ARO Report 93-2

PROCEEDINGS OF THE THIRTY-EIGHTH CONFERENCE ON THE DESIGN OF EXPERIMENTS IN ARMY RESEARCH DEVELOPMENT AND TESTING

Approved for public release; distribution unlimited.
The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

Sponsored by
The Army Mathematics Steering Committee
on Behalf of

THE CHIEF OF RESEARCH, DEVELOPMENT AND ACQUISITION

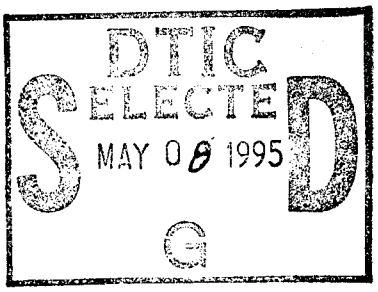
Revised 10 February 1992

Models for Assessing the Reliability of Computer Software

by

Nozer D. Singpurwalla and Simon P. Wilson
The George Washington University, Washington, D.C. 20052

GWU/IRRA/Serial TR-91/13
December 1991



Research Supported by

Contract N00014-85-K-202
Office of Naval Research

Grant DAAL03-87-K-0056
The Army Research Office

and

Grant AFOSR-F49620-92-J-0030
The Air Force Office of Scientific Research

Models for Assessing the Reliability of Computer Software

by

Nozer D. Singpurwalla and Simon P. Wilson

The George Washington University, Washington, D.C. 20052

Abstract

A formal approach for evaluating the reliability of computer software is through probabilistic models and statistical methods. This paper is an expository overview of the literature on the former. The various probability models for software failure can be classified into two groups; the merits of these groups are discussed and an example of their use in decision problems is given in some detail. The direction of current and future research is contemplated.

1. Introduction.

Having been developed over the last 20 years, software reliability is a relatively new area of research for the statistics and the computer science communities. It arose because of interest in trying to predict the reliability of software, particularly when its failure could be catastrophic. Obviously the software that controls an aircraft carrier, a nuclear power station, a submarine or a life-support machine needs to be very reliable, and statistical techniques will aid the computer scientist in deciding if such software has sufficient reliability. The subject is also of commercial importance, as for example when decisions have to be made concerning the release of software into the marketplace.

All software is subject to failure, due to the inevitable presence of errors (or bugs) in the code, so the first aim of the subject has been to develop models that describe software failure. There are various methods of specifying such failure models, and Section 2 discusses these in some detail. It is fair to say that this model derivation has been the focus of research so far. Once a failure model has been specified then it can be applied to problems such as the optimal time to debug software or deciding whether software is ready for release. These applications have received less attention in the literature but are becoming more prevelant. We will mention here that there is another approach to software reliability that differs considerably from the statistical ideas presented here. This approach attempts to prove the reliability, or correctness, of software by formal means of proof, just as one would prove a mathematical theorem. This is an exercise in logic, albeit a rather complex one. It works well on small programs, for example on a program that computes the factorial function, but becomes a forbidding task for even moderately complex pieces of code. Nevertheless, the idea that software can be proved correct is appealing. The approach is not discussed further.

This paper is divided into 5 further sections. Section 2 categorizes the different strategies that have been used to model software failure. Section 3 reviews the historical development of the subject by describing some of the more commonly used models, and Section 4 shows that many of these models can be unified if one adopts a Bayesian position. Section 5 looks at applications of the material developed in Section 3, and Section 6 concludes with a look at the current and future direction of the subject. We assume that the reader has some familiarity with some basic reliability and probability concepts; in particular it is important that he or she has knowledge of some common probability distributions, statistical inference and decision making, Poisson processes and the concept of a failure rate.

2. Model Categorization

All statistical software reliability models are probabilistic in nature. They attempt to specify the probability of software failure in some manner. In looking through the literature, one observes that the models developed so far can be broadly classified into two categories

Type I: Those which propose a probability model for *times between successive failure* of the software, and

Type II: Those which propose a probability model for the *number of failures* up to a certain time.

Time is often taken to be CPU time, or the amount of time that the software is actually running, as opposed to real time. In theory, specification via one of these two methods enables one to specify the other. So a model that specifies time between failure will also be able to tell you the number of failures in a given time, and vice versa. In practice, this may not be straightforward.

The first of these categories, modeling time between failure, is most commonly accomplished via a specification of the *failure rate* of the software as it is running. When this is the case the model is to be of **Type I-1.** The failure rate for the i-th time between failure is given, for i=1, 2, 3 ,... and a probability model results. One distinctive feature of software is that its failure rate may decrease with time, as more bugs are discovered and corrected. This contrasts with most mechanical systems which will age over time and so have an increasing failure rate. An attempt to debug software may introduce more bugs into it, thus tending to increase the failure rate, so the decreasing failure rate assumption is somewhat idealized. However, most of the models of this type that are reviewed here have a decreasing failure rate.

Another way to model time between failure is to define a stochastic relationship between successive failure times. Models that are specified by this method are known as **Type I-2,** and have the advantage over Type I-1 in that they model the times between failure directly, and not via the abstract concept of a failure rate. For example, let T_1, T_2, ..., T_i, ... be the length of times between successive failure of the software. As a simple case, one could declare that $T_{i+1} = \rho T_i + \epsilon_i$, where $\rho \geq 0$ is a constant and ϵ_i is an error term (typically some random variable with mean 0). Then $\rho<1$ would indicate decreasing times between failure (software reliability expected to become worse), $\rho=1$ would indicate no expected change in software reliability whilst $\rho>1$ indicates increasing times between failure

(software reliability expected to improve). Those familiar with time series will recognize the relationship in this example as an auto-regressive process of order 1; in general, one would say $T_{i+1} = f(T_1, T_2, \ldots, T_i) + \epsilon_i$ for some function f.

