CHAPTER 3

APPLICATION OF BASIC AND LOGARITHMIC POISSON EXECUTION TIME MODELS IN SOFTWARE RELIABILITY MEASUREMENT

John D. Musa
Kazuhiro Okumoto
AT&T Bell Laboratories
Whippany, NJ 07981
U.S.A.

ABSTRACT

Two software reliability models that share the advantageous property of being based on execution time are presented. They are compared with a number of other published models. Predictive validity is established using sets of failure data from a varied group of software projects and two different parameter estimation methods. The characteristics and advantages and disadvantages of the two with respect to each other are discussed.

1. BASIC CONCEPTS

Software reliability is defined as the probability that a program will operate without failure for a specified time in a specified environment. It is a function of the inputs to and use of the system as well as the existence of faults in the software. A failure is an unacceptable departure of program output from requirements. It is distinguished from a fault or the software defect that causes a failure. The foregoing definition of
software reliability is an operational one; it offers the greatest utility to software engineers and managers, since it directly measures the impact on the user of a system.

The term "unacceptable" implies that the user must determine what is considered to be a failure; this usually depends on the effect of the particular behavior of the system in question on the user's operation, costs, etc. In fact, the situation is often more complex than "acceptable" or "unacceptable": the user may wish to establish several classes of failures of differing severities and define reliability requirements for each class.

There are two alternative ways of expressing software reliability. The failure intensity is the expected number of failures per unit time. The mean time to failure (MTTF), if it exists, is defined as the expected value of the failure interval.

Measurement of software reliability involves estimation of software reliability or its alternate quantities from failure data. The term software reliability prediction is defined [1] as the process of computing software reliability parameters from program characteristics (not failure data). Typically, software reliability prediction takes into account factors such as the size and complexity of a program, and it is normally performed during a program phase prior to test.

The principal objective of a software reliability model is to forecast failure behavior that will be experienced when the program is operational. This expected behavior changes rapidly and can be tracked during the period in which the program is
tested. Reliability or MTTF generally increases as a function of execution time.

2. TWO SELECTED SOFTWARE RELIABILITY MODELS

The first work in software reliability dates to about 1967. Since then, a number of different models have been developed (see [2] for history and survey). Two models will be presented here: the basic execution time model of Musa [3] and the logarithmic Poisson model of Musa and Okumoto [4]. The reasons for the concentration on these two models will become clear after they are compared with a number of other models.

Both the basic and logarithmic Poisson execution time models are based on the premise that execution time (the actual processor time used in executing the program) is the best time domain for expressing reliability. Execution time is the most practical measure of the failure-inducing stress being placed on a program. The foregoing premise has been verified in [5,6]. Both models consist of two components: an execution time component and a calendar time component. The former component characterizes reliability behavior as a function of execution time \( r \). The latter component relates execution time \( r \) to calendar time \( t \), which is more useful for managers and engineers in expressing when a specified reliability goal is expected to be reached.

2.1 Execution Time Component

The execution time components of these models are defined in terms of a random process \( \{ M(r), r \geq 0 \} \) that represents the
number of failures experienced by execution time \( r \). The process is characterized by specifying the distribution of \( M(\tau) \), including either the mean value function

\[
\mu(\tau) = E[M(\tau)]
\]

or the failure intensity function

\[
\lambda(\tau) = \frac{d\mu(\tau)}{d\tau}.
\]

Both models are assumed to be nonhomogeneous Poisson processes *, but they have different failure intensity functions. The basic execution time model has a failure intensity function which decays exponentially with execution time \( r \), i.e.,

\[
\lambda(\tau) = \lambda_0 \exp(-\phi \tau),
\]

where \( \lambda_0 \) is the initial failure intensity and \( \phi \) is the rate of decrease per unit time. The logarithmic Poisson model has a failure intensity function which decays exponentially with respect to the mean value function, i.e.,

\[
\lambda(\tau) = \lambda_0 \exp [-\theta \mu(\tau)],
\]

where \( \theta \) is the rate of decrease per failure. For the basic execution time model, the initial failure intensity \( \lambda_0 \) is given by

\[
\lambda_0 = \nu_0 \phi.
\]

The parameter \( \nu_0 \) is the expected number of failures in infinite time (note that it is usually finite). The parameter \( \phi \) may be viewed as the (constant) hazard rate that characterizes any individual failure. The failure intensity for the basic model is readily expressed in terms of \( \mu \) by integration of (3) and some manipulation:

* The basic execution time model was not at first described as a nonhomogeneous Poisson process.
We obtain a similar result for the logarithmic Poisson model directly from (4):

\[ \lambda(\mu) = \lambda_0 \exp(-\theta) \mu. \]  

It will be seen from (6) and (7) that the rate of change of failure intensity with respect to failures experienced is constant for the basic execution time model but decreases exponentially with failures experienced for the logarithmic Poisson model. The latter model can account for nonuniform operational profiles, where some functions are executed more frequently than others. In this case, frequently occurring failures tend to be experienced first. Early repairs therefore have the largest effects in reducing failure rate.

