment’. *IEEE Transactions on Software Engineering*. **22,** 2. 97-108.

Bertolino, A. and Strigini, L. (1996c). ‘Predicting Software Reliability from Testing Taking into Account Other Knowledge about a Program’. *Quality Week '96*. San Francisco. Software Research Institute, San Francisco.

Delic, K. A., Mazzanti, F. and Strigini, L. (1997). ‘Formalising Engineering Judgement on

Software Dependability via Belief Networks'. *DCCA-6, Sixth IFIP International Working Conference on Dependable Computing for Critical Applications, "Can We Rely on Computers?"*. Garmisch-Partenkirchen, Germany. IEEE Computer Society Press. 291-305.

FAA (1985). Federal Aviation Administration, Advisory Circular AC 25.1309-1A.

Fetzer, J. H. (1988). 'Program Verification: the Very Idea'. *Communications of the ACM.* **31,** 9. 1048-1063.

Galves, A. and Gaudel, M.-C. (1998). 'Rare Events in Stochastic Dynamical Systems and Failures in Ultra-Reliable Reactive Programs'. *28th International Symposium on Fault-Tolerant Computing, FTCS-98*. Munich, Germany. IEEE Computer Society Press. 324-333.

Garman, J. R. (1981). 'The 'bug' heard round the world'. *ACM SIGSOFT Software Engineering Notes.* **6,** 5. 3-10.

Hamlet, D. (1992). 'Are we testing for true reliability?'. *IEEE Software.* **July,** 21-27.

Hamlet, R. G. (1987). 'Probable correctness theory'. *Information Processing Letters.* **25,** 17-25.

Howden, W. E. and Huang, Y. (1995). 'Software Trustability Analysis'. *ACM Transactions on Software Engineering and Methodology.* **4,** 1. 36-64.

IEC (1995). Draft IEC 1508. Functional Safety: Safety Related Systems. International Electrical Commission, IEC/SC65A/WG 9 and10, Proposed Standard.

Lions, J. L. (1996). Report by the Inquiry Board on the Ariane 5 Flight 501 Failure. ESA/CNES.

Littlewood, B. and Strigini, L. (1993). 'Validation of Ultra-High Dependability for Software-based Systems'. *Communications of the ACM.* **36,** 11. 69-80.

Littlewood, B. and Wright, D. (1995). 'A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability'. *SAFECOMP '95, 14th International Conference on Computer Safety, Reliability and Security*. Belgirate, Italy. Springer. 173-188.

Littlewood, B. and Wright, D. (1997). 'Some conservative stopping rules for the operational testing of safety-critical software'. *IEEE Transactions on Software Engineering.* **23,** 11. 673-683.

Miller, K. W., Morell, L. J., Noonan, R. E., Park, S. K., Nicol, D. M., Murrill, B. W. and Voas, J. M. (1992). 'Estimating the Probability of Failure When Testing Reveals No Failures'. *IEEE Transactions on Software Engineering.* **18,** 1. 33-43.

Neil, M., Littlewood, B. and Fenton, N. (1996). 'Applying Bayesian Belief Networks to Systems Dependability Assessment'. *Safety Critical Systems: The Convergence of High Tech and Human Factors: Proceedings of the 4th Safety-critical Systems Symposium.* Leeds, U.K. Springer. 71-94.

Parnas, D. L., van Schouwen, A. J. and Kwan, S. P. (1990). 'Evaluation of Safety-Critical Software'. *Communications of the ACM.* **33,** 6. 636-648.

RTCA/EUROCAE (1993). Software Considerations in Airborne Systems and Equipment Certification., document RTCA DO-178B/EUROCAE ED-12B.

Strigini, L. (1996). 'On testing process control software for reliability assessment: the effects of correlation between successive failures'. *Software Testing Verification and Reliability.* **6,** 1. 36-48.

Strigini, L. and Fenton, N. (1996). 'Rigorously Assessing Software Reliability and Safety'. *ESA Software Product Assurance Workshop*. Nordvjik, the Netherlands. European Space Agency. 193-198.

Voas, J. M., Michael, C. C. and Miller, K. W. (1995). 'Confidently Assessing a Zero Probability of Software Failure'. *High Integrity Systems.* **1,** 3. 269-275.