Statistical Modelling of Software Reliability

Semi-Annual Status Report No. 1

1 April 1991 through 30 September 1991

National Aeronautics and Space Administration
Grant NAG 1-1241

Douglas R. Miller
Principal Investigator

Department of Operations Research and Applied Statistics
School of Information Technology and Engineering
George Mason University
Fairfax, VA 20030-4444

NASA Grant NAG 1-1241 commenced on 1 April 1991. During the six-month period from 1 April 1991 to 30 September 1991 the following research work in statistical modelling of software reliability appeared:

[1] A. Sofer and D. R. Miller, "A Nonparametric Software Reliability Growth Model," **IEEE Transactions on Software Engineering 40** (1991): 329-337.

[2] P. A. Keiller and D. R. Miller, "On the Use and the Performance of Software Reliability Growth Models," **Reliability Engineering and System Safety 32** (1991): 95-117.

[3] M. Lyu, H. Hecht, H. Kopetz, D. Miller, J. Musa, M. Ohba, and D. Siefert, "Research and Development Issues in Software Reliability Engineering," **Proceedings of the IEEE International Symposium on Software Reliability Engineering** (1991); 80-89. (Reprinted in **Software Engineering Notes 16,2** (1991): 23-30.)

[4] B. Littlewood and D. Miller (guest co-editors), Special Issues on Software, **Reliability Engineering and System Safety, Volume 32, Numbers 1 and 2** (1991).

[5] B. Littlewood and D. Miller (co-editors), Software Reliability and Safety, Elsevier Applied Science, London, 1991.

The above work was started with the support of NASA Grant NAG 1-771 and was continued in the hope of receiving additional NASA support. That support materialized as NASA Grant NAG 1-1241. Final preparations for publication of some of the above works were completed during the period covered by this Semi-Annual Status Report.

Note that items [4] and [5] are essentially the same work in different formats. Elsevier decided that it wanted to publish the papers in the special issues [4] as a separate hardbound volume [5] in order to reach a wider audience.

Copies of [1], [2], and [3] are attached to this report as Appendices 1, 2, and 3, respectively.

Appendix 1

A. Sofer and D. R. Miller, "A Nonparametric Software Reliability Growth Model," **IEEE Transactions on Software Engineering 40** (1991): 329-337.

A Nonparametric Software-Reliability Growth Model

Ariela Sofer
George Mason University, Fairfax
Douglas R. Miller
George Mason University, Fairfax

Key Words — Software reliability, Complete monotonicity, Nonparametric regression, Failure rate estimation, Failure rate extrapolation

Reader Aids —
Purpose: Present a general model
Special math needed for explanations: Statistics and linear algebra
Special math needed to use the results: Statistics
Results useful to: Software reliability theorists and analysts

Abstract — Miller & Sofer previously presented a new nonparametric method for estimating the failure rate of a software program. The method is based on the complete monotonicity property of the failure rate, and uses regression to estimate the current software-failure rate. This paper extends this completely monotone model and demonstrates how it can also provide longer-range predictions of reliability growth. Preliminary evaluation indicates that the method is competitive with parametric approaches, while being more robust.

1. INTRODUCTION

Suppose a program is executed for a length of time T. During this time, n bugs are detected and removed when they manifest themselves as failures. The successive failures occur at times:

$$0 < t_1 < t_2 < \ldots < t_n < T. \tag{1}$$

When bugs are corrected without introducing new faults, the program evolves into a more reliable program, hence the term "reliability growth". Given the past software data (1) we want to make various statistical inferences concerning the current and future reliability of the software. In particular we are interested in —

- the number of failures anticipated over some future horizon
- the future failure rate after an additional specified time of debugging.

Over the years, many competing models for software-reliability growth have been developed, eg, Duane [6], Jelinski & Moranda [8], Goel & Okumoto [7], Littlewood [9], and Musa & Okumoto [16]. These are all parametric models, and have a common property: complete monotonicity of the failure rate.

Notation

$N(t)$	(random) number of failures observed in $[0,t]$
$M(t)$	$E\{N(t)\}$, mean number of failures, viz, the mean function
$r(t)$	$dM(t)/dt, 0<t$, intensity function of the point process $\{N(t), 0 \le t\}$

The $r(t)$ is also referred to as the *failure rate* of the process, although *failure intensity* is probably a better name. A function $r(\cdot)$ is completely monotone if and only if it has derivatives of all orders, and they alternate in sign:

$$(-1)^q \frac{d^q r(t)}{dt^q} \ge 0,\ t \ge 0,\ q = 0,1,2,\ldots. \tag{2}$$

Miller [12] has shown that software under random time-homogeneous testing or usage with perfect fixes shows completely monotone reliability-growth, and conversely that virtually all completely monotone functions can occur as intensity functions of reliability-growth point processes. Thus, a general approach to software-reliability growth modelling should include the entire class of completely monotone intensities. Reliability-growth prediction based on a single parametric family of reliability-growth processes cannot be justified.

Miller & Sofer [13] previously introduced a nonparametric model for software-reliability growth which is based on complete monotonicity of the failure rate. The method uses regression to estimate the current software failure rate. Miller & Sofer [14] show that this method often gives estimates which have a lower s-bias than those of certain (widely-used) parametric methods; using Monte Carlo simulated failure data, these "completely monotone regression" estimates of current failure rate are also shown to be more robust than the estimates based on parametric models.

Chan [4] has estimated the distribution of time-until-next-failure for real data using completely monotone regression estimates of current reliability. He starts with a raw estimate which is an exponential distribution with the estimated current failure rate and then "adapts" it to a more general distribution using the procedure of Littlewood & Keiller [10]. Chan then evaluates these estimates using criteria of Abdel-Ghaly, Chan, Littlewood [1]. The Chan study shows that completely monotone regression gives good estimates that are more robust than estimates from parametric models.

This paper extends the completely monotone software model by developing a method for providing long-range predictions of reliability growth, based on the model. The paper derives upper and lower bounds on extrapolations of the failure rate and the mean function. These are then used to obtain estimates for the future software failure rate and the mean future number of failures.

0018-9529/91/0800-0329$01.00©1991 IEEE

2. NOTATION, DEFINITIONS, ASSUMPTIONS

Notation

a	second order difference defined in (17)
d	order of the highest difference constraint (7)
$D(x,y)$	weighted squared distance between vectors x and y
i,j	dummy indices
k	number of discrete time subintervals over the observation interval $[0,T]$
l	number of subintervals over the future prediction horizon
$M(t)$	mean number of failures observed in the interval $[0,t]$
$\hat{M}(t)$	raw piecewise linear estimate of $M(t)$
$\hat{m}_i$	$\hat{M}(s_i)$
m_i	smoothed least squares estimate of $M(s_i)$
n	number of observed failures over the interval $[0,T]$
$N(t)$	random number of failures occuring in the interval $[0,t]$
$\{N(t),0\le t\}$	stochastic point process of the number of failures
p	defined in (9) in terms of the failure rate, and in (24) in terms of the mean function
$P(j)$	an instance of a third order extrapolation of the failure rate — for j subintervals into the future
q	defined in (12) in terms of the failure rate, and in (27) in terms of the mean function
$r(t)$	$dM(t)/dt$, failure rate at time t,
$\hat{r}_i$	raw estimate of the failure rate over the subinterval i
r_i	smoothed estimate (least squares estimate satisfying completely monotone constraints) of the failure rate over subinterval i
s_i	$i\theta$, the end point of subinterval i
t	time
T	length of the observation interval
t_i	occurence time of failure i
u	defined in (15) in terms of the failure rate and in (30) in terms of the mean function
v	defined in (22)
w_i	weight assigned to subinterval i in the least squares equation
δ	adjustment to the number of failures in subinterval k, see (8)
Δ^j	order j backward difference operator see (5)
θ	T/k, length of each of the k subintervals of $[0,T]$; also the length of the l subintervals in the future prediction horizon

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Definition

A positive function $r(\cdot)$ is *completely monotone* if and only if it has derivatives of all orders, and they alternate in sign, see (2).

Assumptions

1. Parametric models are an approximation to the software-reliability growth process. In general, there is no "correct" parametric reliability-growth model. While a parametric model might work well on some failure-data sets, it might also give bad predictions for other data sets [1].

2. The "goodness of fit" to observed data and "quality of prediction" of future failure behavior are two distinct (not necessarily equivalent) properties of reliability growth models [1].

Background Theory

1. Under very general conditions, if the software usage is random and time homogeneous, and if faults are fixed immediately and perfectly, then the reliability growth process has a completely monotone intensity [12].

2. Conversely, virtually all completely monotone functions can occur as the failure rate of reliability-growth processes [12].

3. PROBLEM FORMULATION

Consider the failure data as in (1). Our goal is to find a completely monotone rate function and/or the associated mean function which best fits the data. The mean function does not strictly satisfy the complete monotonicity property; rather, $M(t)$ is a nonegative function whose derivative $dM(t)/dt$ is a completely monotone function. Our approach is to obtain an initial raw estimate for the required function from the data, and then to smooth it by fitting a completely monotonic function which is closest to it in the least squares sense.

A reasonable raw estimate $\hat{M}(t)$ for the mean function is a piecewise linear function with breakpoints at t_i, $i=1,\ldots,n$, such that $\hat{M}(t_i)=i$:

$$\hat{M}(t) = \begin{cases} i+(t-t_i)/(t_{i+1}-t_i) & t_i\le t\le t_{i+1};\ i=0,\ldots,n-1 \\ n+\delta(t-t_n)/(T-t_n) & t_n\le t\le T. \end{cases} \tag{3}$$

The second term in the final interval reflects the absence of a failure in the period $(t_n,T]$. The choice of δ is somewhat arbitrary, with higher values tending to give more conservative estimates. In this work we consider values of 0.0, 0.5, 1.0 for δ; however one can argue for and against any particular value.

In practice, it is necessary to discretize the problem of finding a completely monotone function to the mathematically more tractable problem of finding a finite set of points along that function. The most plausible and straightforward approach is to consider discrete time points which are equally spaced. We thus divide the time interval $[0,T]$ into k intervals of equal length $\theta=T/k$, and define $s_i=i\theta$, $i=0,\ldots,k$. Thus the sequence $\hat{m}_i=\hat{M}(s_i)$ is an initial estimator for the values of the mean function at the fixed intervals s_i. In general, however, this sequence does not satisfy the complete monotonicity assumptions of the model, and thus needs some modification.

For the problem of estimating the rate function, we obtain an initial estimator from the slope of $M(t)$. Specifically, the sequence —

$$\hat{r}_i = (\hat{m}_i - \hat{m}_{i-1})/\theta;\ i=1,\ldots,k$$

is a raw estimate of the failure rate at the points s_i.

When working with discrete, equally spaced time points, the analogue of a completely monotone function is a completely monotone sequence. The sequence $(r_i,\ i=1,2,\ldots)$ is completely monotone if —

$$(-1)^j\ \Delta^j\ r_i \geq 0,\ j+1 \leq i;\ j=0,1,\ldots \tag{4}$$

where Δ^j is the order j backward difference operator:

$$\Delta^0\ r_i = r_i,\ \Delta^1\ r_i = r_i - r_{i-1},\ \Delta^j\ r_i = \Delta^{j-1}\ r_i - \Delta^{j-1}\ r_{i-1},\ j>1. \tag{5}$$

In general, the initial estimate $(\hat{r}_1,\ldots,\hat{r}_k)$ does not have the complete monotonicity property. Our goal is to find the "closest" completely monotone sequence $(r_1,\ldots,r_k)$, and use it as an estimate of the sequence of failure rates at times s_i. Using the criterion of weighted least squares, the problem is to find a vector r which minimizes —

$$D(r,\hat{r}) = \sum_{i=1}^{k} w_i(r_i - \hat{r}_i)^2 \tag{6}$$

subject to the complete monotonicity constraints of (4), where w_i is a set of prespecified weights.

Numerical experience indicates that the effect of the very high order difference constraints on the optimal solution is at most marginal; moreover, their presence leads to ill-conditioning of the optimization problem. Consequently we relax the constraints in (4) and consider differences of at most d (not ∞), with d being typically 3 or 4. Similarly, it is necessary to constrain the sequence infinitely far into the future; we restrict the number of future intervals to l, rather than ∞. Finally, many of the constraints in (4) are redundant, eg, $\Delta r_{i-1} \leq 0$ and $\Delta^2\ r_i \geq 0$ imply that $\Delta\ r_i \geq 0$. Eliminating those redundant constraints, we obtain the reduced system of equations —

$$\begin{aligned} &(-1)^d \Delta^d\ r_i \geq 0, && d+1 \leq i \leq k+l \\ &(-1)^j \Delta^j\ r_{k+l} \geq 0, && 0 \leq j \leq d-1, \end{aligned} \tag{7}$$

and our problem is to minimize (6) subject to (7).

For $d=1$, the problem is the well known "isotone regression" (Barlow, et al [2]) and addressed in the reliability-growth context by Campbell & Ott [3], and Nagel, et al [17]. If the last interfailure happens to come from the right tail of the interfailure time distribution, $\hat{r}_k$ underestimates $r(T)$, and the monotone constraint on r has no effect; thereby leading to a negative bias. Imposing the additional constraint of convexity tends to pull this estimate up. In most software-reliability applications, a positively biased estimate of the failure rate is safer than a negatively biased estimate; thus, higher order constraints seem to be desirable, and the generalization of isotone regression to completely monotone regression is an improvement.

Return to the problem of estimating the mean function. Its first order derivative is completely monotone, and using the above, our problem is:

$$\min \quad D(m,\hat{m}) = \sum_{i=1}^{k} w_i(m_i - \hat{m}_i)^2$$

$$\begin{aligned} \text{subject to } &(-1)^{d+1}\Delta^d\ m_i \geq 0, && d \leq i \leq k+l \\ &(-1)^j \Delta^j\ m_{k+l} \geq 0, && 0 \leq j \leq d-1 \\ &m_k \equiv n+\delta,\ k>0 \\ &m_0 \equiv 0. \end{aligned} \tag{8}$$

If testing stopped at a failure, $(t_n = T)$, then $\delta = 0$. For truncated testing however, $\delta = 0.5$ is more plausible. Using an argument based on the assumption of Poisson process, $\delta = 1$ is also a plausible choice.

The optimization problems in this section are linearly constrained quadratic programming problems, and algorithms for their solution are readily available in the literature. However, our particular problem of least squares regression under higher order difference constraints becomes increasingly ill-conditioned as the problem size grows [15]. Thus, a numerically stable algorithm should be employed for its solution. For a detailed description of a viable solution approach, see [15].

An additional difficulty when attempting to include monotonicity requirements into the future, is that the Hessian matrix (the matrix of the second order derivatives of the objective function) is singular since the predictions r_i and m_i (where $i=k+1,\ldots,k+l$) do not appear in the objective. Moreover, the optimal future rate or mean estimators obtained by the least squares objective are not unique. Section 4 shows how to overcome the problem of singularity, by reformulating the constraints on the future rates (or mean function estimates) in terms of those of the past. Surprisingly, this approach also provides bounds — lower and upper envelopes for these future estimates.

4. PREDICTIONS

Formulation (8) gives rise to some computational problems, when predictions are requested, ie, when $l>0$. Algorithms for solving quadratic programming problems [11] usually require that the Hessian matrix of the objective function be positive-definite. However, the Hessian matrix of the objective for (6), (ie, diag $(w_1,\ldots,w_k)$) is only positive semi-definite, and does have singularities. As a result, not only do we encounter numerical difficulties when trying to solve the problem directly, but the optimal solution is not unique. Indeed, any two solution vectors where the first k components are equal, yield exactly the same objective value. In other words, if the completely monotone sequence $(r_1,\ldots,r_k)$ can be extrapolated l time intervals into the future, in a way that the resulting sequence $(r_1,\ldots,r_{k+l})$ is completely monotone, then *all* such possible extrapolations have the same least squares objective. We show, that among all such extrapolations, there exist a globally highest and a globally lowest extrapolation, and all other completely monotone extrapolations into the future must lie in between the

highest and lowest bounds. We thus have an envelope in which all completely monotone extrapolations are restricted. In addition, we derive the conditions under which the sequence $(r_1,\ldots,r_k)$ can be extrapolated as a completely monotone sequence into the future.

