

ISSRE'93 Tutorial Proposal

Denver, Colorado

November, 1993

**Software Reliability Modeling Techniques and Tools**

**Michael R. Lyu**

*Bell Communications Research*  
**445 South Street**  
*Morristown, NJ 07962*  
*(201)829-3999*  
*(201)829-5981 (fax)*  
*<lyu@bellcore.com>*

**Allen P. Nikora**

*Jet Propulsion Laboratory*  
*4800 Oak Grove Drive*  
*Pasadena, CA 97709*  
*(818)354-9694*  
*(878)393-7362 (fax)*  
*<bignuke@spa1.jpl.nasa.gov>*

# Half-Day Tutorial Agenda

**Part I: Introduction** (*20 minutes*)

**Part II: Survey of Software Reliability Models** (*60 minutes*)

**Part III: Quantitative Criteria for Model Selection** (*20 minutes*)

**Part IV: Linear Combination Modeling Approach** (*20 minutes*)

**Part V: Software Reliability Modeling Took** (*120 minutes*)

*Total time: 240 minutes*

ISSRE'93 Tutorial

Software Reliability Modeling Techniques and Tools

## **Part I: Introduction**

- 1. Definitions**
- 2. Modeling principles**
- 3. Reliability theory**
- 4. Benefits “ ”**

*Total time: 20 minutes*

# **SOFTWARE RELIABILITY**

## **DEFINITION:**

**“[Software Reliability] is the Probability of Failure-Free Operation of a Computer Program for a Specified Time in a Specified Environment.”**

**IEEE STANDARD**

# Reliability Measurement Definition and Goal

- Reliability measurement is a set of mathematical techniques that can be used to estimate and predict the reliability behavior of software during its development and operation.
- The primary goal of software reliability modeling is to answer the following question:  
Given a system, what is the probability that it will fail in a given time interval, or, what is the expected duration between successive failures?

# System Reliability Measures

---

---

- Reliability function (survival function, survival probability) –  $R(t)$ 
  - the probability that the system will operate failure-free for a time  $t$  (days, weeks, months, or years, whichever is most meaningful for the system under consideration) in the specific customer environment
- Failure intensity (failure rate) –  $\lambda$ 
  - number of failures per unit time (e.g., per day per week or per month)
- Mean time to failure – *MTTF*
  - "average" time (days, weeks, months, or years) expected until system failure in the customer environment
- Expected total number of failures –  $\mu(t)$ 
  - "average" number of failures expected in a time period  $t$  (days, weeks, months) in the customer environment

# Reliability Theory

"T" is a random variable representing the failure time or lifetime of a physical system.

For this system, the probability that it will fail by time "t" is:

$$F(t) = P[T \leq t] \\ = \int_0^t f(x) dx$$

The probability of the system surviving until time "t" is:

$$R(t) = P[T > t] \\ = 1 - F(t) \\ = \int_t^{\infty} f(x) dx$$

# Reliability Theory (cont'd)

Failure rate - probability that a failure will occur in the interval  $[t_1, t_2]$  given that a failure has occurred before time  $t_1$ . This is written as:

$$\frac{P[t_1 \leq t < t_2 \mid T > t_1]}{t_2 - t_1} = \frac{P[t_1 \leq t < t_2]}{(t_2 - t_1)P[T > t_1]}$$
$$= \frac{F(t_2) - F(t_1)}{(t_2 - t_1)R(t_1)}$$

# Reliability Theory (cont'd)

Hazard rate - limit of the failure rate as the length of the interval approaches zero. This is written as:

$$z(t) = \lim_{\Delta t \rightarrow 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)}$$

$$= \frac{f(t)}{R(t)}$$

This is the instantaneous failure rate at time  $t$ , given that the system survived until time  $t$ . The terms hazard rate and failure rate are often used interchangeably.

## *Relationship Between Reliability Measures*

---

---

A reliability objective expressed in terms of one reliability measure can be easily converted into other reliability measures as follows:

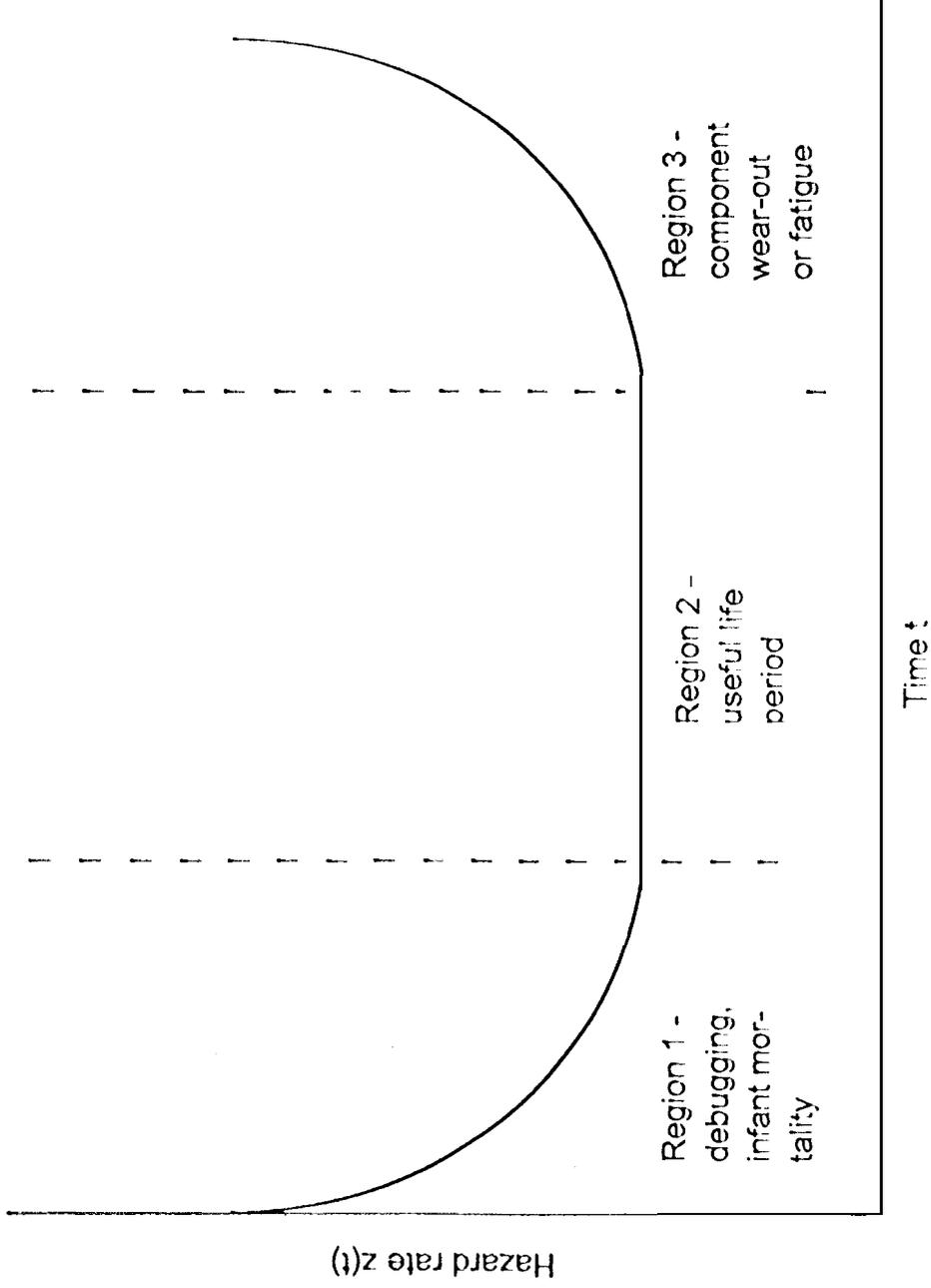
$$\mu(t) = \lambda * t$$

$$MTTF = 1/\lambda$$

$$A = 1/MTTF$$

$$R(t) = \exp(-\mu(t))$$

# Reliability Theory (cont'd)

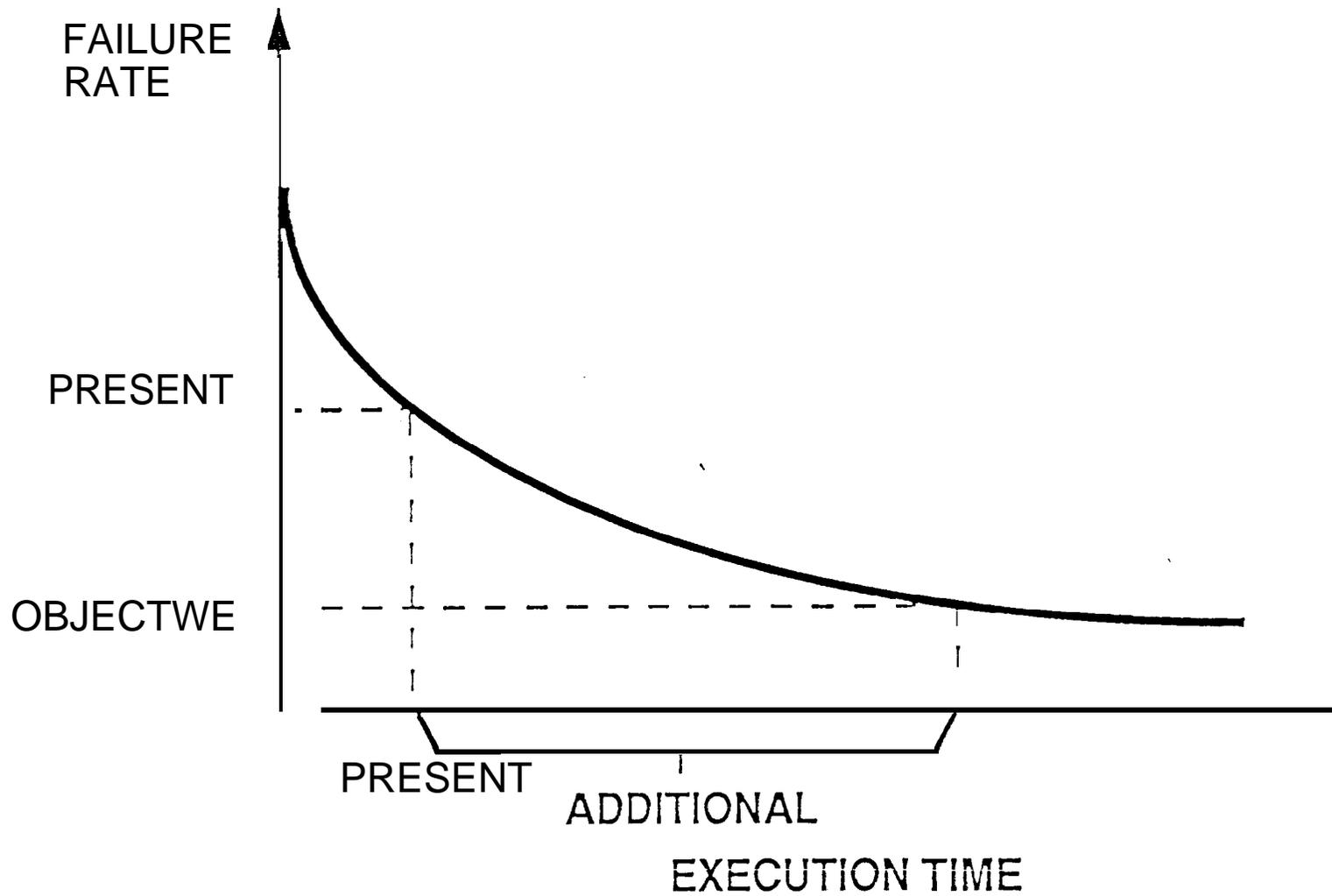


Region 1 applies to both hardware and software

Region 2 applies to both hardware and software

Region 3 does not apply to software

# BASIC IDEAS ON SOFTWARE RELIABILITY MEASUREMENTS



# Benefits

The ultimate objective of developing a model is to use it for making decisions about the software system.

- Planning/Scheduling
  - Measure when reliability goal has been achieved.
  - Control application of test resources.
  - Make trade-offs between resources and schedules.
  - Evaluation and status during test phase.
  - Determine software release date.