The second of these categories, modeling the number of failures, uses a point process to count the failures. Let M(t) be the number of failures of the software that are observed during time [0,t). M(t) is modeled by a *Poisson process*, which is a stochastic process with the following properties:

i) $M(0) = 0$ and if $s<t$ then $M(s) \leq M(t)$. M(t) takes values in $\{0, 1, 2, \ldots\}$

ii) The number of failures that occur in disjoint time intervals are independent. So, for example, the number of failures in the first 5 hours of use has no effect on the number of failures in the next 5 hours.

iii) The number of failures to time t is a Poisson random variable with mean $\mu(t)$, for some non-decreasing function $\mu(t)$; that is to say:

$$P(M(t)=n) = \frac{(\mu(t))^n}{n!} e^{-\mu(t)} \qquad n=0, 1, 2,\ldots$$

The different models of this type have a different function $\mu(t)$, which is called the mean value function. The mean number of failures at time t is indeed $\mu(t)$, as is the variance. The Poisson process is chosen because in many ways it is the simplest point process yet it is flexible and has many useful properties that can be exploited. This second approach has become increasingly popular in recent years. M(t) can also be specified by its intensity function $\lambda(t)$, which is the derivative of $\mu(t)$ with respect to t; either of these functions completely specify a particular Poisson process. One disadvantage of this approach is that it implies that there are conceptually an infinite number of bugs in the program, which is obviously impossible for code of a finite length. Another disadvantage is more subtle; the model implies a positive correlation between the number of failures in adjoining time intervals, a situation which is not true since again the total number of bugs has to be finite.

Figure 1 is a flow-chart showing the above categorization of the statistical models.

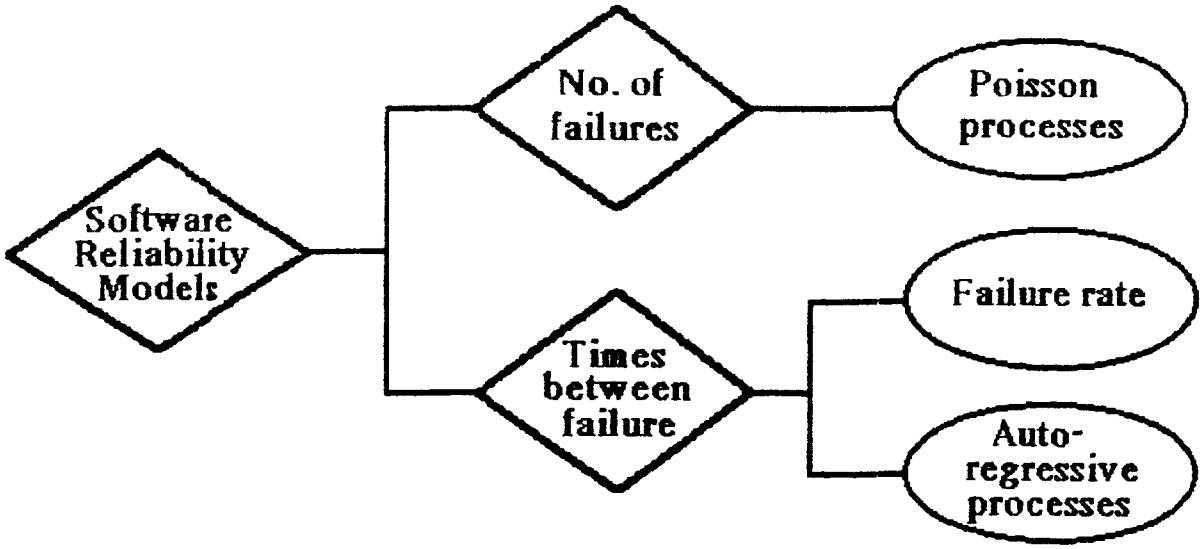


Figure 1. Categorization of Software Reliability Models.

3. Review of Some Software Reliability Models

This section introduces some of the more well known probability models for software reliability. There are examples of models from each of the two main categories that were discussed in the previous section. Since the main purpose of the review is to describe the ideas and assumptions behind the models, technical details will be kept to a minimum in most cases. Those interested in the details of a particular model are advised to reference the papers where they were originally presented.

Some common notation will be assumed throughout this section and is given below :

i) T_i = i-th time between failure of the software [i.e.time between (i-1)th and i-th failure].

ii) $r_{T_i}(t)$ = failure rate for T_i, the i-th time between failure, at time t.

iii) $M(t)$ = number of failures of the software in the time interval [0, t) (a Poisson process).

iv) $\lambda(t)$ = intensity function of M(t).

v) $\mu(t)$ = expected number of failures of software in time [0,t).

$$= \int_0^t \lambda(s)\, ds \quad \text{, since M(t) is a Poisson process.}$$

10 models are presented. Model numbers 1 to 7 are of Type I-1, models 8 and 9 are of Type II and model 10 is of type I-2. A common problem to all the models is the lack of data on which to test their validity; data on software reliability is commercially sensitive and so statisticians in academia have very little information on how software in the marketplace actually performs. For this reason it is important that the assumptions made in deriving these models are carefully thought about.

1. The model of Jelinski & Moranda (1972).

This was the very first software reliability model that was proposed, and has formed the basis for many models developed after. It is a Type I-1 model; it models times between failure by considering their failure rates. Jelinski and Moranda reasoned as follows. Suppose that the total number of bugs in the program is N, and suppose that each time the software fails, one bug is corrected. The failure rate of the i-th time between failure, T_i, is then assumed a constant proportional to N-i+1, which is the number of bugs remaining in the program. In other words

$$r_{T_i}(t \mid N, \Lambda) = \Lambda\,(N\text{-}i+1), \qquad i=1, 2, 3, \ldots, t \geq 0, \quad \text{for some constant } \Lambda.$$

There are some criticisms that one could make of the model. It assumes that each error contributes the same amount Λ to the failure rate, whereas in reality different bugs will have different effects. It also assumes that every time a fix is made, no new bugs are introduced; note [see Figure 2(i)] that the successive failure rates are indeed decreasing. A model like this is sometimes refered to as a "de-eutrophication model", because the process of removing bugs from software is akin to the removal of pollutants in rivers and lakes.