Consider the completely monotone sequence of order d: $R=(r_1,\ldots,r_k)$. The sequence $(r_{k+1},\ldots,r_{k+l})$ is defined to be a feasible completely monotone extrapolation of order d for R, if the sequence $(r_1,\ldots,r_{k+l})$ is completely monotone up to order d, ie, it satisfies (7). In addition, this extrapolation constitutes an upper bound for all feasible extrapolations of order d, if any other such extrapolation, $(\bar{r}_{k+1},\ldots,\bar{r}_{k+l})$ satisfies $\bar{r}_{k+i}\le r_{k+i}$ for $i=1,\ldots,l$. Similarly it constitutes a lower envelope if $\bar{r}_{k+i}\ge r_{k+i}$ for all i. We derive conditions for the existence for such higher and lower envelopes for the completely monotone extrapolations.

For $d=1$ and $d=2$, the sequence $(r_1,\ldots,r_k)$ can be extraplated into the future by letting $r_{k+i}=r_k$, $i=1,\ldots,l$. This extrapolation is clearly the upper envelope for all completely monotone extrapolations of order 1 and 2, and is always feasible. Also for $d=1$, the extrapolation $r_{k+i}=0$ is clearly a lower envelope for all isotone extrapolations. The next proposition shows, that the lower envelope for feasible extrapolations of order $d=2$ is along a piecewise linear function which has slope $\Delta^1 r_k$, until it reaches zero, after which it continues as a constant function zero. We define

$$p = \begin{cases} \text{gilb}(-r_k/\Delta^1 r_k) & \text{,if } \Delta^1 r_k>0 \\ l & \text{,if } \Delta^1 r_k=0 \end{cases} \tag{9}$$

Proposition 1. Consider the constraints (7) with $d=2$ and fixed $l>0$, and let $(r_1,\ldots,r_k)$ be a feasible solution to (7) with $l=0$. Then the extrapolation

$$r_{k+i} = \begin{cases} r_k+i\Delta^1 r_k & i=1,\ldots,p \\ 0 & i=p+1,\ldots,l \end{cases} \tag{10}$$

is a lower envelope for all feasible extrapolations of order 2 to $(r_1,\ldots,r_k)$.

Proof: The solution above is clearly monotone, and $\Delta^2 r_{k+i}=0$ for $i=1,\ldots,p$ and $i=p+3,\ldots,l$. In addition, $\Delta^2 r_{k+p+1}=-(r_k+(p+1)\Delta^1 r_k)$ and $\Delta^2 r_{k+p+2}=r_k+p\Delta^1 r_k$, which, by definition of p are both nonnegative. Thus the constraints of (7) for $d=2$ are satisfied. Note also, that for any other feasible extrapolation $(\bar{r}_{k+1},\ldots,\bar{r}_{k+l})$ we have —

$$\Delta^1 \bar{r}_{k+j}\ge\Delta^1 r_k.$$

Thus, if $i\le p$ then —

$$\bar{r}_{k+i} = r_k + \sum_{j=1}^{i} \Delta^1 \bar{r}_{k+j} \ge r_k+i\Delta^1 r_k = r_{k+i}.$$

It follows that (10) is a lower envelope as proposed. *Q.E.D.*

Proposition 2. Consider the constraints (7) with $d=3$ and fixed $l>0$. A solution $(r_1,\ldots,r_k)$ which satisfies (7) with $l=0$ can be extrapolated to a vector $(r_1,\ldots,r_{k+l})$ which satisfies (7) with $l>0$ if and only if:

$$r_k+j\Delta^1 r_k+\frac{1}{2}j(j+1)\Delta^2 r_k\ge 0;\ j=1,\ldots,l. \tag{11}$$

In addition, let —

$$q = \begin{cases} \text{gilb}(-\Delta^1 r_k/\Delta^2 r_k) & \text{if } \Delta^2 r_k>0 \\ l & \text{if } \Delta^2 r_k=0. \end{cases}$$

Then the upper envelope of all feasible extrapolations for $d=3$ is:

$$r_{k+i} = \begin{cases} r_k+i\Delta^1 r_k+\tfrac{1}{2}i(i+1)\Delta^2 r_k & i=1,\ldots,q \\ r_{k+q} & i=q+1,\ldots,l. \end{cases}$$

Proof: Any feasible extrapolation satisfies:

$$\begin{aligned} r_{k+i} &= r_k + \sum_{j=1}^{i} \Delta^1 r_{k+j} \\ &= r_k + \sum_{j=1}^{i}\left(\Delta^1 r_k + \sum_{h=1}^{j}\Delta^2 r_{k+h}\right) \\ &= r_k + i\Delta^1 r_k + \sum_{j=1}^{i}\sum_{h=1}^{j}\Delta^2 r_{k+h} \\ &\le r_k + i\Delta^1 r_k + \frac{1}{2}i(i+1)\Delta^2 r_k \end{aligned}$$

and the nonnegativity of r_{k+i} implies that (11) must hold.

Conversely, assume that (11) holds. Now since $\{r_i\}$ is completely monotone of order 3, the sequence $\{-\Delta^1 r_i\}$ is completely monotone with order $d=2$. Using proposition 1 for the lowest feasible convex extrapolation for $\{-\Delta^1 r_i\}$, we obtain the upper envelope for completely monotone extrapolations of $(r_1,\ldots,r_k)$ of order 3. *Q.E.D.*

Proposition 3. Consider the constraints (7) with $d=3$ and fixed $l>0$, and let $(r_1,\ldots,r_k)$ be a solution to (7) with $l=0$ satisfying (11). Let p be defined as in (9).

(a). If $p\ge l$, then the extrapolation —

$$r_{k+i} = r_k + i\Delta^1 r_k,\ i=1,\ldots,p$$

is a lower envelope for all feasible extrapolations of order 3 to $(r_1,\ldots,r_k)$.

(b). If $p\le l$, let —

$$u = \min(l,1+\text{gilb}(-2r_k/\Delta^1 r_k)).$$

Then the extrapolation —

$$r_{k+i} = \begin{cases} r_k + i\Delta^1 r_k + \frac{1}{2} i(i+1)\left(\frac{-2(r_k + u\Delta^1 r_k)}{u(u+1)}\right) & i = 1,\ldots,u \\ 0 & i = u+1,\ldots,l \end{cases} \quad (12)$$

is a lower envelope for all feasible extrapolations of order 3 to $(r_1,\ldots,r_k)$.

The proposition states that the lowest envelope is a linear function with slope $\Delta^1 r_k$, provided that such a linear function is feasible (nonnegative); otherwise it starts as a quadratic function with constant second order difference

$$a = -\left(\frac{2(r_k + u\Delta^1 r_k)}{u(u+1)}\right)$$

which flattens to zero at r_{k+u}, and from there continues as zero.

Proof: If $p \geq l$ then (10) is a feasible extrapolation of order 3, thus a follows from proposition 1. If $p \leq l$ then (10) does not satisfy the third order difference constraints. We now show that for this case, the function of (12) is a lower envelope for any feasible extrapolation. First assume that there exists a feasible extrapolation $\bar{r}_{k+1}\ldots\bar{r}_{k+l}$ for which $\Delta^2 \bar{r}_{k+1} < a$. Then

$$\bar{r}_{k+1} + (u-1)\Delta^1 \bar{r}_{k+1} + \frac{1}{2}(u-1)u\Delta^2 \bar{r}_{k+1}$$

$$\leq r_k + u\Delta^1 r_k + \frac{1}{2}u(u+1)\Delta^2 \bar{r}_{k+1}$$

$$= r_k + u\Delta^1 r_k - (r_k + u\Delta^1 r_k) = 0,$$

in contradiction to the conditions given by proposition 2, for a feasible extrapolation for $r_1,\ldots,r_k, \bar{r}_{k+1}$. We therefore conclude that any feasible extrapolation has a second order difference of at least a. If, on the other hand, $\Delta^2 \bar{r}_{k+1} > a$, then $\bar{r}_{k+1} > r_{k+1}$. An inductive argument starting from r_{k+1} completes the proof. *Q.E.D.*

Proposition 4. Consider the constraints (7) with $d=4$ and fixed $l>0$. A solution $(r_1,\ldots,r_k)$ which satisfies (7) with $l=0$ can be extrapolated to a vector $(r_1,\ldots,r_{k+l})$ which satisfies (7) with $l>0$ if and only if —

$$\Delta^1 r_k + j\Delta^2 r_k + \frac{1}{2}j(j+1)\Delta^3 r_k \leq 0,\ j = 1,\ldots,l \quad (13)$$

$$r_k + l\Delta^1 r_k + \frac{1}{2}l(l+1)\Delta^2 r_k \geq 0, \quad (14)$$

$$r_k + \frac{2}{3}(j-1)\Delta^1 r_k + \frac{1}{6}j(j-1)\Delta^2 r_k \geq 0,\ j = 1,\ldots,l. \quad (15)$$

If $\Delta^1 r_k + l\Delta^2 r_k \leq 0$ then the upper envelope of all such extrapolations is:

$$r_{k+i} = r_k + i\Delta^1 r_k + \frac{1}{2}i(i+1)\Delta^2 r_k,\ i = 1,\ldots,p. \quad (16)$$

Otherwise, let —

$$v = \min(l, 1 + \text{gilb}(-2\Delta^1 r_k / \Delta^2 r_k)).$$

Then the upper envelope of all such extrapolations is:

$$r_{k+i} = \begin{cases} r_k + i\Delta^1 r_k + \frac{1}{2}i(i+1)\Delta^2 r_k + \frac{1}{6}i(i+1)(i+2) \\ \quad \left(\frac{-2(\Delta^1 r_k + v\Delta^2 r_k)}{v(v+1)}\right) & i = 1,\ldots,v \\ r_{k+v} & i = v+1,\ldots,l. \end{cases} \quad (17)$$

Proof: If the sequence $\{r_{k+i}\}$ is a feasible extrapolation of order $d=4$ then the sequence $\{-\Delta^1 r_{k+i}\}$ is a feasible extrapolation of order $d=3$. By Proposition 2, the conditions for existence of the latter are given by (13). In addition, the upper envelope of all extrapolations for $d=4$ is the sequence $\{r_{k+i}\}$ for which $\{-\Delta^1 r_{k+i}\}$ constitutes the lower envelope of all extrapolations of order $d=3$. Applying Proposition 3 with respect to the sequence $\{-\Delta^1 r_{k+i}\}$ and integrating over this lower envelope yields the sequence of (16) and (17). Note that by construction, the resulting sequence is nonincreasing, convex with nonpositive third order difference. It remains to determine the conditions under which this sequence is nonnegative. First, we note that condition (14) guarantees that (16) will be nonnegative. From Proposition 2 this is also a necessary condition. Also conditions (15) guarantee that r_{k+v} is nonnegative for any possible value of v between 1 and l. Since (17) represents a decreasing function which becomes constant for $i \geq v$, this guarantees that r_{k+i} is also nonnegative for any i. To show that conditions (15) are also necessary, define

$$P(j) = r_k + \frac{2}{3}(j-1)\Delta^1 r_k + \frac{1}{6}j(j-1)\Delta^2 r_k.$$

It is easy to see that $P(j)$ decreases for $j = 1,\ldots,v$ and increases for $j = v,..,l$. Suppose that (15) is violated for some j. Let $\bar{j}$ be the smallest index to violate this condition. It follows that $\bar{j} \leq v$ and that $P(v) \leq 0$. This in turn implies that $r_{k+v} \leq 0$, and thus no feasible extrapolation with $d=4$ is feasible, hence a contradiction. *Q.E.D.*

We now derive the envelopes for prediction for the mean function. Consider a sequence of order d: $M = (m_1,\ldots,m_k)$ which satisfies (8). The sequence $(m_{k+1},\ldots,m_{k+l})$ is defined to be a feasible extrapolation of order d for M, if the sequence $(m_1,\ldots,m_{k+l})$ satisfies (8). In addition, this extrapolation constitutes an upper bound for all feasible extrapolations of order d, if any other such extrapolation $(\bar{m}_{k+1},\ldots,\bar{m}_{k+l})$ satisfies $\bar{m}_{k+i} \leq m_{k+i}$ for $i = 1,\ldots,l$. Similarly it constitutes a lower envelope if $\bar{m}_{k+i} \geq m_{k+i}$ for $i = 1,\ldots,l$.

We derive conditions for the existence for such higher and lower envelopes for the feasible extrapolations for M. The derivative of the mean function is completely monotone. Therefore, the lower and upper bounds for all feasible extrapolations of order d to $m_1,\ldots,m_k$ are obtained by integrating respectively over the lower and upper bounds for all feasible extrapolations of order $d-1$ to $\Delta m_1,\ldots,\Delta m_k$.

Consequently, for the case $d=1$, $d=2$ and $d=3$, the sequence $(m_1,\ldots,m_k)$ can always be extrapolated into the future. The upper envelope for all feasible extrapolations of order up to 3 is the linear function:

$$m_{k+i} = m_k + i\Delta^1 m_k.$$

For $d=1$ and $d=2$ the extrapolation $m_{k+i} = m_k$ is clearly a lower envelope for all feasible extrapolations. Proposition 5 shows, that the lower envelope for feasible extrapolations of order $d=3$ is along a quadratic which tapers off to a constant function.

Proposition 5. Consider the constraints (8) with $d=3$ and fixed $l>0$, and let $(m_1,\ldots,m_k)$ be a feasible solution to (8) with $l=0$. Let —

$$p = \begin{cases} \text{gilb}(-\Delta^1 m_k/\Delta^2 m_k), & \text{if } \Delta^2 m_k > 0 \\ l, & \text{if } \Delta^2 m_k = 0 \end{cases} \tag{18}$$

Then the extrapolation —

$$m_{k+i} = \begin{cases} m_k + i\Delta^1 m_k + \tfrac{1}{2} i(i+1)\Delta^2 m_k, & i=1,\ldots,p \\ m_{k+p}, & i=p+1,\ldots,l \end{cases}$$

is a lower envelope for all feasible extrapolations of order 3 to $(m_1,\ldots,m_k)$.

Proof: Follows from proposition 1. *Q.E.D.*

Proposition 6. Consider the constraints (8) with $d=4$ and fixed $l>0$. A solution $(m_1,\ldots,m_k)$ which satisfies (8) with $l=0$ can be extrapolated to a solution $(m_1,\ldots,m_{k+l})$ which satisfies (8) with $l>0$ if and only if —

$$\Delta^1 m_k + j\Delta^2 m_k + \frac{1}{2} j(j+1)\Delta^3 m_k \geq 0, \quad j=1,\ldots,l. \tag{19}$$

Let —

$$q = \begin{cases} \text{gilb}(-\Delta^2 m_k/\Delta^3 m_k), & \text{if } \Delta^3 m_k > 0 \\ l, & \text{if } \Delta^3 m_k = 0 \end{cases}$$

$$\alpha = \Delta^1 m_k + q\Delta^2 m_k + \frac{1}{2} q(q+1)\Delta^3 m_k.$$

Then the upper envelope of all such extrapolations is:

$$m_{k+i} = \begin{cases} m_k + i\Delta^1 m_k + 1/2\ i(i+1)\Delta^2 m_k + 1/6\ i(i+1)(i+2)\Delta^3 m_k & i=1,\ldots,q \\ m_{k+q} + (i-q)\alpha & i=q+1,\ldots,l. \end{cases}$$

Proof: Follows from proposition 2. *Q.E.D.*

Proposition 7. Consider the constraints (8) with $d=4$ and fixed $l>0$, and let $(m_1,\ldots,m_k)$ be a solution to (8) with $l=0$ satisfying (19). Let p be defined as in (18).

a. If $p \geq l$, then the extrapolation —

$$m_{k+i} = m_k + i\Delta^1 m_k + \frac{1}{2} i(i+1)\Delta^2 m_k, \quad i=1,\ldots,p$$

is a lower envelope for all feasible extrapolations of order 4 to $(m_1,\ldots,m_k)$.

b. If $p \leq l$, let —

$$u = \min(l, 1+\text{gilb}(-2\Delta^1 m_k/\Delta^2 m_k)).$$

Then the extrapolation —

$$m_{k+i} = \begin{cases} m_k + i\Delta^1 m_k + \dfrac{1}{2} i(i+1)\Delta^2 m_k + \dfrac{1}{6} i(i+1)(i+2) \cdot \left(\dfrac{-2(\Delta^1 m_k + u\Delta^2 m_k)}{u(u+1)} \right), & i=1,\ldots,u \\ m_{k+u} & i=u+1,\ldots,l \end{cases}$$

is a lower envelope for all feasible extrapolations of order 4 to $(m_1,\ldots,m_k)$.