# Benefits (cont'd)

- Risk Assessment

  - Determine whether the system is ready for release

  - Determine confidence about system correctness

  - Estimate failure behavior during system operation

- Other Management Decisions

  - Criteria for evaluating new technology or features added to the system

ISSRE'93 Tutorial

Software Reliability Modeling Techniques and Tools

## **Part II: Survey of Software Reliability Models**

- 1. Criteria for evaluating software reliability models**
- 2. Software reliability estimation models<sup>w</sup>**
- 3. Software reliability prediction models**

*Total time: 60 minutes*

# Criteria for Evaluating Software Reliability Models

## (1) Model validity:

- measurement accuracy for current failure rate
- . prediction of the time to finish testing with associated date and cost
- . prediction of the operational failure rate

## (2) Ease of measuring parameters

- . cost, schedule impact for data collection
- . physical significance of parameters to software development process

## (3) Quality of assumptions

- . closeness to the real world
- . adaptability to a special environment

## (4) Applicability (ability to handle program evolution and change in test and operational environment)

## (5) Simplicity in concept, data collection, program implementation, and validation

## (6) Insensitivity to noise (insignificant changes) in input data and parameters without losing responsiveness to significant differences

# Software Reliability Estimation Models

Definition of *Estimation*:

Apply statistical inference procedures to failure data taken from software testing to determine software reliability.

Available Models:

- Jelinski-Moranda Model
- Generalized Poisson Model
- Geometric Model
- Schneidewind Model
- Non-homogeneous Poisson Process Models
- Muss Calendar Time Model
- Brooks and Motley Binomial Model
- Littlewood-Verrall Bayesian Model

# Software Reliability Prediction Models

## Definition of *Prediction*:

Determine software reliability from properties of the software product and the development process even without software execution.

## Available Models:

- Rome Air Development Center (RADC) Model
- SoftWare Early Error Prediction (SWEEP) Model, also known as Phase-Based Model

# Jelinski-Moranda/Shoorman Models

- Jelinski-Moranda model was developed by Jelinski and Moranda of McDonnell Douglas Astronautics Company for use on Navy NTDS software and a number of modules of the Apollo program. The Jelinski-Moranda model was published in 1971.
- Shoorman's model discovered independently of Jelinski and Moranda's work, was also published in 1971. Shoorman's model is identical to the JM model.

# Jelinski-Moranda/Shooman (cent'd)

Assumptions:

1. The number of errors in the code is fixed.
2. No new errors are introduced into the code through the correction process.
3. The number of machine instructions is essentially constant.
4. Detections of errors are independent.
5. The software is operated in a similar manner as the anticipated operational usage.
6. The error detection rate is proportional to the number of errors remaining in the code.

# Jelinski-Moranda/Shoorman (control)

Let  $\tau$  represent the amount of debugging time spent on the system since the start of the test phase.

From assumption 5, we have:

$$z(t) = K\varepsilon_r(t)$$

where  $K$  is the proportionality constant, and  $\varepsilon_r$  is the error rate (number of remaining errors normalized with respect to the number of instructions).

$$\varepsilon_r(\tau) = \frac{E_T}{I_T} - \varepsilon_c(\tau)$$

$E_T$  = number of errors initially in the program

$I_T$  = number of machine instructions in the program

$\varepsilon_c$  = cumulative number of errors fixed in the interval  $[0, \tau]$  (normalized by the number of instructions).

# Jelinski-Moranda/Shoorman (cent'd)

$E_T$  and  $I_T$  are constant (assumptions 1 and 3).

No new errors are introduced into the correction process (assumption 2).

As  $\tau \rightarrow \infty$ ,  $\epsilon_c(\tau) \rightarrow E_T/I_T$ , so  $\epsilon_r(\tau) \rightarrow 0$ .

The hazard rate becomes:

$$z(t) = K \left\{ \frac{E_T}{I_T} - \epsilon_c(\tau) \right\}$$

# Jelinski-Moranda/Shoorman (cont'd)

The reliability function becomes:

$$R(t) = \exp \left\{ -K \left\{ \frac{E_T}{I_T} \int_0^t \varepsilon_c(\tau) \right\} \right\}$$

The expression for MTBF is:

$$\frac{1}{K \left\{ \frac{E_T}{I_T} \int_0^{\infty} \varepsilon_c(\tau) \right\}}$$

# Jelinski-Moranda/Shoorman (cont'd)

At this point the only unknowns are the  $E_T$  and  $K$ . These can be estimated using maximum likelihood estimation.

## Data Requirements

Run a functional test of the program after two different debugging times. and record the following information:

1. For each testing period, record the number of test runs  $n_1$  and  $n_2$  (usually  $n_1 = n_2$ )
2. For each testing period and for each run record the amount of CPU time that the program successfully executed.

# Generalized Poisson Model

- Proposed by Schafer, Alter, Angus, and Emoto for Hughes Aircraft Company under contract to RADC in 1979.
- Model is analagous in form to the Jelinski-Moranda model but taken within the error count framework. The model can be shown reduce to the Jelinski-Moranda model under the appropriate circumstances.

# Generalized Poisson Model (cont'd)

Assumptions:

1. The expected number of errors occurring in any time interval is proportional to the error content at the time of testing and to some function of the amount of time spent in error testing.
2. All errors are equally likely to occur and are independent of each other.
3. Each error is of the same order of severity as any other error.
4. The software is operated in a similar manner as the anticipated usage.
5. The errors are corrected at the ends of the testing intervals without introduction of new errors into the program.

# Generalized Poisson Model (cent'd)

Construction of Model:

- Given testing intervals of length  $X_1, X_2, \dots, X_n$
- $f_i$  errors discovered during the  $i$ 'th interval
- At the end of the  $i$ 'th interval, a total of  $M_i$  errors have been corrected

First assumption of the model yields:

$$E(f_i) = \phi(N - M_{i-1})g_i(x_1, x_2, \dots, x_i)$$

where

$\phi$  is a proportionality constant

$N$  is the initial number of errors

$g_i$  is a function of the amount of testing time spent, previously and currently.  $g_i$  is usually non-decreasing. If  $g_i(x_1, x_2, \dots, x_i) = x_i$ , then the model reduces to the Jelinski-Moranda model.

# Generalized Poisson Model (cont'd)

- Assumptions 1 and 2 yield the following joint density of the  $f_i$ 's:

$$f(f_1, f_2, \dots, f_n) = \prod_{i=1}^n f(f_i)$$

$$= \prod_{i=1}^n \frac{[f(N - M_{i-1})g_i(x_1, x_2, \dots, x_n)]^{f_i}}{f_i!} \exp(-\phi(N - M_{i-1})g_i(x_1, x_2, \dots, x_i))$$

- Likelihood function,  $L(\phi, N) = \prod_{i=1}^n f(f_i); nL(\phi, N) = \sum_{i=1}^n f_i \ln \phi + \sum_{i=1}^n f_i \ln(N - M_{i-1}) + \sum_{i=1}^n f_i \ln g_i - \sum_{i=1}^n (N - M_{i-1})g_i - \sum_{i=1}^n \ln f_i!$

# Generalized Poisson Model (cont'd)

- MLEs  $\hat{\phi}$ ,  $\hat{N}$  given by solutions to the following pair of equations:

$$\hat{\phi} = \frac{\sum_{i=1}^n f_i}{\hat{N} \sum_{i=1}^n g_i - \sum_{i=1}^n M_{i-1} - g_i}$$

$$\sum_{i=1}^n [f_i / \hat{N} - M_{i-1}] = \hat{\phi} \sum_{i=1}^n g_i$$

# Geometric Model

- Proposed by Moranda in 1975 as a variation of the Jelinski-Moranda model.
- Unlike models previously discussed, it does not assume that the number of errors in the program is finite, nor does it assume that errors are equally likely to occur.
- This model assumes that errors become increasingly difficult to detect as debugging progresses, and that the program is never completely error free.

# Geometric Model (cent'd)

Assumptions:

1. There are an infinite number of total errors.
2. All errors do not have the same chance of detection.
3. The detections of errors are independent.
4. The software is operated in a similar manner as the anticipated operational usage.
5. The error detection rate forms a geometric progression and is constant between error occurrences.

# Geometric Model (cent'd)

The above assumptions result in the following hazard rate:

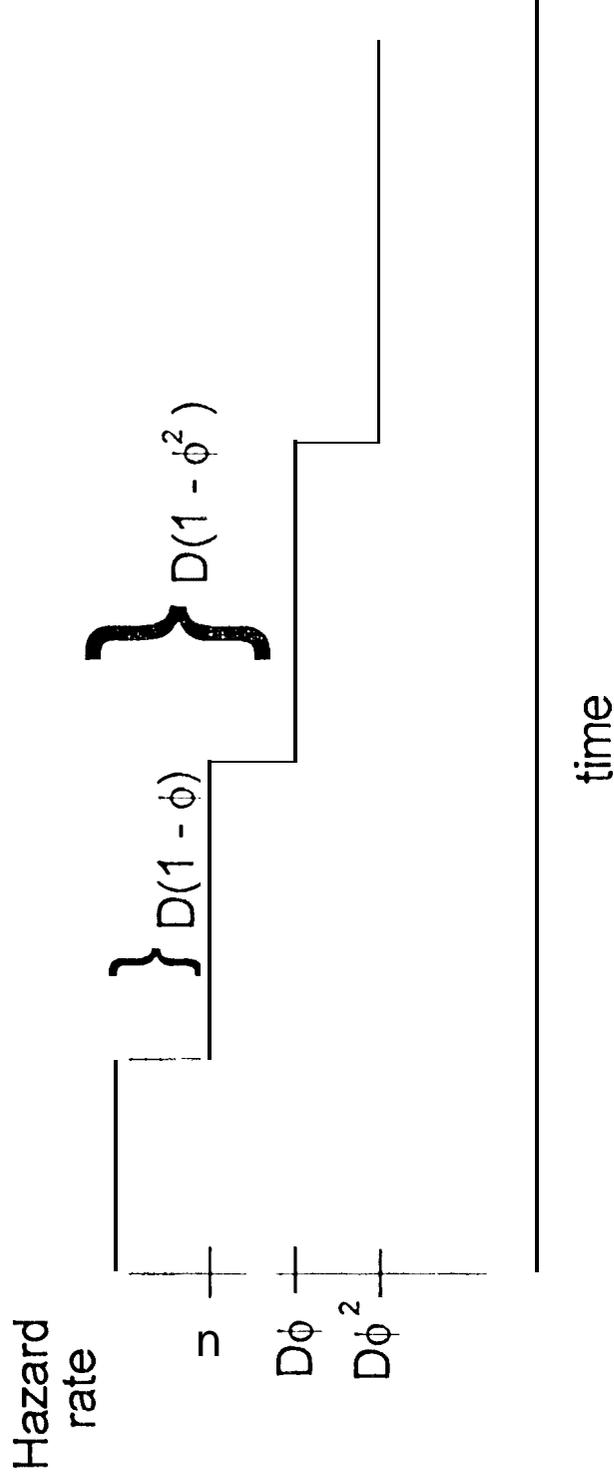
$$z(t) = D\phi^{i-1}$$

for any time "t" between the (i - 1)st and the i'th error.

The initial value of  $z(t) = D$

# Geometric Model (cont'd)

Hazard Rate Graph



$$\frac{\text{Change in } z(t) \propto \text{discovery of } i\text{'th error}}{\text{Change in } z(t) \text{ on discovery of } (i + 1)\text{st error}} = \frac{D\phi^{i-1} - D\phi^{i-2}}{D\phi^i - D\phi^{i-1}} = \frac{1}{\phi}$$

A

# Geometric Model (cent'd)

- From assumptions 3 and 5,

$$f(X_i) = D\phi^{i-1} \exp(-D\phi^{i-1} X_i)$$

where  $X_i = t_i - t_{i-1}$

- Likelihood function for  $X_i$  is written as:

$$L = \prod_{i=1}^n f(X_i) = D^n \prod_{i=1}^n \phi^{i-1} \exp(-D \sum_{i=1}^n \phi^{i-1} X_i)$$

$$\ln(L) = n \ln D + \sum_{i=1}^n (i-1) \ln \phi - D \sum_{i=1}^n \phi^{i-1} X_i$$

# Geometric Model (cont'd)

- MLEs for  $D$  and  $\phi$  are given by the solutions to:

$$\frac{d \ln L}{d D} = \frac{n}{D} - \sum_{i=1}^n \phi^{-1} X_i = 0$$

$$\frac{d \ln L}{d \phi} = \sum_{i=1}^n ((i-1)/\phi - D_i^{-1}) \phi^{-2} X_i = 0$$

- MLEs are:

$$\hat{D} = \frac{\hat{\phi} n}{\sum_{i=1}^n \hat{\phi}^i X_i} \quad \frac{\sum_{i=1}^n i \hat{\phi}^i X_i}{\sum_{i=1}^n \hat{\phi}^i X_i} \quad \frac{n+1}{2}$$

$$\hat{MTBF} = \hat{\xi}(X_{n+1}) = \frac{1}{\hat{D} \hat{\phi}} \quad \text{after } n \text{ errors have been discovered}$$

# Geometric Model (cont'd)

Because of the first assumption (the number of errors in the program is infinite), the number of errors in the program cannot be estimated.

