Predicting Software Reliability from Testing Taking into Account Other Knowledge about a Program

Antonia Bertolino
Istituto di Elaborazione della Informazione del CNR, Pisa, Italy
E-mail: bertolino@iei.pi.cnr.it

Lorenzo Strigini
Centre for Software Reliability, City University, London, U.K.
E-mail: strigini@csr.city.ac.uk

Abstract

Inference from statistical testing is the only sound method available for estimating software reliability. However, if one ignores evidence other than testing (e.g., evidence from the track record of a developer, or from the quality of the development process), the results are going to be so conservative that they are often felt to be useless for decision-making. *Bayesian inference* is the main mathematical tool for taking into account such knowledge. Evidence from sources other than testing is modelled as *prior* probabilities (for values of the failure rate of the program) and is updated on the basis of test results to produce *posterior* probabilities. We explain these methods and demonstrate their use on simple examples. The measure of interest is the probability that a program satisfies a given reliability requirement, given that it has passed a certain number of tests. The procedures of Bayesian inference explicitly show the weights of prior assumptions *vs.* test results in determining this probability. We also demonstrate how one can model different assumptions about the fault-revealing efficacy of testing. We believe that these methods are a powerful aid for improving the quality of decision-making in matters related to software reliability.

1. Introduction

With the ever-increasing reliance on computers, reliability requirements on their behaviour become more stringent. For software, this requires not only improving the development process so as to achieve higher levels of reliability, but also being able to demonstrate, before delivery, that the required reliability has in fact been achieved.

The obvious language for discussing software reliability is that of probabilities: although software failures are deterministic, in the sense that they are not caused by random physical failures, there is uncertainty about whether faults (bugs, defects) are present and when they will cause failures. So, we are interested in predictions in a quantitative form, expressed as the probability of observing, or not observing, failures over given periods of future operation. The only well-developed *and* trustworthy approach to obtaining a quantitative prediction of software reliability is via statistical testing. The software is executed in a test environment reproducing (as closely as possible) operational usage [1, 2, 3], and then from the observed failures (or lack thereof) one can estimate a probability of the software failing over a stated period of operation.

The purpose of this paper is to demonstrate the use of *Bayesian* probabilistic techniques for predicting reliability measures from the results observed in testing. In the Bayesian interpretation, a probability is seen as describing the strength of the belief which an observer

Presented at Quality Week'96, San Francisco, May 1996

can justifiably hold that a certain event will take place (*subjective* probability). The subject, upon observing the outcome of an "experiment" (i.e., collecting new data), updates the belief held before the experiment ("prior probability"), producing a "posterior probability". In our case, the experiment consists in testing the software, and the prior probabilities (prior belief, prior knowledge) must describe what expectations one may have about the reliability of the software *before* testing it. The need to assume prior beliefs is the main reason why Bayesian analysis is often opposed in favour of the alternative "classical" approach to statistical inference. The "classical" methods produce "confidence levels" on stated hypotheses, i.e., statements like "I have confidence C that the software has failure rate lower than q". However, we argue that in our case of interest, i.e., the assessment of software reliability, this may be misleading. Considering the knowledge one can have about a program before testing is not only useful, but also necessary. This may be demonstrated by a hypothetical case: let us imagine that we are assessing the reliability of two programs, one written by notorious incompetents and one by the best company in the market. After any (even very small) number of successful tests, the confidence levels for a given hypothesis on the reliability of the two programs would be equal: they would certainly not measure one's rationally based trust in the two programs.

We certainly do not claim that Bayesian methods can solve all problems in software reliability assessment. Many such problems are due to the inadequacy of the available factual knowledge to support a desired conclusion. For instance, observing a short period of successful operation does not allow one to predict high reliability over a much longer period [4]; the apparent soundness of a software engineering method does not allow us to conclude that it will deliver high reliability, unless we actually measure a consistent, high reliability in its products [5]. What we do expect from Bayesian methods is an aid in obtaining sensible conclusions from available evidence (test results), and in testing how these conclusions depend on additional assumptions (represented in the prior probabilities) on the characteristics of a software product. They are an aid for decision makers in spelling out what they really know and believe about a piece of software, and making sure that their decisions do not depend on intuitive assumptions that remain implicit and thus not subjected to any conscious analysis.

Bayesian methods are well established in many fields, including reliability assessment, and have been applied specifically to software reliability assessment [6, 7, 8, 9, 10]. However, their practical use is hindered by computational difficulties, and thus, for instance, the selection of prior probability distributions is often restricted by concerns of analytical tractability. Our goal here is to show how Bayesian inference can be used to represent different scenarios of practical interest, and to derive the relevant probabilities for decision-making. We avoid the problem of mathematical complexity by using numerical, rather than analytical, computations, and concentrate on presenting alternative hypotheses on the software to be assessed and deriving their consequences

Our examples in this paper concern an assessment scenario which is typical of safety-critical software: acceptance testing starts after the last change to the program and reveals no failure. Otherwise, the software would have to be fixed and the testing session would be restarted from scratch. This scenario lends itself especially well to our illustrative purposes; however, applying the same methods to cases in which testing does reveal failures is straightforward.

In the next Section, we describe the standard application of Bayesian inference [7] to the results of statistical testing. Sections 3 and 4 show how different measures can be derived and discuss their interest for a decision maker. Section 5 illustrates the implications of different

assumptions about a program under test, represented by different prior distributions of its failure rate. Section 6 briefly shows how the basic model presented in section 2 can be extended to represent other situations of practical interest. Section 7 summarises the limits and advantages of this approach, and the possible extensions to our work.

2. Application of Bayesian inference to statistical testing

Under a given input profile (that is, given probabilities for all the possible input values), a program has a certain probability of failure per execution, or, as it is usually called, although somewhat improperly, a given *failure rate* . We can then describe our uncertainty about the program's true failure rate by considering it as a random variable, Θ. The next step is postulating a probability distribution for Θ. Θ is a continuous random variable over the interval [0,1], and in principle its probabilities of falling in any given sub-interval can be represented by any probability density function that is defined as non-null only in the [0,1] interval. A common problem with Bayesian methods is that they require very complex calculations, unless the distribution functions used are severely constrained to make them analytically tractable. In this paper, we take the opposite approach: we perform all the calculations via numerical approximations, so as to be free to consider any shape of the probability density function, if it seems to represent an interesting real-world situation. We will then represent these distributions in an approximate form for the purpose of computation. In particular, for the sake of simplicity, we will often approximate Θ as a discrete random variable, which may take a finite sequence of discrete values, ϑ_0, ϑ_1,...ϑ_N. These values are the possible failure rates of the program. By convention, we define $\vartheta_0 = 0$.