Proof: Follows from proposition 3. *Q.E.D.*

Proposition 7 states that the lowest envelope is either along a quadratic function, or it starts as a cubic function which tapers off to a constant function.

5. MONTE CARLO STUDY OF PERFORMANCE

To get an idea of how well the prediction envelopes estimate future behavior, we conducted a small Monte Carlo simulation experiment. Our goal is to estimate the number of events over some finite horizon. As in [13], we compare the completely monotone approach to some of the more popular parametric models. A value of $d=4$ is used for the completely monotone estimation (δ is taken as 1). Thus the least squares problem (8) is solved for $d=4$, with the constraints of (19) replacing the constraints of (8) for $i=k+1,\ldots,k+l$. Propositions

6 and 7 are applied to the resulting solution to obtain the upper and lower envelopes for the future mean function. Finally, we need a point estimate of the mean number of failures. We arbitrarily decided to use the midpoint of the envelope.

Our choice of parameter models consists of three families of nonhomogeneous Poisson processes (NHPP). The mean functions of the NHPPs can have exponential, power or logarithmic form:

$$M_{\exp}(t) = \gamma(1-e^{-\eta t}),$$

$$M_{\text{pow}}(t) = \gamma t^{\alpha},$$

$$M_{\log}(t) = \gamma \log(\beta t+1).$$

Those models are fit to data by using the method of maximum likelihood [16]. Furthermore, we define a fourth model which is a mixture of the above three. It is fit by selecting the best fitting (maximum likelihood) of the three models. This is the "best" parametric model, among the three possibilities.

We draw our data from 16 different Poisson processes. Each process is observed over the interval [0,100] and the future interval is [100,125], — 25% into the future. We used $k=20$ and $l=5$. The 16 cases provide a variety of growth patterns. Each case is replicated 400 times. The cases are summarized in table 1.

TABLE 1. Data Models (Poisson Processes).
[All models are scaled so that $E(N(100)) = M(100) = 40$]

Model Number	Type of NHPP	Parameter	M(125)-M(100)
1	Homogeneous		10.00
2	Power	$\alpha = .749$	7.28
3	Power	$\alpha = .557$	5.29
4	Power	$\alpha = .410$	3.83
5	Power	$\alpha = .296$	2.73
6	Power	$\alpha = .208$	1.90
7	Logarithmic	$\beta = .0124$	6.42
8	Logarithmic	$\beta = .0429$	4.43
9	Logarithmic	$\beta = .131$	3.16
10	Logarithmic	$\beta = .461$	2.27
11	Logarithmic	$\beta = 2.43$	1.62
12	Exponential	$\eta = .00808$	5.88
13	Exponential	$\eta = .0167$	3.17
14	Exponential	$\eta = .0265$	1.47
15	Exponential	$\eta = .0385$	0.54
16	Exponential	$\eta = .0550$	0.12

The performance of the parametric models and the completely monotone approach are summarized in tables 2–4. Table 2 shows the average prediction made by each model for the 400 replicates of each case. Table 3 shows the average percentage error, or bias. Table 4 shows the root-mean-square percentage error for the 400 estimates made by each model for the 16 test cases. When the data come from a certain model, then that particular model gives the best predictions. However in most cases, the completely monotone comes in as "second best", ie, it gives better predictions than those given by using the incorrect parametric model. In practice, of course, it is highly unlikely that a parametric model used for prediction will indeed be the "correct" model from which the failure data were generated. Table 5 summarizes the performance of the prediction envelopes. The majority of the envelopes have zero width, ie, the upper envelope is identical to the lower envelope.

TABLE 2. Average Predictions of Mean Number over Future Horizon

Model Number	True Mean	EXP	LOG	POW	BEST	CM. Mdpt.
1	10.00	8.67	8.86	9.33	8.73	9.52
2	7.28	5.62	6.11	7.34	6.38	7.72
3	5.29	2.97	3.73	5.36	4.70	5.88
4	3.83	1.36	2.23	3.88	3.62	4.50
5	2.73	0.51	1.31	2.76	2.65	3.40
6	1.90	0.15	0.75	1.92	1.87	2.52
7	6.42	6.10	6.76	8.20	6.48	7.44
8	4.43	3.41	4.68	6.63	4.19	5.38
9	3.16	1.64	3.29	5.27	2.89	4.06
10	2.27	0.64	2.35	4.11	2.28	3.08
11	1.62	0.18	1.66	3.10	1.71	2.30
12	5.88	5.86	6.57	8.09	6.26	7.31
13	3.17	3.21	4.70	6.72	3.67	4.73
14	1.47	1.51	3.62	5.63	1.91	2.84
15	0.54	0.57	2.95	4.77	0.75	1.60
16	0.12	0.14	2.48	4.07	0.21	0.88

TABLE 3
Percent Prediction Error (Bias) for Mean Future Number

Model Number	Fitted Model				
	EXP	LOG	POW	BEST	CM
1	−13.	−11.	−7.	−13.	−5.
2	−23.	−16.	+1.	−12.	+6.
3	−44.	−30.	+1.	−11.	+11.
4	−65.	−42.	+1.	−6.	+17.
5	−81.	−52.	+1.	−3.	+24.
6	−92.	−61.	+1.	−2.	+33.
7	−5.	+5.	+28.	+1.	+16.
8	−23.	+5.	+50.	−6.	+21.
9	−48.	+4.	+67.	−8.	+29.
10	−72.	+3.	+81.	0.	+36.
11	−89.	+3.	+98.	+6.	+42.
12	0.	+12.	+32.	+6.	+24.
13	+1.	+48.	+112.	+16.	+49.
14	+3.	+146.	+282.	+30.	+93.
15	+6.	+448.	+788.	+40.	+199.
16	+14.	+1921.	+3216.	+74.	+615.

Perspective

We stress that some components in the formulation of the completely monotone model were chosen arbitrarily. Other definitions of the raw estimates and other objective functions

TABLE 4
Percent Root Mean Square Error
for Mean Future Number Prediction

Model Number	Fitted Model				
	EXP	LOG	POW	BEST	CM
1	26.	24.	29.	24.	29.
2	39.	32.	23.	31.	30.
3	53.	39.	23.	32.	39.
4	69.	46.	23.	29.	48.
5	83.	55.	23.	26.	60.
6	93.	62.	23.	26.	73.
7	37.	32.	39.	36.	38.
8	44.	31.	59.	43.	50.
9	57.	26.	75.	47.	62.
10	75.	23.	89.	40.	74.
11	90.	21.	99.	27.	85.
12	38.	34.	47.	37.	45.
13	45.	61.	120.	53.	80.
14	54.	155.	292.	82.	142.
15	67.	461.	805.	134.	278.
16	95.	1956.	3268.	336.	776.

TABLE 5
Performance of Completely Monotone Prediction Windows

Data Model Number	Fraction Zero Width	Fraction Non-zero Width	Av. width Non-zero Envelope	True Mean Coverage		
				Fraction Overestimate	Fraction Correct	Fraction Underestimate
1	.715	.285	0.348	.408	.067	.525
2	.515	.485	0.816	.475	.105	.320
3	.503	.497	0.966	.542	.182	.275
4	.548	.452	0.759	.550	.167	.283
5	.570	.430	0.588	.585	.160	.255
6	.600	.400	0.414	.595	.155	.250
7	.420	.580	1.116	.610	.193	.197
8	.417	.583	1.334	.515	.283	.202
9	.505	.495	0.897	.560	.243	.197
10	.573	.427	0.616	.600	.193	.208
11	.573	.427	0.441	.605	.182	.213
12	.363	.637	1.228	.648	.190	.162
13	.305	.695	1.828	.520	.400	.080
14	.321	.679	1.300	.574	.333	.093
15	.503	.497	.0656	.652	.243	.105
16	.698	.302	0.209	.925	.063	.013

will give different, and possibly better estimates. Nevertheless, the completely monotone approach shows a robustness not exhibited by the individual parametric models. The procedure has quite low bias, which is less than that caused by using the incorrect parametric models for prediction. Comparisons to the "best" parametric model are unfair because the Monte Carlo data are, in effect, drawn from that model. We could use other models to generate data for which this "best" parametric model is inferior to the more robust completely monotone approach.

ACKNOWLEDGMENT

AS gratefully acknowledges support of the US National Science Foundation Grant ECS-8709795. DRM gratefully acknowledges support of the US National Aeronautics and Space Administration Grant NAG-1-771.

REFERENCES

[1] A. A. Abdel-Ghaly, P. Y. Chan, B. Littlewood, "Evaluation of competing software reliability predictions", *IEEE Trans. Software Engineering*, vol SE-12, 1986, pp 950–967.

[2] R. E. Barlow, D. J. Bartholomew, J. M. Bremner, H. D. Brunk, *Statistical Inference Under Order Restrictions*, 1972; John Wiley & Sons.

[3] G. Campbell, K. O. Ott, "Statistical evaluation of major human errors during the development of new technological systems", *Nuclear Science and Engineering*, vol 71, 1979, pp 267–279.

[4] P. Y. Chan, *Software Reliability Prediction* (PhD Thesis), 1986, Department of Mathematics, The City University, London.

[5] L. H. Crow, Reliability analyses for complex repairable systems, *Reliability and Biometry*, (Proschan & Serfling, eds), 1974, pp 379–410; SIAM, Philadelphia.

[6] J. T. Duane, "Learning curve approach to reliability monitoring", *IEEE Trans. Aerospace*, vol 2, 1964, pp 563–566.

[7] A. K. Goel, K. Okumoto, "Time independent error detection rate model for software reliability and other performance measures", *IEEE Trans. Reliability*, vol R-28, 1979, pp 206–211.

[8] Z. Jelinski, P. Moranda, "Software reliability research", *Statistical Computer Performance Evaluation*, (W. Ferberger, ed), 1972, pp 465–484; Academic Press.

[9] B. Littlewood, "Software reliability growth: A model for fault removal in computer-programs and hardware-design", *IEEE Trans. Reliability*, vol R-30, 1981, pp 313–320.

[10] B. Littlewood, P. A. Keiller, "Adaptive software reliability modelling", *Proc. 14-th Int'l Conf. Fault-Tolerant Computing*, 1984, pp 108–113; IEEE Computer Society Press.

[11] G. P. McCormick, *Nonlinear Programming*, 1983; John Wiley & Sons.

[12] D. R. Miller, "Exponential order statistics models for software reliability growth", *IEEE Trans. Software Engineering*, vol SE-12, 1986, pp 12–24.

[13] D. R. Miller, A. Sofer, "Completely monotone regression estimates of software failure rates", *Proc. Eighth Int'l Conf. Software Engineering*, 1985, pp 343-348; IEEE Computer Society Press.

[14] D. R. Miller, A. Sofer, "A nonparametric approach to software reliability, using complete monotonicity", *Software Reliability: A State of the Art Report*, (A. Bendell, P. Mellor, eds), 1986, pp 183-195; Pergammon Press.

[15] D. R. Miller, A. Sofer, "Least squares regression under convexity and higher order difference constraints with application to software reliability", *Advances in Order Restricted Inference*, (Dykstra, Robertson, Wright, eds), 1986, pp 91-124; Springer Verlag.

[16] J. D. Musa, K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement", *Proc. Seventh Int'l Conf. Software Engineering*, 1984, pp 230-238; IEEE.

[17] P. M. Nagel, F. W. Scholz, J. A. Skrivan, "Software reliability: Additional investigations into modeling with replicated experiments", CR-172378, 1984; NASA.

AUTHORS

Professor Ariela Sofer; Department of Operations Research and Applied Statistics; George Mason University; Fairfax, Virginia 22030 USA.

Ariela Sofer received her BSc in Mathematics and her MSc in Operations Research, both from the Technion, Technological Institute of Israel. She received her DSc degree in Operations Research from The George Washington University in 1984. Dr. Sofer joined George Mason University in 1983, where she holds the rank of Associate Professor. Her areas of interest are software reliability, mathematical programming, and numerical optimization. She is a member of ORSA, SIAM, and the Mathematical Programming Society.

Professor Douglas Miller; Department of Operations Research and Applied Statistics; George Mason University; Fairfax, Virginia 22030 USA.

Douglas R. Miller received his BS in Mathematics from Carnegie Institute of Technology, Pittsburgh in 1966, and the MA in Mathematics and PhD in Operations Research from Cornell University, Ithaca in 1969 & 1971. Dr. Miller held positions at the University of Missouri-Columbia, and the George Washington University, before joining George Mason University, Fairfax in 1989, where he is Professor of Operations Research and Applied Statistics in the School of Information Technology and Engineering. He has also held visiting positions at Universidad Nacional del Sur, Argentina, and the City University, London. His current research involves probability modeling and statistical analysis, with applications to software reliability, queueing systems, and polymerization processes. Since 1977 he has been associated with the advanced digital avionics program at NASA Langley Research Center. He is a member of ORSA, TIMS, ASA, ACM, and the IEEE Computer Society.

Manuscript TR88-216 received 1988 December 15; revised 1990 August 15.

IEEE Log Number 42712 ◄TR►

Appendix 2

P. A. Keiller and D. R. Miller, "On the Use and the Performance of Software Reliability Growth Models," **Reliability Engineering and System Safety 32** (1991): 95-117.

Reliability Engineering and System Safety 32 (1991) 95–117

On the Use and the Performance of Software Reliability Growth Models

Peter A. Keiller

IBM Corporation, Library Management and Systems Control Department,
10401 Fernwood Road, Bethesda, Maryland 20817, USA

&

Douglas R. Miller

Department of Operations Research and Applied Statistics,
School of Information Technology and Engineering,
George Mason University, Fairfax, Virginia 22030, USA

ABSTRACT

We address the problem of predicting future failures for a piece of software. The number of failures occurring during a finite future time interval is predicted from the number of failures observed during an initial period of usage by using software reliability growth models. Two different methods for using the models are considered: straightforward use of individual models (simple models), and dynamic selection among models based on goodness-of-fit and quality-of-prediction criteria (super models). Performance is judged by the relative error of the predicted number of failures over future finite time intervals relative to the number of failures eventually observed during the intervals. Six simple models and eight super models are evaluated based on their performance on twenty data sets. This study is by no means comprehensive. Some conclusions can be drawn, but many open questions remain regarding the use and the performance of software reliability growth models.

INTRODUCTION

Software sometimes fails to perform as desired. These failures may be due to errors, ambiguities, oversights or misinterpretations of the specification

Reliability Engineering and System Safety 0951-8320/91/$03·50

which the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software, or other unforeseen problems. All of these potential sources of failure create an environment of uncertainty for the behavior of any software: will the software fail or not? If so, when? Statistical modeling and analysis provide tools to investigate this phenomenon.