The expressions for expected failures for the two models are, respectively

\[ \mu(\tau') = \lambda_0 [1 - \exp(-\phi \tau)] \]  

and

\[ \mu(\tau) = \frac{1}{\theta} \ln \left( \lambda_0 \theta \tau + 1 \right). \]  

The name "logarithmic Poisson model" is derived from the form of (9). Plots of expected failures and failure intensity are provided for both models in Figs. 1 and 2, respectively.

The reliability \( R(\tau' | \tau) \) for either model at a time \( \tau' \) measured from time \( \tau \) is given by [7]

\[ R(\tau' | \tau) = \exp\left\{-[\mu(\tau+\tau') - \mu(\tau)]\right\} \]  

(10)
Fig. 1. Expected failures vs. execution time.

Fig. 2. Failure intensity vs. expected failures.
Expressions for several useful quantities can be derived from (3), (4), (6), (7), and (9), including the additional execution time $\Delta \tau$ and additional expected failures $\Delta \mu$ to be experienced to reach a failure intensity goal of $\lambda_P$, given a present failure intensity $\lambda_F$. For the basic execution time model we have

$$\Delta \tau = \frac{1}{\phi} \ln \frac{\lambda_P}{\lambda_F}$$  \hspace{1cm} (11)

$$\Delta \mu = \frac{1}{\phi} (\lambda_P - \lambda_F)$$  \hspace{1cm} (12)

For the logarithmic Poisson model

$$\Delta \tau = \frac{1}{\theta} \left( \frac{1}{\lambda_F} - \frac{1}{\lambda_P} \right)$$  \hspace{1cm} (13)

$$\Delta \mu = \frac{1}{\theta} \ln \frac{\lambda_P}{\lambda_F}$$  \hspace{1cm} (14)

2.2 Calendar Time Component

The calendar time component relates execution time to calendar time based on resources available to a project. That is, the rate of testing is constrained by the failure identification or test team personnel, the failure correction or debugging personnel, and the computer time available. The quantities of these resources available to a project may be more or less established in its early stages, but increases are generally not feasible during the system test phase because of the long lead times required for training and computer procurement. At any point in testing, one of these resources will be limiting and will be determining the rate at which execution time can be spent per
unit calendar time. The test phase may consist of from one to three periods, each characterized by a different limiting resource.

The following is a common scenario. At the start of testing one identifies a large number of failures separated by short time intervals. Testing must be stopped from time to time in order to let the people who are fixing the faults keep up with the load. As testing progresses, the intervals between failures become longer and longer and the failure correction personnel are no longer fully loaded. The test team becomes the bottleneck. The effect required to run tests and analyze the results is occupying all their time. Finally, at even longer intervals, the capacity of the computing facilities becomes limiting.

The calendar time component of the model is derived by assuming that the quantities of the resources available are constant for the remainder of the test period and that the rates of resource expenditure $dX_k/d\tau$ for resource $k$ ($k = C$ for computer time, $k = I$ for failure identification personnel, and $k = F$ for failure correction personnel) can be approximated by

$$\frac{dX_k}{d\tau} = \theta_k + \mu_k \frac{d\mu(t)}{d\tau},$$

where $\theta_k$ is an execution time coefficient of resource expenditure and $\mu_k$ is a failure coefficient of resource expenditure.

For both models we find that

$$\frac{dt}{d\tau} = \max_k \left( \frac{1}{P_k} \frac{dX_k}{d\tau} \right), \quad k = C, I, F,$$
where $P_k$ is the resource quantity available and $f_k$ the utilization for resource $k$.