However, a “purification level” after  $n$  errors have been observed can be defined by:

$$\frac{z(t) - z(t_0)}{z(t_0)} = \frac{D - D\phi^n}{D} = 1 - \phi^n$$

# Schneidewind Model

- Proposed by Norman Schneidewind in 1975.
- Model's basic premise is that as the testing progresses over time, the error detection process changes. Therefore, recent error counts are usually of more use than earlier counts in predicting future error counts.
- Schneidewind identifies three approaches to using the error count data. These are identified in the following slide.

# Schneidewind Model (cont'd)

- First approach is to use all of the error counts for all testing intervals. Schneidewind argues that this approach is applicable when it is felt that the error counts from all of the test intervals are useful in predicting future error counts.
- Second approach is to use only the error counts from test intervals through  $m$  and ignore completely the error counts from the first  $s - 1$  test intervals, assuming that there have been  $m$  test intervals to date. This approach is to be used when it is felt that there has been a significant change in the error detection process in the last  $m - s + 1$  intervals.
- Third approach is a hybrid approach which uses the cumulative error count for the first  $s - 1$  intervals and the individual error counts for the last  $m - s + 1$  intervals. This approach is to be used when it is felt that the combined error count from the first  $s - 1$  intervals and the individual error counts for the last  $m - s + 1$  intervals are representative of the error detection behavior for future testing intervals.

# Schneidewind Model (cent'd)

## Assumptions:

1. The number of errors detected in one interval is independent of the error count in another.
2. The error correction rate is proportional to the number of errors to be corrected.
3. The software is operated in a similar manner as the anticipated operational usage.
4. The mean number of detected errors decreases from one interval to the next.
5. The intervals are all of the same length.

# Schneidewind Model (cont'd)

Assumptions (cont'd):

6. The rate of error detection is proportional to the number of errors within the program at the time of test. The error detection process is assumed to be a nonhomogeneous Poisson process with an exponentially decreasing error detection rate. This rate of change is:

$$d_i = \alpha \exp(-\beta i)$$

for the  $i$ 'th interval where  $\alpha > 0$  and  $\beta > 0$  are constants of the model

# Schneidewind Model (cont'd)

- From assumption 6, the cumulative mean number of errors is:

$$D_i = \frac{\alpha}{\beta} [1 - \exp(-\beta i)]$$

for the  $i$ 'th interval, the mean number of errors is:

$$m_i = D_i - D_{i-1} = \frac{\alpha}{\beta} [\exp(-\beta(i-1)) - \exp(-\beta i)]$$

# Schneidewind Model (cont'd)

- Assuming a Poisson process, the likelihood function is given by:

$$L_{S-1} = \frac{\exp(-M_{S-1})}{(S-1)!} \prod_{i=1}^{S-1} \frac{m_i^{f_i} \exp(-m_i)}{f_i!}$$

where  $M_{S-1}$  is the mean number of errors in the interval through  $S-1$ , with  $s$  chosen such that  $2 \leq s \leq m$  and  $F_{S-1} = \sum_{i=1}^{S-1} f_i$ .

$$m_i = \frac{\alpha}{\beta} \exp[-\beta(i-1)(1 - \exp(-\beta))]$$

$$M_{S-1} = \frac{\alpha}{\beta} [1 - \exp(-S(S-1)\beta)]$$

# Schneidewind Model (cent'd)

- Assuming a Poisson process, the likelihood function is given by:

$$\frac{M_{s-1}^{F_{s-1}} \exp(-M_{s-1})}{F_{s-1}!} \prod_{i=s}^m m_i^{f_i} \frac{\exp(-m_i)}{f_i!}$$

where  $M_{s-1}$  is the mean number of errors in the interval 1 through  $S-1$ , with  $s$  chosen such that  $2 < S \leq m$  and  $F_{s-1} = \sum_{i=1}^{s-1} f_i$ .

$$m_i = \frac{\alpha}{\beta} \exp[-\beta(i-1)(1 - \exp(-\beta))]$$

$$M_{s-1} = \frac{\alpha}{\beta} [1 - \exp(-S(S-1)\beta)]$$

# Schneidewind Model (cent'd)

- Determination of MLEs for  $\alpha$  and  $\beta$  is done by solving the following system of equations:

$\hat{\beta} = [n(y)]$ , where  $y$  is the solution to:

$$\frac{(s-1)F_{s-1}}{y} + \frac{F_{s,m}}{y-1} - \frac{mF_m}{y^{m-1}} = A$$

$m-s$

$$A = \sum_{i=1}^m \alpha (s+i-1) f_{s+i}$$

$$F_{s,m} = \sum_{i=s}^m f_i$$

$$\hat{\alpha} = \frac{(\sum_{i=1}^m f_i) \hat{\beta}}{1 - \exp(-\hat{\beta}_m)}$$

# Schneidewind Model (cont'd)

- Once the MLEs have been obtained, the following estimates can be made:

Expected number of errors in  $(m - 1)$ st testing interval

$$= m_{i+1} = \frac{\alpha [\exp(-\beta(i - 1)) - \exp(-\beta i)]}{\beta}$$

Time to detect a total number,  $M$ , of errors

$$= \log\left(\frac{\alpha}{\alpha - \beta M}\right) \frac{1}{\beta}$$

Correction rate for the  $i$ 'th interval

$$= \alpha \exp(-\beta(i - \tau_i))$$

where  $\tau_i$  is the time lag between error detection and correction

# Nonhomogeneous Poisson Process

- Proposed by Amrit Goel of Syracuse University and Kazu Okumoto in 1979.
- ..
- Model assumes that the error counts over non-overlapping time intervals follow a Poisson distribution.
- It is also assumed that the expected number of errors in an interval of time is proportional to the remaining number of errors in the program at that time.

# NHPP (cont'd)

## Assumptions:

1. The software is operated in a similar manner as the anticipated operational usage.
2. The numbers of errors,  $(f_1, f_2, f_3, \dots, f_m)$  detected in each of the respective time intervals  $[(0, t_1), (t_1, t_2), \dots, (t_{m-1}, t_m)]$  are independent for any finite collection of times  $t_1 < t_2 < \dots < t_m$ .
3. Every error has the same chance of being detected and is of the same severity as any other error.
4. The cumulative number of errors detected at any time  $t$ ,  $N(t)$ , follows a Poisson distribution with mean  $m(t)$ .  $m(t)$  is such that the expected number of error occurrences for any time  $(t, t + \Delta t)$ , is proportional to the expected number of undetected errors at time  $t$ .

# NHPP (continued)

Assumptions (contd):

5. The expected cumulative number of errors function,  $m(t)$ , is assumed to be a bounded, nondecreasing function of  $t$  with:

$$m(t) = 0 \text{ for } t = 0$$

$$m(t) = a \text{ for } t = \infty$$

where  $a$  is the expected total number of errors to be eventually detected in the testing process.

## NHPP (cont'd)

- Assumptions 4 and 5 give the number of errors  $d$  discovered in the interval  $t, t + \Delta t$  as:

$$m(t + \Delta t) - m(t) = b((a - m(t))\Delta t + O(\Delta t^2))$$

where

$b$  is a constant of proportionality

$a$  is the total number of errors in the program

$$\frac{m(t + \Delta t) - m(t)}{\Delta t} = \frac{b((a - m(t))\Delta t + O(\Delta t^2))}{\Delta t}$$

$$m'(t) = ab - b m(t) \lim_{\Delta t \rightarrow 0} \frac{O(\Delta t^2)}{\Delta t} = 0$$

# NHPP (cent'd)

- Solving the above differential equation gives:

$$m(t) = a(1 - e^{-bt})$$

which satisfies the initial conditions  $m(0) = 0$ ,  $m(\infty) = a$

- For  $f_i = N(t_i) - N(t_{i-1})$  and the error counts being independent of one another, the likelihood function is:

$$\prod_{i=1}^n \frac{[m(t_i) - m(t_{i-1})]^{f_i} \exp(m(t_{i-1}) - m(t_i))}{f_i!}$$

$$\prod_{i=1}^n \frac{[a(e^{-bt_{i-1}} - e^{-bt_i})]^{f_i} \exp(a(e^{-bt_i} - e^{-bt_{i-1}}))}{f_i!}$$

## NHPP (cont'd)

- MLEs for a and b are given by the following system of equations:

$$\hat{a} = \frac{\sum_{i=1}^m f_i}{(1 - e^{-\hat{b}t_m})}$$

$$\frac{t_m e^{-\hat{b}t_m} \sum_{i=1}^m f_i}{1 - e^{-\hat{b}t_m}} = \sum_{i=1}^m \frac{f_i (t_i e^{-\hat{b}t_i} - t_{i-1} e^{-\hat{b}t_i - 1})}{e^{-\hat{b}t_i - 1} - e^{-\hat{b}t_i}}$$

# Musa Calendar Time

- Developed by John Musa of AT&T Bell Labs between 1975 and 1980.
- This model is one of the most popular software reliability models, having been employed both inside and outside of AT&T.
- Basic model is based on the amount of CPU time that occurs between successive failures rather than on wall clock time.
- Calendar time component of the model attempts to relate CPU time to wall clock time by modeling the resources (failure identification personnel, failure correction personnel, and computer time) that may limit various time segments of testing.

# Musa Calendar Time (cent'd)

- Muss's basic model is essentially the same as the Jelinski-Moranda model
- The importance of the calendar component of the model is:
  - Development of resource allocation (failure identification staff, failure correction staff, and computer time)
  - Determining the relationship between CPU time and wall clock time

Let  $\alpha_{ti}/\alpha_{\tau}$  = instantaneous ratio of calendar to execution time resulting from effects of failure identification staff

Let  $\alpha_{tf}/\alpha_{\tau}$  = instantaneous ratio of calendar to execution time resulting from effects of failure correction staff

Let  $\alpha_{tc}/\alpha_{\tau}$  = instantaneous ratio of calendar to execution time resulting from effects of available computer time

# Muss Calendar Time (cent'd)

- Increment in calendar time is proportional to the average amount by which the limiting resource constrains testing over a given execution time segment:

$$\Delta t = \int_{\tau_1}^{\tau_2} \left( \frac{\Delta t_i}{\Delta \tau} + \frac{\Delta t_F}{\Delta \tau} + \frac{\Delta t_c}{\Delta \tau} \right) d\tau$$

- Resource requirements associated with a change in  $MTBF$  from  $\tau_1$  to  $\tau_2$  can be approximated by:

$$\Delta X_k = \theta_k \Delta \tau + \mu_k \Delta m$$

$\Delta \tau$  = execution time increment

$\Delta m$  = increment of failures experienced

$\theta_k$  = execution time coefficient of resource expenditure (e.g. 100 seconds of CPU time per hour of failure identification staff)

$\mu_k$  = failure coefficient of resource expenditure (e.g. 1 failure detected for every 1.4 hours of failure identification staff)

# Musa Calendar Time (cent'd)

- Given  $P_K$  represents the number of available personnel (failure identification or failure correction,  $K = I$  or  $F$ ) or available number of computer shifts ( $K = C$ )

$\rho_K$  = utilization factor for  $K$ 'th resource. We assume  $\rho_I = 1$ .