Intuitively, a program only has one failure rate (under the given input profile). By assigning probabilities to the different plausible values of Θ, we describe our uncertainty about it: although we do not know its value, we know - through qualitative appraisal of our program and through testing - something about which values of Θ are more or less likely. In a sense, the program whose reliability we wish to predict can be seen as a program extracted at random from the "population" of all the programs that *could* have been developed under the same general conditions (which determine the distribution of Θ). Seen in these terms, subjective probabilities can be described in the usual "frequentist" terms, i.e., in terms of the fraction of trials on which a certain event happens: each notional "trial" is the production of a program under the same circumstances under which the actual program was produced, and the "event" is "this program will have a given failure rate, under the assumed type of operational use". Then, by assigning probabilities to the different possible values of the failure rate, we represent our knowledge and beliefs about the programs that we could develop: for instance, we could plausibly believe that programs so different from the requirements as to have very high failure rates are very unlikely to be produced or, that programs with failure rates in a certain interval are twice as likely to be produced as those with failure rates in another interval.

We hence describe our knowledge or belief about the given program via a probability distribution for its failure rate Θ. This will be represented by a probability density function:

$$a_\Theta(\vartheta) = \frac{d}{d\vartheta}\left(P(\Theta \le \vartheta)\right)$$

or, for our discrete approximation, by a succession $a_0, a_1, ..., a_N$, where:

$$a_i = P(\Theta = \vartheta_i)$$

As assigned before testing, the succession of the a_i values represents our *prior* (subjective) distribution of the failure rate, because it represents our knowledge *before* we observe some new evidence about this program. This distribution must be obtained from experience: in an ideal case, we would have measured the failure behaviours of many other programs previously developed for similar applications and under approximately the same conditions, and we could thus consider the new program as a new sample from a known population for which we have a reasonably good knowledge of this distribution.

Note that, in particular, $a_0 = P(\Theta = 0)$ is the prior probability that the program is perfectly correct, or defect-free: in the case of simple programs and good development and V & V practices it certainly makes sense to believe that a non-null subset of the programs that one could produce would be defect-free when presented for acceptance testing.

We then test the program (on T independent test cases) and observe no failures. Clearly, this means that the program is more likely to be, in the "population of the possible programs", one of those with a lower failure rate than one of those with a higher failure rate. In other words, after testing, we can update our belief about the reliability of the program, taking into account the test results. The updated belief is expressed by a *posterior* distribution, which we will represent here by a succession $b_0(T), b_1(T), ..., b_N(T)$. This posterior distribution can be obtained by straightforward application of Bayes' theorem. By definition:

$b_i(T) = P(\Theta = \vartheta_i \mid$ the program has passed T independent tests from the given input profile)

and by applying Bayes' theorem:

$$P(\Theta = \vartheta_i \mid T \text{ tests passed}) = \frac{P_{prior}(\Theta = \vartheta_i)\, P(T\text{testspassed} \mid \Theta = \vartheta_i)}{P_{prior}(T\text{testspassed})} \qquad (2.1)$$

If $\Theta = \vartheta_i$, the probability of T failure-free tests is $(1-\vartheta_i)^T$ (assuming that the testers detect all failures, the "perfect oracle" assumption); substituting in (2.1) we finally obtain the values of the b_is as:

$$b_i(T) = \frac{a_i(1-\vartheta_i)^T}{\sum_{j=0}^{N} a_j(1-\vartheta_j)^T} \qquad (2.2)$$

Notice that, for i = 0, the probability of T failure-free tests is 1 (irrespective of T). Hence,

$$b_0(T) = P(\text{the program is perfect} \mid T \text{ tests passed}) = \frac{a_0}{\sum_{i=0}^{N} a_i(1-\vartheta_i)^T}$$

This simple, yet rigorous, model allows us first to give unambiguous mathematical statements of different plausible hypotheses about the program under test, by varying the *prior* distribution of the failure rate, and then to observe how these hypotheses affect the predicted reliability or other probabilities of interest, derived from the *posterior* distribution of the failure rate.

3. Expected failure rate after successful testing

We are now in a position to study examples of dependability assessment. There is a further issue to be decided: which measures of predicted reliability we should use in decision-making. Throughout this paper, we consider the case that we are interested in a program's

"failure rate", Θ (for arguments about alternative measures, see e.g. [9]). So, we may be interested in the expected value of the failure rate after testing, i.e., the mean value of its posterior distribution:

$$E(\Theta) = \sum_{i=0}^{N} \vartheta_i b_i \qquad (3.1)$$

For instance, the next figure shows how this expected value decreases as the number of successful tests increases. For the prior distribution, we chose here the simple case of a uniform prior distribution: $a_\Theta(\vartheta) \equiv 1$ for all $\vartheta \in [0,1]$. We shall see later that this distribution occupies a privileged position in the literature.

As we test the program and observe no failures, the expected value of Θ decreases as 1/(2+T), as shown in the figure below.

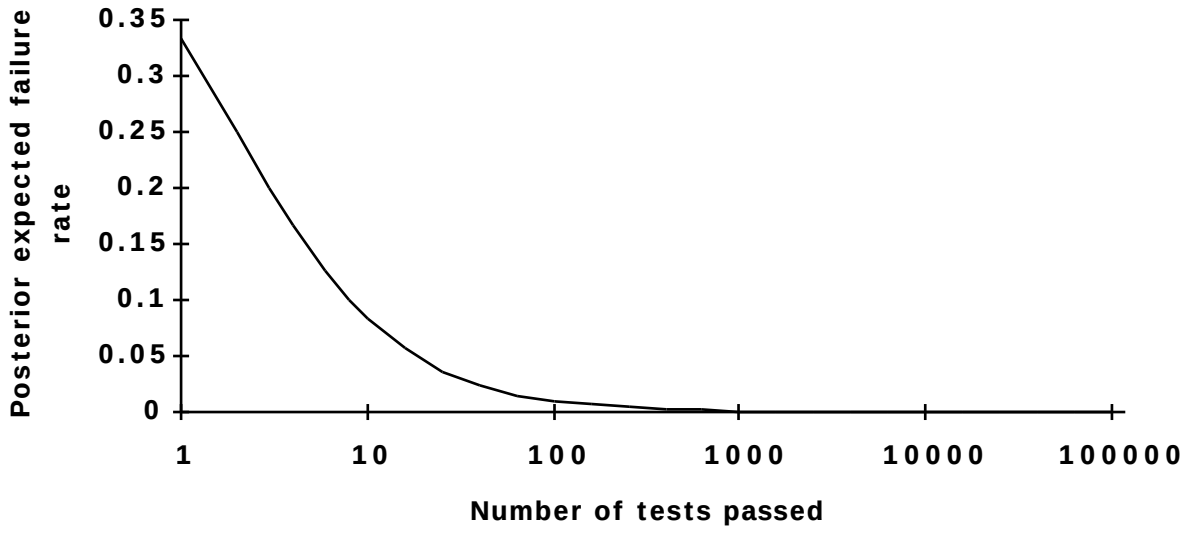

Fig. 3.1: Expected failure rate as successful tests accumulate, assuming that the prior failure rate distribution is uniform between 0 and 1.

However, the mean of Θ is not a very useful measure for a manager who needs to decide whether the software can be released. It only indicates how many failures we could expect from executing many programs in our notional population (of which we have only produced *one* in actual fact), but not the probability that *this* program is satisfactory, because a mean does not show whether most programs would have similar, satisfactory failure rates, or whether there would be a large spread between very reliable and very unreliable programs.