For the basic execution time model we have

$$\frac{d\chi}{d\tau} = \theta_k + \mu_k \lambda_0 \exp(-\phi \tau) \quad (17)$$

and for the logarithmic Poisson

$$\frac{d\chi}{d\tau} = \theta_k + \mu_k \frac{\lambda_0}{\lambda_0 \theta + 1} \quad (18)$$

The foregoing quantities are plotted in Fig. 3. Note that the difference is due solely to the difference in the failure intensity curves.
2.3 Determination of Model Parameters

In the case of the two models described above, two parameters must be estimated for the execution time components: initial failure intensity $\lambda_0$ in both cases, total failures experienced $o$ for the basic execution time model, and the rate of reduction in the failure intensity per failure $\theta$ for the logarithmic Poisson model. Once failure data is available in terms of execution time, these parameters may be estimated, using a statistical inference method (e.g., maximum likelihood estimation). The accuracy with which they are known generally increases with the size of the sample of failures. The accuracy may be characterized by constructing confidence intervals.

Procedures for predicting the values of the execution time component parameters before failure data is available are presently developed just for the basic execution time model.

The parameter $v_o$ may be estimated, prior to test, from the number of inherent faults $v_0$ and the fault reduction factor $B$, since

$$v_o = \frac{v_0}{B}$$  \hspace{1cm} (19)

The number of inherent faults is dependent on the size of the program. Some data has been taken on average faults per instruction. A range of 3.36 to 7.98 faults/1000 delivered executable source instructions for assembly language programs at the start of system test has been reported [3]. It may be that measures of program complexity can improve the prediction of $v_o$;
this is an active current research area.

The parameter \( \lambda_0 \) may be predicted from

\[
\lambda_0 = fK \nu_0
\]

(20)

where \( f \) is the linear execution frequency of the program or the average instruction execution rate divided by the number of object instructions in the program and \( K \) is a fault exposure ratio which relates failures to "fault velocity". The fault velocity is the rate at which faults in the program would pass by if the program were executed linearly. It accounts for the following facts:

a. programs are not generally executed in "straight line" fashion, but have many loops and branches, and

b. the machine state varies and hence the fault associated with an instruction may or may not be exposed at one particular execution of the instruction.

It may be that the range of values of \( K \) over different software systems is small. This could be due to program dynamic structure averaging out in some fashion for programs of any size. Thus it may be possible to determine a value for \( K \), perhaps as a function of some program characteristic.

There are two categories of parameters that must be established for the calendar time components of the two models; the parameters are the same and have the same values for both models. The categories are planned and resource usage. The planned parameters are established by project objectives and available resources. The resource usage parameters relate to the resources required for failure identification and correction. It is hoped
that ultimately values of these parameters can be determined for all software projects or for large classes of projects. The values of the resource usage parameters may be related to such factors as batch debugging versus interactive debugging, debugging aids available, computer used, language used, administrative and documentation overhead associated with corrections, etc.

3. COMPARISON OF MODELS

It is assumed that comparison should be done with relation to a variety of software systems; it does not appear likely that the evaluation of the models will be application-dependent but one must watch for this possibility. It is expected that comparisons will cause some models to be rejected because they meet few of the criteria that we will discuss. On the other hand, there may or may not be a clear choice between the more acceptable models. The relative weight to be placed on the different criteria may depend upon the context in which the model is being applied. However, the criteria have been ranked in approximate order of importance. When comparing two models, all criteria should be considered simultaneously (i.e., models should not be eliminated by one criterion before other criteria are considered).

3.1 Comparison Criteria

The comparison criteria that are described below represent an approximate consensus among a number of researchers in the field [8]. They are:
a. predictive validity,
b. capability,
c. quality of assumptions,
d. applicability, and
e. simplicity.

3.1.1 Predictive Validity. Predictive validity is the ability of the model to determine future failure behavior during either the test or the operational phases from present and past failure behavior in the respective phase. Note that "predictive validity" does not carry the connotation of "prediction" in referring to determination from program characteristics. In comparing predictive validity of models, one must keep in mind that differences should be larger than other sources of error (especially measured error) before any advantage can be attributed to one model over another.

We will evaluate predictive validity by attempting to predict number of failures that will be experienced by the end of the period of execution over which data has been collected and compare this with actuals. It is an excellent but not necessarily the only possible approach.

In this approach, the failure random process is described by \( (M(t), t \geq 0) \), representing the number of failures experienced by time \( t \). Such a counting process is characterized by specifying the distribution of \( M(t) \), including the mean value function \( \mu(t) \).

