Acceptance Criteria for Critical Software Based on Testability Estimates and Test Results

Antonia Bertolino
Istituto di Elaborazione della Informazione, CNR
Pisa, Italy

Lorenzo Strigini
Centre for Software Reliability, City University
London, U.K.

Abstract

Testability is defined as the probability that a program will fail a test, conditional on the program containing some fault. In this paper, we show that statements about the testability of a program can be more simply described in terms of assumptions on the probability distribution of the failure intensity of the program. We can thus state general acceptance conditions in clear mathematical terms using Bayesian inference. We develop two scenarios, one for software for which the reliability requirements are that the software must be completely fault-free, and another for requirements stated as an upper bound on the acceptable failure probability.

1. Introduction

The only direct method for predicting the operational reliability of a program is inference from "statistical" testing under an operational input profile [1, 2]. For safety-critical software, acceptance requires a long testing campaign with no failures. However, an amount of operational testing sufficient to warrant a high confidence that the software is as reliable as required in some current applications is infeasible [3, 4, 5].

A way forward is to combine the evidence from testing with any other evidence available. This combination can be made rigorous through *Bayesian* methods, in which the assessor can update the *prior* probability of an event, on the basis of new observed data, to produce a *posterior* probability, representing how the strength of belief allowable in the event taking place varies with new evidence. In particular, the use of prior probabilities in Bayesian reasoning explicitly describes the fact that predictions on the basis of statistical inference must also depend on pre-existing information about the events in question.

When judging on the basis of the results of testing, a kind of clearly helpful information is how effective the testing is at discovering faults. One would think that a series of successes in highly effective tests would give the same confidence as a longer series with less effective tests. A measure of test effectiveness that has gained some popularity is *testability*, the probability of a test detecting a failure conditional on the program being faulty, introduced by Voas and co-authors [6, 7, 8, 9, 10] and proposed as a basis for assessing software. The underlying intuition is that a statement about the internal structure of a program (to the effect that any bugs are likely to produce a high failure rate) allows one to draw stronger conclusions from testing than allowed by black-box considerations alone. In [11], we gave a rigorous, Bayesian inference procedure for obtaining the probability that a program is correct, knowing its testability, the test results, and the prior probability of it being correct. However, in that paper we used a *point* estimate of program

 Published in Lecture Notes in Computer Science (http://www.springer.de/series/lncs/)

15th International Conference on Computer Safety, Reliability and Security SAFECOMP '96 - Vienna, Austria

testability. This amounts to assuming that, if a program does contain faults, it is bound to have a certain, known probability of failure per execution (failure intensity), which is clearly a simplifying, but unrealistic assumption. In reality, we will instead have at most an understanding of which values of the failure intensity are more or less likely. In this paper, we offer two improvements:

1) we describe testability in terms of the prior distribution of the failure intensity of a program. This yields a prediction method which is more applicable in realistic situations, and eliminates the need to reason with a rather abstruse concept like the probability that a program would fail, if it were possible for it to fail;
2) we show how to use this prediction method when the criterion for accepting a program is either the probability that the program is correct (completely fault-free), or the probability that the program has an acceptable failure intensity in operation.

We consider a scenario in which software undergoes a long series of independent test cases, without failure. This is the typical case of interest for safety-critical software, but the mathematics can easily be extended to the case of any number of observed failures. For reasons of space, we only consider a few examples of simple prior distributions, to illustrate some essential facts about reasoning with testability. The other assumptions are that testing takes place with the operational input distribution, and all and only the actual failures of the program under test are detected (*perfect oracle* assumption).

In Section 2, we introduce the notion of failure intensity as a random variable and its probability distribution. Section 3 deals with the representation of assumptions about testability in terms of this distribution. Sections 4 and 5 describe the use of Bayesian inference in judgement about accepting software, according to the acceptance criteria 1) and 2), respectively, and illustrate the method with numerical examples. Section 6 summarises our results and their possible developments and discusses their practical uses.

2. Distributions of the Failure Intensity

In the assessment of software reliability, uncertainty derives from two sources: we do not know which inputs, if any, will cause the software to fail; and we do not know when and whether such inputs will be presented to the software in operation. It is reasonable to describe this uncertainty by stating that, under a given input profile, a program has a certain *probability* of failure when executed once, called a *failure intensity* (often called a "failure rate"). The failure intensity of a program is uncertain because of our limited knowledge about the program: we thus consider it as a random variable, Θ, with a certain *probability distribution*. A way of picturing this is to think about the program to be assessed as having been "extracted at random" from the population of all the programs that *could* have been produced for the same purpose and under the same known conditions: they have different values of Θ, and the distribution describes the frequencies with which different values of Θ appear in this population. Or, the distribution of Θ can be thought of as representing the degrees of one's beliefs ("subjective probabilities", based on whatever evidence is available) that different possible values are the *actual* value of Θ for *this* program. The latter is the Bayesian interpretation. With Bayesian inference, we represent what we expect about the program before testing it via a prior distribution: we can for instance take into account the reliability levels achieved in past products of the same development process. By applying Bayes' rule, we then obtain a (*posterior*) distribution for Θ which also takes account of test results.