2. Bayesian Reliability Growth Model (Littlewood & Verall (1973)).

Like the Jelinski & Moranda model, the model proposed by Littlewood and Verall looked at times between failure of the software. However, they did not develop the model by characterizing the failure rate; rather they stated that the model should *not* be based on fault content (as Jelinski & Moranda had assumed) and then declared that T_i has an exponential distribution with scale Λ_i, and that Λ_i itself has a gamma distribution with shape α and scale $\Psi(i)$, for some function Ψ. Despite this it is still considered to be a Type I-1 model.

Specifically :

$$f_{T_i}(t \mid \Lambda_i) = \Lambda_i\, e^{-\Lambda_i t} \qquad t \geq 0$$

$$\Pi_{\Lambda_i}(\lambda \mid \alpha, \Psi(i)) = \frac{(\Psi(i))^{\alpha}}{\Gamma(\alpha)}\, \lambda^{\alpha-1}\, e^{-\Psi(i)\lambda} \qquad \lambda \geq 0$$

$\Psi(i)$ was supposed to describe the quality of the programmer and the programming task. As an example, they chose $\Psi(i)=\beta_0+\beta_1 i$. One can show that this makes the failure rate of each T_i decreasing in t and that each time a bug is discovered and fixed there is a downward jump in the successive failure rates; see Figure 2(ii). In fact

$$r_{T_i}(t \mid \alpha, \beta_0, \beta_1) = \frac{\alpha}{\beta_0+\beta_1 i+t}, \text{ for } t \geq 0.$$

If $\beta_1>1$ then the jumps in the failure rate decrease in i, if $\beta_1<1$ they increase whilst if $\beta_1=1$ they remain a constant. So if β_1 differs from 1 then the fixing of each bug is making a different contribution to the reduction in the failure rate of the software, an apparent advantage over the model by Jelinski &

Moranda. This model has received quite a lot of attention and has been the subject of various modifications; see models 6 and 7 later in this section.

3. The De-eutrophication model of Moranda (1975).

Another (de-eutrophication) model of Moranda (1975) attempted to answer some of the criticisms of the Jelinski & Moranda model, in particular the criticism concerning the equal effect that each bug in the code has on the failure rate. He hypothesized that the fixing of bugs that cause early failures in the system reduces the failure rate more than the fixing of bugs that occur later, because these early bugs are more likely to be the bigger ones. With this in mind, he proposed that the failure rate should remain constant for each T_i, but that it should be made to decrease geometrically in i after each failure i.e. for constants D and k

$$r_{T_i}(t \mid D, k) = D\, k^{i-1} \qquad t \geq 0,\ D>0 \text{ and } 0<k<1.$$

Compared to the Jelinski & Moranda model, where the drop in failure rate after each failure was always Λ, the drop in failure rate here after the i-th failure is $D\, k^{i-1}(1-k)$ see Figure 2(iii). The assumption of a perfect fix, with no introduction of new bugs during the fix, is retained.

4. Imperfect Debugging Model (Goel & Okumoto (1978)).

This model is another generalization of the Jelinski & Moranda model which attempts to address the criticism that a perfect fix of a bug does not always occur. Goel & Okumoto's *Imperfect Debugging Model* is like the Jelinski & Moranda model, but assumes that there is a probability p, $0 \leq p \leq 1$, of fixing a bug when it is encountered. This means that after i faults have been found, we expect $i \times p$ faults to have been corrected, instead of i. Thus the failure rate of T_i is

$$r_{T_i}(t \mid N, \Lambda, p) = \Lambda\,(\,N-p(i-1)\,)$$

When p=1 we get the Jelinski & Moranda model.

5. A model by Schick & Wolverton (1978).

This is yet another Type I model, and this time the failure rate is assumed proportional to the number of bugs remaining in the system and the time elapsed since the last failure. Thus

$$r_{T_i}(t \mid N, \Lambda) = \Lambda\,(N\text{-}i+1)t, \qquad t \geq 0$$

This model differs from models 1-4 in that the failure rate does not decrease monotonically. Immediately after the i-th failure, the failure rate drops to 0, and then increases linearly with slope (N−i) until the (i+1)th failure; see Figure 2(iv).

6. Bayesian Differential Debugging Model (Littlewood (1980)).

This model can be considered as an elaboration of model 2 proposed by Littlewood & Verall. Recall that in model 2 it was assumed that Λ_i, the failure rate of the i-th time between failures, was declared to have a gamma distribution. In this new model Littlewood supposed that there were N bugs in the system (a return to the bug counting phenomenon), and then proposed that Λ_i be specified as a function of the remaining bugs. In particular, he stated $\Lambda_i = \phi_1 + \phi_2 + \ldots + \phi_{N\text{-}i}$, where ϕ_i were independent and identically distributed gamma random variables with shape α and scale β. This implied that Λ_i would have a gamma distribution with shape α(N-i) and scale β. In other respects its assumptions are identical to the original Littlewood/Verall model.

7. Bayes Empirical Bayes or Hierarchical Model (Mazzuchi & Soyer (1988)).

In 1988 Mazzuchi and Soyer proposed a *Bayes Empirical Bayes* or *Hierarchical* extension to the Littlewood & Verall model (model 2). As with the original model, they assumed T_i to be exponentially distributed with scale Λ_i. Then they proposed two ideas for describing Λ_i, here called model A and model B.