4. Probability of achieving a reliability target

We can choose a better indicator than the expected failure rate if we consider that the very purpose of assessing a program via statistical testing is to check that it satisfies its reliability requirements. In many cases a reliability requirement is stated as an allowed upper bound on the failure rate in operation, i.e., it is required that :

$$\Theta \leq \vartheta_R$$

Then, an appropriate output of the assessment procedure is *the probability that the program satisfies this reliability requirement*, given that it passed T tests. We will call this $P_{succ}(T)$. Thus, by definition:

$P_{succ}(T)$ = P(program has satisfactory Θ | program passed T tests)

Again, this probability is easily derived:

$$P_{succ}(T) = \sum_{i=0}^{R} b_i = \frac{\sum_{i=0}^{R} a_i (1-\vartheta_i)^T}{\sum_{i=0}^{N} a_i (1-\vartheta_i)^T} \qquad (4.1)$$

Figure 4.1 shows an example of how P_{succ} varies as successful tests accumulate.

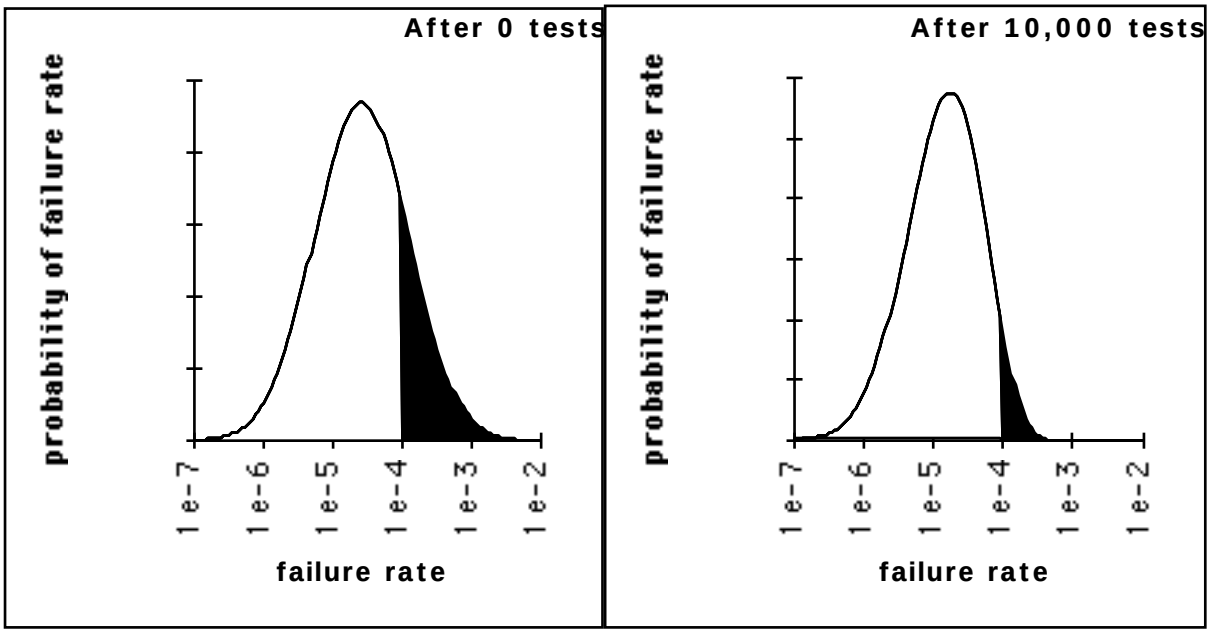

Fig. 4.1 Comparison of prior(left) and posterior (right) probability distributions. The prior is an arbitrary, plausible bell-shaped curve chosen for the sake of illustration. A requirement $\Theta \leq 10^{-4}$ is assumed, and the coloured areas represent the probability that Θ exceeds this value. Observing 10,000 successful tests reduces this probability from 21 % to approximately 5 %, i.e., it increases P_{succ} from 79% (prior probability) to 95 % (posterior probability).

The prior distribution implies, for each value of T, an assignment of probabilities to the four events:

i) "**program OK**": the software satisfies its reliability requirement;

ii) "**program not OK**": the software does *not* satisfy this requirement;

iii) "**test success**": the software will pass the T tests;

iv) "**test failure**": the software will *not* pass the T tests;

as well as for the four non-null intersections of these four events.

The following figure plots how one's knowledge about the reliability of the software evolves as tests accumulate. The Cartesian plane is divided into four bands, and their heights represent the probabilities of the different events, as indicated by the labels. The two upper bands

together represent the probability that the software fails at least once in the T tests: clearly, this probability increases with T. This event represents a project failure, requiring the software to be modified before it can be resubmitted for acceptance testing. Within the event "the software passes T tests", the two lower bands represent respectively the case in which the software is indeed satisfactory, and the case in which it is not, but it does pass the acceptance test, thus possibly deceiving us into accepting it. The ratio between the height of the bottom band and the combined height of the two bottom bands represents the probability that the software is satisfactory ($\Theta \leq \vartheta_R$), given that it has passed T tests.

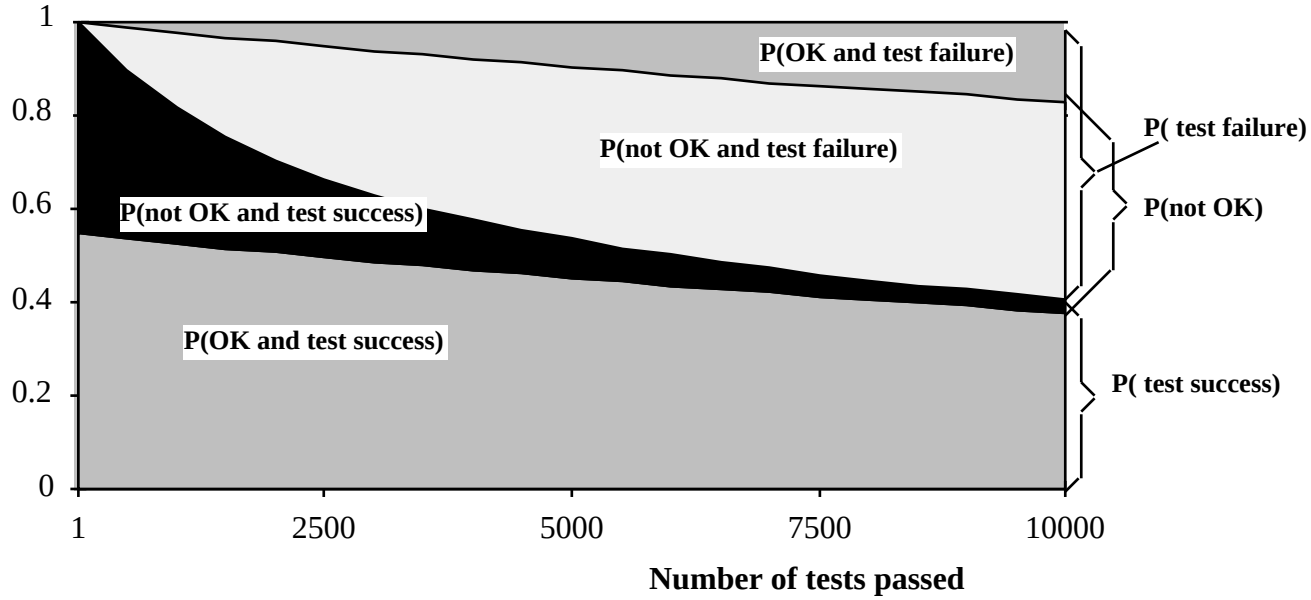

Fig. 4.2: Probabilities of interest in the assessment of software reliability via statistical testing. This plot was derived assuming that the reliability requirement is $\Theta \leq 10^{-4}$ that the prior distribution for Θ is uniform between 10^{-5} and 10^{-3} and 0 elsewhere, except for a non-null probability that the program is perfect ($P(\Theta = 0) = 0.01$).