Choosing a prior distribution for Θ is a difficult task. It may be appealing to look for a prior distribution that represents "ignorance" about the failure intensity. However, absolute ignorance cannot be uniquely defined. Any representation of "ignorance" embodies a statement about which events are deemed to be equally likely. Many authors (e.g. [3, 4]), represent "ignorance" via the (mathematically convenient) *uniform* prior:

$$P(\Theta = \vartheta) \equiv 1 , \quad \text{for all } \vartheta \in [0,1]$$

This means that the true value of Θ is as likely to lie within the interval [0.1, 0.2] as is within [0.2, 0.3], [0.3, 0.4], etc. But one can imagine different forms of ignorance, e.g., one might believe that the failure intensity is as likely to fall in [0.1, 1] as in [0.01, 0.1], as in [0.001, 0.01], etc., and this belief would result in a totally different distribution than the uniform prior. This latter way of dividing the event space seems closer to the usual ways of reasoning about software. Our examples will use a variant of this second form of "ignorance", fitting the assumptions of [6, 8, 9]: Θ may only take values in an interval $[\vartheta_1, \vartheta_N]$, $0<\vartheta_1 < \vartheta_N = 1$, *plus* the isolated value $\vartheta_0=0$ (correct software), and the logarithm of Θ has a uniform distribution over the interval $[\log(\vartheta_1), \log(\vartheta_N)]$.

To avoid mathematical complexity and simplify our explanations, we use discrete approximations to our distributions and numerical computations. We thus represent Θ as a discrete random variable, and describe its distribution via a succession

$$a_i = P(\Theta = \vartheta_i), \quad i=0, 1, ..., N;$$

in particular we define $\vartheta_0 = 0$ and $a_0 = P(\Theta = \vartheta_0 = 0)$.

After observing T successful tests, we obtain, by applying Bayes' theorem, a discrete *posterior* distribution $b_0, b_1, ..., b_N$ with:

$$(1) \qquad b_i(T) = \frac{a_i(1-\vartheta_i)^T}{\sum_{i=0}^{N} a_i(1-\vartheta_i)^T}$$

where $(1-\vartheta_i)^T$, the probability that no failures happen in T tests *if* $\Theta=\vartheta_i$, is usually called a *likelihood function*.

3. Representing Testability in terms of the Prior Distribution of the Failure Intensity

We refer to the concept of testability introduced by Voas and co-authors, but use our, more precise definition [11]: the testability of a program is the conditional probability that the program fails a test, on an input randomly drawn from a given input profile, *given* a specified oracle and *given that* the program is faulty.

Testability is a rather un-intuitive concept: if the program contained no faults, then it could not fail. Testability describes how likely it would be to fail, *if* it did contain faults, and is then used to decide how likely the program is *not* to contain faults.

If we consider Θ as a random variable, to represent our uncertainty about its actual value, it becomes natural to see testability as a random variable as well. If we consider a point estimate, τ, for this random variable, we can study the relationship between τ and the distribution of Θ. By definition:

$$(2) \qquad \tau = P(\text{failure of a test} \mid \text{the program is faulty}),$$

that is

$$(3)\ \tau = \frac{\mathrm{P}(\text{failure of a test AND the program is faulty})}{\mathrm{P}(\text{the program is faulty})} = \frac{\mathrm{P}(\text{failure of a test})}{\mathrm{P}(\text{the program is faulty})}$$

where the last equality is justified since only faulty programs can fail a test, so that the event "failure of a test" is contained in the event "the program is faulty". In our representation of the prior probability distribution of Θ,

$$(4)\qquad \mathrm{P}(\text{the program is faulty}) = 1 - a_0 = \sum_{i=1}^{N} a_i$$

and:

$$(5)\qquad \mathrm{P}(\text{failure of a test}) = \mathrm{E}(\Theta) = \sum_{i=0}^{N} a_i\ \vartheta_i = \sum_{i=1}^{N} a_i\ \vartheta_i$$

where $\mathrm{E}(\Theta)$ represents the expected value, or mean, of the random variable Θ. So,

$$(6)\qquad \tau = \frac{\sum_{i=1}^{N} a_i\ \vartheta_i}{\sum_{i=1}^{N} a_i} = \frac{\sum_{i=1}^{N} a_i\ \vartheta_i}{1 - a_0}$$