Assume that \( q \) failures have been observed by the end of test time \( t_q \). The failure data up to time \( t_e (\leq t_q) \) is used to estimate the
parameters of $\mu(t)$. Then, the number of failures by $t_q$ can be predicted by substituting the estimates of the parameters in the mean value function to obtain $\hat{\mu}(t_q)$, which is compared with the actually observed number $q$. This will be repeated for various values of $t_e$.

The predictive validity can be checked visually by plotting the normalized relative error $((\hat{\mu}(t_q) - q)/q)$ against the normalized time $t_e/t_q$ (Fig. 4). The error will approach zero as $t_e$ approaches $t_q$. If the points are positive (negative), the model tends to overestimate (underestimate). Numbers closer to zero imply more accurate prediction and hence the better model.

The use of normalization enables one to overlay relative error
curves obtained from different failure data sets. For an overall conclusion as to the relative predictive validity of models, we may compare plots of the medians (taken with respect to the various data sets). The model which yields the curve that is the closest to zero will be considered superior.

Note that predictive validity may be a function of both the model and the inference procedure.

3.1.2 Capability. Capability refers to the ability of the model to estimate with satisfactory accuracy quantities needed by software managers, engineers, and users in planning and managing software development projects or controlling change in operational software systems. The degree of capability must be gauged by looking at the relative importance as well as number of quantities estimated. The quantities, in approximate order of importance, as denoted by the letters, are:

a. present reliability, MTTF, or failure intensity,

b. expected date of reaching a specified reliability, MTTF, or failure intensity goal (it is assumed that the goal is variable and that dates can be computed for a number of goals, if desired. If a date cannot be computed and the goal achievement can be described only in terms of additional execution time or failures experienced, this limited facility is preferable to no facility, although it is very definitely inferior),

c. resource and cost requirements related to achievement of the foregoing goal(s).
Any capability of a model for prediction of software reliability in the system design and early development phases would be extremely valuable because of the resultant value for system engineering and planning purposes. It appears that these predictions must be made through measurable characteristics of the software (size, complexity, structure, etc.), the software development environment and the operational environment.

3.1.3 Quality of Assumptions. The following considerations of quality should be applied to each assumption in turn. If it is possible to test an assumption, the degree to which it is supported by actual data is an important consideration. This is especially true of assumptions that may be common to an entire class of models. If it is not possible to test the assumption, its plausibility from the viewpoint of logical consistency and software engineering experience should be evaluated. Finally, the clarity and explicitness of an assumption should be judged; these characteristics are often necessary to determine whether a model applies to particular circumstances.

3.1.4 Applicability. Another important characteristic of a model is its applicability. A model should be judged on its degree of applicability across different software products (size, structure, function, etc.), different development environments, different operational environments, and different life cycle phases. However, if a particular model gives outstanding results for just a narrow range of products or development environments, it should not necessarily be eliminated.

There are at least five situations that are encountered commonly
enough in practice that a model should either be capable of dealing with them directly or should be compatible with procedures that can deal with them. There are:
a. phased integration of a program during test (i.e., testing starts before the entire program is integrated, with the result that some failure data is associated with a partial program),
b. design and requirements changes to the program,
c. classification of severity of failures into different categories,
d. ability to handle incomplete failure data or data with measurement uncertainties (although not without loss of predictive validity),
e. operation of same program on computers of different performance.

Finally, it is desirable that a model be robust with respect to departures from its assumptions, errors in the data or parameters it employs, and unusual conditions.

3.1.5 Simplicity. A model should be simple in three aspects. The most important consideration is that it must be simple and inexpensive to collect the data that is required to particularize the model. If the foregoing is not the case, the model will not be used. Second, the model should be simple in concept. Software engineers without extensive mathematical background should be able to understand the nature of the model and its assumptions, so they can determine when it is applicable and the extent to
which the model may diverge from reality in an application. Parameters should have readily understood interpretations that relate to characteristics of the program, the development environment, or the execution environment. This property makes it more feasible for software engineers to estimate the values of the parameters where data is not available. Finally, a model must be readily implementable as a program that is a practical management and engineering tool. This means that the program must run rapidly and inexpensively with no manual intervention required other than the initial input.

3.2 Classification of Models
Some models share common characteristics, particularly predictive validity. Hence it is efficient to classify models for the purpose of making comparisons. We will base our classification on five attributes [9]:

a. time domain - calendar time or execution (CPU or processor) time,

b. category - the expected number of failures that can be experienced in infinite time is finite or infinite,

c. type - the failure quantity distribution,

d. class (finite failures category only) - functional form of the failure intensity in terms of time,

e. family (infinite failures category only) - functional form of the failure intensity in terms of the expected value of failures experienced.