$\rho_F$  assumes that error correction is a Poisson process, with servers randomly assigned in time. This determines the length of the error queue,  $Q$ , which in turn determines  $\rho_F$ .

$$\rho_F = (1 - e^{-1/P_F})^{1/Q}$$

Effective amount of  $K$ 'th resource =  $\rho_K P_K$ .

- Correspondence between resources and calendar time is:

$${}^2\tau = \frac{MT_0}{C} \int_{T_1}^{T_2} \max\left[\frac{\theta_K T + C_{\mu_K}}{P_K \rho_K T}\right] dT$$

# Musa Calendar Time (cent'd)

- Correspondence between resourced and calendar time is:

$$z_t = MT_0 \sum_k \frac{1}{P_k \rho_k} \left[ \frac{\theta_k}{C} \ln \left( \frac{T_{K2}}{T_{K1}} \right) + \mu_k \left( \frac{1}{T_{K2}} - \frac{1}{T_{K1}} \right) \right]$$

where:

$T_0 =$  initial  $MTBF$

$M =$  number of failures required to be experienced before all errors in the program have been uncovered

$T_{K1}, T_{K2} =$   $MTBF$  at the boundary of above periods

# Musa Calendar Time (cent'd)

- Boundaries are  $T_1, T_2$  and the transition points:

$$T_{kk'} = \frac{C(P_{k\mu k'r_k} - P_{k'\mu k'r_{k'}})}{P_{k'r_k\theta_k} - P_{k'r_{k'}\theta_{k'}}$$

Transition points are the values of  $T$  at which the derivative of calendar time with respect to execution time for one resource becomes greater than another.

- The limiting resource,  $K$ , for a given MTBF,  $T$ , is the one that maximizes:

$$\frac{\theta_k T + C_{\mu k}}{P_{k'r_k T}}$$

# Brooks and Motley Models

- Published by Brooks and Motley of IBM in 1980.
- Models attempt to account for the following:

During a given testing period, not all of the program is tested equally.

During a given development period, only some portion or modules may be available for test.

When errors are corrected, additional errors may be introduced as part of the correction that was made.

# Brooks and Motley Models (cent'd)

## Assumptions:

1. The number of errors detected on each test occasion is proportional to the number of errors at risk for detection. This, in turn, is proportional to the remaining number of errors.
2. The proportionality factor or probability of detecting any error during a specified unit interval of testing is constant over all occasions and independent of error detections.
3. The errors reintroduced in the correction process are proportional to the number of errors detected.

# Brooks and Motley (cont'd)

## Binomial Model

- Given that the  $j$ 'th module from a program is tested for the first time. From assumptions 1 and 2, the expected number of errors for the 1st interval is:

$$n_{1j} = \omega_j N q \text{ where}$$

$\omega_j$  = weight assigned to module  $j$  (module size divided by the total code size)

$N$  = the number of errors initially in the system

$q$  = the error detection probability (from assumption 2)

- For the second interval of testing on module  $j$ , the expected number of errors detected is:

$$q[\omega_j N - \omega_j N q] = \omega_j N(1 - q)q$$

- For the  $i$ 'th interval, the expected number of errors detected is:

$$\omega_j N(1 - q)^{i-1} q$$

# Brooks and Motley (cent'd)

## Binomial Model

- The total number of errors expected for the first testing occasion is:

$$n_{1j} = \sum_{i=1}^{K_{1j}} \omega_j N (1 - q)^{i-1} q$$

$$= \omega_j N [1 - (1 - q)^{K_{1j}}]$$

$$= \omega_j N q_{1j}$$

$K_{1j}$  = number of unit intervals making up the first test occasion

$$q_{1j} = [1 - (1 - q)^{K_{1j}}]$$

# Brooks and Motley (cent'd)

## Binomial Model

- When testing module  $j$  for the second test occasion, the number of errors at risk is:

$$\omega_j N - n_{1j} + r n_{1j} \text{ where}$$

$r n_{1j}$  = the number of new errors introduced into the module as a result of previous corrections  
(from assumption 3)

- The total number of errors in the second test period is:

$$n_{2j} = (\omega_j N - \alpha n_{1j}) q_{2j} \text{ where}$$

$\alpha = 1 - r$ : the probability of correcting code without introducing new errors

# Brooks and Motley (cent'd)

## Binomial Model

- For the  $i$ 'th testing period, the expected number of errors detected is:

$$n_{ij} = (\omega_j N - \alpha N_{i-1,j}) q_{ij} = \bar{N}_{ij} q_{ij} \text{ where}$$

$$\bar{N}_{ij} = (\omega_j N - \alpha N_{i-1,j}) \text{ and } N_{i-1,j} = \sum_{m=1}^{i-1} n_{mj}$$

$$q_{ij} = (1 - (1 - q)^{K_{ij}})$$

# Brooks and Motley (cont'd)

## Binomial Model

- MLEs for  $(N, q, \alpha)$  are solutions to the following system of equations:

$$\frac{\partial \ln L}{\partial N} = \sum_{j=1}^K \sum_{i=1}^J [\omega_j n \left( \frac{\bar{N}_{ij}}{\bar{N}_{ij} - n_{ij}} \right) + \omega_j K_{ij} \ln(1 - q)] = 0$$

$$\frac{\partial \ln L}{\partial q} = \sum_{i=1}^K \sum_{j=1}^J \left[ \frac{n_{ij} K_{ij}}{(1 - (1 - q)^{K_{ij}})} - K_{ij} \bar{N}_{ij} \right] = 0$$

$$\frac{\partial \ln L}{\partial N} = \sum_{j=1}^K \sum_{i=1}^J \left[ N_{j-1} \cdot n \left( \frac{N_{ij}}{N_{ij} - n_{ij}} \right) + N_{j-1} K_{ij} \ln(1 - q) \right] = 0$$

# Brooks and Motley (cent'd)

## Binomial Model

MLE for  $(N, q, \alpha)$  - continued from previous slide

$L$  is the likelihood function given below:

$$\prod_{i=1}^K \prod_{j=1}^J \binom{N_{ij}}{n_{ij}} q^{n_{ij}} (1-q)^{N_{ij}-n_{ij}}$$

where

$K$  is the number of test occasions

$J$  is the number of modules in the system

$n_{ij}$  is the number of errors observed on the  $i$ 'th testing occasion of the  $j$ 'th module. Note that there is no solution if  $N_{ij} - n_{ij}$  becomes negative.

# Littlewood-Verral Bayesian Model

- Littlewood's model, a reformulation of the Jelinski-Moranda model, postulated that that all errors do not contribute equally to the reliability of a program. For example, a program with errors in rarely exercised sections of the code is more reliable than the same program with the same number of errors in frequently exercised sections of the code.
- Littlewood's model postulates that the error rate, assumed to be a constant in the Jelinski-Moranda model, should be treated as a random variable.
- The Littlewood-Verral model of 1978 attempts to account for error generation in the correction process by allowing for the probability that the program could be made less reliable by correcting an error.

# Littlewood-Verral (cent'd)

Assumptions:

1. Successive execution times between failures are independent random variables with probability density functions

$$f(X_i|\lambda_i) = \lambda_i e^{-\lambda_i X_i} \quad \text{where the } \lambda_i \text{ are the error rates}$$

2.  $\lambda_i$ 's form a sequence of random variables, each with a gamma distribution of parameters  $\alpha$  and  $\psi(i)$ , such that:

$$g(\lambda_i) = \frac{[\psi(i)]^\alpha}{\Gamma(\alpha)} \lambda_i^{\alpha-1} e^{-\psi(i)\lambda_i}$$

$\psi(i)$  is an increasing function of  $i$  that describes the “quality” of the programmer and the “difficulty” of the task.

# Littlewood-Verral (cent'd)

## Assumptions (cent'd)

Imposing the constraint  $P(\lambda(j) < x) > P(\lambda(j-1) < x)$  for any  $x$  reflects the intention to make the program better by correcting errors. It also reflects the fact that sometimes corrections will make the program worse.

3. The software is operated in a similar manner as the anticipated operational usage.

# Littlewood-Verral (cont'd)

$$f(X_i|\alpha, \psi(i)) = \int_0^{\infty} f(X_i|\lambda_i)g(\lambda_i)d\lambda_i$$

$$= \int_0^{\infty} \frac{\lambda_i^{\alpha} e^{-\psi(i)\lambda_i} \lambda_i^{\alpha-1} e^{-\psi(i)\lambda_i} d\lambda_i}{\Gamma(\alpha)}$$

$$= \frac{\alpha[\psi(i)]^{\alpha}}{[\psi(i) + \alpha]^{1-\alpha}}$$

This is a Pareto distribution

# Littlewood-Verral (cent'd)

- Joint density for the  $X_i$ 's is given by:

$$f[X_1, X_2, \dots, X_n | a, \psi(i)] = \frac{a^n \prod_{i=1}^n [\psi(i)]^\alpha}{\prod_{i=1}^n [X_i + \psi(i)]^{\alpha+1}}$$

For  $\psi$ , Littlewood and Verral suggest:

$$\psi(i) = \beta_0 + \beta_1 i$$

$$\psi(i) = \beta_0 + \beta_1 i^2$$

- Maximum Likelihood Estimates for  $a$ ,  $\beta_0$ , and  $\beta_1$  are found by evaluating the likelihood function  $L(\hat{\alpha}, \hat{\beta}_0, \hat{\beta}_1)$ , which is  $f(X_1, X_2, \dots, X_n | \hat{\alpha}, \hat{\beta}_0, \hat{\beta}_1)$

## Littlewood-Verrill (cont'd)

MLEs for  $\alpha$ ,  $\beta_0$ , and  $\beta_1$  are found by solving the following system of equations:

$$\frac{\partial L}{\partial \alpha} = \frac{n}{\alpha} + \sum_{i=1}^n \ln \hat{\psi}(i) - \sum_{i=1}^n \ln(X_i + \hat{\psi}(i)) = 0$$

$$\frac{\partial L}{\partial \beta_0} = \frac{1}{\alpha} \sum_{i=1}^n \frac{1}{\hat{\psi}(i)} - (\hat{\alpha} + 1) \sum_{i=1}^n \frac{1}{X_i + \hat{\psi}(i)} = 0$$

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{\alpha} \sum_{i=1}^n \frac{i}{\hat{\psi}(i)^2} - (\hat{\alpha} + 1) \sum_{i=1}^n \frac{i}{X_i + \hat{\psi}(i)} = 0$$

# Littlewood-Verral (cent'd)

Assuming a uniform a priori distribution for  $\alpha$ , Bayesian inference gives:

$$f(x_i | \beta_0, \beta_1) = \frac{\gamma}{x_{i+1} - \psi(i)} \left[ \gamma + \ln\left(\frac{x_{i+1} - \psi(i)}{\psi(i)}\right) \right]^{-\gamma-1}$$

where  $\gamma$  is the percentage certainty that the failure rate at the  $i + k$ 'th error detected is less than a pre-defined target failure rate.

$\beta_0$  and  $\beta_1$  can then be found by using MLE techniques.

# Other Models

- ROME AIR DEVELOPMENT CENTER (RADC) MODEL
- PHASE-BASED ERROR DISCOVERY MODELS

# RADC Model

- This model yields a prediction of initial fault density based on the following static characteristics of the software:

A = Application Type (e.g. real time flight control system)

D = Development Environment (characterized by development methodology, available tools)

“Requirements and Design Representation Metrics”

SA = Anomaly Management, ST = Traceability, SQ = Quality Review Results

“Software Implementation Metrics”

SL = Language Type, SS = Program Size, SM = Modularity, SU = Extent of Reuse, SX = Complexity, SR = Standards Review Results

- Initial Fault Density is given as:

$$A * D * (SA * ST * SQ) * (SL * SS * SM * SU * SX * SR)$$

# RADC Model (cent'd)

- Numbers for these software characteristics are given in RADC TR-87-171. These numbers are based on a large-scale survey of software development efforts. This is “therefore an empirical model.
- Unlike the models described in earlier slides, this model can be used prior to the testing phases of a development effort. Use of this model is similar to the use of other static, multivariable estimation models (e.g. COCOMO)

# RADC Model (cent'd)

- This model also yields an initial failure rate prediction based on the initial fault density.

$$\lambda_0 = F * K * W_0 \text{ where}$$

$\lambda_0$  = the initial failure rate

F = linear execution frequency of the program

K = fault exposure ratio (from  $1.4 \times 10^{-7}$  and  $10.6 \times 10^{-7}$ , with the average being  $4.2 \times 10^{-7}$ )

$W_0$  = number of inherent faults

# RADC Model (cent'd)

Moreover,  $F = R/I$  where

$R$  = average instruction execution rate

$I$  = number of object instructions

Also,  $I = I_s * Q_x$  where

$I_s$  = number of source instructions

$Q_x$  = code expansion ratio (average value of 4)

Therefore,  $\lambda_0$  can be expressed as:

$$(R * K/Q_x) * (W_0/I_s)$$

All of these can be estimated during requirements analysis, design, and coding.