Voas and co-authors assumed that a lower bound h can be estimated for the testability of a program, i.e., *if* the program is faulty, then its probability of failing a test is at least h. Our chosen prior distribution satisfies this assumption. By increasing ϑ_1, the lower bound on the values of Θ that have non-zero probability, we can thus study the effect of assuming increasing values for testability (i.e., of the failure intensity of faulty programs).

Notice, however, that events with a zero prior probability also have a zero posterior probability. So, stating that a lower bound exists is a very strong statement about the program, as it cannot be changed by any amount of new evidence. By comparison, we observe that the point estimate of testability, τ, will in general change as the probability distribution for Θ changes with the number of successful tests.

The adopted test method affects testability. For instance, test inputs could be taken from an input distribution different from the operational profile. The coverage of the test oracle could be smaller, (or, conceivably, greater) than 1 (in practical terms, failures during testing could go undetected, or vice versa the oracle could, by monitoring internal program variables, detect erroneous behaviour even when this is not propagated to program outputs). In these cases, the probability of a test failure for those programs which are faulty, i.e., testability, will differ from their operational failure intensity. These scenarios can be modelled by a probability distribution for testability, conditional on Θ. For reasons of space, we will not do so in this paper. Simple examples are shown in [12].

4. Acceptance Based on the Probability of Correctness

The reasoning in [8, 9] and similar papers uses the evidence of successful tests to increase confidence that the software under test is defect-free. In Bayesian terms, this means having a prior probability $a_0 = \mathrm{P}(\Theta=0) \neq 0$ (if it were $a_0 = 0$, no amount of successful test could produce a posterior $b_0 \neq 0$), and observing how the corresponding posterior probability increases with the number T of successful tests. From equation (1):

$$(7) \qquad b_0(T) = \frac{a_0}{\sum_{i=0}^{N} a_i (1-\vartheta_i)^T}$$

A non-negligible prior probability (i.e., belief held before observing any testing) that a program is defect-free is implausible in many fields of application of software. However, it may be plausible for simple software developed under very stringent quality criteria, which includes some safety-critical software.

Estimating the probability of the software being correct, rather than its probability of failure or similar reliability measures, has some important advantages. Firstly, it does not depend on testing under the operational profile, which is difficult to derive, but can use any test profile chosen for its effectiveness in revealing faults. This is important because testing in many organisations is organised on the assumption that other test selection criteria are more efficient than operational profiles for revealing faults (the controversy cannot be settled for lack of conclusive evidence; it is also clear that different answers might be true for different organisations and specific situations). Secondly (as we discuss in detail in [13]), a probability of correctness is a lower bound on the probability of correct behaviour over any arbitrary length of time, while an estimated failure intensity or rate implies progressively less favourable predictions with longer times of operation.

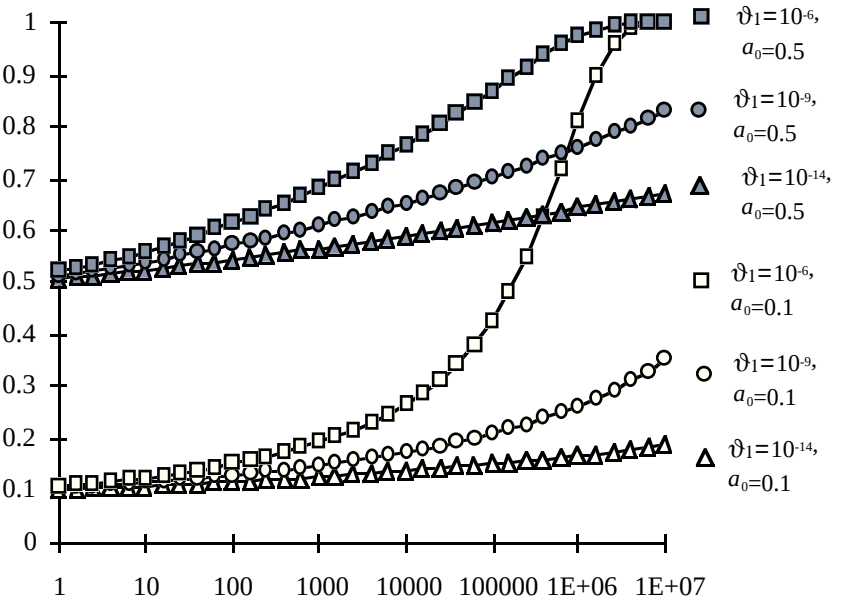

Figure 1 Probability that the program under test is correct, $b_0(T)$, as a function of the number of successful tests, T.