Model A :
Still assume that Λ_i is described by a gamma distribution, but with parameters α and β. Now assume that α and β are independent and that they themselves are described by probability distributions; α by a uniform and β by another gamma. In other words :

$$\Pi_{\Lambda_i}(\lambda \mid \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}, \qquad \lambda \geq 0$$

$$\pi(\alpha \mid \nu) = \frac{1}{\nu}, \qquad 0 \leq \alpha \leq \nu$$

$$\pi(\beta \mid a, b) = \frac{b^{a}}{\Gamma(a)} \beta^{a-1} e^{-b\beta}, \qquad \beta \geq 0, \ a>0, \ b>0.$$

Model B:

Assume that Λ_i is described exactly as in Littlewood and Verall i.e.

$$\Pi_{\Lambda_i}(\lambda \mid \alpha, \Psi(i)) = \frac{(\Psi(i))^{\alpha}}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\Psi(i)\lambda}, \qquad \lambda \geq 0$$

and that $\Psi(i)=\beta_0+\beta_1 i$, except now place probability distributions on α, β_0 and β_1 as follows:

$$\pi(\alpha \mid \omega) = \frac{1}{\omega}, \qquad 0 \leq \alpha \leq \omega$$

$$\pi(\beta_0 \mid a, b, \beta_1) = \frac{b^{a}}{\Gamma(a)} (\beta_0+\beta_1)^{a-1} e^{-b(\beta_0+\beta_1)}, \qquad \beta_0 \geq -\beta_1, \ a>0, \ b>0$$

$$\pi(\beta_1 \mid c, d) = \frac{d^{c}}{\Gamma(c)} \beta_1^{c-1} e^{-d\beta_1} \qquad \beta_1 \geq 0, \ c>0, \ d>0.$$

So α is described by a uniform distribution, β_0 by a shifted gamma and β_1 by another gamma, and there is dependence between β_0 and β_1. By assuming β_1 to be degenerate at 0, model A is obtained from model B. The authors were able to find an approximation to the expectation of T_{n+1} given that $T_1=t_1$, $T_2=t_2$, ..., $T_n=t_n$, and so use their model to predict future reliability of the software in light of the previous failure times.

8. Time-dependent Error Detection Model (Goel & Okomuto (1979)).

This is the first Type II model that we will consider. It assumes that M(t), the number of failures of the software in time [0,t), is described by a Poisson process with intensity function given by

$$\lambda(t) = ab\, e^{-bt}$$

where a is the total expected number of bugs in the system and b is the fault detection rate; see Figure 2(v). Thus the expected number of failures to time t is :

$$\mu(t) = \int_0^t ab\, e^{-bs}\, ds = a\,(1 - e^{-bt}).$$

The function $\mu(t)$ completely specifies a particular Poisson process, and the distribution of M(t) is given by the well known formula

$$P(M(t)=n) = \frac{(\mu(t))^n}{n!}\, e^{-\mu(t)}, \qquad n=0,1,2,...$$

Experience has shown that often the rate of faults in software increases initially before eventually decreasing, and so in Goel (1983) the model was modified to account for this by letting

$$\lambda(t) = abc\, t^{c-1}\, e^{-bt^c}$$

where a is still the total number of bugs and b and c describe the quality of testing.

9. Logarithmic Poisson Execution Time Model (Musa and Okumoto (1984)).

The *Logarithmic Poisson Execution Time Model* of Musa and Okomuto is one of the more popular software failure models of recent years. It is a type II model, but the model is not derived by directly assuming some intensity function $\lambda(t)$, as was the case with model 8 of Goel & Okumoto. Here $\lambda(t)$ is expressed in terms of $\mu(t)$, the expected number of failures in time [0,t), via the relationship

$$\lambda(t) = \lambda_0\, e^{-\theta\mu(t)}.$$

Put simply, this relationship encapsulates the belief that the intensity (or rate) of failures at time t decreases exponentially with the number of failures experienced, and so bugs fixed, up to time t. The fixing of earlier failures will reduce $\lambda(t)$ more than the fixing of later ones. Since we are modeling the number of failures by a Poisson process, then we have another relationship between $\lambda(t)$ and $\mu(t)$, namely

$$\mu(t) = \int_0^t \lambda(s)\, ds \quad .$$

Using these two relationships between $\lambda(t)$ and $\mu(t)$, there is a unique solution for the two functions:

$$\lambda(t) = \frac{\lambda_0}{\lambda_0\theta t+1} \qquad ; \qquad \mu(t) = \frac{1}{\theta}\ln(\lambda_0\theta t+1) \quad .$$

Figure 2 (vi) shows a plot of $\lambda(t)$ versus t; it is similar to the plot of figure 2 (v) except that the tail is thicker.

It now follows from the above that by using $P(M(t)=n) = (\mu(t))^n e^{-\mu(t)}/n!$ we can say

$$P(M(t)=n) = \frac{(\ln(\lambda_0\theta t+1))^n}{\theta^n\ (\lambda_0\theta t+1)^{1\backslash\theta} \times\ n!}, \qquad n=0,1,2....$$

As a final remark, we mention that in their paper the authors go into some detail on estimation of λ_0 and θ by maximum likelihood methods; however, one of the likelihoods appears to be incorrect.

10. Random Coefficient Autoregressive process model (Singpurwalla & Soyer (1985)).

This is a Type I-2 model, that is one that does not consider the failure rate of times between failure. Instead it assumes that there is some pattern between successive failure times and that this pattern can be described by a functional relationship between them. The authors declare this relationship to be of the form

$$T_i = T_{i-1}^{\theta_i} \quad , \quad i=1,2,3,...$$

where T_0 is the time to the first failure and θ_i is some unknown coefficient. If all the θ_i's are bigger than 1 then we expect successive lifelengths to increase, and if all the θ_i's are smaller than 1 we expect successive lifelengths to decrease.