A general goal is to understand, predict and control the uncertainty in software failure behavior. Statistical models and analysis can investigate various aspects of software and its failure, at different levels of detail. Our study treats a piece of software as a 'black box' operating in a random environment. We ignore factors in the development of the software, the internal structure and functioning of the software, and details of the operating environment. In contrast, Eckhardt & Lee,[1] Littlewood[2] and Littlewood & Miller[3] present models that deal more closely with the structure of the software.

In this paper we consider the sequence of times at which a piece of software fails. After each failure, the software is fixed so that (hopefully) it will not fail again from the same cause. From these data we want to predict future failure behavior. In particular, we will try to predict the number of additional failures which will occur during a future time interval of finite length. Our approach is to use 'reliability growth models'. The questions are: 'What is the best way to do this?' and 'How well do these models predict future failure behavior?'. Many reliability growth models have been proposed. For a given piece of software it is very difficult (perhaps impossible) to know which reliability growth model to use. (Iannino *et al.*[3a] give qualitative guidelines for choosing different software reliability growth models.) It is also difficult to know much about the accuracy of the predictions about future failures. Our study looks at these problems.

We have taken failure data for 20 programs, fitted reliability growth models to initial segments of each data set, predicted the number of remaining failures in the data set, and computed the prediction errors. Our reliability growth models include several of the usual models in the literature and additional models that we call 'super models'. These super models are based on a set of the usual reliability growth models plus a selection criterion which identifies one of the set to use for predictions at each point of time; selection criteria may be based on 'goodness-of-fit' or 'quality-of-past-prediction' measures. We have tried to identify the best models or approaches, conditional on our 20 failure data sets. We cannot make any strong recommendations, but we do see that many of the models give useful predictions if only nominal levels of reliability are of concern. The major conclusion is that there are still important open questions in the area of reliability growth modeling and prediction. We hope that this paper will

serve as an example of an objective study of this important problem, and that more work will be done.

An important negative conclusion can be drawn from any study of reliability growth modeling: only moderate levels of reliability can be treated. Extremely high levels of reliability such as those required in safety critical systems cannot be treated; see Miller.[4,5] Some software reliability growth models will occasionally predict that no failures will occur in the future; however, this cannot be done with levels of confidence required in safety critical software. Even the most casual examination of the numbers presented in this paper should lead the reader to that conclusion.

THE RELIABILITY GROWTH SCENARIO

A system contains design flaws, each of which eventually manifests itself at some point in time, whereupon the system is redesigned in order to remove the design flaw. Design flaws are often called 'bugs', and the time points of

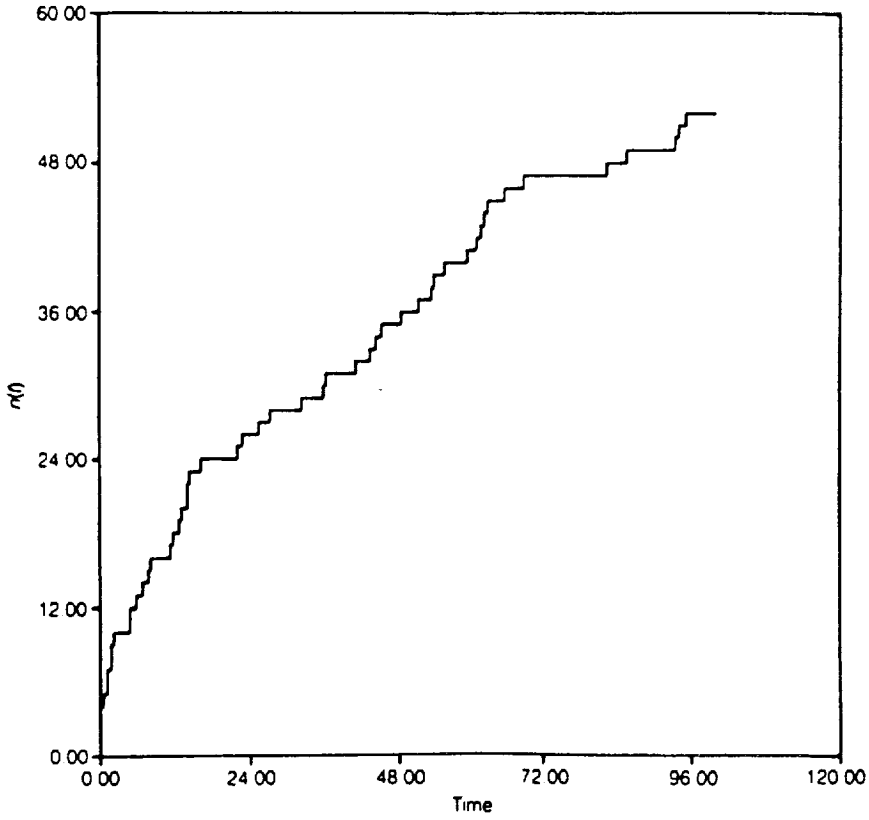

Fig. 1. Observed cumulative number of failures as a function of time.

manifestation mentioned above will be called 'failure times'. If the failure times are indexed chronologically they can be represented as

$$0 \le t_1 \le t_2 \le t_3 \le t_4 \le \cdots \le t_c \tag{1}$$

where t_c is the 'current' time, the length of time that the system has been investigated, i.e. execution time for software. A convenient way to graphically present these failure time data and stochastic processes is with a plot of cumulative number of failures versus cumulative time: let

$$n(t) = \max \{i: t_i \le t\} \qquad 0 \le t \le t_c \tag{2}$$

be the sample path of such data as depicted in Fig. 1. If system redesigns successfully remove design flaws, system reliability will improve and eqn (1) should show a general pattern of stochastically increasing interfailure times, and plots of the cumulative number of failures in eqn (2) should show a positive but stochastically decreasing slope (negative second derivative). This phenomenon is called 'reliability growth', i.e. the reliability of the system is improving as successive redesigns remove design flaws. We wish to make predictions about future behavior of the software, i.e. for t such that $t_c \le t$.

RELIABILITY GROWTH MODELS

It is convenient to consider eqn (1) as the realization of a random process:

$$0 \le T_1 \le T_2 \le T_3 \le T_4 \le \cdots \tag{3}$$

where the T_i's are random variables (the t_i's are real scalars) and the process is observed for $t: 0 \le t \le t_c$. The stochastic process of which eqn (2) is a realization is

$$\{N(t) = \max (i: T_i \le t), 0 \le t\} \tag{4}$$

The stochastic processes, eqns (3) and (4), are 'reliability growth processes'. Numerous reliability growth models have been proposed for the analysis of software reliability. The first one specifically for software was proposed by Jelinski & Moranda.[6] There are numerous surveys[7-11] of the software reliability growth modeling literature.

There tend to be three general classes of software reliability growth models: interfailure time models, order statistic models, and Poisson process models. Examples include the Littlewood–Verall model,[12] Littlewood's Pareto model[13] and Duane's power law nonhomogeneous Poisson process,[14,15] respectively. (For general discussions of the interrelationships between these classes of models, some additional modeling considerations,

derivations of these models from more basic principles, underlying assumptions and complications, see Gray[16,17] and Miller.[18]) All reliability growth processes can be thought of as consisting of noisy behavior around a smooth trend curve. One obvious way of describing the trend curve is with the average number of failures occurring by time t, i.e. the expected value of the number of failures, thus a trend curve for stochastic processes in eqns (3) and (4) is

$$M(t) = E[N(t)] \qquad 0 \leq t \tag{5}$$

Several members of a logarithmic family of trend (or growth) curves are shown in Fig. 2. A rough approach to the prediction problem is to pick a member from a parametric family of growth curves (as in Fig. 2) which best fits some software failure data (as in Fig. 1) and then extrapolate along the curve to the right in order to predict the expected number of failures during a future time interval. In fact, most reliability growth modeling is equivalent to this kind of curve fitting. Sophisticated statistical techniques may be used to

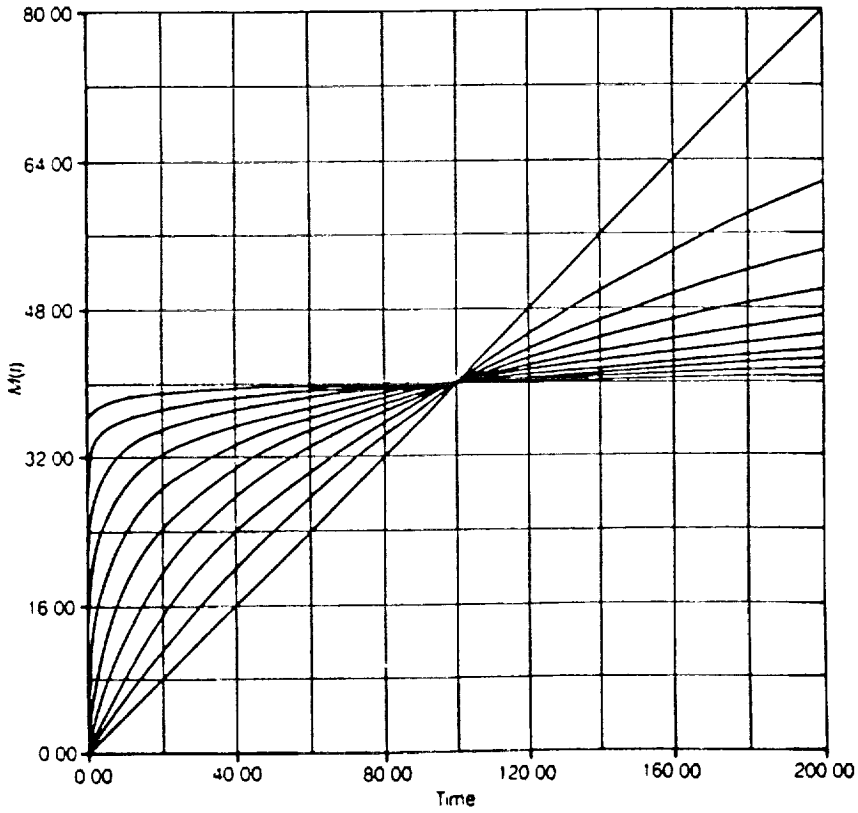

Fig. 2. Subset of mean functions for a parametric family of reliability growth models with logarithmic trend.

fit the models. But it had not been proven that these statistical techniques are superior to a simple qualitative 'eye-ball' fit. This fact should be kept in mind when interpreting the accuracy of the predictions from reliability growth models. In particular, these models are not refined enough to distinguish between whether there are zero bugs or one bug remaining in a piece of software.

Another conclusion from the point of view that software reliability growth is noisy behavior around a growth curve can be used in defining a rich family of reliability growth models: we want a family which is characterized by the mean function, eqn (5), and we want a rich set of mean functions. (For a discussion of necessary and sufficient conditions for reliability growth mean functions, see Miller.[18] For a nonparametric approach, see Miller & Sofer.[19]) An attractive family of stochastic processes, eqn (4), characterized by their mean functions are the nonhomogeneous Poisson processes (NHPP); Musa & Okumota[20,21] have promoted this idea. NHPPs have an independent Poisson number of failures in disjoint intervals:

$$P(N(t+s)-N(t)=n)=\mathrm{e}^{-(M(t+s)-M(t))}\frac{(M(t+s)-M(t))^n}{n!}$$
$$0\le t,\ 0\le s;\ n=0,1,2,3,\ldots \quad (6)$$

This characterizes the processes.

We shall use six parametric families of NHPPs, characterized by their mean functions:

M1	Power:	$M_1(t)=\gamma t^{\alpha}$	$0<\alpha\le 1$
M2	Exponential:	$M_2(t)=\gamma(1-\mathrm{e}^{-\eta t})$	$0<\eta$
M3	Logarithmic:	$M_3(t)=\gamma\log(1+\beta t)$	$0<\beta$
M4	Pareto:	$M_4(t)=\gamma(1-(1+\beta t)^{-\alpha})$	$0<\alpha,\ 0<\beta$
M5	General Power:	$M_5(t)=\gamma((1+\beta t)^{-\alpha}-1)$	$-1<\alpha<0,\ 0<\beta$
M6	Weibull:	$M_6(t)=\gamma(1-\exp(-\eta t^{\alpha}))$	$0<\alpha,\ 0<\eta,\ 0<\gamma$

The 'Power' law was first suggested in a reliability growth context by Duane[14] and specifically as a NHPP model by Crow.[15] The 'Exponential' law is the trend encountered in the Jelinski–Moranda[6] model and the trend of the Goel–Okumoto[22] NHPP software reliability growth model. The 'Logarithmic' trend is used by Musa and Okumoto.[20,23] (Figure 2 shows some of the mean functions for this family.) The 'Pareto' curve occurs in Littlewood's[13] order statistic model. The 'General Power' curve arises naturally when considering order statistics of independent but non-identically exponentially distributed failure times.[18] The 'Weibull' NHPP is discussed by Musa & Okumoto,[20] Abdalla-Ghaly *et al.*,[7] Miller[18] and

others. Taken together, these parametric families include many of the reliability growth models proposed in the literature; see Miller[18] for plots of selected mean functions of these models.

FITTING MODELS

We fit the six NHPP models (M1, M2, M3, M4, M5 and M6) to data in the form of eqns (1) or (2) as depicted in Fig. 1. In effect, for each of the above six parametric families, we want to find the 'best' fitting curve. We use the method of maximum likelihood as suggested and described by Musa & Okumoto[20] for fitting these models to data consisting of single sample paths. For each parametric family we get maximum likelihood estimates (MLEs) of the appropriate parameters: α, β, γ, etc.; this gives us a curve

$$\hat{M}(t) \qquad 0 \leq t \leq t_c \tag{7}$$

uniquely determining the MLE NHPP. To solve for the MLEs we used the Nelder–Meade[24] simplex search algorithm. We wanted a general algorithm to solve general MLE problems for this study; in practice one would want to devote more effort to finding the MLEs as Chan[25] does for some models.

There is no unique definition of 'best-fitting'. The best way to fit a stochastic process model to an observed realization of the process is an open question. As mentioned before, an 'eye-ball' fitting may work well. Least-square or Kolmogorov–Smirnov distances could be used. The definition of 'best-fitting' is certainly dependent on how the fitted curve is to be used. In our context we could define 'best-fitting' as equivalent to 'best-predicting'; Brocklehurst[26] has investigated this approach of fitting some reliability growth models by optimizing certain quality-of-prediction measures.

We are faced with two problems: finding the best-fitting member of a given parametric family and choosing among the best-fitting from several parametric families. We have rather arbitrarily decided to use the MLE for a given family. To choose among different families we shall try several approaches: minimum Kolmogorov–Smirmov distance, maximum likelihood and three others (Retro-U, Retro-Y and Retro-PL) to be described later.

PREDICTIONS

Various predictions can be made from the fitted NHPP with mean function (eqn (7)): the expected number of failures during a future time interval

$$(t, t+s]\text{: } \hat{M}(t+s) - \hat{M}(t)$$

the current failure rate, at time t_c: $\hat{m}(t_c)$, where

$$\hat{m}(t) = \frac{d}{ds}\hat{M}(s)|_{s=t}$$

the time until a target failure rate

$$r_0\text{: } t_0 = \min\{t\text{: } \hat{m}(t) = r_0\}$$

the distribution of the time until next failure from the current time

$$t_c\text{: } \hat{F}_{t_c}(s) = 1 - P(0 \text{ failures in } (t_c, t_c + s]) = 1 - \exp(-[\hat{M}(t_c + s) - \hat{M}(t_c)])$$

and the density of time until next failure from current time

$$t_c\text{: } \hat{f}_{t_c}(s) = \frac{d}{ds}\hat{F}_{t_c}(s)$$

A standard approach is to consider the modeling, fitting and prediction steps as separate activities. Since the ultimate goal is good prediction, Abdalla-Ghaly *et al.*[7] argue convincingly that an integrated approach should be taken: they introduce the idea of a 'prediction system' which integrates the above three phases. We are taking such an integrated point of view in this paper.