We show in Fig. 1 how the probability of correctness, i.e., the chance that this program is actually one of the perfect ones in our notional population, grows rapidly as we observe successful tests. We show two groups of curves, respectively with a prior probability of correctness equal to 0.1 and 0.5, and within each group we assume different values for ϑ_1, the lower bound on the failure intensity. As

expected, the higher the lower bound ϑ_1, the higher the estimated probability of correctness. We can also note that for small numbers of tests the prior probability of correctness greatly affects the posterior; however, as successful tests accumulate, its influence becomes less important: for instance, after 10^6 successful tests, a prior distribution with a_0=0.1 and ϑ_1=10^{-6} would give a more favourable prediction than a prior starting with a_0=0.5 and ϑ_1=10^{-14}.

In the following figure we also show the posterior probability that the program is correct as a function of ϑ_1, under the assumption that, when we vary ϑ_1, a_0 does not change, and the distribution of Θ when $\Theta \neq 0$ (i.e., for faulty programs) remains uniform between ϑ_1 and 1. We remind the reader that we are assuming that a test input causes the tester to adjudge a test failure if and only if the same input would cause the program to fail in operation. We are thus modelling the effects of the different failure intensities in different programs, not the effects of different degrees of instrumentation of programs to facilitate error detection in testing.

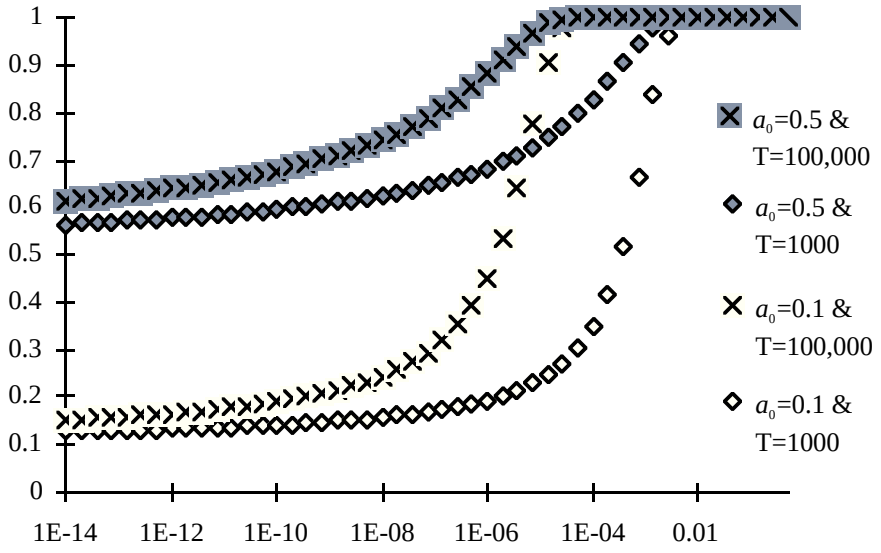

Figure 2 Probability that the program under test is correct, $b_0(T)$, as a function of the lower bound on the failure intensity, ϑ_1, after T successful tests.

5. Acceptance Based on the Probability of Not Exceeding a Given Failure Intensity

Fig. 2 confirms that $b_0(T)$ (for a fixed T) increases with ϑ_1, implying that, out of two programs that pass the same number of tests, the program having a higher testability is more likely to be correct. But this is only one side of the coin. The probability of correctness alone gives us no indication of how likely a program would be to fail, *if* it were faulty. In other words, we may have estimated that our program is very likely to be correct, but if the program happens to be faulty, what risk are we accepting that it is *too* unreliable? In this section, we discuss another acceptance criterion, which seems to apply to most practical situations: a reliability

requirement is stated in terms of an allowed upper bound, ϑ_R, on the failure intensity in operation, i.e., it is required that $\Theta \leq \vartheta_R$. So, an appropriate measure on which an acceptance criterion can be based is *the probability that the program satisfies this reliability requirement.* This cannot be 1, since perfect prediction is impossible, but one would reasonably require it to be close to 1 (more or less close depending on the cost of operating unsatisfactory software). Once we have inferred the posterior probability distribution for Θ, the probability of "success", i.e., of the program satisfying its reliability requirement is:

$$(8) \qquad P_{succ}(T) = \sum_{i=0}^{R} b_i(T) = \frac{\sum_{i=0}^{R} a_i (1-\vartheta_i)^T}{\sum_{i=0}^{N} a_i (1-\vartheta_i)^T}$$