Uncertainty in the above relationship is expressed via an error term δ_i, so that

$$T_i = \delta_i\ T_{i-1}^{\theta_i}\ .$$

The authors then make the following assumptions, which greatly facilitate the analysis of this model. They assume the T_i's to be lognormally distributed, that is to say that log T_i's have a normal distribution, and that they are all scaled so that $T_i \geq 1$. The δ_i's are also assumed to be lognormal, with median 1 and variance σ_1^2 (the conventional notation is $\Lambda(1,\sigma_1^2)$). Then by taking logs on the relationship above they obtain

$$\log T_i = \theta_i \log T_{i-1} + \log \delta_i$$

$$= \theta_i \log T_{i-1} + \epsilon_i \quad , \text{ say.}$$

Since the T_i's and the δ_i's are lognormal so the log T_i's and the ϵ_i's (= log δ_i's) will be normally distributed, and in particular ϵ_i has mean 0 and some variance σ^2 (the conventional notation is $\mathcal{N}(0,\sigma^2)$). The log-lifelengths therefore form what is known as an *autoregressive process of order 1 with random coefficients* θ_i. There is an extensive literature on such processes which can now be used on this model.

All that remains to do is to specify θ_i, and the authors consider several alternative models. For example, one could make θ_i itself an autoregressive process :

$$\theta_i = \alpha\theta_{i-1} + \omega_i \quad , \qquad \text{where } \omega_i \text{ is } \mathcal{N}(0, W_i) \text{ with } W_i \text{ known.}$$

When α is known, the expressions for log T_i and θ_i together form a *Kalman filter model*, on which there is also an extensive literature. When α is not known the solution is via an *adaptive Kalman filter* algorithm for which the above authors propose an approach. As an alternative to the above, one could place a two stage distribution on θ_i, and the authors considered the idea of θ_i being $\mathcal{N}(\lambda, {\sigma_2}^2)$, with λ also a normal random variable having mean m_0 and variance ${s_0}^2$. In this latter case one can employ standard hierarchical Bayesian inference techniques to predict future reliability in the light of previous failure data.

Figure 2 shows the various failure rates for models 1, 2, 3 and 5, and the intensity function for models 8 and 9.