QUALITY OF PREDICTIONS

We wish to evaluate the accuracy of the predictions of future failure behavior which we make. If we predict an observable quantity, we can wait and compare the observation with the prediction, and then compute a measure of discrepancy. When predicting the number of failures in finite future time intervals, the error is simply the difference between the predicted number and the observed number. For a given piece of software undergoing execution, failure and fix, as time passes we can make predictions up to various time horizons, then when that horizon is reached we can compare the prediction with observation. So for a given piece of software we can make a sequence of predictions which can be checked against observation; from this a measure of quality-of-prediction can be computed.

When we use a reliability growth model to predict the distribution or the density of the time until the next failure, we must compare predicted distributions to observed times in order to get a measure of quality-of-prediction. The procedure is as follows: after each failure is observed, the model is fitted to the data observed, thus far giving a new estimated mean function. The mean function fitted to the first i observed failures is denoted as $\hat{M}_i(t)$, $0 \le t$. The estimates of the distribution and the density of the time

until the next failure are then computed:

$$\hat{F}_{i+1}(s) = \hat{F}_{t_i}(s) = 1 - \exp(-[\hat{M}_i(t_i + s) - \hat{M}_i(t_i)]) \quad \hat{f}_{i+1}(s) = \frac{\mathrm{d}}{\mathrm{d}s}\hat{F}_{i+1}(s) \quad i \geq 1$$

Then the next failure is observed, at time t_{i+1}, so the interfailure time is

$$x_{i+1} = t_{i+1} - t_i$$

Thus we have a sequence of predictions of successive interfailure time distributions and a sequence of observed interfailure times. The goal is to evaluate how well the predictive distributions actually predicted the observed interfailure times. We would like a quantitative measure of quality-of-prediction. Littlewood and co-workers[7,25,27] have provided three such measures, which we now summarize. (These measures are used mainly for comparison purposes. It is difficult to interpret the deviations quantitatively.)

The first quality-of-prediction measure is the 'u-plot': it is well known that $U = F_X(X)$ has a uniform distribution on the interval $[0, 1]$. Using this fact, if $\hat{F}_{i+1}(\cdot)$ is the true distribution of X_{i+1} then $u_{i+1} = \hat{F}_{i+1}(x_{i+1})$ will be an observation from a $U[0, 1]$ distribution. Thus the empirical distribution formed from the u's should be closed to that of $U[0, 1]$. If we observe $n_0 + n$ failures, starting to make predictions after the n_0th failure, the plot of

$$\{(u_{n_0+1}, i/(n+1)), i = 1, 2, 3, \ldots, n\}$$

is the u-plot. The maximum deviation of the u-plot from the identity function is a measure of quality-of-prediction.

The second measure of quality-of-prediction is the 'y-plot': if the predictive distributions are good the u's should look like a random sequence of independent $U[0, 1]$ variables, and $-\log(1-u)$'s like exponential random variates. In this case, let

$$y_i = \sum_{j=1}^{i} \log(1 - u_{n_0+j}) \Big/ \sum_{j=1}^{n} \log(1 - u_{n_0+j}) \qquad i = 1, 2, 3, \ldots, n$$

and plot the pairs $\{(y_i, i/(n+1)), i = 1, 2, 3, \ldots, n\}$. If the predictive distributions are good, this plot should be close to the identity function. A quantitative measure of the quality-of-prediction is the maximum deviation between the y-plot and the identity function.

The third measure of quality-of-prediction is the prequential likelihood: based on Dawid's[28] generalization of likelihood to a sequential situation, we have

$$PL_n = \prod_{i=1}^{n} \hat{f}_{n_0+i}(x_{n_0+i})$$

For comparison purposes, the best predictive system should have the largest prequential likelihood.

For a detailed discussion of these three measures of quality-of-prediction, see Abdalla-Ghaly *et al.*,[7] Chan[25] and Keiller *et al.*[27] These three measures give a dynamic real-time evaluation of how well a given parametric model has done predicting interfailure times up to the present. It would seem logical to calculate the next prediction from the parametric family which has performed best up to the present on the particular software failure data set under consideration. These three measures give a basis for making this choice.

There are other possible measures of quality-of-prediction. For example, one such measure could be based on past predictions of the number of failures to be observed in finite time intervals which have subsequently elapsed.

SUPER MODELS

We consider eight super models. A super model is a set of parametric reliability growth models and a selection criterion; for a given software failure data set and for a given time, the selection criterion chooses the parametric model in the set that is to be used for making predictions of future failure behavior. As time passes for a given data set, a given super model may change its choice of parametric family to use for predictions.

Our procedure for a super model is as follows: using maximum likelihood estimation, we fit all six of the parametric models (M1–M6). Next, the selection criterion picks one parametric class based on the fitted models. The current fitted model of the chosen class is used for making predictions at the current time.

We consider two goodness-of-fit criteria: Kolmogorov–Smirnov deviations between the fitted mean function and the sample path, i.e.

$$\sup_{0 \le t \le t_e} |\hat{M}(t) - n(t)|$$

and the maximum likelihood of the fitted models. We comment that three-parameter models (M4, M5 and M6) should fit the data better than two-parameter models (M1, M2 and M3), and some correction should be made to the goodness-of-fit criterion to reflect this; see Akaike.[29] We do not pursue this here. It is another example of one of the open questions in this research area.

We consider three pure quality-of-prediction measures: the u-plot, the y-plot and the prequential likelihood. Using these criteria requires fitting each

of the six NHPP models (M1–M6) after each failure and calculating the predictive distribution and density of the time until next failure. So these three super models require more computationally intensive implementation.

Finally, we consider three hybrid super models. We use the three quality-of-prediction measures in a goodness-of-fit mode. At current time, t_c, the six NHPP models (M1–M6) are fitted to the data; each fitted model is then used retroactively to predict ('retrodict') the already elapsed interfailure time distributions and densities; from these retrodictions and the observed data, u-plots, y-plots and prequential likelihood can be computed, and the best fitted model chosen from among the six simple models (M1–M6).

We comment that it is possible to define other selection criteria. In a related piece of work, Littlewood & Keiller[30] and Chan[30a] have shown how to improve the prediction of the time until next failure by adapting a reliability growth model, basing the adaption on past quality-of-prediction.

To summarize, our eight super models are all based on six NHPP models (M1, M2, M3, M4, M5 and M6). The selection criteria for the eight super models are:

M7	Maximum likelihood
M8	Minimum K–S distance
M9	U-plot
M10	Y-plot
M11	Prequential likelihood
M12	Retro. U-plot
M13	Retro. Y-plot
M14	Retro. PL

EXPERIMENTS

We investigate the performance of 14 reliability growth models: six simple models (M1–M6) and eight super models (M7–M14). We see how well the different models can predict the number of new failures manifested during finite future time intervals.

We base our experiment on 20 sets of software failure data, denoted D1–D20 in Table 1. The first 15 data sets are the same as those used in performance experiments by Musa and co-workers,[20,21,31] with two modifications: D5 consists of only the first 288 interfailure times of Musa's[32] System 5 because a major code change occurred at that point; D11 consists of the last 100 interfailure times of Sukert's[33] data set because the data set was huge with several major changes. The data consist of execution times between successive failures. To give the reader a rough idea of the data, we

TABLE 1
Summary of Software Failure Data Sets

Data set designation	*Original source and reference*	*Designation in original source*	*Designation in references[20,31]*
D1	Musa[32]	1	T1
D2	Musa[32]	2	T2
D3	Musa[32]	3	T3
D4	Musa[32]	4	T4
D5	Musa[32]	5	T5
D6	Musa[32]	6	T6
D7	Musa[32]	27	T16
D8	Musa[32]	40	T17
D9	Musa[34]	—	T18
D10	Musa[32]	17	T19
D11	Sukert[33]	—	T20
D12	Musa[32]	—	T21
D13	Miller[35]	ISEE-C	T22
D14	Miller[35]	AEM	T23
D15	Miller[35]	SMM	T25
D16	Abdalla-Ghaly *et al.*[7]	Fig. 2	—
D17	Abdalla-Ghaly *et al.*[7]	Fig. 3	—
D18	Mock[36]	A	—
D19	Mock[36]	B	—
D20	Mock[36]	C	—

present them in an aggregated form in Table 2: we split the total cumulative time for each data set into 10 equal intervals, and show the cumulative number of failures occurring up to each of 10 elapsed time points. From Table 2 it is possible to construct very rough plots of reliability growth as in Fig. 1. The original raw unaggregated data are used to fit reliability growth models.

The experiment is designed as follows. For each data set we select nine time points, equal to $k/10$ of the total execution time for the entire data set, $k = 1, 2, 3, \ldots, 9$; for each of our six simple models (M1–M6), we find the MLE and make predictions. We have 180 ($= 9 \times 20$) data intervals: $[0, (k/10)T_j^{\text{tot}}]$, $k = 1, 2, 3, \ldots, 9$, $j = 1, 2, 3, \ldots, 20$, where T_j^{tot} is the total execution time for data set *Dj*. Using maximum likelihood estimation, we fit the model *Mi* to the failure data observed from data set *Dj* in the interval $[0, (k/10)T_j^{\text{tot}}]$, then the fitted model is used to predict the number of failures to occur in the future interval $((k/10)T_j^{\text{tot}}, T_j^{\text{tot}}]$, for $k = 1, 2, 3, \ldots, 9$, $j = 1, 2, 3, \ldots, 20$ and $i = 1, 2, 3, \ldots, 6$. Let $n_i(j, k)$ equal the number of predicted failures for data set *Dj* in the time interval $((k/10)T_j^{\text{tot}}, T_j^{\text{tot}}]$, predicted by model *Mi* fitted to data observed from data set *Dj* over the time interval $[0, (k/10)T_j^{\text{tot}}]$. Let $n(j, k)$

TABLE 2
Cumulative Failures Occurring in Percentage of Total Time

Data set	*Elapsed percentages of time*									
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*	*100%*
D1	49	74	85	93	104	114	122	128	132	136
D2	20	28	30	41	42	46	48	50	52	54
D3	22	24	28	30	30	33	35	35	36	38
D4	24	37	45	50	50	50	51	51	51	53
D5	53	103	153	172	192	235	251	264	273	288
D6	15	26	32	33	47	58	66	68	69	73
D7	15	23	25	28	31	33	39	40	40	41
D8	63	75	76	78	79	79	85	89	92	101
D9	74	103	123	137	146	146	152	155	158	163
D10	7	16	24	27	30	33	36	36	36	38
D11	28	50	54	68	79	87	93	97	99	100
D12	14	20	27	33	38	50	60	65	70	75
D13	23	38	61	62	73	80	98	102	110	117
D14	25	48	78	89	98	127	133	142	167	179
D15	44	69	95	106	129	137	162	170	185	210
D16	15	28	39	49	54	60	68	71	75	81
D17	36	74	100	117	145	158	175	189	198	207
D18	11	20	29	31	33	38	40	41	42	43
D19	21	27	32	33	34	35	36	37	38	40
D20	2	4	6	10	12	14	14	14	16	17

equal the observed number of failures for data set Dj in $((k/10)T_j^{\text{tot}}, T_j^{\text{tot}}]$. The prediction errors are

$$e_i(j,k) = \hat{n}_i(j,k) - n(j,k)$$

and the relative prediction errors are

$$r_i(j,k) = (\hat{n}_i(j,k) - n(j,k))/n(j,k)$$
$$i = 1,\ldots,6,\ j = 1,\ldots,20 \text{ and } k = 1,\ldots,9$$

(For the 20 data sets there happens to always be at least one failure in the last interval, so we avoid division by 0.) The prediction errors and the relative prediction errors for model M3 are tabulated in Tables 3 and 4, respectively.

For each super model (M7–M14) we have a selection criterion. Based on these criteria, each one of the super models chooses one of the simple models and makes a prediction. For $i = 7, 8, 9, \ldots, 14$, $j = 1, 2, 3, \ldots, 20$ and $k = 1, 2, 3, \ldots, 9$, let $c_i(j,k)$ equal the index of the simple model (1, 2, 3, 4, 5 or 6) that super model Mi likes the best for data from data set Dj over the interval $[0, (k/10)T_j^{\text{tot}}]$; for example, Table 5 shows the choices made by model M8, which uses Kolmogorov–Smirnov goodness-of-fit as its selection criterion.

TABLE 3
Prediction Errors, $e(j,k)$, for Model M3

Data set	*Percentage of time elapsed*								
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
D1	−22·8	1·0	−8·7	−12·0	−6·4	−1·7	0·8	1·4	0·4
D2	−2·7	−4·7	−11·0	3·6	−1·3	0·5	−0·2	−0·4	−0·3
D3	19·9	−0·8	0·5	−0·2	−2·7	−0·7	0·1	−1·2	−1·1
D4	25·1	21·2	20·0	17·1	9·5	5·2	3·4	1·2	−0·6
D5	242·0	227·0	222·0	26·62	0·8	42·6	23·4	11·2	0·5
D6	−36·5	−16·3	−18·6	−27·1	−2·4	14·8	14·8	5·2	0·1
D7	109·0	10·8	−0·8	−1·6	−1·1	−1·5	3·8	2·4	0·5
D8	54·2	20·6	2·6	−3·9	−8·5	−12·7	−9·1	−7·6	−6·9
D9	−19·5	−1·1	8·7	12·7	12·2	2·6	2·7	0·4	−1·1
D10	32·0	42·0	42·0	16·0	8·8	6·8	6·2	2·3	−0·2
D11	179·0	103·4	−4·1	7·9	11·9	11·2	9·6	6·3	2·5
D12	−53·4	−45·3	−34·4	−27·5	−24·0	−2·9	8·1	2·5	0·7
D13	113·0	−10·8	60·4	−17·7	−11·7	−13·5	5·1	−1·6	−0·4
D14	−115·9	−62·7	71·6	−23·1	−36·5	6·5	−13·5	−17·4	0·8
D15	−17·4	−54·8	−28·2	−48·2	−26·0	−36·3	−14·1	−20·0	−14·7
D16	−0·7	38·1	13·2	13·6	0·7	−0·6	3·1	−0·9	−1·7
D17	−123·1	33·1	4·4	−14·1	16·4	3·9	7·0	6·8	2·2
D18	−9·3	10·7	27·6	6·3	1·8	5·2	3·8	1·9	0·8
D19	3·8	2·7	4·8	1·7	0·2	−0·6	−1·0	−1·1	−1·1
D20	−10·1	−8·1	−4·0	8·0	7·0	6·3	0·9	−1·1	0·1

Next, model Mi predicts that $\hat{n}_i(j,k) = \hat{n}_{c_i(j,k)}(j,k)$ failures will occur for data set Dj over the time interval $((k,10)T_j^{tot}, T_j^{tot}]$. Errors are then computed as before; for example, Table 6 shows the relative errors for model M8.