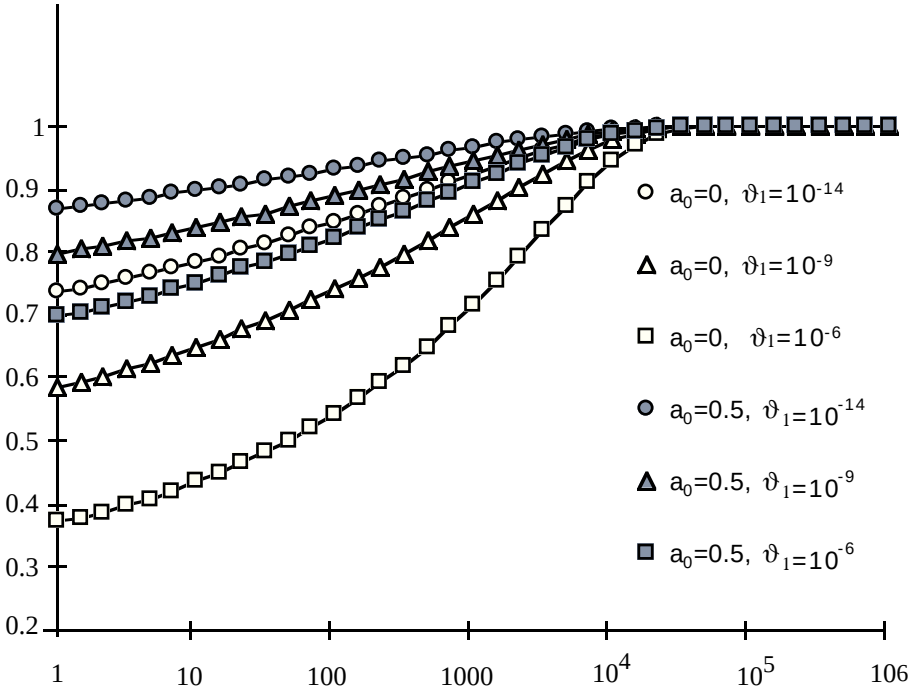

Figure 3. Probability that the failure intensity Θ of the program under test is lower than $\vartheta_R = 10^{-4}$, $P_{succ}(T)$, as a function of the number of successful tests, T.

Fig. 3 shows the function $P_{succ}(T)$. Notice that P_{succ} increases with a_0 but decreases with increasing values of the lower bound on Θ, ϑ_1. This is caused by the assumption that the logarithm of Θ is uniformly distributed over $[\log(\vartheta_1), \log(\vartheta_N)]$. Increasing ϑ_1, i.e., assigning a zero probability to a wider range of values of Θ starting from 0, while keeping a_0 constant, means that the remaining possible values have increased probabilities (i.e., the values of the probabilities a_i for $i \geq 1$ are increased). So, increasing ϑ_1, decreases the reliability of the programs and changes the scenario so that the evidence before testing (in particular, the prior probability of "success", $P_{succ}(0)$) is less favourable to the program under test.