This figure illustrates the meaning of Bayes' theorem applied to our problem. The sum of the heights of the "OK" bands stays constant as T varies. It equals the prior probability of success, i.e., the fraction of satisfactory programs in our notional population of possible programs. However, when we consider the *posterior* probability that a program is satisfactory, conditional on its passing T tests, we concentrate our attention on those possible programs that would indeed pass T tests. These are represented by the two bands collectively labelled P(test success). This subset of the population obviously shrinks as we perform more tests, but, more importantly, within this subset the fraction of programs with an unsatisfactory failure rate decreases as well, and the fraction of those with satisfactory failure rate increases. This latter fraction is our posterior probability $P_{succ}(T)$: the probability that a program has the required low failure rate, *conditional* on its passing T tests.

5. Reliability assessment with different prior distributions

5.1. Uniform prior

Assigning prior distributions is difficult. An appealing notion is to look for a prior distribution that represents "ignorance" about the variable of interest. This is actually impossible. Any representation of "ignorance" actually embodies a statement about which events are equally

likely. For instance, do we think that the true value of Θ is as likely to lie within the interval [0.1, 0.2] is as within [0.2, 0.3], [0.3, 0.4], etc.? Or do we think that its is as likely to fall in [0.1, 1] as in [0.01, 0.1], as in [0.001, 0.01], etc.? Both these beliefs could be construed as "ignorance", yet they describe very different distributions.

However, a prior distribution which can be said plausibly to represent minimal knowledge, if one accepts certain requirements for mathematical symmetry [4, 7], is the *uniform* prior,

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

We will thus use this prior distribution as a term of comparison with the others that we are going to study in this paper. For this prior, the posterior distribution is a "Beta" distribution, with a probability density function given by:

$$f_{\Theta}(\vartheta) = \frac{\vartheta^{a-1}(1-\vartheta)^{b-1}}{B(a,b)} \qquad (5.1)$$

where a and b are two real valued, positive, parameters and B(a, b) is the "Beta" function.

With a uniform prior, the parameters of the posterior Beta distributions after T successful tests take the values a=1, b=T+1 (Beta distributions are further examined in Section 5.4).

5.2. Implications of the prior probability that the program is correct

A case of some interest is that of software that has a non-negligible probability of being defect-free. Some practitioners would consider this plausible for, e.g., simple software developed under very stringent quality criteria, as mandated for some safety-critical software (they might qualify this belief by limiting it to defects with safety-critical implications). This knowledge can be represented as a certain non-zero probability for the event Θ = 0. If Θ is represented as a continuous random variable, the value of its probability density function for Θ = 0 is then represented by an impulse or "delta" function.

Fig. 5.1 shows the effects of this kind of prior distribution. The prior distributions that give rise to these three curves only differ in the prior probability of perfection, a_0; in all three, the probability density function is constant over the rest of the x axis, $a(\vartheta) = 1\text{-}a_0$ for all $0<\vartheta \leq 1$. The striking feature is that a high P_{succ} can be reached with very few tests. So, it is correct to believe that some confidence in correctness should bolster one's trust in a program a great deal. The real problem is in justifying one's confidence in correctness, since P_{succ} is very sensitive to variations in a_0. Of course, failing one test would make the posterior probability of correctness $b_0 = 0$.

We can also see in Fig. 5.2 how the probability of correctness, i.e., the chance that this program is actually one of the perfect ones in our population, grows rapidly as we observe successful tests. The curves in Fig 5.2 are quite similar to the two top curves in Fig. 5.1, because the chosen prior density functionimplies $P(0<\Theta\leq 10^{-4} \mid T=0)=10^{-4}\ (1\text{-}a_0)$. This makes $P(0<\Theta\leq 10^{-4}) \ll P(\Theta=0)$ (both when the latter is 1% and when it is 50%), for most values of T. Since, by definition, $P_{succ}= P(\Theta=0)+P(0<\Theta\leq 10^{-4})$, this inequality implies $P_{succ}\approx P(\Theta=0)$.

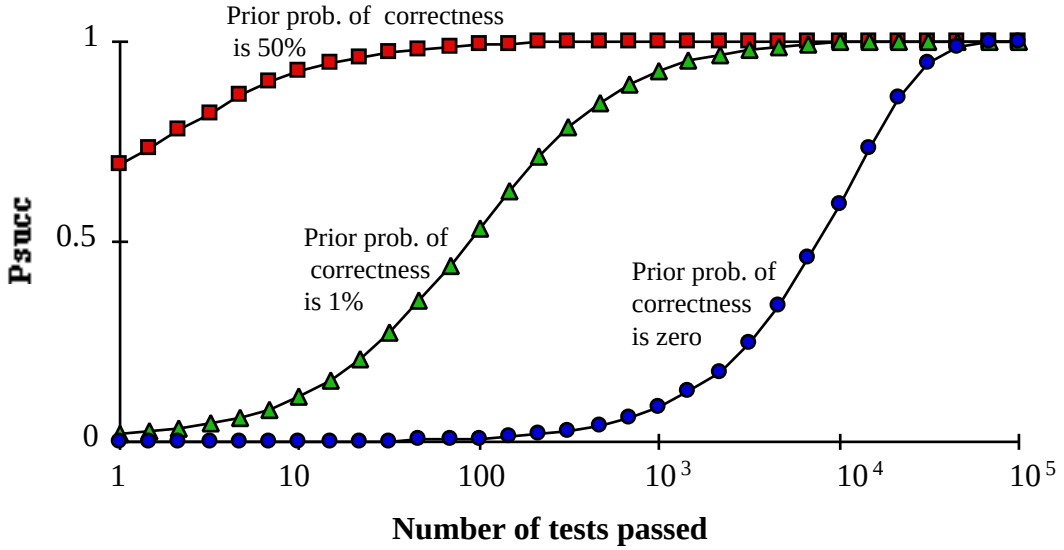

Fig. 5.1: Effects of a non-null prior probability of correctness on the posterior probability of achieving a reliability target (ϑ_R = 10-4). The number of tests passed is shown on a logarithmic scale.