Table I illustrates the classification scheme with respect to the last four attributes (it is identical for both kinds of time) and
Table 1. Software reliability model classification scheme

- Finite Failures Category Models

<table>
<thead>
<tr>
<th>Class</th>
<th>Type</th>
<th>OtherTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Poisson</td>
<td>Littlewood-Verrall general with rational ( \Phi() ) suggested by Musa [2] Goel-Okumoto imperfect debugging [18]</td>
</tr>
<tr>
<td></td>
<td>Binomial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Moranda geometric</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Poisson</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schneidewind [11]</td>
<td></td>
</tr>
<tr>
<td>Weibull</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wagoner [14]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schick/Wolverton [15]</td>
<td></td>
</tr>
<tr>
<td>Pareto</td>
<td>Littlewood differential [16]</td>
<td></td>
</tr>
</tbody>
</table>

- Infinite Failures Category Models

<table>
<thead>
<tr>
<th>Family</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric</td>
<td>Moranda geometric</td>
</tr>
<tr>
<td></td>
<td>De-estophication [10]</td>
</tr>
<tr>
<td>Inverse Linear</td>
<td>Littlewood/Verrall</td>
</tr>
<tr>
<td></td>
<td>general with ( \Phi() ) linear [19]</td>
</tr>
<tr>
<td>Inverse Polynomial (2nd degree)</td>
<td>Littlewood-Verrall general with ( \Phi() ) polynomial [19]</td>
</tr>
<tr>
<td>Power</td>
<td>Crow [20]</td>
</tr>
</tbody>
</table>
notes where most of the published models fit in it. Table II summarizes the functional relationships of the failure intensity of various models with respect to (execution) time and the expected number of failures experienced (see [6] for detailed derivations).

3.3 Comparison of Predictive Validity
We will make comparison using the following seven model groups (classes or families), which include most published models: exponential class, Weibull class, Pareto class, geometric family, inverse linear family, inverse polynomial (2nd degree only) family, and power family. As can be seen from (7), the logarithmic Poisson model is a member of the geometric family. We do not consider different types because the mean value functions of the models are independent of type, and the mean value function is the primary determinant of the model's predictive validity characteristics. The approach previously described for evaluating predictive validity will be employed.

The failure data used is composed of 15 sets of data on a variety of software systems (such as real time command and control, real time commercial, military, and space systems) with system sizes ranging from small (5.7 K object instructions) to large (2.4 M object instructions). The data sets were all taken during system test (except for one taken during subsystem test). Consult [6] for detailed descriptions of the data source and system characteristics.
Table II. Functional relationships for failure intensity with respect to time and expected number of failures experienced \( \mu(a_0, a_1, a_2, a_3, \phi_0, \phi_1, \phi_2 \text{ are real}).

- **Finite Failures Category (All Types)**

<table>
<thead>
<tr>
<th>Class</th>
<th>( \lambda(t) )</th>
<th>( \lambda(\mu) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>( a_0 e^{-a_1 t} )</td>
<td>( \phi_0 (\phi_1 - \mu) )</td>
</tr>
<tr>
<td>Weibull</td>
<td>( a_0 a_1 t^{a_1 - 1} e^{-a_1 t} )</td>
<td>( \phi_0 [-\ln(1 - \mu/\phi_1)]^{a_1} (\phi_1 - \mu) )</td>
</tr>
<tr>
<td>Pareto</td>
<td>( a_0 (a_1 + t)^{-a_1} )</td>
<td>( \phi_0 (\phi_1 - \mu)^{a_1} )</td>
</tr>
</tbody>
</table>

- **Infinite Failures Category (All Types)**

<table>
<thead>
<tr>
<th>Family</th>
<th>( \lambda(t) )</th>
<th>( \lambda(\mu) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric</td>
<td>( a_0 (a_1 + t)^{-1} )</td>
<td>( \phi_0 \phi_1 )</td>
</tr>
<tr>
<td>Inverse Linear</td>
<td>( a_0 (a_1 + t)^{-1/2} )</td>
<td>( 1/(\phi_0 + \phi_1 \mu) )</td>
</tr>
<tr>
<td>Inverse Polynomial (2nd Degree)</td>
<td>( \frac{a_0}{\sqrt{1 + a_0 t^2 + a_1 t^2}} )</td>
<td>( 1/(\phi_0 + \phi_1 \mu^2) )</td>
</tr>
<tr>
<td>Power</td>
<td>( a_0 a_1 t^{a_1 - 1} )</td>
<td>( \phi_0 \phi_1^{a_1} )</td>
</tr>
</tbody>
</table>