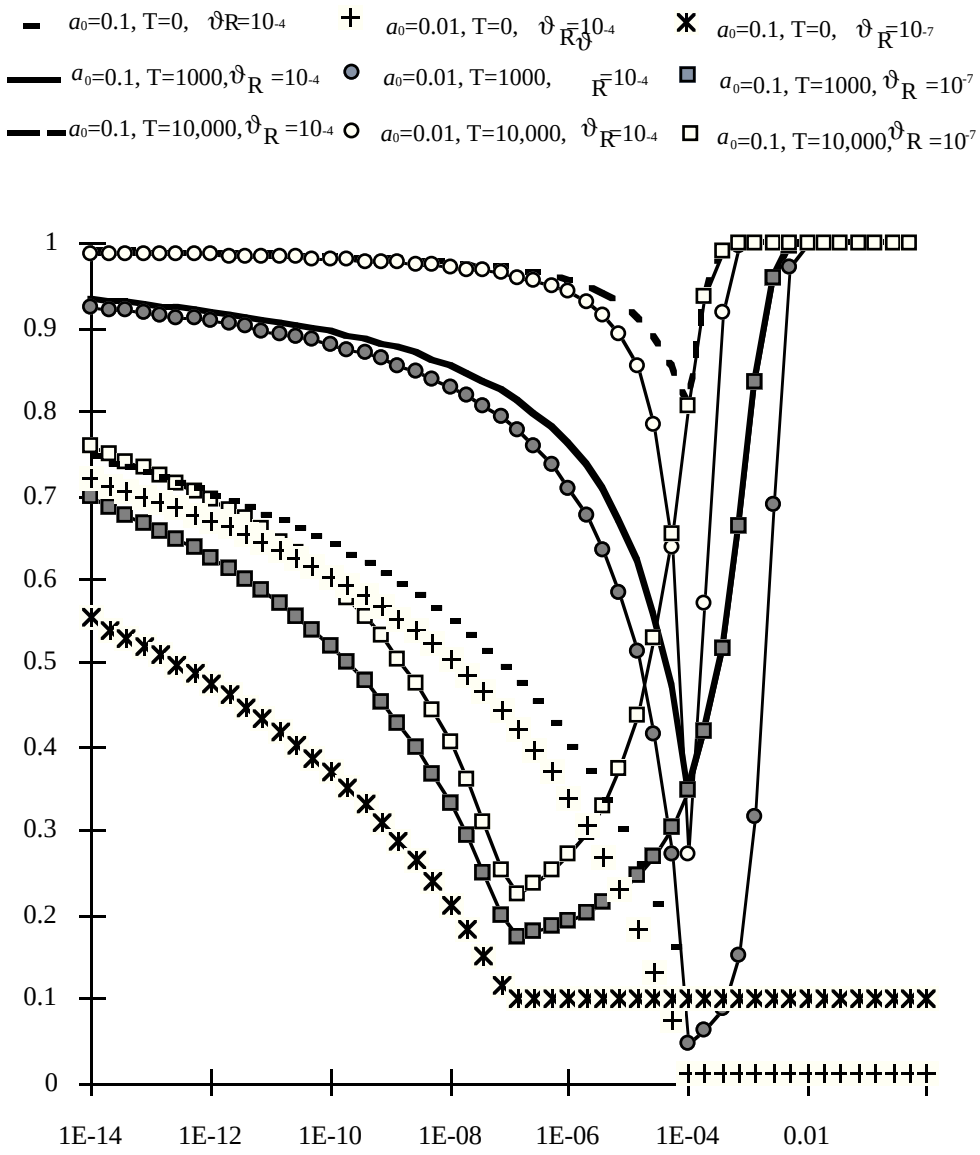

Figure 4 Probability $P_{succ}(T)$ that the failure intensity Θ of the program under test is lower than ϑ_R, as a function of the lower bound on the failure intensity, ϑ_1. The probability of correctness, a_0, is kept constant, and the logarithm of Θ is uniformly distributed between $\log(\vartheta_1)$ and 0. Notice that the curves to the right of the point $\vartheta_1=\vartheta_R$ are just the curves of $b_0=P(\Theta=0)$, since $\vartheta_1>\vartheta_R$ means that all acceptable, non-zero failure intensities have zero probability.

The next two figures offer more insight into the effects of varying ϑ_1. They show $P_{succ}(T)$ as a function of ϑ_1 (as fig. 2 did for $b_0(T)$). However, in Fig 4 the prior distribution of Θ varies with ϑ_1 as we just explained. Fig. 5 studies a completely

different scenario: as we increase ϑ_1, we do not change the probability density function for $\Theta > \vartheta_1$, but we increase a_0, the probability that the software is correct, so as to preserve the property $\sum_{i=0}^{N} a_i = 1$. So, Fig. 4 and Fig. 5 represent the reasoning of two assessors with different information. Both know with certainty that there is a lower bound ϑ_1 on the failure intensity of the program, if faulty. However, in Fig. 4 the assessor is considering software for which (s)he has a clear idea of the probability of it being correct, which does not change with different assessments of ϑ_1. The assessor in Fig. 5, on the contrary, knows the distribution of Θ for $\Theta > \vartheta_1$, which does not vary with different assessed values of ϑ_1, and therefore has to change his assessment of a_0 as (s)he varies ϑ_1. In Fig. 4, increasing ϑ_1 causes P_{succ} to plummet, until ϑ_1 exceeds ϑ_R, that is, until one believes that any bug in the software will cause it to be too unreliable. In Fig. 5, increasing ϑ_1 causes P_{succ} to increase slightly ($P_{succ}(0)$ remains constant as ϑ_1 varies), until ϑ_1 reaches the threshold ϑ_R. These are extreme scenarios, of course. In general, we can expect that different situations change all aspects of the distribution of Θ, rather than leaving just some aspects unchanged. However, these figures show how the whole distribution of Θ is relevant, rather than just ϑ_1 or *h*.