# Phase-Based Model

- The phase-based model is another approach to predicting reliability prior to the testing phase. This approach was developed by John E. Gaffney, Jr. and Charles F. Davis of the Software Productivity Consortium.
- This is a dynamic model, and makes use of error statistics obtained during technical reviews of requirements, design, and source code. It also uses failure history data obtained during the testing phases.

# Phase-Based Model (cent'd)

## Model Assumptions

1. The development effort's current staffing level is directly related to the number of errors discovered during a development phase.
2. Error discovery curve is monomodal.
3. Code size estimates are available during early phases of a development effort. The model expects that error densities will be expressed in the number of errors per thousand lines of source code, which means that error found during requirements analysis and design will have to be "normalized".

# Phase-Based Model (continued)

- Using assumptions 1 and 2, plus Norden's observation that the Rayleigh curve is the "correct" way of applying resources to a development project in general, the model is expressed as follows:

$$V_t = E[e^{-B(t-1)^2} - e^{-Bt^2}]$$

where

$E$  = Total Lifetime Error Rate (errors per KSLOC)

$t$  = Error Discovery Phase index

$B = 1/2\tau_p$ ,  $\tau_p$  = Defect Discovery Phase Constant, the location of the 'peak' in a continuous fit to the data. This is the point at which 39% of the errors have been discovered

- Cumulative form of model is:

$$V_t = E[1 - e^{-Bt^2}]$$

where  $V_t$  is the number of errors per KSLOC discovered through phase "t".

# Phase-Based Model (cont'd)

## Model Application

- Once two or more data points have been obtained,  $\Xi$  and  $\zeta$  can be estimated.
- The equation for  $^2V_t$  is used to estimate the error discovery rates after the initial estimates for B and E have been made.
- As more data becomes available from later phases, new estimates for E and  $\Xi$  can be made to improve the accuracy of the model.

# Phase-Based Model (cent'd)

Estimation of number of latent errors

- The number of errors per KSLOC removed through the n'th phase is:

$$V_t = E[1 - e^{-Bn^2}]$$

- The number of errors remaining in the software is just

$$Ee^{-Bn^2}$$

times the number of source line statements,  $S$ .

ISSRE'93 Tutorial

Software Reliability Modeling Techniques and Tools

# Part I& Quantitative Criteria for Model Selection

1. Relative accuracy ("prequential likelihood")
2. Bias ("u-plot")
3. Bias trend ("y-plot")
4. Scatter plot of u
5. Model noise

*Total time: 20 minutes*

# Model Concerns

- Accurate Data Collection During Test
- Accurate Operational Profile description during test
- Models are Primarily used during the testing phase, which is late in the development cycle
- Estimation of parameters is not always possible, and sometimes it is mathematically intractable.
- Reliable models for multiple systems have not been developed
- No well-established criteria for model selection. A model should:
  - Give good predictions of future failure behavior
  - Compute useful quantities
  - Be simple to use
  - Be applicable to a wide variety of situations
  - Be based on sound assumptions

# Criteria for Model Selection

- When software reliability models first appeared, it was felt that a process of refinement would produce “definitive” models that would apply to all development and test situations.
- Current situation
  - Over 40 models have been published in the literature.
  - Recent studies indicate that the accuracy of models is variable
  - Analysis of the particular context in which reliability measurement is to take place so as to decide a priori which model to use does not seem possible.

# Criteria for Model Selection (cont'd)

Analysis of a model's predictive quality can help user decide which model(s) to use.

- Simplest question a SRM user can ask is, "How reliable is the software at this moment?"
- The time to the next failure,  $\tau_i$ , is usually predicted using observed times to failure  $t_1, t_2, \dots, t_{i-1}$
- In general, predictions of  $\tau_i$  can be made using observed times to failure  $t_1, t_2, \dots, t_{i-K}, K > 0$

The results of predictions made for different values of  $K$  can then be compared. If a model produced "self-consistent" results for differing values of  $K$ , this indicates that its use is appropriate for the data on which the particular predictions were made.

HOWEVER, THIS PROVIDES NO GUARANTEE THAT THE PREDICTIONS ARE CLOSE TO THE TRUTH.

# Criteria for Model Selection (cent'd)

## Pre-quential likelihood

- The pdf of  $F_i(t)$  for  $T_i$  is based on observations  $t_1, t_2, \dots, t_{i-1}$ . The pdf of  $F_i(t)$ ,  $f_i(t)$ , is  $\partial F_i(t)/\partial t$ .
- For one-step ahead predictions of  $T_{j+1}, T_{j+2}, \dots, T_{j+n}$ , the prequential likelihood is:

$$PL_n = \prod_{i=j+1}^{j+n} \tilde{f}_i(t_i)$$

- Two prediction systems, A and B, can be evaluated by finding the prequential likelihood ratio:

$$PLR_n = \frac{\prod_{i=j+1}^{j+n} \tilde{f}_i^A(t_i)}{\prod_{i=j+1}^{j+n} \tilde{f}_i^B(t_i)}$$

If  $PLR_n \rightarrow 00$  as  $n \rightarrow 00$ , then B is discarded in favor of A.

# Criteria for Model Selection (cont'd)

## Prequential Likelihood Example

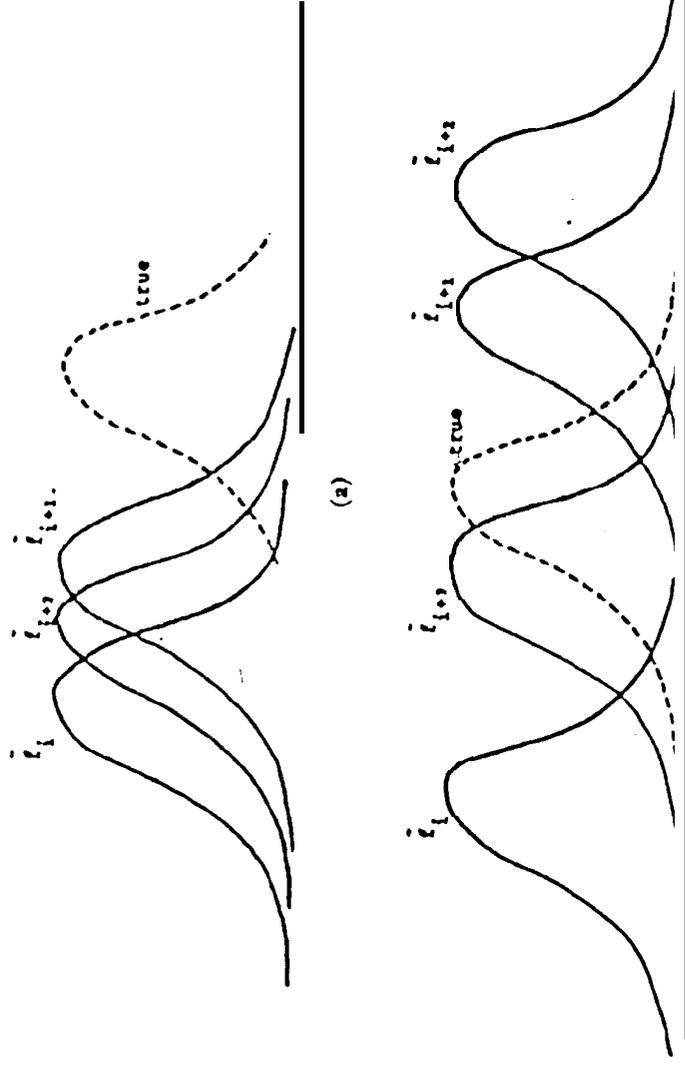


Fig. 8. These predictions have (a) high "bias" and low "noise," (b) high "noise" and low "bias."

# Criteria for Model Selection (cent'd)

The “u-plot” can be used to assess the predictive quality of a model

- Given a predictor,  $\tilde{F}_i(\tilde{t})$ , that estimates the probability that the time to the next failure is less than  $t$ . Consider the sequence

$$u_i = \tilde{F}_i(t_i)$$

where each  $u_i$  is a probability integral transform of the observed  $t_i$  using the previously calculated predictor  $\tilde{F}_i$  based upon  $t_1, t_2, \dots, t_{i-1}$

- If each  $\tilde{F}_i$  were identical to the true, but hidden,  $F_i$ , then the  $u_i$  would be realizations of independent random variables with a uniform distribution in  $[0, 1]$ .

The problem then reduces to seeing how closely the sequence  $\{u_i\}$  resembles a random sample from  $[0, 1]$

# Criteria for Model Selection - U-Plots for JM and LV Models

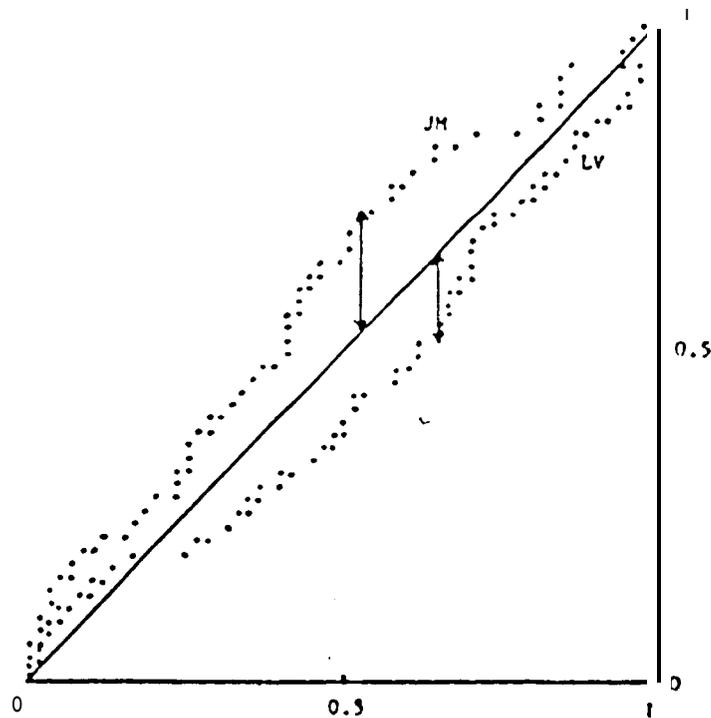


Fig. 4. LV, JM u-plots. data of Table I. Steps omitted for clarity. Note that these are reproduced from line-printer plots and do not correspond exactly to true plot.

# Criteria for Model Selection (cent'd)

The y-plot

- Temporal ordering is not shown in a u-plot. The y-plot addresses this deficiency
- To generate a y-plot, the following steps are taken:
  - Compute the sequence of  $\{u_i\}$
  - For each  $u_i$ , compute  $x_i = \ln(1 - u_i)$

Obtain  $y_i$  by computing  $\frac{\sum_1^i x_j}{\sum_1^m x_j}$ ,  $i \leq m$

If the  $\{u_i\}$  really do form a sequence of independent random variables in  $[0, 1]$ , the slope of the plotted  $y_i$  will be constant

# Criteria for Model Selection - Y-Plots for JM and LV Models

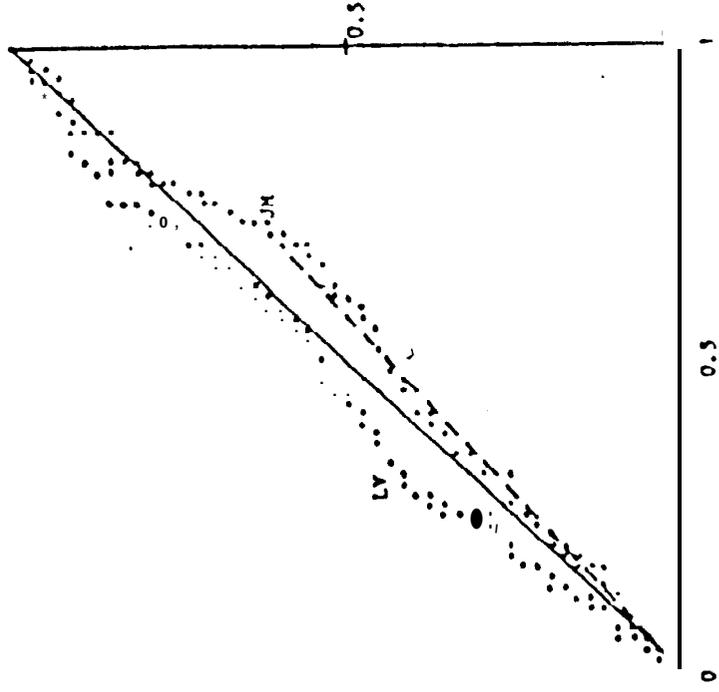


Fig. 6. JM and LV y-plots for data of Table I. Again, these are line-printer plots and points do not correspond exactly to true points.