We wish to compute some summary statistics of how the different models perform over the different data sets, $Dj, j = 1,2,3,\ldots,20$. There does not seem to be any obvious best way to summarize the performance. It seems that relative error is preferred to absolute error in order to prevent one or two data sets with large numbers of failures dominating the summarizing statistics. Also, initially we want to summarize over independent test cases. Therefore we consider average relative errors (averaged over the 20 data sets) for each of the nine elapsed time percentiles. We consider two averages:

The average bias: $$\bar{b}_i(k) = \sum_{j=1}^{20} r_i(j,k)/20$$

The average deviation: $$\bar{d}_i(k) = \sum_{j=1}^{20} |r_i(j,k)|/20$$

TABLE 4
Relative Prediction Errors, $r(j,k)$, for Model M3

Data set	*Percentage of time elapsed*								
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
D1	−0·26	0·02	−0·17	−0·28	−0·20	−0·08	0·06	0·17	0·09
D2	−0·08	−0·18	−0·46	0·27	−0·11	0·06	−0·03	−0·10	−0·14
D3	1·24	−0·05	0·05	−0·02	−0·34	−0·15	0·03	−0·39	−0·57
D4	0·87	1·32	2·50	5·71	3·17	1·72	1·70	0·58	−0·29
D5	1·03	1·23	1·64	0·23	0·01	0·80	0·63	0·47	0·03
D6	−0·63	−0·35	−0·45	−0·68	−0·09	0·98	2·11	1·05	0·01
D7	4·19	0·60	−0·05	−0·12	−0·11	−0·19	1·91	2·44	0·49
D8	1·43	0·79	0·10	−0·17	−0·39	−0·58	−0·57	−0·63	−0·77
D9	−0·22	−0·02	0·22	0·49	0·72	0·15	0·25	0·05	−0·22
D10	1·03	1·91	3·00	1·45	1·10	1·36	3·09	1·14	−0·10
D11	2·49	2·07	−0·09	0·25	0·57	0·86	1·32	2·10	2·53
D12	−0·87	−0·82	−0·72	−0·65	−0·65	−0·12	0·54	0·25	0·14
D13	1·20	−0·14	1·08	−0·32	−0·27	−0·36	0·27	−0·11	−0·06
D14	−0·75	−0·48	0·71	−0·26	−0·45	0·13	−0·29	−0·47	0·07
D15	−0·10	−0·39	−0·25	−0·46	−0·32	−0·50	−0·29	−0·50	−0·59
D16	−0·01	0·72	0·31	0·43	0·03	−0·03	0·24	−0·08	−0·29
D17	−0·72	0·25	0·04	−0·16	0·26	0·08	0·22	0·38	0·25
D18	−0·29	0·47	1·97	0·52	0·18	1·05	1·26	0·96	0·76
D19	0·20	0·21	0·60	0·24	0·03	−0·11	−0·24	−0·37	−0·55
D20	−0·67	−0·63	−0·36	1·14	1·40	2·11	0·29	−0·36	0·08

These two average performance measures are tabulated in Tables 7 and 8, respectively. Finally, succumbing to the temptation to try to quantify the overall behavior of each model, we compute two grand averages:

$$\text{Grand average bias:} \qquad \bar{b}_i = \sum_{k=1}^{9} \bar{b}_i(k)/9$$

$$\text{Grand average deviation:} \qquad \bar{d}_i = \sum_{k=1}^{9} \bar{d}_i(k)/9$$

These grand averages are tabulated in Table 9 for the 14 models.

There are many other ways to summarize data. Musa and co-workers[20,31] used a normalized error (dividing the prediction error by the total number of failures in the data set); they then summarized by considering the median normalized error at each elapsed time point. The interested reader has probably already thought of other variations. One of the open problems in

TABLE 5
Simple Models Chosen by Super-Model M8

Data set	Percentage of time elapsed								
	10%	20%	30%	40%	50%	60%	70%	80%	90%
D1	M1	M1	M6	M6	M6	M6	M6	M6	M6
D2	M6	M2	M2	M1	M5	M5	M5	M5	M5
D3	M4	M6	M6	M6	M6	M3	M3	M3	M3
D4	M2	M6	M2	M2	M2	M6	M2	M2	M6
D5	M6	M6	M1	M2	M2	M2	M2	M2	M2
D6	M6	M5	M5	M3	M2	M2	M2	M1	M5
D7	M6	M6	M6	M4	M4	M4	M3	M3	M3
D8	M4	M2	M2	M2	M2	M4	M4	M4	M3
D9	M2	M3	M3	M3	M3	M3	M3	M3	M3
D10	M1	M6	M1	M6	M6	M4	M4	M6	M6
D11	M1	M6	M6	M6	M6	M1	M3	M3	M5
D12	M5	M5	M1	M1	M1	M4	M3	M2	M2
D13	M1	M2	M1	M6	M6	M6	M3	M4	M4
D14	M6	M3	M2	M3	M6	M1	M3	M3	M5
D15	M6	M2	M5	M5	M5	M5	M5	M5	M5
D16	M6	M5	M2	M3	M6	M6	M3	M2	M2
D17	M5	M1	M1	M2	M3	M3	M3	M3	M3
D18	M6	M5	M5	M2	M6	M2	M2	M2	M6
D19	M6	M4	M3	M6	M6	M6	M6	M6	M6
D20	M6	M2	M3	M1	M4	M4	M2	M6	M6

this research area is to identify the best ways to define and evaluate performance statistics.

INTERPRETATION OF EXPERIMENTS

There is a great temptation to end such an experimental investigation with the conclusion: 'The winner is model...'. That would be very misleading for this type of experiment. The experiment is based on only 20 data sets. The statistical estimation methods, performance measures and general design of the experiment are quite arbitrary; other choices could be made. It is hoped that this experiment gives rough ideas of how reliability growth models perform, what can be expected of them, and of numerous open questions arising about their usage. However, there are several observations that can be made from this experiment.

A cursory glance at the size of the errors in Tables 3, 4, 6, 7, 8 and 9 leads us to the conclusion that the predictions have some value. They are not

TABLE 6
Relative Prediction Errors, $r(j,k)$, for Model M8

Data set	*Percentage of time elapsed*								
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
D1	0·91	1·09	−0·44	−0·62	−0·32	−0·03	0·12	0·20	0·64
D2	−0·76	−0·86	−0·96	1·43	0·06	0·34	0·15	0·04	−0·02
D3	1·24	−0·95	−0·74	−0·91	−0·91	−0·15	0·03	−0·39	−0·57
D4	0·07	0·95	0·23	0·87	−0·43	−0·81	−0·82	−0·93	−0·98
D5	7·38	−0·64	1·64	−1·00	−1·00	0·55	0·28	0·07	−0·30
D6	−0·99	0·22	−0·05	−0·68	0·88	1·16	2·26	1·43	0·28
D7	6·72	−0·99	−0·97	−0·89	−0·93	−0·96	1·91	2·44	0·49
D8	−1·00	−0·89	−1·00	−0·25	−0·45	−0·99	−1·00	−1·00	−1·00
D9	−0·96	−0·02	0·22	0·49	0·72	0·15	0·25	0·05	−0·22
D10	1·03	−0·92	3·00	−0·76	−0·72	1·32	3·00	0·66	−0·92
D11	2·49	−0·85	−1·02	−0·74	−0·30	2·04	1·32	2·10	3·67
D12	−0·63	−0·62	−0·50	−0·48	−0·51	−0·83	0·54	0·28	0·14
D13	1·20	−0·50	1·36	−0·93	−0·64	−0·65	0·27	−0·11	−0·06
D14	−0·96	−0·48	0·72	−0·26	−0·65	0·29	−0·29	−0·47	0·22
D15	−0·96	−0·78	−0·25	−0·44	−0·21	−0·48	−0·16	−0·44	−0·53
D16	−0·77	0·85	0·12	0·43	−0·37	−0·32	0·24	−0·08	−0·29
D17	−0·59	0·62	0·54	−0·48	0·26	0·08	0·22	0·38	0·25
D18	−0·81	1·30	2·66	−0·20	−0·60	0·26	0·27	−0·05	−0·14
D19	−0·65	0·21	0·53	−0·62	−0·75	−0·79	−0·79	−0·79	−0·82
D20	−1·00	−0·93	−0·36	1·14	1·40	1·76	0·05	−0·89	−0·38

TABLE 7
Average Relative Bias of Predictions, $\bar{b}$

Model	*Percentage of time elapsed*								
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
M1	1·56	1·24	1·25	1·32	0·97	0·99	1·38	0·99	0·54
M2	−0·03	−0·23	−0·06	−0·35	−0·39	−0·15	0·05	−0·23	−0·41
M3	0·45	0·33	0·48	0·38	0·23	0·36	0·62	0·33	0·04
M4	0·30	−0·06	−0·01	−0·03	−0·18	−0·29	−0·20	−0·43	−0·55
M5	1·05	0·84	0·91	0·94	0·47	0·58	0·92	0·65	0·31
M6	1·12	−0·15	−0·12	−0·24	−0·44	−0·18	−0·14	−0·08	0·32
M7	0·61	0·21	0·33	0·28	0·09	0·24	0·60	0·28	0·07
M8	0·55	−0·21	0·24	−0·24	−0·31	0·10	0·39	0·06	−0·06
M9	0·25	0·24	−0·04	0·08	−0·07	0·19	0·39	0·38	0·52
M10	1·56	0·64	0·65	0·11	0·06	0·35	0·76	0·49	0·20
M11	0·82	0·49	0·77	0·29	0·10	0·27	0·47	0·50	0·59
M12	0·25	−0·12	0·08	0·17	−0·02	−0·01	0·23	0·27	−0·22
M13	1·56	0·46	0·77	0·13	−0·13	0·19	0·53	0·01	−0·04
M14	0·82	0·11	0·32	−0·17	−0·34	0·15	0·27	0·01	−0·26

TABLE 8
Average Relative Deviations of Predictions, $\bar{d}$

Model	*Percentage of time elapsed*								
	10%	*20%*	*30%*	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
M1	1·66	1·31	1·30	1·41	1·04	1·06	1·41	1·09	0·70
M2	1·03	0·81	0·91	0·67	0·59	0·66	0·66	0·57	0·47
M3	0·91	0·63	0·74	0·69	0·52	0·57	0·77	0·63	0·40
M4	1·03	0·87	0·91	0·90	0·72	0·74	0·79	0·65	0·60
M5	1·39	1·03	1·06	1·12	0·63	0·71	1·00	0·89	0·59
M6	2·25	0·94	0·82	0·84	0·64	0·67	0·56	0·89	1·26
M7	1·13	0·90	0·89	0·80	0·76	0·94	1·26	0·89	0·79
M8	1·56	0·73	0·87	0·68	0·57	0·70	0·70	0·64	0·57
M9	1·22	0·87	0·77	0·66	0·57	0·74	0·78	0·85	1·09
M10	1·66	1·05	1·16	0·61	0·54	0·78	1·05	0·91	0·76
M11	1·48	0·87	1·16	0·63	0·39	0·64	0·77	0·90	1·07
M12	1·22	0·76	0·97	0·91	0·75	0·67	0·72	0·82	0·49
M13	1·66	0·83	1·29	0·75	0·44	0·72	0·83	0·60	0·60
M14	1·48	0·72	1·03	0·65	0·57	0·62	0·75	0·62	0·43

extremely accurate, but they are good enough to be helpful in some situations. It appears that reliability growth estimates may be useful for forecasting future maintenance activities on moderately reliable software. (For a successful application of software reliability growth modeling to certifying software, see Currit *et al.*[37])

This experiment is strongly conditional on the data sets used. If one of the simple models (M1–M6) was truly a superior fitting model, we would expect that model to perform best, and we would expect super models (M7–M14) with good selection criteria to consistently choose that simple model. The fact that this is not happening suggests that none of the simple models is a superior fit. Furthermore, the super model approach is not an improvement. This needs further study, perhaps in a more controlled experiment based on Monte Carlo data.

There is an interesting trade-off between the two-parameter simple models (M1, M2 and M3) and the three-parameter models (M4, M5 and M6). The

TABLE 9
Summary Performance Measures for 14 Models

	M1	*M2*	*M3*	*M4*	*M5*	*M6*	*M7*	*M8*	*M9*	*M10*	*M11*	*M12*	*M13*	*M14*
$\bar{b}$	1·14	−0·20	0·36	−0·16	0·74	0·01	0·30	0·06	0·22	0·54	0·48	0·07	0·39	0·10
$\bar{d}$	1·22	0·71	0·65	0·80	0·93	0·98	0·93	0·78	0·83	0·96	0·88	0·81	0·86	0·76

three-parameter models should generally fit better because they have richer flexibility. However, it is more difficult to fit a higher parameter family, especially with a general-purpose search algorithm like Nelder–Meade; so we may not always be getting the best-fitting member of a three-parameter family. Thus the three-parameter simple models may not be performing as well as they might with a perfect search algorithm.

There is also a trade-off between 'goodness-of-fit' and 'quality-of-prediction'. A model that fits the observed data well is not guaranteed to give good predictions into the future, especially if it is a rich family parameterized with several parameters. (Akaike[29] suggests handicapping parametric families based on the number of parameters.) The idea of an integrated 'prediction system'[7] suggests that 'quality-of-prediction' measures should be used for fitting models. However, in our experiment there is not an obvious difference between the super models based on the two different concepts.

There is an effect caused by how the prediction errors are measured. A model may overestimate the number of future errors by any amount, but it may underestimate the number by at most 100%. This means that a few wild overestimates will hurt the average performance (Tables 7, 8 and 9) much more than wild underestimates (which is probably a more serious error). This may be making model M2 (which tends to underestimate) appear better than it really is and model M1 (which tends to overestimate) appear worse than it really is. A more reasonable error summary might weight an 'underestimate by one-half' equivalent to an 'overestimate by two-fold', for example. There are other possibilities; in fact, how to evaluate and summarize performance is an open question.

Model M3 seems to be doing slightly better than all others. This model also performed the best in Musa & Okumoto's[20] performance studies using different summary statistics (median normalized prediction errors). Nagel[38,39] and others have observed a log-linear pattern among occurrence rates of bugs in a program and hypothesize that this may be a frequently occurring pattern; Miller[18] has shown that this pattern is approximately modeled by model M3. Another interesting fact is that model M3 plays a central role among the models M1–M5; see Miller.[18] So it is not surprising that model M3 scores best in Table 9. But, of course, it is all conditional on the 20 data sets.

Phillips (see Adams[40]) has observed that the occurrence rates for bugs in some large operating systems show a power law pattern which is equivalent to model M1 (see Miller[18]), but model M1 does not perform well for our 20 data sets. For different sets of data the performance of M1 and M3 might be reversed. This is why we want the freedom to pick the best model for each piece of software. These experiments imply either that this is impossible or that we have not figured out how to do it yet.

Looking at the prediction errors at the 90% elapsed time point in Tables 3, 4 and 6 reveals moderately sized errors. This should be a fairly easy prediction problem: we are predicting for a future time interval equal in length to $\frac{1}{9}$ of the previous observed interval. From these moderately sized errors, we conclude that it is not reasonable to ask these reliability growth models to accurately predict that software will perform error free for long future time intervals.

NON-APPLICABILITY TO SAFETY-CRITICAL SOFTWARE

Safety-critical software must be extremely reliable. The question is how to achieve extremely high levels of reliability. The reliability growth scenario would start with faulty software. Through execution of the software, bugs are discovered. The software is then modified to correct for the design flaws represented by the bugs. Gradually the software evolves into a state of higher reliability. There are at least two general reasons why this is an unreasonable approach to highly reliable safety-critical software. The time required for reliability to grow to acceptable levels will tend to be extremely long. Extremely high levels of reliability cannot be statistically guaranteed *a priori*.

For a discussion of the limitations of the statistical approach to high reliability, see Miller.[4,5] For a good discussion about the reliability growth scenario, see Gray.[16,17] Gray points out many aspects of reliability growth, some of which are difficult to quantify and thus ignored by the usual reliability growth models; ignoring these aspects may not lead to unacceptable results when dealing with nominal levels of reliability, but they cannot be ignored when dealing with extremely high levels of reliability. See Hamlet[41,42] for discussion of some additional complications.

CONCLUSIONS

We conclude that reliability growth models are useful for predicting the number of failures over finite future time intervals when we are dealing with software which is of low or moderate reliability. Maintenance of large moderately reliable software systems might be usefully predicted by these models.

There are numerous open questions about software reliability growth models. When making predictions into the future, it is very important to use a good model; how to choose the best model is an open question. The quantification of prediction errors (by confidence intervals or other

methods) is yet to be solved. The best ways to evaluate performance of models have not been identified.

Our experiment shows that an apparently reasonable way to improve reliability growth modeling prediction based on super models results in no improvement. This may be due to the particular data sets we used or to other factors mentioned in the paper. A controlled Monte Carlo study may be useful in answering these questions. Regardless, the experiment reveals some of the problems arising in reliability growth modeling.

Through this experiment and the errors calculated, we have tried to convey a rough idea of how well software reliability growth models perform.