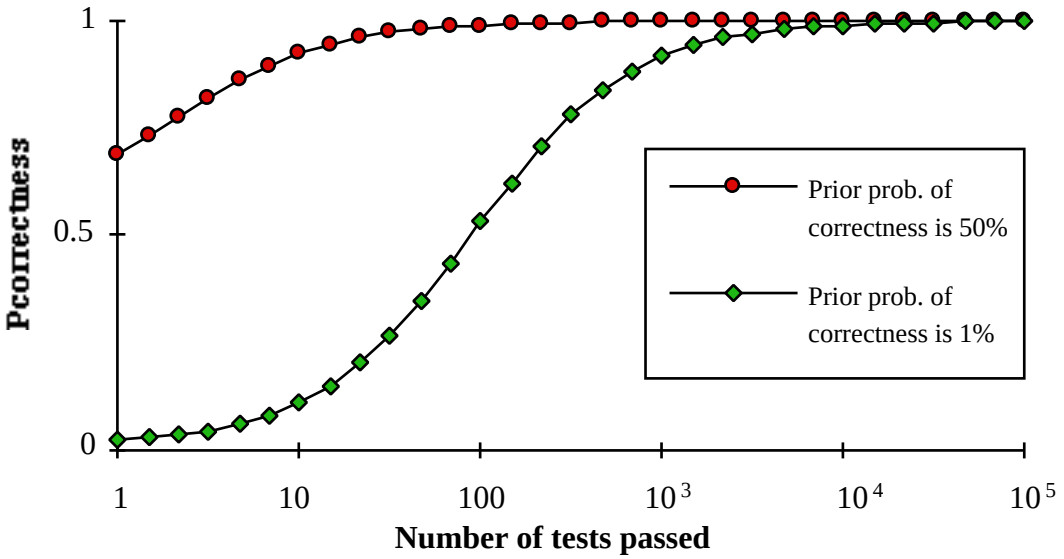

Fig. 5.2: Posterior probability of correctness as successful tests accumulate

5.3. Prior distributions where high values of Θ are very unlikely

We may expect that usually, with a quality development process, a program that is ready for acceptance testing is very unlikely to have a high failure rate. This is by no means certain: at least one release of the Space Shuttle software had a bug which would cause it to fail in the start-up phase more frequently than once in 100 start-ups [11]. However, let us examine the consequences of such assumptions. We model the assumption in a stylised fashion, by assigning zero prior probability to high failure rates: the upper bound on Θ is no longer 1, but a smaller number. The figure 5.3 shows the consequences of such prior distributions on $P_{succ}(T)$.

Four prior distributions for the failure rate are considered, all of them uniform in the interval between 0 and an upper bound, which is (respectively for the four curves, from top to bottom): $5*10^{-4}$, 10^{-3}, 10^{-2} and 1.

Notice that:

- if we assumed an upper bound of 10^{-4}, then the curve would be constantly 1: we would know from the beginning that our software is "good enough". This shows that stating a belief that high failure rates are unlikely in the form of an upper bound on Θ actually amounts to a quite strong assumption;
- after about 10,000 tests (when $P_{succ}(T)$ 0.5, the four curves do not differ by much. At this stage, we have practically eliminated the possibility of a very high failure rate, independently of how high its prior probability was;
- these are all quite unfavourable prior distributions: a developer believing these would expect (for the most optimistic of the three) that 8 out of 10 programs do not achieve the target reliability.

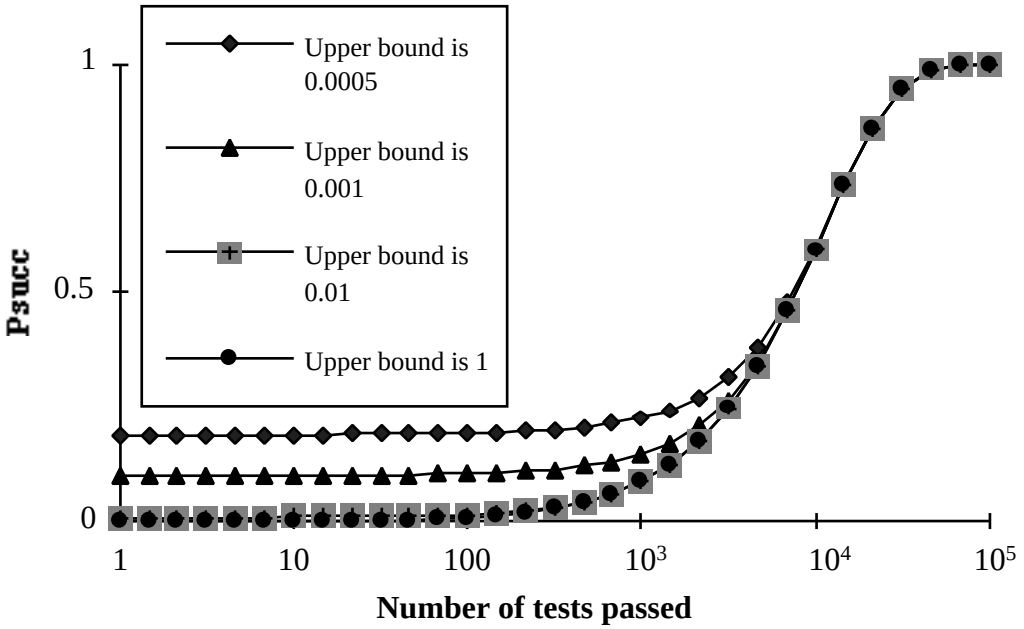

Fig. 5.3: Effects of prior knowledge that the programs cannot have a very high failure rate on the posterior probability of achieving a reliability target ($\vartheta_R = 10^{-4}$)

5.4. Bell-shaped prior distributions of Θ

Human error, which causes software failures, is a natural phenomenon arising in an activity which combines a large number of intellectual tasks. This may lend some credibility to the conjecture that the distribution of failure rates will approximate a Gaussian distribution or other bell-shaped curve. We have considered here a few prior distributions of the Beta family, which are shown in figure 5.4 below.

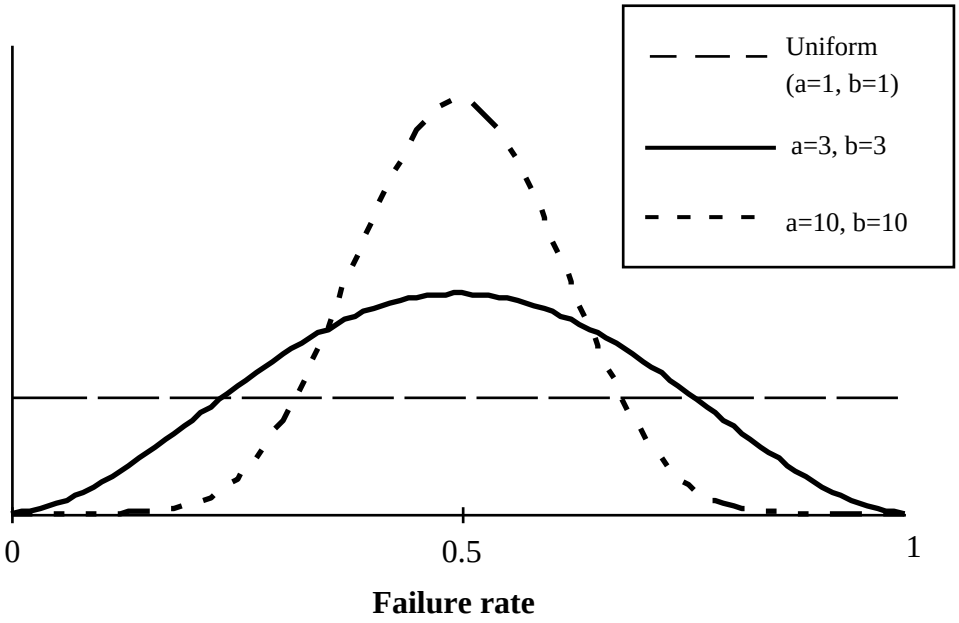

Fig. 5.4 Three probability density functions for distributions in the Beta family