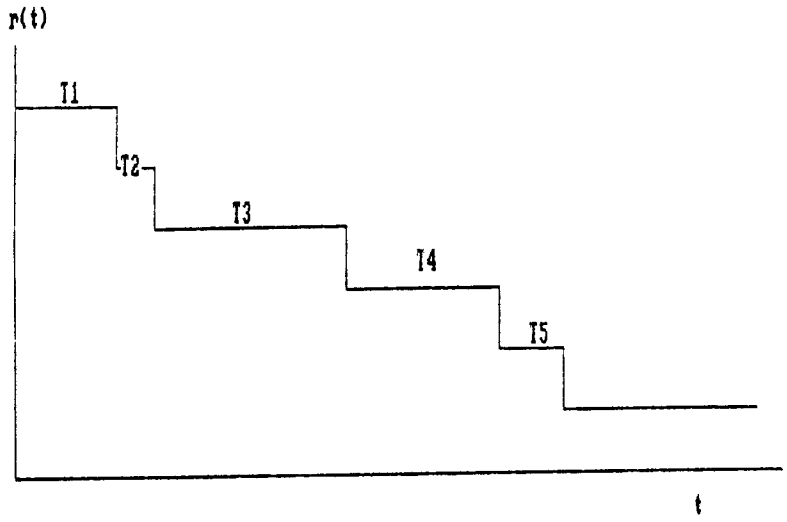


Figure 2 (i) The failure rate of the model of Jelinski and Moranda

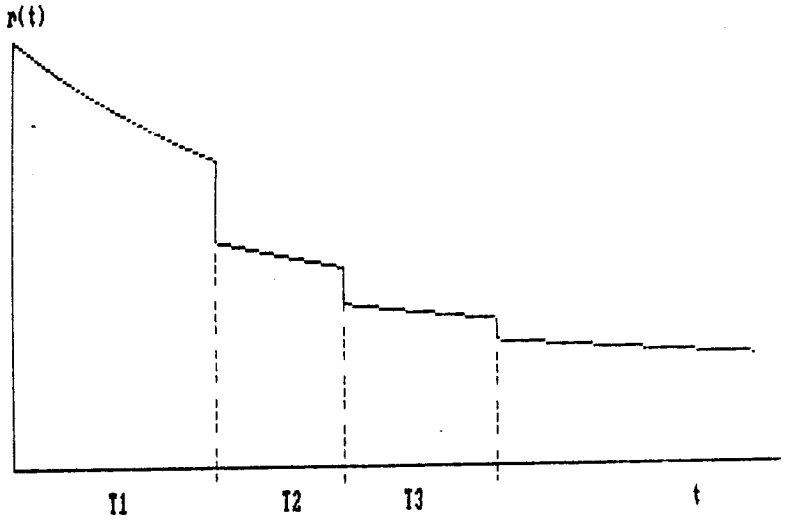


Figure 2 (ii) The failure rate of the model of Littlewood and Verall

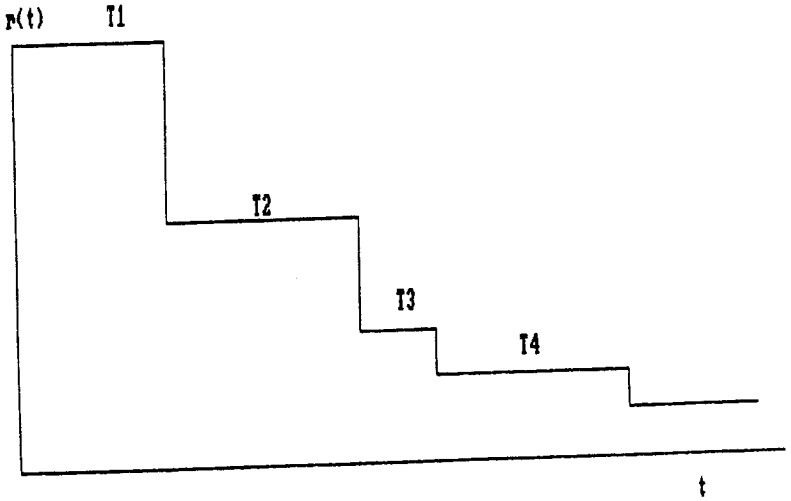


Figure 2 (iii) The failure rate of the model of Moranda

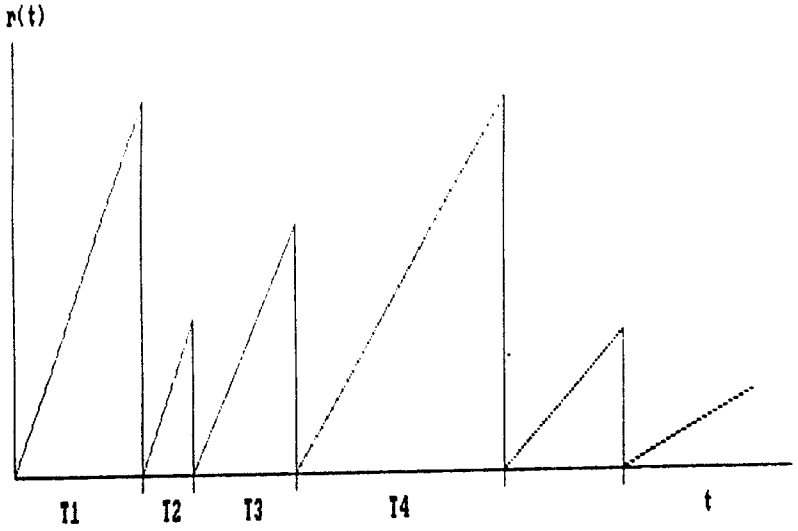


Figure 2 (iv) The failure rate of the model of Schick and Wolverton

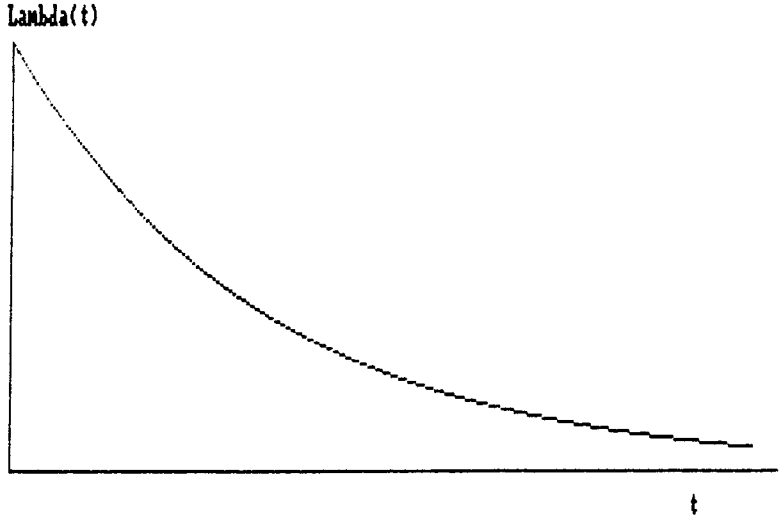


Figure 2 (v) The intensity function for the model of Goel and Okumoto

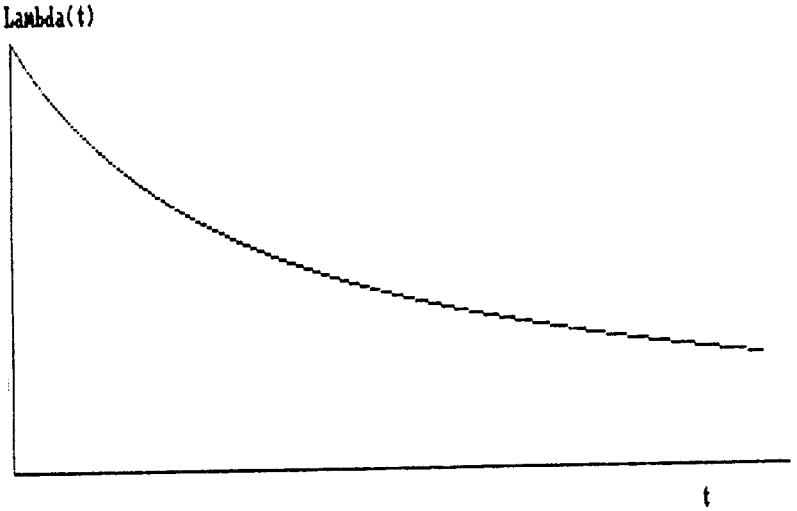


Figure 2 (vi) The intensity function for the model of Musa and Okumoto

4. Model Unification.

By adopting a Bayesian approach, it turns out that one can unify models 1, 2 and 8 - the models by Jelinski & Moranda, Littlewood & Verrall and Goel & Okomuto respectively - under a general framework. Observe that this also provides a link between the two types of models, since models 1 and 2 are of type I whilst model 8 is of type II.

We begin by recalling the first model, that by Jelinski & Moranda. Each T_i is assumed to have a constant failure rate $\Lambda(N-i+1)$. It is well known that this implies each T_i must therefore be exponentially distributed with mean $(\Lambda(N-i+1))^{-1}$. Now assume that Λ and/or N is unknown; in true Bayesian fashion prior distributions are placed upon them.

To obtain model 8 by Goel & Okomuto, we let Λ be degenerate at λ and N have a Poisson distribution with mean θ. One can calculate M(t) using the T_i's as defined by Jelinski & Moranda, and then by averaging out over N one finds that M(t) is indeed a Poisson process with mean :

$$\mu(t) = \theta\ (1\text{-}e^{-\lambda t})$$

which is the form of $\mu(t)$ for Goel & Okomuto's model.