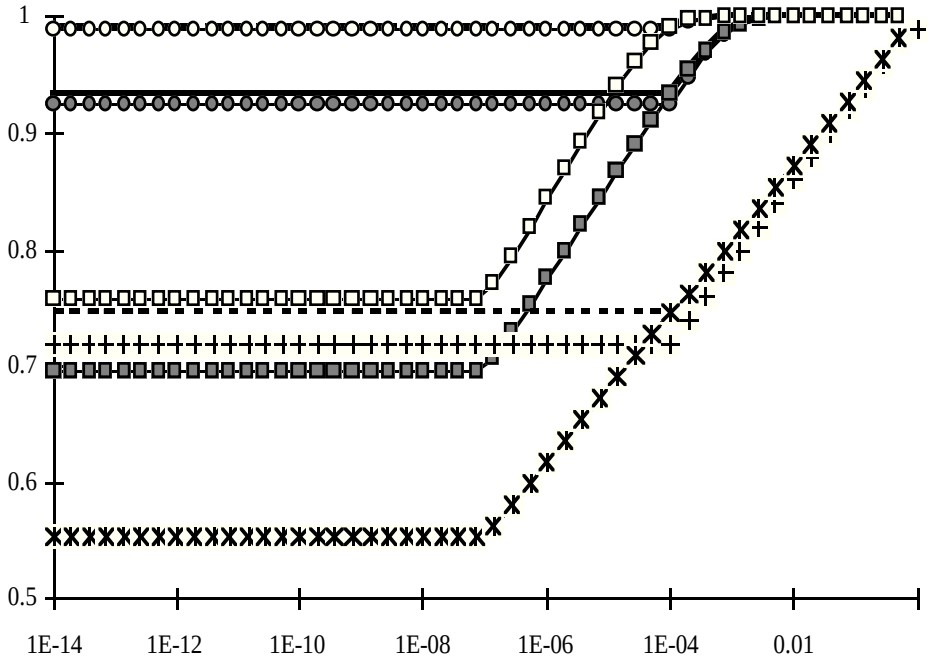

Figure 5 Probability $P_{succ}(T)$ that the program's failure intensity Θ is lower than ϑ_R, as a function of the lower bound on the failure intensity, ϑ_1. The curves are identified by the same symbols as in Fig. 4. However, for each curve, the probability that the software is correct, a_0, starts, for $\vartheta_1 = 10^{-14}$, at the value shown in the legend of Fig. 4, and then increases as ϑ_1 increases, leaving unchanged the probabilities of the values of Θ greater than ϑ_1.

6. Conclusions

In [11] we reached these conclusions:

- the only ways of increasing testability that are unconditionally beneficial are those that improve one's ability to *detect* undesired behaviour of a program, *without* increasing the likely failure intensity if the program happens to be faulty;
- if program A has higher testability than program B, obtained via a structure that makes faults, if present, more likely to cause failures, and programs A and B pass the same number of tests, this does *not* indicate that program A is a better program than B.

We have now also given precise quantitative expressions of what can be inferred from evidence on testability. In comparison with previous work, we have shown that testability is indeed an interesting measure for an assessor, but is only one aspect of our possible knowledge on a program, and using it without considering other aspects of the prior distribution of Θ may be misleading.

We have instead shown that knowledge about program testability can be integrated in reasoning based on the prior distribution of Θ. In particular, we have shown (equations (7) and (8)) how to compute two specific measures corresponding to reasonable acceptance criteria - the probability that the software is correct, and the probability that it has an acceptable failure intensity. We have shown numerical examples for both cases, with particular prior distributions, and discussed some of the effects of assumptions about testability, i.e., of a lower bound on the non-zero values of failure intensity.

We think that testability-based arguments cannot now be used to evaluate critical software, because there is no trustworthy way of measuring the testability of the programs concerned. Our reasoning in this paper does not explicitly depend on an estimate of testability. Still, in many cases one would be hard pressed to assign a prior distribution of Θ which is perceived as representing a soundly based belief. So, the main advantage of our new representation of testability-based arguments is that it is natural: it uses primitive concepts (the probability that a program has a certain failure intensity) which are easier to grasp than the concept of testability, and subsumes testability-based arguments in a general, standard form of arguments using test results [3, 4, 12].