Note that failure data is used to estimate the parameters of the failure intensity function that represents the model group. We will utilize two different inference methods, maximum likelihood and least squares, to see if inference method has any substantial effect on predictive validity. Since most models are associated with particular inference procedures, we may not be precisely representing them in some cases; the difference will be significant only if different inference procedures yield substantially different results in regard to predictive validity.
3.3.1 Maximum Likelihood Estimation. An evaluation of predictive validity for the case of inference by maximum likelihood estimation is shown in Fig. 5. (See [4] for a detailed discussion on the inference method). This plot represents a summary of the data from the system test periods of about 15 different software systems. Plots of the median (over the 15 systems) error curves for the model groups are shown. The geometric family yields the best prediction at most points in time, followed closely by the inverse polynomial family. Note that the exponential group is inferior for small values of time, but not markedly so past 60% of the period in question. In fact, the prediction error is well under 10% after this point.

3.3.2 Least Squares Estimation. We will now repeat the comparison among model groups for predictive validity, using inference by least squares estimation. The failure intensity is first estimated based on groups of failures. The functional relationship $\lambda(\mu)$ of the failure intensity with respect to the mean value function (Table II) is then used to estimate the model parameters by fitting the function to the estimated failure intensity data.

Assume that the failure data is available in the form of $m$ successive failure intervals, denoted by $r_1', i=1,\ldots,m$. Then, the cumulative time to the $i$-th failure is given by $r_i = \sum_{l=1}^{i} r_1'$. The observation interval $(0, r_m]$ is partitioned at every $k$-th failure occurrence time so that there are $p$ (the largest integer of $m/k$) disjoint subintervals. Then, the failure intensity for the $j$-th subinterval ($r_{k(j-1)}, r_{kj}$] may be estimated by
Fig. 5. Median curves of relative error for seven model groups, maximum likelihood estimation.

\[
y_j = \begin{cases} 
  \frac{k}{\tau_{kj} - \tau_{k(j-1)}}, & j=1,\ldots,p-1 \\
  \frac{m - k(p-1)}{\tau_m - \tau_{k(j-1)}}, & j=p 
\end{cases}
\]  

(21)

The corresponding estimate of the mean value function for the j-th subinterval is

\[x_j = k(j-1), \quad j=1,\ldots,p.\]  

(22)

Note that grouping a small number of failures (a small value of \(k\)) will result in large variations in the estimated failure intensity while grouping a large number of failures (a large
value of $k$) will result in too much smoothing. A group of five failures (i.e., $k = 5$) has been selected as a reasonable compromise in the following analysis. Although some information may be lost due to grouping of failures, an advantage of the foregoing approach is that no specific model or distribution is assumed.

The method of least squares is used to estimate the model parameters by fitting the functional relationship $\lambda(\mu)$ to the estimated failure intensity data $((x_j, y_j), j=1,\ldots,p)$. Let $\epsilon_j$ represent the logarithm of the ratio of the $j$-th data point to the model. Then, we have

$$\ln y_j = \ln \lambda(x_j) + \epsilon_j.$$  \hspace{1cm} (23)

The estimates of the model parameters can be found so that the sum of the squares of $\epsilon_j$'s is minimized. In the case of the logarithmic Poisson model, if we denote by $S(\lambda_0, \theta)$ the sum of the squares of $\epsilon_j$'s from (7) and (23) we have

$$S(\lambda_0, \theta) = \sum_{j=1}^{p} \epsilon_j^2 = \sum_{j=1}^{p} (\ln y_j - \ln \lambda_0 + \theta x_j)^2$$  \hspace{1cm} (24)

which is to be minimized. Results from a simple linear regression yield the least squares estimates of $\lambda_0$ and $\theta$. Consult Okumoto [21] for a detailed description of the foregoing approach. It should be pointed out that the least squares estimate of $\lambda_0$ based on the mid-point, i.e., $x_j = k(j-1)+k/2$ was found to be biased.