# Criteria for Model Selection

Scatter Plot of  $u_i$

- If  $\{u_i\}$  are a sequence of independent random variables in  $[0, 1]$ , a scatter plot of  $\{u_i\}$  vs.  $i$  should be uniform.

ISSRE'93 Tutorial

Software Reliability Modeling Techniques and Tools

## **Part IV: Linear Combination Modeling Approach**

- 1. Application procedure**
- 2. The linear combination models**
- 3. Model application results**

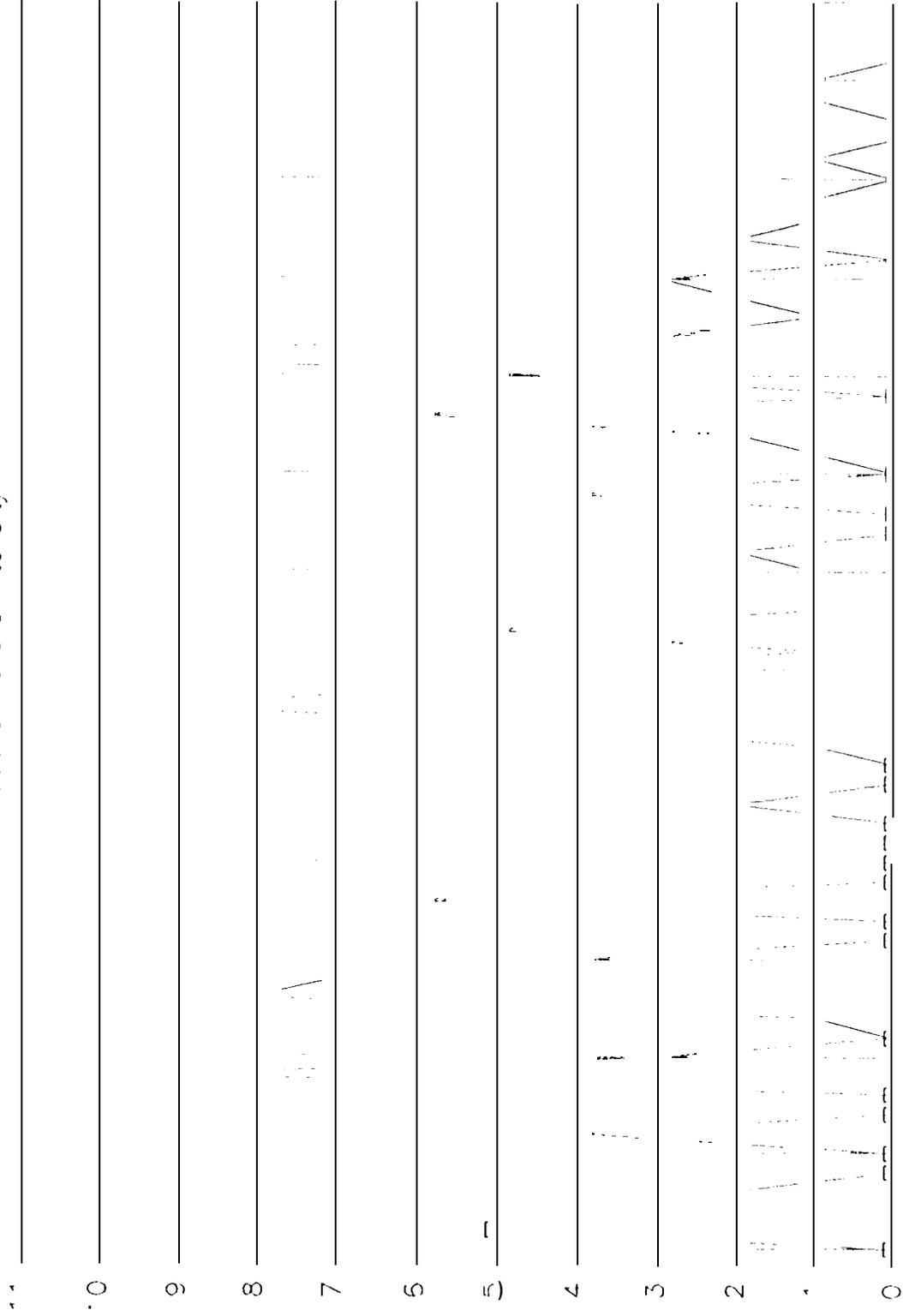
*Total time: 20 minutes*

## A PROCEDURE TO APPLY SOFTWARE RELIABILITY MODELS

- (1) Examine model assumptions and **constraints**
- (2) Identify model selection criteria
- (3)** Select a candidate set of models
- (4)** Collect data for model applications
- (5) Choose procedure to apply models (use of database)
- (6) Identify software tools
- (7)** Select best models
- (8) Tailor to the special environment (e.g., JPL)

# VOYAGER Software Reliability

Operational Failure Intensity



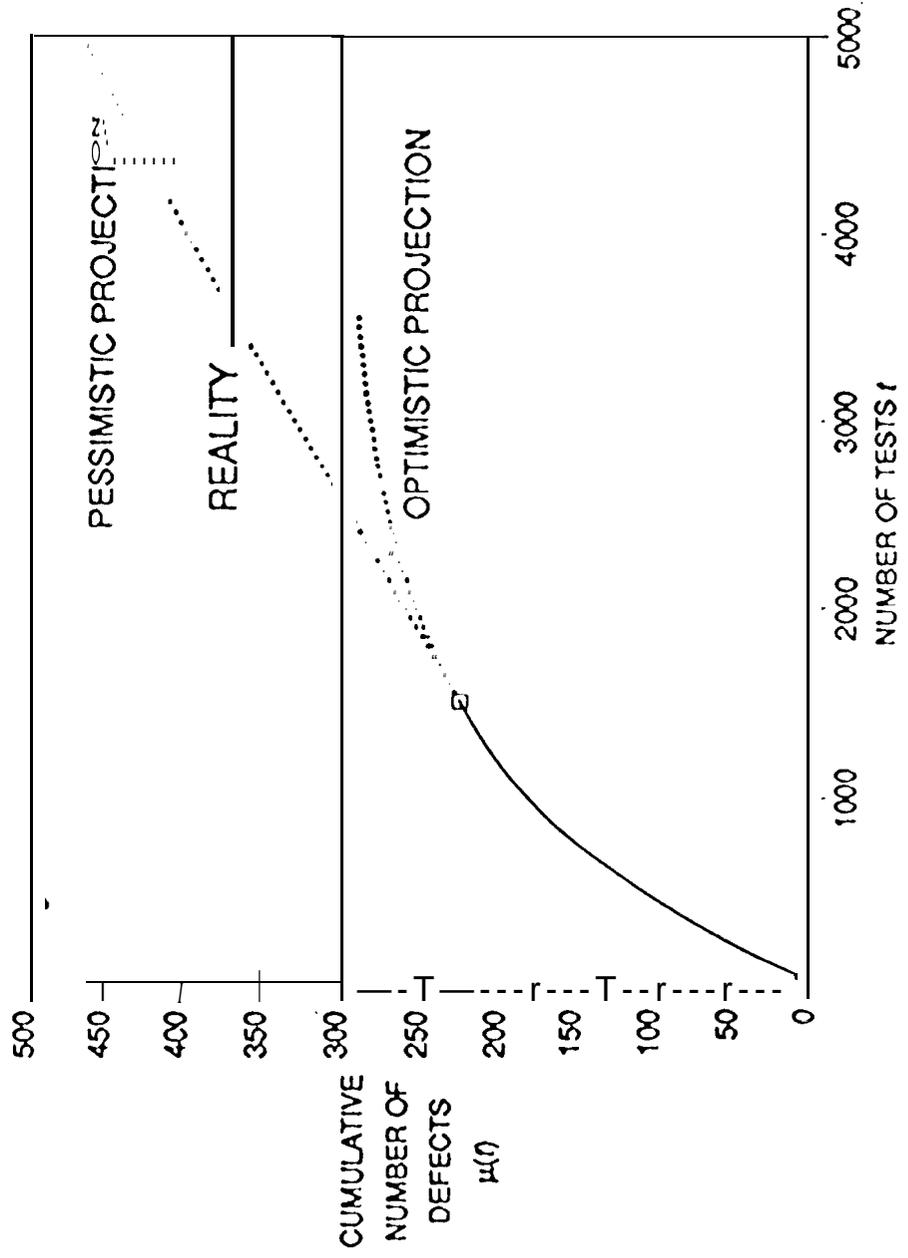
Failures per Calendar Week

Calendar Week

## Problems in Software Reliability Modeling

- Over 40 models have been published in the literature.
- Significant differences exist among the performance of these historical models.
- It was felt that a process of refinement would produce a "definitive" model.
- Software reliability measurers' fantasy: one model for all cases.
- The reality is: no single model could be determined *a priori* as the best model during measurement.

# THE REALITY IS IN BETWEEN



# A New Approach in Measuring Software Reliability

- (1) Identify a basic set of models (called *component* models).
- (2) Select models that tend to cancel out in their biased predictions.
- (3) Keep track of the software the software failure data with all the component models.
- (4) Apply certain criteria to weigh the selected component models and form one or several *Linear Combination Models* for final predictions.

# A Set of Linear Combination Models

Selected component models: *GO*, *MO*, *LV*

(1) *ELC* - Equally-Weighted Linear Combination Model

$$ELC = \frac{1}{3} GO + \frac{1}{3} MO + \frac{1}{3} LV$$

(2) *MLC* - Median-Oriented Linear Combination Model

(3) *ULC* - Unequally-Weighted Linear Combination Model

$$ULC = \frac{1}{6} O + \frac{4}{6} M + \frac{1}{6} P$$

O: optimistic M: median P: pessimistic

(4) *DLC* - Dynamically-Weighted Linear Combination Model

Weighings are determined by dynamically calculating the posterior "pre-quential Likelihood" of each model as a meta-predictor.