One can also obtain model 2 by assuming N to be degenerate and Λ to have a gamma distribution. The derivations which lead to the above are complex; readers are referred to Langberg and Singpurwalla (1985) for the details.

5. An application : optimal testing of software.

The failure models that have been reviewed in the preceding sections can be used for more than inference or the prediction of software failure. They can also be applied in the framework of decision theory to solve decision problems. An important example of such a problem is the optimal time to test software before releasing it. This involves the balancing of the costs of testing and the risk of software obsolesence with the cost of in-service failure, should a bug not be corrected during the testing period. The following is taken from Singpurwalla (1991), in which a strongly Bayesian approach is taken.

To implement a decision theoretic procedure requires two key ingredients. The first is a probability model, and here we take a generalization of the Jelinski & Moranda model. The second is a consideration of the costs and benefits, or *utilities*, associated with a particular decision i.e the costs of testing, the benefits and costs of fixing a bug etc. Decision theory states that the optimal decision (in this case time of test) is that which *maximizes expected utility.*

If the software is to be tested for some time, say T units, and then released the problem is to find a T that maximizes expected utility. This is called *single stage testing.* There is a more complex, yet realistic, scenario called *two stage testing.* Here the software is tested for T units of time, and then depending on how many failures M(T) were observed during that test, a decision is made on whether to continue testing for a further T^* units. The problem here is to find the optimal T and T^*, with T^* to be determined before M(T) is observed. Finally there is a third testing scenario, namely *sequential testing.* Here T^* is determined after M(T) is observed; this procedure can continue for several stages, with T^{**} being determined after $M(T^*)$ is observed and so on. Here we consider the case of single stage testing. Figure 3 is a graph of the decision process associated with single stage testing.

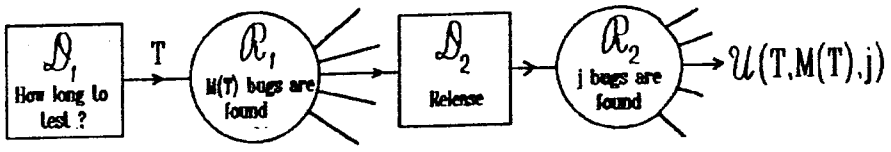


Figure 3 Decision process for single-stage testing

The model chosen in this paper is an extension to Jelinski & Moranda's model. We have

$$f_{T_i}(t \mid N, \Lambda) = \Lambda(N-i+1)\, e^{-\Lambda(N-i+1)t} \qquad t \geq 0$$

In the previous section we placed prior distributions on one of N or Λ. Now we place priors on both the parameters, and say That N has a Poisson distribution with mean θ, Λ has a gamma distribution with scale μ and shape α and that N and Λ are independent.

We now turn to the choice of utility function. The following assumptions are made :

i) The utility of a program that encounters j bugs during its operation is $a_1 + a_2\, e^{-a_3 j}$.
ii) The cost of fixing a bug is some constant C_1.
iii) Let f(T) be the cost of testing and lost opportunity to time t; here we say $f(T) = dT^a$

Note from i) that the utility of a bug-free program is $a_1 + a_2$, and the utility of a program with a very large number of bugs is near a_1, so that typically a_1 is a large negative number (because there is a great loss associated with software that is constantly failing in the marketplace) and $a_2 > 0$. Combining these assumptions gives us the utility of a program that is tested for T units of time, during which M(T) bugs are found and corrected, and then released where j bugs are encountered by the customer as

$$\mathcal{U}(T, M(T), j) = e^{-bT} \times \{a_1 + a_2\, e^{-a_3 j} - C_1 M(T) - dT^a\}$$

where e^{-bT} is some devaluating factor.

Now the two parts of the decision process - the probability model and the utility function - are brought together. We wish to find the time T that maximizes expected utility. In other words find $\hat{T}$ such that $E(\mathcal{U}(T, M(T), j))$ is a maximum, where we take expectation, using our failure model, with respect to M(T) and j. This maximization is quite complex, and must be done numerically via computer. The details are found in the paper, but the end result is best displayed as a graph of time against expected utility (figure 4); in this case one can see that the time one should test the software for is about 3.5 units.