Although the above method is attractive due to its simplicity and practicability, if the fitted mean value function is used to predict the number of failures in some additional execution time,
it is not certain that the additional number of failures is always greater than or equal to zero. Since in this paper the number of failures predicted by a model is used to evaluate predictive validity, the above method is modified so that the fitted mean value function passes through a point \((r_m, m)\), i.e.,

\[ \mu(r_m) = m. \] (25)

In other words, the model parameters are estimated so as to minimize the sum of the squares of \(\epsilon_j\)'s under the condition given by (25). Because of the nonlinear constraints, we cannot find an analytical solution but must obtain it numerically.

The predictive validity is now evaluated once more for the different model groups, using the same procedure as before but with the parameters found by least squares estimation. The results are shown in Fig. 6.

It will be seen that the results, both relatively and in absolute terms, are very similar to those for the case of maximum likelihood estimation. Thus, it does not appear likely that different methods of estimation will have a substantial effect on predictive validity of the models with which they are associated.

3.4 Evaluation of Other Criteria
The capabilities of both the basic and logarithmic Poisson models are superior to those of other published models. Currently, they are the only two models that perform reliability modeling in execution time and then employ calendar time components to convert execution time quantities to calendar time. They readily yield present failure intensity. They also provide expected date
Fig. 6. Median curves of relative error for seven model groups, least squares estimation.

of reaching a specified failure intensity and the associated resource and cost requirements. To the best of the authors' knowledge, the latter capability is unique.

The basic model has parameters that can be related to the characteristics of the software and the development process (although not with high accuracy at present). Thus it possesses prediction capability or the ability to project software reliability prior to execution. This capability does not appear to exist at present for models outside the exponential class of the finite failures category. Although models such as Jelinski-Moranda, Shooman, and Goel-Okumoto share this property with the
basic execution time model, the latter is used because it incorporates some of the concepts involved in the former models.

There has not been a general evaluation of all the assumptions on which the published models are based; hence it would be difficult to draw any conclusions in regard to their relative merits on this criterion.

In general, the published models seem to be widely applicable to most types of software products under various conditions. If the operational profile (set of input states experienced and associated probabilities) is highly nonuniform, then the decrement in failure intensity per failure experienced will tend to be nonuniform. In this situation, models of class or family other than exponential may tend to fit better and yield better predictive validity. On the other hand, the published models are generally developed for programs that are stable in size, while most programs change as the result of the phasing of integration, design changes, etc. A method of compensating for changing size has been developed [22], but it is dependent on being able to relate model parameters to program characteristics, particularly size. Hence only the basic execution time model can be used at present for the large class of programs that are changing in size.

In point of fact, the basic execution time model is the one that has been most widely applied to actual projects, as development was proceeding. Hence, a lot of information and lore is available concerning its use [23-27] and convenient programs have been
Both the basic and logarithmic Poisson execution time models are simple in concept. Both have execution time components that are based on only two parameters. These parameters are readily interpretable as physical quantities, especially in the case of the basic model. Models of the Weibull and Pareto types have three parameters, and their physical significance is not as readily apparent. The Littlewood general model uses Bayesian inference. Most engineers find this a difficult approach to comprehend. The analysis is frequently very complex. The computer programs that implement the Littlewood model are substantially more difficult to develop and debug, and the run times are sometimes several orders of magnitude greater than those of the two execution time models with maximum likelihood estimation. It should be noted that Kremer [30] presents an interesting analysis of the fault repair process that gives general conceptual insight into the nature of finite failure models (the results do not favor any particular model, however).

4. CONCLUSIONS
After considering all of the foregoing evaluations, it will be seen that the basic execution model is generally superior in capability and applicability to the other published models. It and the logarithmic Poisson are superior in simplicity. The logarithmic Poisson is second in capability to the basic model, but superior to the others. The logarithmic Poisson is superior in predictive validity; the basic model is not, although the
deficit is not significant after about 60% of the way through the test period. Thus the foregoing appear to be the two models of choice. One possible approach is to use the basic model for pretest studies and estimates and for periods of phased integration. You would switch to the logarithmic Poisson model when integration is complete and the program is stable. However, the additional complexity of this approach must be considered against the possibly limited improvement in predictive validity.

ACKNOWLEDGMENTS
The authors are indebted to John Greene and Wilhelm Kremer for their helpful comments and suggestions.

REFERENCES


[25] J.D. Musa, "Software reliability measures applied to system engineering", *1979 NCC Proceedings*, New York, N.Y., June 4-