ACKNOWLEDGEMENTS

P. A. Keiller thanks Fred Waters for many helpful discussions. D. R. Miller gratefully acknowledges research support from the National Aeronautics and Space Administration, Grant NAG 1-771.

REFERENCES

1. Eckhardt, D. E. & Lee, L. D., A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transactions on Software Engineering*, SE-12 (1985) 1511–16.
2. Littlewood, B., A reliability model for systems with Markov structure. *Applied Statistics*, 24 (1975) 172–7.
3. Littlewood, B. & Miller, D. R., Conceptual modelling of coincident failures in multiversion software. *IEEE Transactions on Software Engineering* (in press).

3a. Iannino, A., Musa, J. D., Okumoto, K. & Littlewood, B., Criteria for software reliability model comparisons. *ACM Sigsoft Software Engineering Notes*, 8(3) (1983) 12–16.

4. Miller, D. R., Making statistical inferences about software reliability. 1986 Joint Statistical Meetings Invited Paper, Chicago, August 1986. Available as CR-4197, National Aeronautics and Space Administration, December 1988.
5. Miller, D. R., The role of statistical modeling and inference in software quality assurance. In *Software Certification*, ed. Bernard de Neuman. Elsevier Applied Science, London, 1989, pp. 135–52.
6. Jelinski, Z. & Moranda, P. B., Software reliability research. In *Statistical Computer Performance Evaluation*, ed. W. Freiberger. Academic Press, New York, 1972, pp. 465–84.
7. Abdalla-Ghaly, A. A., Chan, P. Y. & Littlewood, B., Evaluation of competing reliability predictions. *IEEE Transactions on Software Engineering*, SE-12 (1986) 950–67.
8. Dale, C. J., Software reliability evaluation methods. ST-26750, British Aerospace, September 1982.

9. Dale, C. J., Software reliability models. In *Software Reliability: State of the Art Report 14:2*, ed. A. Bendell & P. Mellor. Pergamon Infotech, London, 1986, pp. 31–44.
10. Farr, W. H., A survey of software reliability modeling and estimation. AD-A154.874, Naval Surface Weapons Center, Dahlgren, VA, 1983.
11. Goel, A. L., Software reliability modelling and estimation techniques. RADC-TR-82-263, Rome Air Development Center, Griffiss Air Force Base, New York, 1982.
12. Littlewood, B. & Verrall, J. L., A Bayesian reliability growth model for computer software. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **22** (1973) 332–46.
13. Littlewood, B., Stochastic reliability-growth: a model for fault-removal in computer-programs and hardware-designs. *IEEE Transactions on Reliability*, **R-30** (1981) 313–20.
14. Duane, J. T., Learning curve approach to reliability monitoring. *IEEE Transactions on Aerospace*, **AS-2** (1964) 563–6.
15. Crow, L. H., Reliability analysis for complex repairable systems. In *Reliability and Biometry: Statistical Analysis of Lifelength*, ed. F. Proschan & R. J. Serfling. SIAM, Philadelphia, PA, 1974, pp. 379–410.
16. Gray, C. T., Superposition models for reliability growth. PhD thesis, University of Birmingham, 1985.
17. Gray, C. T., A framework for modelling software reliability. In *Software Reliability: State of the Art Report 14:2*, ed. A. Bendell & P. Mellor. Pergamon Infotech, London, 1986, pp. 81–94.
18. Miller, D. R., Exponential order statistic models of software reliability growth. CR-3909, National Aeronautics and Space Administration, July 1985. (Abridged version: *IEEE Transactions on Software Engineering*, **SE-12** (1986) 12–24.)
19. Miller, D. R. & Sofer, A., A nonparametric approach to software reliability using complete monotonicity. In *Software Reliability: State of the Art Report 14:2*, ed. A. Bendell & P. Mellor. Pergamon Infotech, London, 1986, pp. 183–95.
20. Musa, J. D. & Okumoto, K., A logarithmic Poisson execution time model for software reliability measurement. *Proceedings of the 7th International Conference on Software Engineering*. IEEE Computer Society Press, Washington, DC, 1984, pp. 230–8.
21. Musa, J. D. & Okumoto, K., A comparison of time domains for software reliability models. *Journal of Systems and Software*, **4** (1984) 277–87.
22. Goel, A. L. & Okumoto, K., Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, **R-28** (1979) 206–11.
23. Okumoto, K., A statistical method for software quality control. *IEEE Transactions on Software Engineering*, **SE-11** (1985) 1424–30.
24. Nelder, J. A. & Mead, R., A simplex method for function minimization. *Computer Journal*, **7** (1965) 308–13.
25. Chan, P. Y., Software reliability prediction. PhD thesis, City University, London, 1986.
26. Brocklehurst, S., Private communication, 1988.
27. Keiller, P. A., Littlewood, B., Miller, D. R. & Sofer, A., Comparison of software reliability predictions. *13th International Symposium on Fault-Tolerant*

Computing, Digest of Papers. IEEE Computer Society Press, Washington, 1983, pp. 128–34.
28. Dawid, A. P., Statistical theory: the prequential approach. *Journal of the Royal Statistical Society, A*, **147** (1984) 278–92.
29. Akaike, H., *Prediction and Entropy*. Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, USA, June 1982.
30. Littlewood, B. & Keiller, P. A., Adaptive software reliability modelling. *14th International Symposium on Fault-Tolerant Computing, Digest of Papers*. IEEE Computer Society Press, Washington, 1984, pp. 108–13.
30a. Chan, P. Y., Adaptive models. In *Software Reliability: State of the Art Report 14:2*, ed. A. Bendell & P. Mellor. Pergamon Infotech, London, 1986, pp. 3–18.
31. Musa, J. D., Iannino, A. & Okumoto, K., *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, 1987.
32. Musa, J. D., Software reliability data. Data and Analysis Center for Software, Rome Air Development Center, Rome, New York, 1979.
33. Sukert, A. N., A software reliability modeling study. Rome Air Development Center, Technical Report RADC-TR-76-247, Rome, New York, 1976.
34. Musa, J. D., Private communication, 1988.
35. Miller, A. M. B., A study of the Musa reliability model. MS thesis, University of Maryland, College Park, MD, 1980.
36. Moek, G., Comparison of some software reliability models for simulated and real failure data. 4th IASTED (International Association of Science and Technology for Development) International Symposium and Course 'Modelling and Simulation', Lugano, Italy, 21–24 June 1983.
37. Currit, P. A., Dyer, M. & Mills, H. D., Certifying the reliability of software. *IEEE Transactions on Software Engineering*, **SE-12** (1986) 3–11.
38. Nagel, P. M. & Skrivan, J. A., Software reliability: repetitive run experimentation and modeling. NASA CR-165836, 1982.
39. Nagel, P. M., Scholz, F. W. & Skrivan, J. A., Software reliability: additional investigations into modeling with replicated experiments. NASA CR-172378, 1984.
40. Adams, E. N., Optimizing preventive service of software products. *IBM Journal of Research and Development*, **28** (1984) 2–14.
41. Hamlet, R. G., Probable correctness theory. *Information Processing Letters*, **25** (1987) 17–25.
42. Hamlet, D. & Taylor, R., Partition testing does not inspire confidence. *Proceedings of Second Workshop on Software Testing, Verification, and Analysis*. IEEE Computer Society Press, Washington, DC, 1988, pp. 206–15.

Appendix 3

M. Lyu, H. Hecht, H. Kopetz, D. Miller, J. Musa, M. Ohba, and D. Siefert, "Research and Development Issues in Software Reliability Engineering," **Proceedings of the IEEE International Symposium on Software Reliability Engineering** (1991); 80-89.

(Reprinted in **Software Engineering Notes 16,2** (1991): 23-30.)

PANEL: RESEARCH AND DEVELOPMENT ISSUES IN SOFTWARE RELIABILITY ENGINEERING

Panel Chair:	*Michael Lyu*	(University of Iowa)
Panelists:	*Herbert Hecht*	(SoHaR Inc.)
	Hermann Kopetz	(Technical University of Vienna)
	Douglas Miller	(George Mason University)
	John Musa	(AT&T Bell Labs.)
	Mits Ohba	(IBM Corporation)
	David Siefert	(NCR)

Introduction

Michael R. Lyu, University of Iowa

Computers are bringing revolutionary changes to our life with their involvement in most human-made systems for sensing, communication, control, guidance and decision-making. As the functionality of computer operations becomes more essential and complicated in the modern society, the reliability of computer software becomes more important and critical.

Research activities in software reliability engineering have been vigorous in the past 20 years. Numerous statistical models have been proposed in the literature for the prediction and estimation of software reliability, and many research efforts and paradigms have been conducted for the design and engineering of reliable software. However, there seems to be a gap in between the achievements of software reliability research and the results from software reliability practice. We keep on hearing troublesome software projects, horrible software failures, and misconceptions in software reliability applications.

It is the purpose of this panel to bring together researchers and practitioners of this field to discuss the software reliability problems which will have tremendous impact to our daily life. The panel is expected to raise research and development issues under this concern, to address existing and potential problems, to resolve some misunderstandings and conflicts, and to reach a fundamental basis for the advancement of this field.

The panelists are invited to discuss those topics including, but not limited to, the following:

(1) What are the most urgent needs for software reliability practitioners?

(2) What kind of issues practitioners would like researchers to pursue?

(3) Did practitioners get satisfactory results from software reliability researchers?

(4) What are the most challenging software reliability issues researchers are facing today?

(5) Did researchers gain enough support to perform software reliability research?

(6) What kind of inputs or feedbacks researchers are seeking from practitioners?

(7) What practices should be developed and conducted based on the current research results?

(8) What is the gap in between software reliability modelers and measurers? How to abbreviate it?

(9) What kind of multi-institutional efforts have been, or should be conducted for acquiring software reliability standards, handbooks, benchmarks, database, tools, etc.?

The following sections consist the position statements written by each panelist under the panel title and the suggested topics.

TH0336-5/91/0000/0080$01.00 © 1991 IEEE

Quantitative and Qualitative Concepts

Herbert Hecht, SoHaR

For Project Managers the reliability of the computing function as a whole is of primary concern, and for that purpose a combined quantitative hardware/software reliability expression is required. The responsibility for hardware and software functions is frequently separated immediately below the project management level, and therefore the project manager also needs separate models for allocating and controlling the achievement of adequate reliability. For these purposes broad statistical reliability metrics are suitable, particularly failures per unit time of computer usage or time unit loss of computer availability due to failures. Examples: failures per CPU-hr or outage-hrs per month.

The software manager is responsible for achieving the statistical reliability goals but in order to know where and how to improve the reliability more specific measurements are required. Quantitative approaches have so far been only of limited use in this domain. Audits, employment of software development and test tools, and test planning are largely guided by purely qualitative considerations. Therefore there exists at present no consistent methodology that permits the software manager to meet the quantitative requirements imposed by systems considerations with the tools at their disposal.

Two activities can bring about a connection between the quantitative and qualitative approaches, and can provide sorely needed advances toward achieving more reliable software. The first activity is the quantitative analysis of failures in terms of software development and test techniques that could have prevented them. The resulting data, particularly if they are weighted by severity of the failure, can provide the software manager with concrete information on the means of improving the reliability of his/her product.

The second step deals with the use of quantitative data as a test termination criterion. The present practice of ending test on the basis of schedule, budget, or (in the very best cases) attainment of a period of failure free operation, provides little useful feedback to the team that developed the software or for the test planning in other projects. Reliability growth measurement during formal test will permit termination on demonstration of a defined reliability level and will also provide insights into the effectiveness of different development and test methodologies.

I will present examples of these integrated practices.

Reliability of Real Time Systems

Hermann Kopetz, Technical University of Vienna

Since my background is in the area of fault-tolerant distributed real-time systems, my view is determined from this position.

In hard real-time systems, i.e., systems where a failure can have catastrophic consequences, a result must be correct, both in the domains of value and time. Since the behavior in the domain of time depends on the properties of the underlying hardware, an integrated software/hardware view has to be taken. The functional correctness of the software per se (i.e., correctness in the value domain) is not sufficient.

Many failures of real-time systems are related to synchronization and performance errors which manifest themselves as 'transient' system failures. In a failure statistics of a complex real-time system [Gebman 1988], it is recorded that less than 10% of the failures observed in the operation of the system can be reproduced within the sophisticated test environment. Similar results have been reported by other manufacturers of real-time systems. This implies that we do not fully understand the character and the interactions of the execution sequences which unfold over time in complex real-time systems and do not know how to build effective test procedures.

This problem has to be attacked from the perspective of design. We have to build real-time architectures that are easier to reason about. Most of the present day real-time systems are event triggered, i.e., as soon as an event occurs, the computer system takes a decision whether to process the task associated with this event immediately or the delay processing until sometimes later. These dynamic scheduling decisions can take a significant amount of processing time, which is then not available for the application software. Every different order of the events can give rise to a different scheduling decision and thus to a different execution sequence. The potential input space of event-triggered systems is enormous. It is difficult to reproduce an input scenario because the exact timing of input cases cannot be controlled easily. There are no methods known which can be applied to reason formally about the timing behavior (i.e. the performance) of complex real-time systems.

If we introduce a time-granularity in the system operation by looking at the events only at predefined points in the time domain (i.e., a time triggered architecture), the plurality of input cases can be substantially reduced. Furthermore, static scheduling strategies become feasi-

ble. The system structure will be more regular, i.e., more predictable and easier to understand and test. The price paid for this reduction in complexity is a reduced flexibility.

We feel that in the field of real-time systems every effort must be made to make the system clear and understandable. In our research on distributed real-time systems [Kopetz 1989] this has always been our primary goal. We have found that time-triggered real-time software is inherently easier to understand and test than event-triggered software. Further research efforts in this area seem to be well justified.

Statistical Issues in Software Reliability Engineering Research and Development

Douglas R. Miller, George Mason University

There are two major issues concerning software reliability: achievement and assurance. They are both very important. Obviously, software in critical applications must achieve high reliability in order for the system to function safely. But it is also necessary to have strong "a priori" assurance that the software is highly reliable before it can be put into use. For example, without reasonable assurance that high reliability has been achieved, flight critical avionics software in commercial aircraft should not be certified for public use.

So, the central focus of Software Reliability Engineering R&D is methodologies for achieving and assuring required levels of software reliability. The goal is reliable software. How do you do it? How do you know when you've done it? Furthermore, what are the most efficient ways to achieve and assure the reliability?

A central idea concerning reliability is "uncertainty." A given piece of software may or may not contain design flaws which will manifest themselves as system failures when the software is used at some time in the future. The point is that uncertainty is inherent to this phenomenon: we do not know if failures will happen and, if they do, when they will happen. To deal with this uncertainty, a scientific approach should be taken. The scientific approach involves experimentation, data collection, statistical modelling and analysis, and drawing inferences and conclusions which will support decisions about developing, testing and using software. The existence of probability seems inevitable here. It is necessary to quantify the uncertainty in terms of probabilities of various events occurring.

Based on information or data concerning software development, testing, previous failures, the usage environment, and any other observables, we would like to estimate (with confidence) the probability that a particular piece of software fails during a given time interval.

Reliability growth models attempt to estimate current reliability and predict future reliability growth for a given piece of software. These models base their estimates and predictions only on past failure times of the given piece of software. IBM's Clean Room used reliability growth models successfully. At the May 1990 Meeting of the IEEE Subcommittee on Software Reliability Engineering, successes were also reported by AT&T, HP and Cray Research. Unfortunately, the reliability growth modelling approach is limited in many ways: The models treat the software as a black box and are only valid for random batch (memoryless) testing or usage. The distribution of usage must be well know. The models do not make use of additional data or information which comes out during testing or usage. The approach does not give useful estimates for extremely high levels of reliability (e.g., avionics software and other safety-related systems).