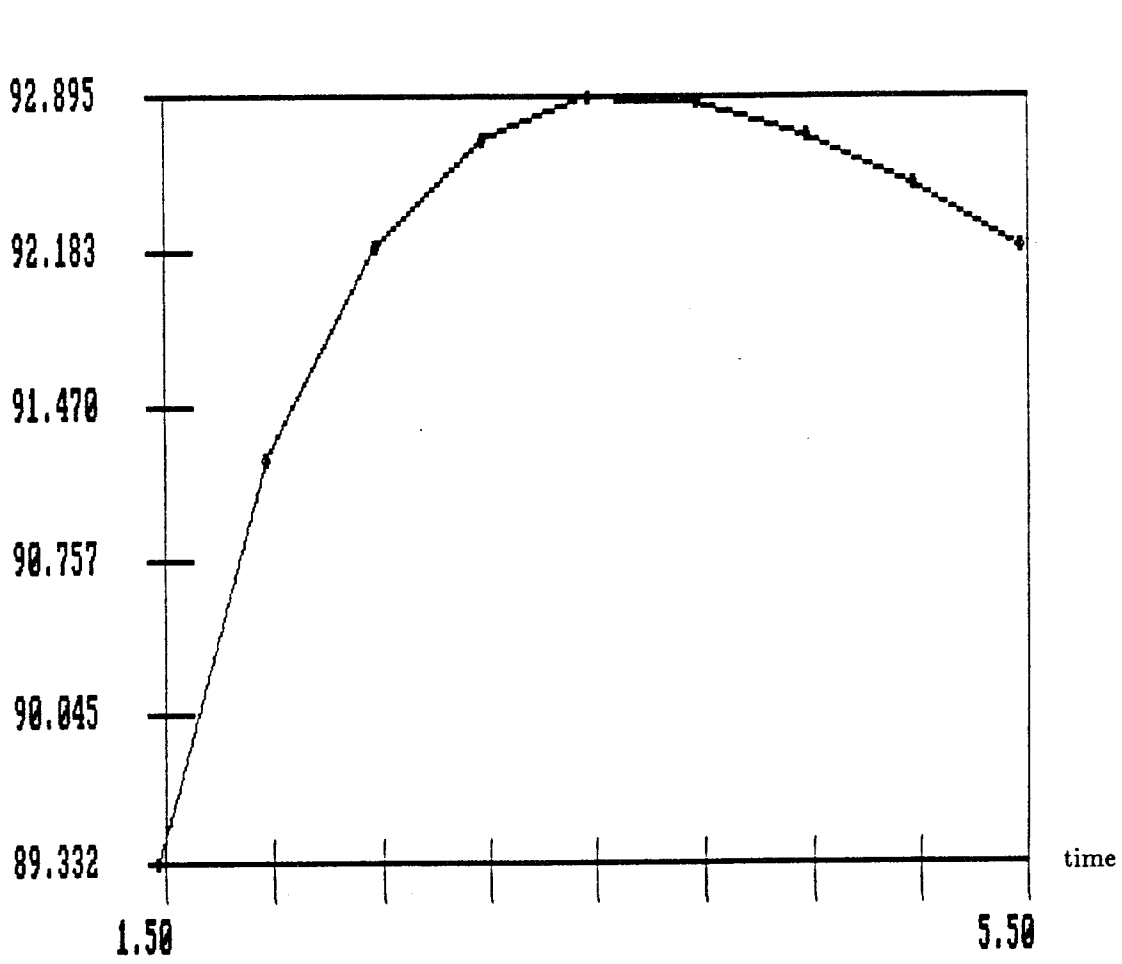


Figure 4 Time of testing versus expected utility for the model in Section 5

6. Conclusion.

This paper has attempted to review the main methods, and some of the more well known models, that have been used by the statistics community in the area of software reliability. The first models were almost always based on looking at the failure rate of the software; later on the idea of modeling number of failures by a Poisson process was used and then most recently auto-regressive processes have been suggested as an alternative to the failure rate method. Application of the failure models, such as to the optimal testing decision problem, is another important aspect to the field.

Earlier it was pointed out that there is almost no data on the reliability of commercial software, due to the sensitive nature of that information. A possible method of overcoming this problem would be to have more interaction between the statistics and computer science communities. In the future, such interaction seems essential if models are to become more realistic and useful, and it is perhaps surprising that there are so few links between the two groups today.

There still remains much to be researched in this field. In the case of optimal testing, plans for two-stage and sequential testing need to be developed, whilst the verification of current and future models is likely to remain a problem. Nevertheless, because of the increasing presence of computers in all aspects of our daily lives, the topic of software reliability can only become more important in the future.

Acknowledgements

This report is based on a series of lectures given by the first author during the Fall of 1991 when he was the C. C. Garvin Endowed Visiting Professor of Computer Science and of Statistics at the Virginia Polytechnic Institute and State University. The comments of Professors I. J. Good, R. Krutchkoff, P. Palletas and K. Hinkelmann, and of students in the class, are gratefully acknowledged. Also acknowledged are the inputs of Ms. Sylvia Campodonico and the assistance of Mr. Jingxian Chen.

References

Goel, A.L. (1983) A Guidebook for Software Reliability Assessment. Technical Report RADC-TR-83-176

Goel, A.L. & Okomuto, K. (1978) An Analysis of Recurrent Software Failures on a Real-time Control System. Proc. ACM Annu. Tech. Conf., ACM, Washington D.C., pp. 496-500

Goel, A.L. & Okomuto, K. (1979) Time-dependent Error Detection Rate Model for Software Reliability and other Performance Measures. IEEE Transactions on Reliability, vol. R-28, pp. 206-211

Jelinski, Z. & Moranda, P. (1972) Software Reliability Research. Statistical Computer Performance Evaluation, W. Freiberger editor. New York: Academic, pp. 465-484

Landberg, N. and Singpurwalla, N.D. (1985) A Unification of Some Software Reliability Models. SIAM J. Sci. Stat. Comput., vol. 6, no. 3, pp.781-790

Littlewood. B. (1980) A Bayesian Differential Debugging Model for Software Reliability. Proceedings of IEEE COMPSAC

Littlewood, B. & Verall, J.L. (1973) A Bayesian Reliability Growth Model for Computer Software. Applied Statistician, vol. 22, pp. 332-346

Mazzuchi & Soyer (1988) A Bayes Empirical-Bayes Model for Software Reliability. IEEE Transactions on Reliability, vol. 37, no. 2, pp. 248-254

Moranda, P.B. (1975) Prediction of Software Reliability and its Applications. Proceedings of the Annual Reliability and Maintainability Symposium, Washington D.C., pp. 327-332

Musa, J.D. & Okumoto, K. (1984) A Logarithmic Poisson Execution Time Model for Software Reliability Measurement. Proceedings of the 7th International Conference on Software Engineering, Orlando, Florida, pp. 230-237

Schick, G.J. & Wolverton, R.W. (1978) Assessment of Software Reliability. Proc. Oper. Res., Physica-Verlag, Wirzberg-Wien, pp. 395-422

Singpurwalla, N.D. (1991) Determining an Optimal Time for Testing and Debugging Software. IEEE Transactions on Software Engineering, vol. 17, no. 4, pp. 313-319

Singpurwalla, N.D. and Soyer, R. (1985) Assessing (Software) Reliability Growth Using a Random Coefficient Autoregressive Process and its Ramifications. IEEE Transactions on Software Engineering, vol. SE-11, no. 12, pp. 1456-1464