The Beta distribution has the property of being the *conjugate* distribution for this particular inference process: if our prior distribution is a Beta distribution with parameters {a, b} (including its limit case, the uniform distribution), the posterior distribution (after any number T of tests with any number *r* of test failures) will also be Beta, with parameters {a+*r*, b+T}. For instance, in Fig. 5.5 we assume that the prior distribution for the failure rate is a Beta distribution with parameters *a*=3 and *b*=3 and show the posterior distributions derived after different numbers of successful tests.

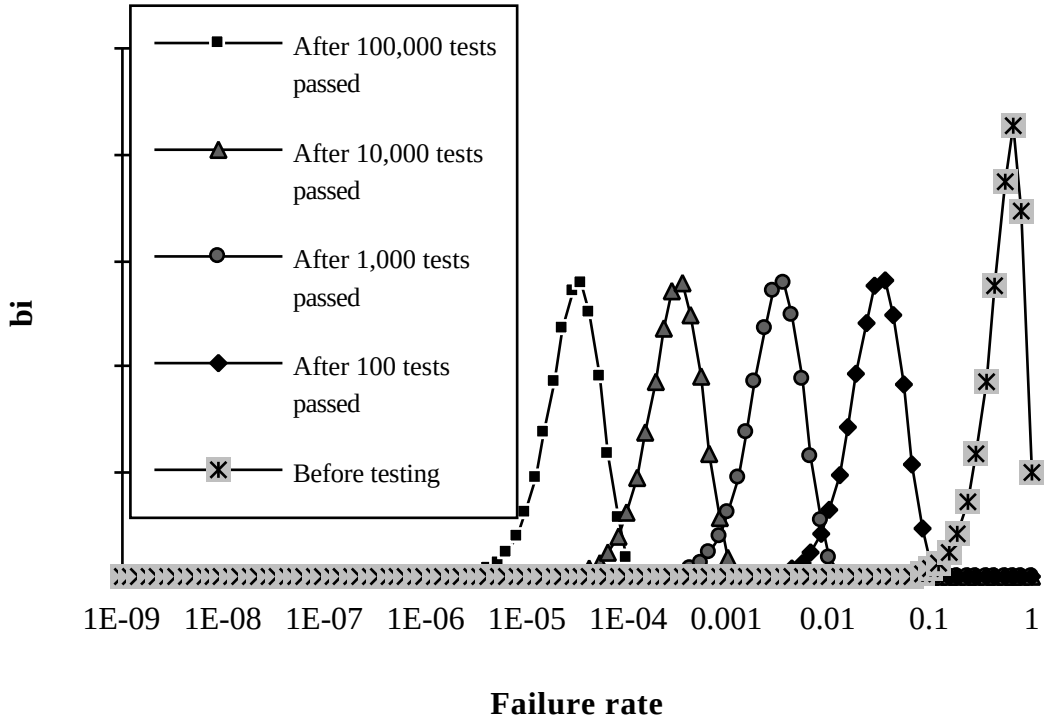

Fig 5.5 Posterior probabilities after various numbers of tests, assuming a prior Beta with a=3 and b=3 (rightmost curve). Note the logarithmic scale for Θ.

It should be noticed that, given a certain experience of previous programs, assigning a mean for a bell-shaped prior distribution would be relatively non-controversial. However, it is the shape of its "tails" that would be problematic. For instance, if some past program was never observed to fail, would this mean a comparatively high probability that the program is correct, or simply that those programs had a very small, non-zero failure rate? These problems of inference can be solved, but the beliefs about the shape of the distribution for failure rates that are too low to be frequently observed would greatly affect the solution.

A problem with the curves in Fig. 5.4 is that they imply a high prior expected value of the failure rate, which is implausible. Nobody would start a software project if the probability of success were as small as these prior beliefs imply. A rough approximation of a bell curve with a low probability of a high failure rate is a Beta distribution "scaled down" so as to take non-zero values only between 0 and a certain upper bound, as we assumed in section 5.2[1].

Thus, in the Fig. 5.6 we show how P_{succ} varies vs. the number of successful tests, for different prior distributions in the Beta family. The better cases are obviously those assuming an upper bound smaller than 1, here 10^{-3}.