If estimating the important parameters is so difficult, what is the use of the way of reasoning we described? In the fact that it is a sound decision method, and can thus be used as a check on the informal, haphazard way in which judgement is often passed in software acceptance, *especially* for highly reliable and safety-critical software. When the required failure intensity is orders of magnitude lower than can be proven directly by statistical testing only, much effort is spent in documenting other forms of evidence, like the formal satisfaction of requirements on the form of documentation, the thoroughness of debug testing, and such. All this evidence is then used to reach a decision in an informal, un-auditable, intuitive way, subject to all the common fallacies of intuitive judgement [14, 15]. The method we have described allows checks of the form "After T tests, I wish to be 99 % sure that the software is satisfactory. What kind of prior distribution of Θ would warrant that conclusion? Is this distribution at all plausible, in view of the existing evidence?". Likewise, one can describe a distribution that appears plausible, and then check whether the desired conclusions are sensitive to minor changes in that distribution: acceptance based on such grounds would appear unsound.

In addition, this kind of reasoning may help with design and project management decisions (e.g., regarding program structure or testing regimes, respectively), because it may show how alternative options, by affecting the knowledge available about the program, would help or hinder the final assessment and acceptance of the program.

Further development of this work are under way in several directions:

- acceptance criteria based on the probability of failure per execution ("failure intensity") can easily be substituted by criteria based on the probability of failure over the operational life of the software, or other periods of operation, as in [16];
- different scenarios can be studied as to the relationship between the probability of failure in operation and the probability of test failure, to take into account factors like imperfect oracle coverage, stress testing, etc. For initial examples of these extensions, the reader is referred to [12].

Acknowledgements

The authors wish to thank Peter Bishop, whose criticism of the use of point estimates for testability prompted them to start this work. This research was funded in part by the European Commission via the "OLOS" research network (Contract CHRX-CT94-0577) and the ESPRIT Long Term Research Project 20072 "DeVa".

References

[1] Musa JD. Operational profiles in software-reliability engineering. IEEE Software 1993; March: 14-32.

[2] Parnas DL, van Schouwen AJ, Kwan SP. Evaluation of safety-critical software. Communications of the ACM 1990; 33: 636-648.

[3] Miller KW, Morell LJ, Noonan RE, et al. Estimating the probability of failure when testing reveals no failures. IEEE Transactions on Software Engineering 1992; 18: 33-43.

[4] Littlewood B, Strigini L. Validation of ultra-high dependability for software-based systems. Communications of the ACM 1993; 36: 69-80.

[5] Butler RW, Finelli GB. The infeasibility of experimental quantification of life-critical software reliability. In Proc. ACM Conference on Software for Critical Systems, in ACM SIGSOFT Software Eng. Notes, Vol. 16 (5). New Orleans, Louisiana, 1991, pp 66-76.

[6] Hamlet D, Voas J. Faults on its sleeve: amplifying software reliability testing. In Proc. 1993 Int. Symposium on Software Testing and Analysis (ISSTA), in ACM SIGSOFT Software Eng. Notes, Vol. 18 (3). Cambridge, Massachusetts, U.S.A., 1993, pp 89-98.

[7] Voas JM, Miller KW. Improving the software development process using testability research. In Proc. of the Third Int. Symposium on Software Reliability Engineering. 1992, pp 114-121.

[8] Voas JM, Michael CC, Miller KW. Confidently assessing a zero probability of software failure. In Proc. SAFECOMP '93 12th International Conference on Computer Safety, Reliability and Security. Poznan-Kiekrz, Poland, 1993, pp 197-206.

[9] Voas JM, Michael CC, Miller KW. Confidently assessing a zero probability of software failure. High Integrity Systems 1995; 1: 269-275.

[10] Voas JM, Miller KW. Software testability: The new verification. IEEE Software 1995; May: 17-28.

[11] Bertolino A, Strigini L. On the use of testability measures for dependability assessment. IEEE Transactions on Software Engineering 1996; 22: 97-108.

[12] Bertolino A, Strigini L. Predicting software reliability from testing taking into account other knowledge about a program. In Proc. Quality Week '96. San Francisco, 1996.

[13] Bertolino A, Strigini L. Is it more convenient to assess a probability of failure or of correctness? Submitted for publication 1996.

[14] Kahnemann D, Slovic P, Tversky A (ed). Judgment under uncertainty: heuristics and biases. Cambridge University Press, 1982.

[15] Strigini L. Engineering judgement in reliability and safety and its limits: what can we learn from research in psychology? SHIP project Technical Report T/030, July, 1994.

[16] Littlewood B, Wright D. On a stopping rule for the operational testing of safety critical software. In Proc. FTCS25 (25th Annual International Symposium on Fault -Tolerant Computing). Pasadena, 1995, pp 444-451.