## The Selected Models for Comparisons

- (1) Jelinski-Moranda Model (JM)
- (2) Goel-Okumoto Model(GO)
- (3) Musa-Okumoto Model (MO)
- (4) Duane Model (DU)
- (5) Littlewood Model (LM)
- (6) Littlewood-Verrall Model (LV)

## Data Set 1: Model Comparisons for the Voyager Project

Voyager Flight Software (133 data points/starting data-2)										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Accuracy	-894.7 (10)	-573.7 (7)	-571.5 (6)	-586.6 (8)	-829.9 (9)	-549.1 (2)	-554.0 (3)	-557.8 (4)	-570.3 (5)	-543.1 (1)
Bias	.2994 (9)	.2849 (6)	.2849 (6)	.2703 (5)	.2994 (9)	.0793 (1)	.2084 (3)	.2438 (4)	.2849 (6)	.2078 (2)
Trend	.0995 (9)	.0965 (7)	.0957 (5)	.2551 (10)	.0994 (8)	.0876 (3)	.0872 (2)	.0951 (4)	.0962 (6)	.0866 (1)
Noise	$\infty$ (9)	13.81 (4)	9.225 (3)	8.402 (1)	$\infty$ (9)	24.51 (8)	15.15 (6)	12.64 (5)	9.129 (2)	17.47 (7)
Rank	(10)	(7)	(6)	(7)	(9)	(2)	(2)	(4)	(5)	(1)

RECOMMENDED MODELS: 1. DLC 2. ELC 2. LV

## Data Set 2: Model Comparisons for the Galileo Flight Project

Galileo Flight Software (224 data points/starting data-24)										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Accuracy	-1074 (5)	-1075 (7)	-1078 (9)	-1098 (10)	-1074 (5)	-1051 (4)	-1019 (2)	-1035 (3)	-1077 (8)	-984.7 (1)
Bias	.3378 (6)	.3378 (6)	.3379 (9)	.1944 (1)	.3382 (10)	.2592 (5)	.1991 (2)	.2569 (4)	.3378 (6)	.2159 (3)
Trend	.4952 (6)	.4954 (7)	.5041 (10)	.4618 (5)	.4954 (7)	.1082 (1)	.2781 (3)	.3271 (4)	.5017 (9)	.2484 (2)
Noise	2.607 (4)	2.593 (3)	2.395 (1)	4.541 (6)	2.624 (5)	23.33 (9)	17.72 (8)	13.80 (7)	2.584 (2)	23.96 (10)
Rank	(5)	(7)	(10)	(6)	(9)	(4)	(1)	(3)	(8)	(2)

RECOMMENDED MODELS: 1. ELC 2. DLC 3. ULC

## Data Set 3: Model Comparisons for the Galileo CDS Subsystem

Galileo CDS Flight Software (358 data points/starting data-152)										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Accuracy	-643.0 (6)	-639.3 (5)	-681.1 (8)	-728.5 (10)	-643.0 (6)	-612.3 (2)	-618.7 (3)	-626.9 (4)	-681.1 (8)	-606.1 (1)
Bias	.1783 (6)	.1783 (6)	.1700 (2)	.1748 (5)	.1784 (8)	.2581 (10)	.1732 (4)	.1599 (1)	.1700 (2)	.1845 (9)
Trend	.3450 (6)	.3408 (5)	.4262 (9)	.4282 (10)	.3450 (6)	.2426 (1)	.2855 (3)	.3072 (4)	.4261 (8)	.2618 (2)
Noise	4.042 (8)	3.908 (7)	2.673 (4)	2.287 (1)	4.042 (8)	2.564 (2)	2.853 (5)	2.958 (6)	2.672 (3)	11.19 (10)
Rank	(8)	(6)	(6)	(8)	(10)	(1)	(1)	(1)	(4)	(5)

RECOMMENDED MODELS: 1. LV 1. ELC 1. ULC

## Data Set 4: Model Comparisons for the Magellan Project

Magellan Flight Software (197 data points/starting data-50)										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Accuracy	-627.1 (6)	-627.1 (6)	-627.1 (6)	-616.0 (2)	-627.1 (6)	-622.9 (5)	-619.1 (3)	-622.3 (4)	-627.1 (6)	-609.2 (1)
Bias	.2968 (6)	.2968 (6)	.2969 (8)	.1858 (1)	.2969 (8)	.3483 (10)	.2140 (2)	.2461 (4)	.2968 (5)	.2396 (3)
Trend	.2399 (6)	.2399 (6)	.2399 (6)	.2180 (5)	.2399 (6)	.1429 (2)	.1400 (1)	.1871 (4)	.2399 (6)	.1049 (3)
Noise	1.007 (1)	1.007 (1)	1.007 (1)	2.003 (6)	1.009 (5)	5.563 (10)	4.260 (8)	3.363 (7)	1.007 (1)	4.982 (9)
Rank	(5)	(5)	(8)	(1)	(9)	(10)	(1)	(5)	(4)	(3)

RECOMMENDED MODELS: 1. DU 1. ELC **3. DLC**

## Data Set 5: Model Comparisons for the Alaska SAR Project

Alaska SAR Ground Software (367 data points/starting data-67)										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Accuracy	-915.7 (2)	-915.8 (6)	-915.7 (2)	-925.5 (10)	-915.7 (2)	-920.5 (9)	-916.2 (8)	-915.9 (7)	-915.7 (2)	-914.8 (1)
Bias	.3023 (1)	.3023 (1)	.3033 (1)	.4249 (10)	.3023 (1)	.3672 (9)	.3434 (7)	.3209 (6)	.3023 (1)	.3468 (8)
Trend	.0606 (4)	.0615 (6)	.0620 (7)	.0918 (9)	.0606 (4)	.1009 (10)	.0586 (3)	.0528 (1)	.0620 (7)	.0569 (2)
Noise	1.587 (4)	1.540 (3)	1.395 (1)	1.650 (6)	1.589 (5)	3.189 (10)	2.220 (9)	1.853 (7)	1.413 (2)	1.949 (8)
Rank	(1)	(5)	(1)	(9)	(3)	(10)	(8)	(7)	(3)	(6)

Recommended MODELS: 1. JM 1. MO

## Overall Model Comparisons Using All Four Criteria

Summary of Model Ranking for Each Data by All Four Criteria										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Voyager	(10)	(7)	(6)	(7)	(9)	(2)	(2)	(4)	(5)	(1)
Galileo	(5)	(7)	(10)	(6)	(9)	(4)	(1)	(3)	(8)	(2)
Galileo CDS	(8)	(6)	(6)	(8)	(10)	(1)	(1)	(1)	(4)	(5)
Magellan	(5)	(5)	(8)	(1)	(9)	(10)	(1)	(5)	(4)	(3)
Alaska SAR	(1)	(5)	(1)	(9)	(3)	(10)	(8)	(7)	(3)	(6)
Sum of Rank	29	30	31	31	40	27	13	20	24	17
“Handicap”	+9	+10	+11	+11	+20	+7	-7	0	+4	-3
Total Rank	(6)	(7)	(8)	(8)	(10)	(5)	(1)	(3)	(4)	(2)

## Overall Model Comparisons by the Accuracy Measure

Summary of Model Ranking for Each Data Using the Accuracy Measure										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Voyager	(10)	(7)	(6)	(8)	(9)	(2)	(3)	(4)	(5)	(1)
Galileo	(5)	(7)	(9)	(10)	(5)	(4)	(2)	(3)	(8)	(1)
Galileo CDS	(6)	(5)	(8)	(10)	(6)	(2)	(3)	(4)	(8)	(1)
Magellan	(6)	(6)	(6)	(2)	(6)	(5)	(3)	(4)	(6)	(1)
Alaska SAR	(2)	(6)	(2)	(10)	(2)	(9)	(8)	(7)	(2)	(1)
Sum of Rank	29	31	31	40	28	22	19	22	29	5
“Handicap”	+9	+11	+11	+20	+8	+2	-1	+2	+9	-15
Total Rank	(6)	(8)	(8)	(10)	(5)	(3)	(2)	(3)	(6)	(1)

## Overall Model Comparisons Using All Four Criteria

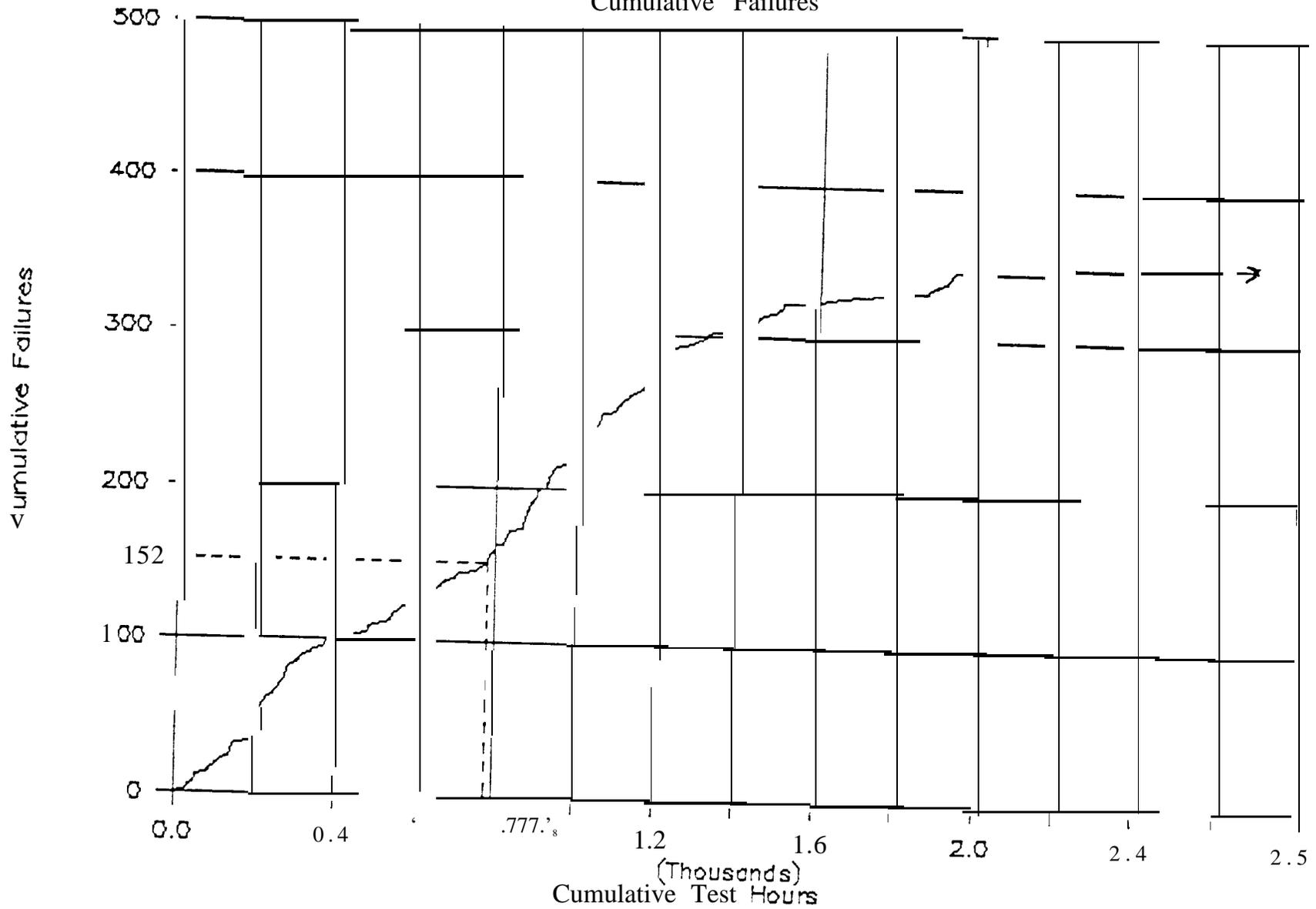
Summary of Model Ranking for Each Data by All Four Criteria										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Data 1 in RADC	(10)	(9)	(1)	(6)	(8)	(6)	(4)	(2)	(3)	(5)
Data 2 in RADC	(9)	(10)	(6)	(7)	(8)	(1)	(4)	(5)	(2)	(2)
Data 3 in RADC	(6)	(8)	(4)	(9)	(9)	(6)	(4)	(3)	(2)	(1)
Voyager	(10)	(7)	(6)	(7)	(9)	(2)	(2)	(4)	(5)	(1)
Galileo	(5)	(7)	(10)	(6)	(9)	(4)	(1)	(3)	(8)	(2)
Galileo CDS	(8)	(6)	(6)	(8)	(10)	(1)	(1)	(1)	(4)	(5)
Magellan	(5)	(5)	(8)	(1)	(9)	(10)	(1)	(5)	(4)	(3)
Alaska SAR	(1)	(5)	(1)	(9)	(3)	(10)	(8)	(7)	(3)	(6)
Sum of Rank	54	57	42	53	65	40	25	30	31	25
"Handicap"	+ 2 2	+25	+10	+21	+33	+8	-7	- 2	-1	-7
Total Rank	(8)	(9)	(6)	(7)	(10)	(5)	(1)	(3)	(4)	(1)