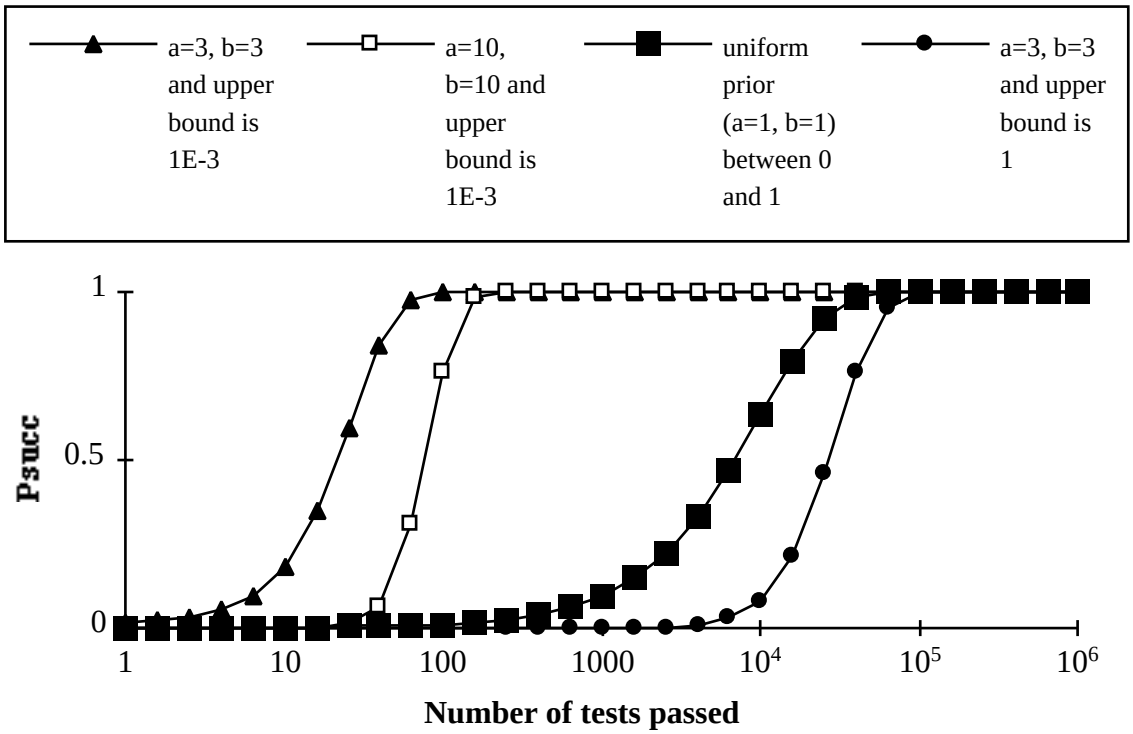

Fig. 5.6 Posterior probability P_{succ} of achieving a reliability target ($\vartheta_R = 10^{-4}$), for Beta prior distributions and for Beta distributions limited to a sub-interval of the (0,1) interval. *a* and *b* represent the two parameters of a Beta distribution

[1] One can instead select a Beta distribution with a high P_{succ} and non-zero values between 0 and 1 by varying the two parameters, a and b, separately. We chose to show here intuitively simpler distributions. In an actual software assessment exercise, we would not recommend adherence to any analytically simple distribution, but, rather, experimenting with various plausible representations of one's assumptions, to make sure that one's predictions are not just an artefact of a peculiar mathematical representation. Numerical calculations via software tools make this approach reasonably inexpensive.

6. Modelling more complex scenarios

All the examples shown this far rely on two assumptions: the outcome of one test case for our program is independent of the outcome of the previous tests; and the probability of failing a test is the same as that of failure on one execution during actual operation. These assumptions are satisfied if we test with test cases independently sampled from the usage profile, and we recognise every program failure during test ("perfect oracle" assumption). In an actual testing exercise, some of these assumptions may be violated, by accident or on purpose. For instance:

- our oracle may be less than perfect: some erroneous behaviours are not recognised;
- our oracle may be "better than perfect": by probing internal variables of the program, we can recognise anomalous behaviours (and thus detect defects, and send the program back to be fixed) even if they do not manifest themselves via an erroneous output [10];
- our test profile may not faithfully represent the future operational input profile, because of the uncertain knowledge about it;
- we may choose to alter the test profile, for instance because we have a notion that we can make it more "stressful" for the program; we may even choose test cases not by a statistical method, but by some method which we think more cost-effective for finding defects (if they are present), and yet would like to use the fact that testing produced no failure to improve our reliability predictions (an example of this is analysed in [7]).

Some of these changes (especially among those concerning test directed by human testers on the basis of some defect-seeking method) would violate the independence assumption, making our mathematical model rather more complicated. We will only consider here those that do not violate it, but only change the conditional probability of a program failing a test, given the program's operational failure rate. Until now, we had

$P(\text{test fails} \mid \Theta = \vartheta_i) \equiv \vartheta_i$ for all i

Now, we will have instead a separate function $\delta(\vartheta)$ (that is, in our discrete representation, a succession δ_i), defined as

$\delta_i = P(\text{test fails} \mid \Theta = \vartheta_i)$

Each probability δ_i may greatly differ from the corresponding ϑ_i, except for the obvious constraint that $\delta_0 = \vartheta_0 = 0$ (fault-free program). Then, the probability of the program passing T tests, *if* $\Theta = \vartheta_i$, is:

$P(\text{T tests passed} \mid \Theta = \vartheta_i) = 1-(1-\delta_i)^T$

and substituting this expression in the previous expressions for $b_i(T)$ (2.1), we obtain, instead of equation (2.2):

$$b_i(T) = \frac{a_i(1-\delta_i)^T}{\sum_{j=0}^{N} a_j(1-\delta_j)^T} \qquad (6.1)$$

and can derive expressions of $E(\Theta)$ (3.1) and P_{succ} (4.1) for this more general case.

The values of the succession δ_i represent the combined effect of all the factors listed above: discrepancies between the input distributions in testing and in operation, the oracle coverage,

etc. So, for instance, stating that $\delta_i > \vartheta_i$ means that for those programs in our notional population that have failure rate ϑ_i, the probability of failing a test is greater than the probability of failure in operation. This means that *for those programs, on average,* the oracle is especially effective and/or the test selection criterion we use is especially effective in causing defects to manifest themselves, etc.

It is clear that a given succession δ_i may represent the effects of many different combinations of the above circumstances. We have not yet begun to explore the variety of knowledge or intuition about a program's structure and behaviour that could be mapped into the conditional distribution δ_i**.** This will depend, for instance, on whether a given failure rate is more likely to be the product of one or another type of defects, whether these defects are more likely to be found by statistical testing or by testing specifically aimed at activating those defects that the testers consider most likely, etc.

We now consider some illustrative situations in which δ_i can be described as a simple function of Θ. Simple distributions that it seems interesting to explore include:

- δ_i is proportional to ϑ_i, i.e., $\delta_i = k^* \vartheta_i$. Setting $k < 1$ would represent the case in which the oracle used fails to detect a fraction of the failures, and this fraction is the same irrespective of the program's actual failure rate (a similar scenario is studied in [12], deriving "classical" confidence bounds). Setting $k > 1$ (under the condition, of course, that $\delta_i \leq 1$ for all i) would represent, for instance, the case in which we are able to effectively "stress-test" the software through our selection of test cases. In Fig. 6.1 we show again P_{succ} vs. the number of tests passed, but this time we vary the probability with which we assume errors or failures are detected under test. As one could expect, the higher the detection rate (compared to the failure rate in operation), the more rapidly P_{succ} increases.

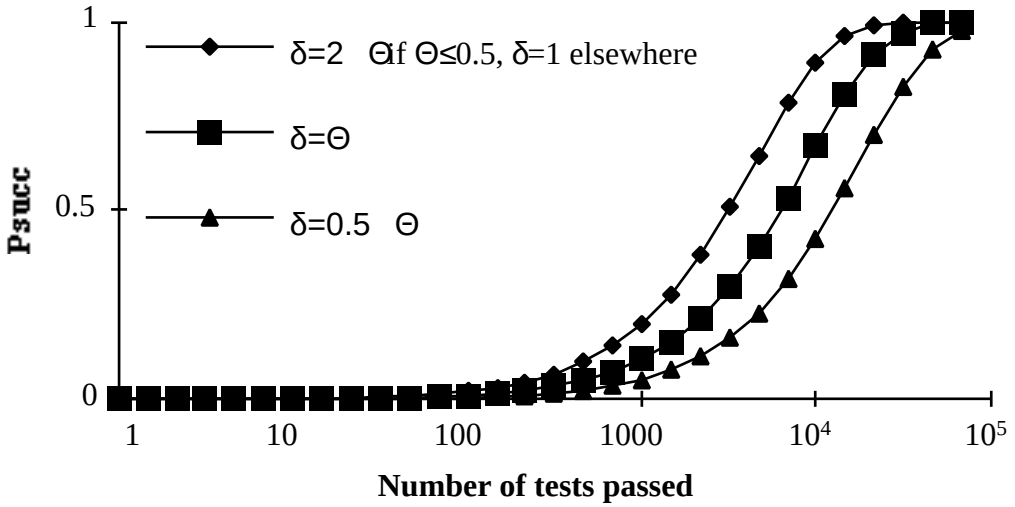

Fig. 6.1 Posterior probability P_{succ} of achieving a reliability target ($\vartheta_R = 10^{-4}$), assuming that the prior failure rate is uniformly distributed between 0 and 1, but that the detection rate δ is different from the failure rate