There are many factors which contribute to the reliability of a piece of software. Case studies such as those sponsored by NASA Goddard's Software Engineering Laboratory explore the effect of various factors on software quality. Factors of interest include different development scenarios, different testing strategies, characteristics of programmers, and others. It can be shown that software quality correlates with various known factors, but calculating reliabilities from these factors seems difficult if not impossible. One very important category of information which should have significant value in predicting reliability of a piece of software is the programmer's personal subjective estimate of its reliability, especially after he has seen and done a post mortem on the first few bugs discovered.

Current practice is often based on engineering judgement. For example, commercial avionics software must be produced following guidelines presented in DO-178A, "Software Considerations in Airborne Systems and Equipment Certification," prepared by Special Committee 152 of the RTCA and currently under revision by Special Committee 167. If appropriate documentation supports compliance, the FAA certifies the software. The actual software is never examined as part of the certification. A major challenge facing the discipline of Software Reliability Engineering involves justifying this

type of approach (also contained in various Military Standards) in some objective, scientific sense.

To summarize: i)For certain classes of software projects, quantitative reliability estimation and prediction is possible (and is done) for individual programs. ii)Through general case studies it is possible to identify factors effecting reliability and thus a get qualitative sense of what constitutes good software development practice. iii)For many critical software systems requiring high reliability, the approach to reliability is very subjective.

It is clear that a quantitative, objective approach to software reliability should be applied to more software projects. This means going beyond the current practice of software reliability growth modelling. The key seems to be: It is necessary to use available data much more efficiently (and imaginatively). There are two categories of data sources: Additional data can be collected (and used) specific to any particular piece of software whose reliability is being assessed. More importantly, there is data from similar and related pieces of existing software; I don't think we know how to make effective use of this data.

The goal is better quantitative understanding (and exploitation of that knowledge) of many software phenomena: behavior of real-time control systems, intricacies of fault-tolerant systems, efficacy of testing, identification of usage distributions, etc. All this knowledge is related to classes of software. (It is necessary to understand more than single software systems individually, one at a time.) Software metrics must be a key feature in this general quantitative understanding, because the similarity between pieces of software must be measured in order to define classes of software.

To progress it is necessary to acquire data. An ideal (but expensive) source is controlled experimentation. For example, NASA Langley continues to sponsor experiments where replicated software is written. A better understanding of replicated batch-processing software has emerged from such experiments. Current experiments should improve understanding of replicated real-time control software. A second general source of data are real software projects. A prime example is the data collected and published by Musa; his data stimulated a flurry of activity in reliability growth modelling. Such experimentation and data collection is crucial. Experimenting and collecting useful data across general classes of software projects is a tremendous challenge.

The Software Reliability Gap: An Opportunity

John D. Musa, AT&T Bell Labs.

We are in the middle of both a problem and an opportunity. I like to call it the "software reliability gap" because the needs of software customers have outrun the current *practice* of software engineering. You can't tell whether they have outrun the technology, because there is much technology that hasn't been refined and applied.

The core of the problem is that intense international competition has made unidimensional needs obsolete. If we only needed to add reliability to software products, we would have many tools and methodologies to help us. The problem is that other customer requirements, such as level of cost and delivery date, would not be met. Customers have *multidimensional* needs that are interdependent and hence must be set and met more precisely than ever before. The precision required can only increase in the future.

Thus measurement is inevitable. Models are also inevitable; we need to know the factors that influence product attributes and how much each of them does, so that the software development process can be controlled to yield the desired objectives for the attributes. In short, competition is creating a technological vacuum or gap.

The principal quality attributes that customers cite as being significant are reliability, cost, and delivery date. Software reliability engineering is the last to develop of the three technologies supporting the measurement and modeling of these attributes. It is the keystone that makes quantitative software quality engineering possible. Since quantitative hardware quality engineering already exists, the development of software reliability engineering also makes quantitative *system* quality engineering possible.

Thus there is an enormous and rare opportunity to fill a widening gap, which makes this an exciting and challenging time.

What must software reliability engineering do to meet the challenge? In my opinion, several general things:

(1) We need to induce a variety of projects to try it. This is already happening, but greater variety would be useful. Care must be taken that it be applied correctly.

(2) The experience on these projects must be recorded, critiqued by others knowledgeable in

the field (to guard against misinformed applications), and published.

(3) Published experience should be organized and digested, so it can be more easily taught to practitioners and future practitioners.

(4) Problems that are blocking further progress and opportunities for new areas of application need to be identified, and they should be addressed by researchers.

These activities clearly offer major possibilities for practitioners, researchers, and educators. People who acquire and use software play an important role in clarifying the needs of the customer that are at the core of the driving forces acting on software reliability engineering.

Can I say anything more specific? I would like to close by entering brainstorming mode and throwing out some thoughts for you to discuss:

(1) We need research to tie software reliability more strongly to the earlier part of the development process. Part of this effort involves determining how fault density is affected by product and process variables.

(2) Little has been done to fulfill the promise of software reliability engineering for evaluating software engineering methodologies and tools. We need to help people do this.

(3) We need data on human and computer resource usage in test, so that resource usage parameters can be determined.

(4) The AIAA software reliability engineering guidelines effort, which includes development of a handbook, looks promising. Because of the diversity of contributors involved, it will be important to devote much effort to interaction between and integration of their views. We don't want a catalog.

(5) We need to strongly support our newsletter and our conference through personal participation in exchanging practical experience and research results. We need to keep the exchange flowing all year through our working committees.

(6) We need software tools (with as many generic elements as possible) to record as large a proportion of failures as possible automatically, particularly in the field but also in test. We need to integrate this system with manually-reported failure systems, but consider implementing the manual reporting online rather than on paper.

(7) The Software Engineering Institute has a methodology for assessing the quality level of software development processes. It does not currently directly include a software reliability engineering program among its assessment criteria. It should, and we should discuss with them how to add it.

I hope you will not only discuss these ideas here, but chew on them later as well. I hope you will add to this necessarily partial list of opportunities for action. I hope you will then seize some of them that appeal to you, and return as significant contributors next year or the year after.

Software Reliability Engineering from Japanese Perspective

Mits Ohba, IBM Corporation

"The wave comes from the East."

Both the computer technology and the quality control method were invented and matured in the US, and they were brought into Japan later. Japan has so far caught up quickly and become competitive in both areas. Especially, Japan is viewed as the leader in the area of quality control and quality management.

"Technology transfer begins when it is imported."

If we carefully review the processes by which Japan has caught up and gone further, we can find some similar patterns of technology development. The processes generally begin at the importing phase where technology is investigated and evaluated. Then there is the deployment phase, the migration phase, and finally, the Japanization phase.

"How does it go through?"

The deployment phase is the phase where the imported technology is widely used and the know-hows associate with it are accumulated. The migration phase is the phase where components of the technology are adjusted for the target environment(s). The Japanization phase is the phase where something additional and unique to Japan is added to the technology.

"How has Japanese software engineering evolved?"

Software engineering is a case in point. It was introduced into Japan in 1977, which was two years later

than the first IEEE Transaction on Software Engineering issued. Two years were spent on the importing phase followed by two years of deployment. The migration phase began in 1982 and lasted six years. The Japanization phase began in 1988. An example of the Japanization phase is what has become known as the "Software Factory" concept.

"Software reliability research is not an exception."

As a domain of research, software reliability engineering is not an exception to the Japanese process. The earlier work done in the US by Musa, Goel and Okumoto drew the attention of Japanese reliability researchers as their new field of study.

"What have Japanese researchers done in this field?"

To date they have: 1) evaluated the basic models proposed by the American researchers by applying them to real project data, 2) modified the models in order to fit the data, 3) developed new models by examining the implication of data and the assumptions of the basic models, and 4) addressed the new research issues of models to be resolved.

"Software factory did not need theories."

On the other hand, software reliability engineering as a practice has evolved differently. It was begun as a branch of software quality control practices in order to determine whether a product developed by a vendor was acceptable. The logistic curve model and the Gompertz curve model were widely used in the industry and became de facto standard models for software factories.

"Technology transfer is really the problem."

The implementation of the theory which has been developed by Japanese researchers is very slow. This is because the old models, with which the practitioners are familiar, are still sufficient for their needs. They will not change as long as the old practices work or until they recognize the advantages of the new theory. This is similar to the fact that people had believed the stars were rotating.

"How can we convince the people that the earth rotates?"

The most serious issue of software reliability engineering as a practice in Japan is the education of the people. It is similar to teach them that the earth rotates, not the stars. The models are not crystal balls. Prediction is made based on a set of assumptions. If the assumptions are not valid, a model based on them becomes a great nonsense. The Gompertz curve fits most of practical project data because of its flexibility. But, no one can explain what the model really means.

"Why do we believe that the earth is rotating?"

The most serious issue as a domain of research is to explain the relationship between test cases and reliability growth using reasonable models, which is also similar to explain the reason why the earth seems to be rotating. What software reliability growth tells is characterization of the state of software under evaluation. It does not tell how we can improve testing. Obviously, time is not the real factor for improving software reliability during the test phase.

"Can measurements and data be standardized?"

A serious issue for both practitioners and researchers is to establish standard ways of measuring software reliability in practice. The models are based on a set of assumptions. The models should be categorized based on 1) what they can predict (e.g., MTTF, number of errors), 2) what type of data they need (e.g., time between failures, number of failures between observations), 3) what assumptions they are based on, and 4) what type of software they can analyze.

Back To The Future

David Siefert, NCR

For the past 20 years, Software Engineering has provided us with the capability for producing highly reliable software. Software reliability is achieved, in part, through the applied discipline of standardized practices, methodologies, tools, and processes comprising the "science" of Software Engineering. Today, dependence on automation is greater than at any point in time in the world's history. Highly reliable products are expected and assumed! The very nature of the level of sophistication and complexity of modern systems are intended to be transparent to the end-user.

Applying Software Reliability Engineering Disciplines

Interestingly, the same practices, methodologies, etc. that lead to the development of reliable software are also the downfall! Why after all these years of "learning" is the world still not applying and improving Software Engineering disciplines etc.? Why do practitioners still develop and maintain software based upon the

approaches used 20 years ago (lack of applied discipline)? Why is it that researchers do not yet know exactly what is the minimum that should be done to develop reliable software? In support of consistently producing reliable software, why after 20 years is there still not a national database leading to the consistent project data collection, analysis, and ultimate determination of practices, tools, and therefore required disciplines? Shouldn't a Software Engineering "Bluebook" exist?

Software Reliability Engineering is addressed in the following two ways:

(1) Technical Aspects of Software Reliability

Technical software reliability consists of many items. Determining reliability goals is one activity. Reliability goals are typically referred to in "technical" terms. These technical terms are placed in product specifications. As it pertains to Software Reliability Engineering, these terms or goals are then tracked through product production to the achievement of the goals. The environment that the software was produced in, plays a significant impact on the results. These specified reliability goals often are determined through the application of software reliability models. An AIAA effort addressing Software Reliability is in the process of providing guidance to industry on which models to use and when. The computing industry has yet to standardize these specific models.

(2) End-User Software Reliability

The second form of Software Reliability Engineering is that of the end-user. The technical specifications which include the software reliability goals are expected to be mapped directly to the end-user's needs and expectations. Too often there is no known methodology to take qualitative and rather subjective unstructured feedback from the end-user and transform them into quantifiable and technically oriented input for use in determining software reliability. Without this methodology, there will remain to be software reliability difficulties. Meeting "specification" infers meeting the end-user's expectations. Meeting specification is certainly one essential form of measurement. Technical specifications are the result of analysis of the end-user's expectation - not the other way around. Too often the technical specification and the end-user's expectations are distinctly separate with no relationship between each other. This results in minimal confidence that the product will achieve it's expectations.

Environmental issues are also important. To understand software reliability, one must understand the environment software resides. The environment for software is systems! System components include other software and hardware. Reliability should be computed or budgeted in such a manner that reliability for each of the components of the computer environment can be determined, evaluated, measured, and tracked separately Reliability should also address a "total" system or enterprise-wide solution. Typically, the end-user is affected by using or experiencing the "total" system They typically have no ability to decipher the type o defect or anomaly that has occurred. It is not clear tha they should. At any rate, Software Reliability Engineering needs to address the "total" system as well as the individual system components.

The Software Engineering community has reliability models that lead to establishing reliability goals. "High Confidence" goals (outputs) produced through the use of these models are dependent upon past history. This history should be retained in the form of a database. Interestingly, no new significant software estimation models have been revealed in the past 5 years. Without the use of such databases as input to and the "tuning" of such models, the community is no closer to estimating with high confidence levels the goals produced from the models as was able to be attained 5 years ago. The goals produced through the use of these models may not be any better than the "guess" of you or I.

Besides past history, the technically specified software reliability goals are established and dependent on some basic items of information:

- How is end-user's "needs" quantified?
- What is a software error, fault, and failure?
- What are the categories of software?
- How is Defect and Fault Density computed?
- What and how is line-of-code or Function Point, by language, determined?
- How is line-of-code or Function Point translated between languages?
- How is Defect Density affected by software production environmental issues?
- How is software to be tracked?

Recommendations in Improving Software Reliability

- For Practitioners:

(1) Practitioners must apply the disciplines considered to Software Engineering. Techniques, methods, tools, etc. as associated with planning, design, development, testing (including verification and validation), should be learned and rigidly applied.

(2) Each software production (or maintenance) organization should develop and maintain a Software Engineering Environment Process (SEEP). This process should consist of all disciplines, tools, etc. actually used in the production of the software - including the measurement systems, of which software reliability is a part.

(3) Practitioners should develop a database of past projects. The database should consist of such information as: the environment that produced the software, skill and types of personnel producing the software, Defect Densities, etc. This database is to be used as a basis for a Software Reliability Measurement Program (SRMP) and positioning for continuous improvement in Software Engineering.

(4) A software reliability measurement program (SRMP) should be put into place that consists of measures that address both the scope of the Software Engineering Environment Process and specific product related results. Measures should consist of indicator measures, e.g., Test Coverage and estimator measures - models to estimate reliability. The measurement program should consist of a methodology that addresses the use of the models beginning with the "how to" develop reliability goals and ending with an approach of a project post mortem. The previously mentioned database would maintain all data. The database would provide for causal root cause analysis and process improvement of the Software Engineering Environmental Process.

- For Computer Scientist Researchers:

(1) Researchers are to develop and maintain a national database (see above). The information contained in the database as previously noted should contain both product and environmental information. Researchers should evaluate the information in such a manner as to determine the best practices, methods, required skills etc. to continuously improve software reliability.

(2) Researchers should provide standards on such subjects as: language constructs, line-of-code definitions, Function Point, etc.

(3) Researchers should determine minimum impacts as to how to conclude with deriving "high confidence" software reliability goals, etc. Models are to be evaluated and maintained.

(4) Researchers should also determine education curricula for software engineering enabling the continuous achievement of high confidence reliable software.

(5) Researchers should determine how to quantify results from evaluating user's needs. These results are used as input into various different reliability tools, models, etc. as discussed earlier.

(6) Researchers should establish and maintain a "Blue Book for Software Engineering."

Concluding Comments

The world continues to embrace higher and higher levels of technology. Software is at the heart of the demand for complex features and functions which are packaged to make the complexity transparent to the end-user. High confidence software reliability is in jeopardy. Software Engineering processes that consist of disciplines, tools, methods, etc. are not being utilized consistently. The science of Software Engineering is not being practiced.

A need exists to focus on the basics; in the simplest form of understanding software and Software Engineering. Data needs to drive decisions. Attaining highly reliable software - consistently - positioned through processes for the purpose of improvement is essential. Researchers need to provide the "data driven" credibility in the baseline evaluations of software and software environments (and processes). Researchers need to see that the appropriate Software Engineering disciplines are applied - consistently and appropriately, evaluating the results, and improving the disciplines and processes.

The disciplines exist in the form of Software Engineering to produce reliability software! The discipline and formality required to achieve the results remain to be the challenge! The solution is: *"go BACK and apply the discipline TO get to THE FUTURE..."*