- δ_i is constant: it does not depend on the operational failure rate, except of course that δ_0=0. One of the scenarios in which this could be true is a special case of programs which are tested with inputs chosen by human testers. One may conjecture that the failure

behaviour of the programs is mostly determined by the following phenomenon. There are in the program's input space many "sensitive spots" (determined by the program's specifications, not by the specific program actually produced), such that if there are defects in the code, the resulting failure regions will be likely to be centred in one of these spots. Furthermore, variations in failure rate between possible programs do not imply a marked variation in the number of failure regions in the input space. Rather, programs with higher failure rates tend to have approximately the same number of failure regions as programs with lower failure rates, but with a higher average probability of being "hit" by an input in operation. In addition, tests are chosen by human testers who have an intuitive notion of where the "sensitive spots" are, so that each test does hit a sensitive spot: if that sensitive spot has a failure region around it, the program will fail the test. So, the probability of test failure is not affected by the size of failure regions (hence by the failure rate of the program), which is constant, but only by their number relative to the number of sensitive spots.
It is clear that this is an extreme scenario, but useful to consider as an extreme of what may happen in reality, e.g. if "difficult spots" in the functional specification are the main cause for defects.
This hypothesis has interesting consequences. Since $\delta_i = 0$ for i=0, $\delta_i = \delta$ elsewhere,

$$b_0(T) = \frac{a_0}{\sum_{i=0}^{N} a_i (1-\delta_i)^T} = \frac{a_0}{a_0 + \sum_{i=1}^{N} a_i (1-\delta)^T} = \frac{a_0}{a_0 + (1-a_0)(1-\delta)^T}$$

and for $i \neq 0$:

$$b_i(T) = \frac{a_i (1-\delta_i)^T}{\sum_{j=0}^{N} a_j (1-\delta_j)^T} = \frac{a_i (1-\delta)^T}{a_0 + \sum_{j=1}^{N} a_j (1-\delta)^T} = \frac{a_i (1-\delta)^T}{a_0 + (1-a_0)(1-\delta)^T} = \frac{a_i}{\frac{a_0}{(1-\delta)^T} + 1 - a_0}$$

i.e., b_0 (the probability that the program is correct) increases with successful testing, and all the b_i for $i \neq 0$ (probabilities of non-zero values of Θ) decrease by the same factor. As for the probability of having the required reliability:

$$P_{succ}(T) = \sum_{i=0}^{R} b_i = \frac{a_0 + (1-\delta)^T \sum_{i=1}^{R} a_i}{a_0 + (1-a_0)(1-\delta)^T}$$

and, in particular, in the limiting case in which $a_0 = 0$, this probability *does not* increase with successful tests. In other words, the tester has no information as to whether successful tests "eliminate" from consideration failure-prone programs or highly reliable programs.

7. Conclusions

We have shown through some examples how a standard Bayesian inference procedure can clarify reasoning about evidence from testing. This method allows one to state one's expectations derived from factors others than testing, and then to calculate the measure of added trust that can be derived from successful testing. The method can also be used to model the effects of many important factors affecting testing, such as the fault-revealing power of debug-oriented testing, imperfect test oracles, testability measures.

The whole method depends on assigning a prior distribution for the failure rate of a program. This is clearly difficult. However, many decisions in software projects are now made on an

intuitive basis, i.e., on the basis of assumptions that are not subjected to any experimental verification. These assumptions can be modelled as prior distributions, and the decision makers can thus see what could be inferred from testing if their assumptions were true; whether the conclusions are sensitive to small errors in these assumptions, or rather robust, and what would be the consequences if alternate plausible conditions held instead. They can see, given a decision that seems intuitively correct, which assumptions would be needed to justify it, and whether they are confident enough in those assumptions. Last, but not least, this process allows learning about the assumptions themselves, by the usual scientific method of conjecture and refutation. If the predictions based on our priors prove to be wrong (e.g., if of ten programs predicted to have $\Theta < \vartheta_R$ with probability 0.9, only 1 turns out to be so reliable), we will correctly be drawn to suspect our assumptions and attempt to correct them.

In intuitive reasoning in uncertain situations, people often manifest "overconfidence bias" : they are more confident in the truth of their beliefs than is warranted by empirical measurement. This risk must obviously be taken into account when relying heavily on expert judgement for assigning prior distributions. It is then useful to be able to visualise these opinions and their consequences in clear mathematical terms.

Many intuitions that practitioners have about software are in terms of "microscopic" behaviour: which kinds of defects predominate, whether a certain testing method is more effective against a kind of defects or another, etc. Translating these intuitions into clear statements about probabilities on a "macroscopic" level - as our a_i distributions, for a start - will allow more consistent decision-making and some learning about when such intuitions are true. We see this "translation" - exploring more sets of assumptions of practical interest - as the next task in this work.

Acknowledgements

This work was funded in part by the European Commission via the "OLOS" research network (Contract No. CHRX-CT94-0577), and via the ESPRIT Long Term Research Project 20072 "DeVa". The authors thank Marc Vinck and Karama Kanoun for their comments on an earlier version of this paper.

References

[1] D. L. Parnas, A. J. van Schouwen and S. P. Kwan, "Evaluation of Safety-Critical Software", Communications of the ACM, 33, pp. 636-648, 1990.

[2] J. D. Musa, "Operational Profiles in Software-Reliability Engineering", IEEE Software, March, pp. 14-32, 1993.

[3] J. May, G. Hughes and A. D. Lunn, "Reliability estimation from appropriate testing of plant protection software", Software Engineering Journal, 10, pp. 206-218, 1995.

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

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

[6] B. Littlewood and J. L. Verrall, "A Bayesian Reliability Growth Model for Computer Software", J. Royal Statist. Soc. C, 22, pp. 332-346, 1973.

[7] 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, 18, pp. 33-43, 1992.

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

[9] B. Littlewood and D. Wright, "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.

[10] A. Bertolino and L. Strigini, "On the use of testability measures for dependability assessment", IEEE Transactions on Software Engineering, 22, pp. 97-108, 1996.

[11] J. R. Garman, "The 'bug' heard round the world", ACM SIGSOFT Software Engineering Notes, 6, pp. 3-10, 1981.

[12] P. E. Amman, S. S. Brilliant and J. Knight, "The Effect of Imperfect Error Detection on Reliability Assessment via Life Testing", IEEE Transactions on Software Engineering, 20, pp. 142-148, 1994.