## Overall Model Comparisons by the Accuracy Measure

Summary of Model Ranking for Each Data Using the Accuracy Measure										
Model	JM	GO	MO	DU	LM	LV	ELC	ULC	MLC	DLC
Data 1 in RADC	(10)	(9)	(2)	(8)	(6)	(7)	(5)	(4)	(3)	(1)
Data 2 in RADC	(7)	(9)	(4)	(10)	(7)	(1)	(4)	(4)	(3)	(2)
Data 3 in RADC	(4)	(7)	(4)	(10)	(8)	(9)	(2)	(2)	(4)	(1)
Voyager	(10)	(7)	(6)	(8)	(9)	(2)	(3)	(4)	(5)	(1)
Galileo	(5)	(7)	(9)	(10)	(5)	(4)	(2)	(3)	(8)	(1)
Galileo CDS	(6)	(5)	(8)	(10)	(6)	(2)	(3)	(4)	(8)	(1)
Magellan	(6)	(6)	(6)	(2)	(6)	(5)	(3)	(4)	(6)	(1)
Alaska SAR	(2)	(6)	(2)	(10)	(2)	(9)	(8)	(7)	(2)	(1)
Sum of Rank	50	56	41	68	49	39	30	32	39	9
"Handicap"	+18	+24	+9	+36	+17	+7	-2	0	+7	-23
Total Rank	(8)	(9)	(6)	(10)	(7)	(4)	(2)	(3)	(4)	(1)

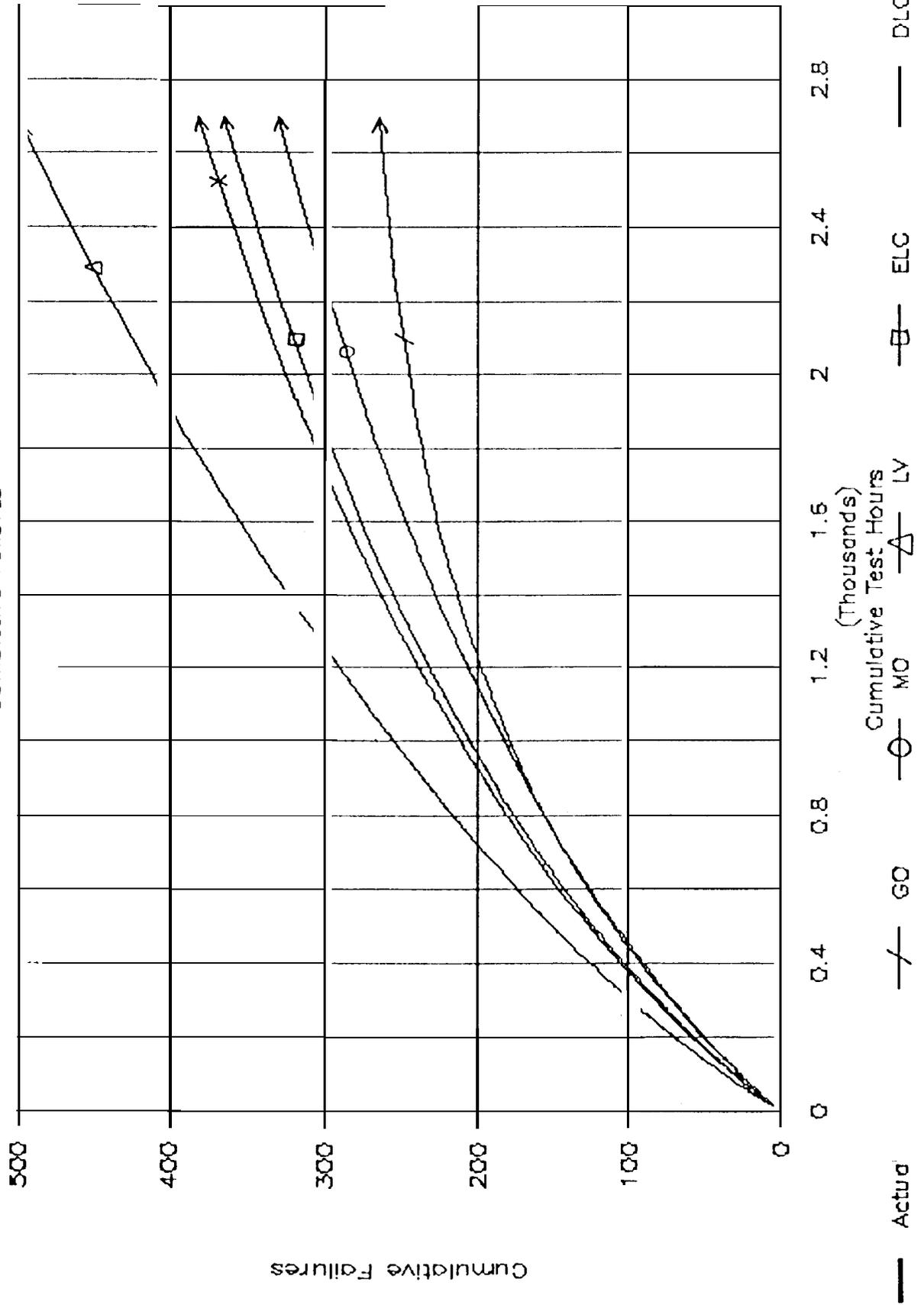
# GALILEO CDS

Cumulative Failures



# GALILEO CDS

Cumulative Failures



## Summary of Long-Term Predictions

Summary of Model Ranking for Long-Term Predictions Using M.S.E.					
	GO	MO	LV	ELC	DLC
Data 1 in RADC	2117(5)	687.4(4)	567.7(3)	266.7(2)	169.7(1)
Data 2 in RADC	1455(5)	1421(4)	246.1(1)	930.5(2)	955.7(3)
Data 3 in RADC	480.0(2)	253.2(1)	2067(5)	745.5(3)	779.8(4)
Voyager	1089(4)	782.9(2)	5283(5)	130.1(1)	876.7(3)
Galileo	4368(4)	4370(5)	539.3(1)	2171(3)	1791(2)
Galileo CDS	4712(5)	3073(3)	43 18(4)	1322(2)	1141(1)
Magellan	3247(4)	3248(5)	219.5(1)	1684(3)	1354(2)
Alaska SAR	60.22(3)	60.12(1)	104.45(5)	68.44(4)	60.15(2)
Sum of MSEs	17528.5	13896.6	13345.0	7317.3	7128.3
Sum of Ranks	(32)	(25)	(25)	(20)	(18)
Overall Rank	(5)	(4)	(3)	(2)	(1)

## Possible Extensions of the New Approach

- (1) Apply models other than GO, MO, and LV as component models.
- (2) Apply more than three component models.
- (3)** Apply other meta-predictors for weight assignments in DLC-type models.
- (4)** Apply user-determined weighting schemes, subject to project criteria and engineering judgments.
- (5) Apply combination models as component models for a hybrid combination.



SSRE'93 Tutorial

Software Reliability Modeling Techniques and Tools

## **Part V: Software Reliability Modeling Tools**

1. **AT&T Toolkit** (*15 minutes*)
2. **SMERFS** (*30 minutes*)
3. **SRMP** (*30 minutes*)
4. **CASRE** (*45 minutes*)

*Total time: 120 minutes*

# Software Reliability Measurement Tools

- AT&T Tool Chest (RELTAB, RELPLOT, RELSIM)
  - *John Musa* of AT&T
- Software Reliability Modeling Programs (SRMP)
  - *Bev Littlewood* of City University, London
- Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS)
  - *William Farr* of Naval Surface Warfare Center
- Computer-Aided Software Reliability Estimation Tool (CASRE)
  - *Allen Nikora* of JPL & *Michael R. Lyu* of Bellcore

# SMERFS Main Features

- Multiple Models (10)
- On-line Model Description Manual
- Two Parameter Estimation Methods
  - Least Square Method
  - Maximum Likelihood Method
- Main Goodness-of-fit Criteria – the Chi-square Statistic
- Simple Plots

# The SMERFS Tool Main Manual

- (1) Data Input
- (2) **D**ata Edit
- (3) Data Transformation
- (4) Data Statistics
- (5) Plot(s) of the Raw Data
- (6) Executions of the Models
- (7) Analyses of Model Fit
- (8) Stop Execution of SMERFS

# CASRE Main Features

- Multiple Models ( 0)
- Multiple Evaluation Criteria (4)
  - Accuracy ("Prequential Likelihood")
  - Bias ("U-Plot")
  - Trend ("Y-Plot")
  - Noise
- Enhanced Graphical User Interface
- Capability
  - Make Linear Combination Models

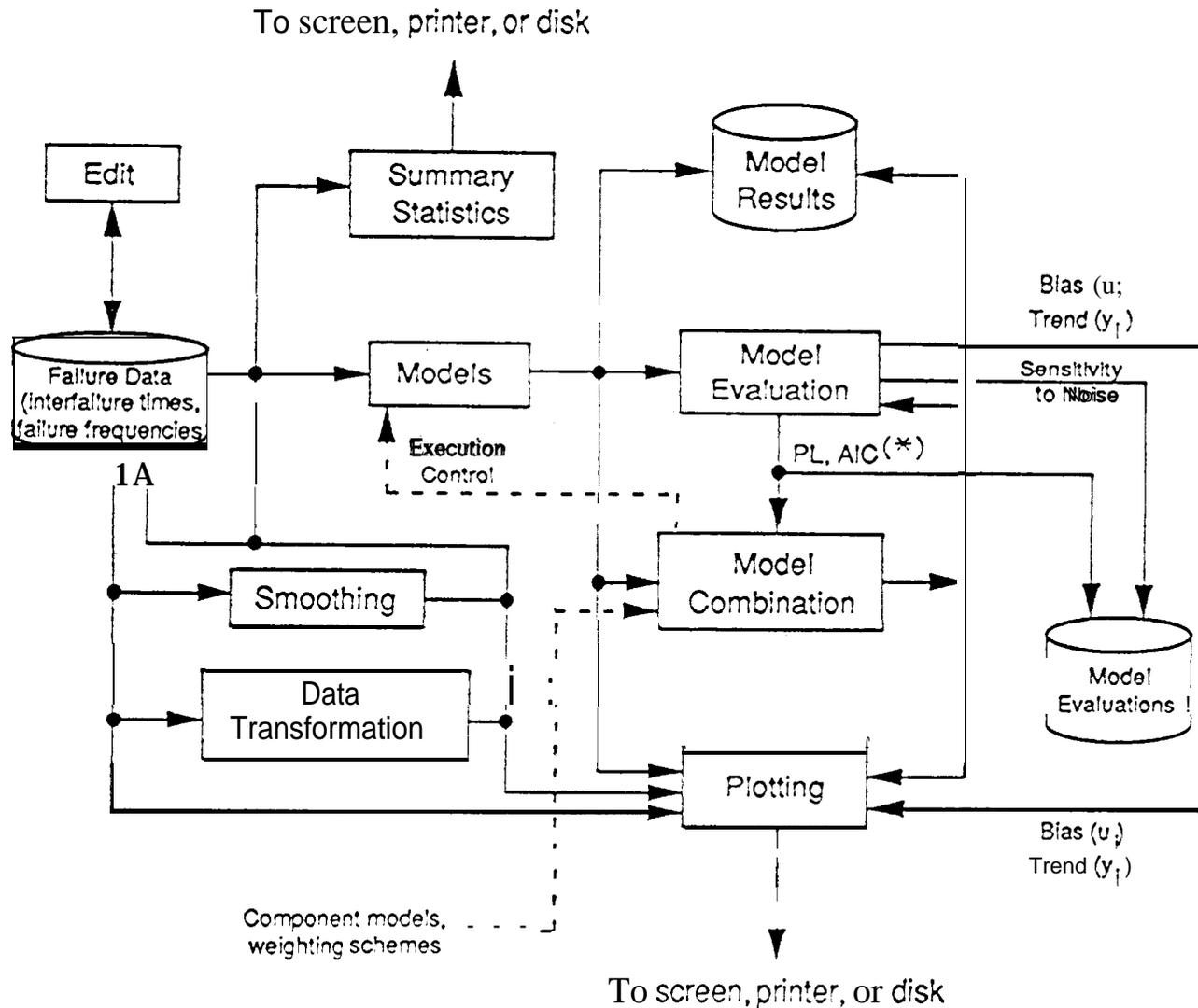
# The CASRE Tool for Computer-Aided Software Reliability Engineering

- (1) Data Modification
- (2) Failure Data Analysis
- (3) Modeling and Measurement
- (4) Modeling/Measurement Results **B**isplay

## The Model Evaluation Criteria

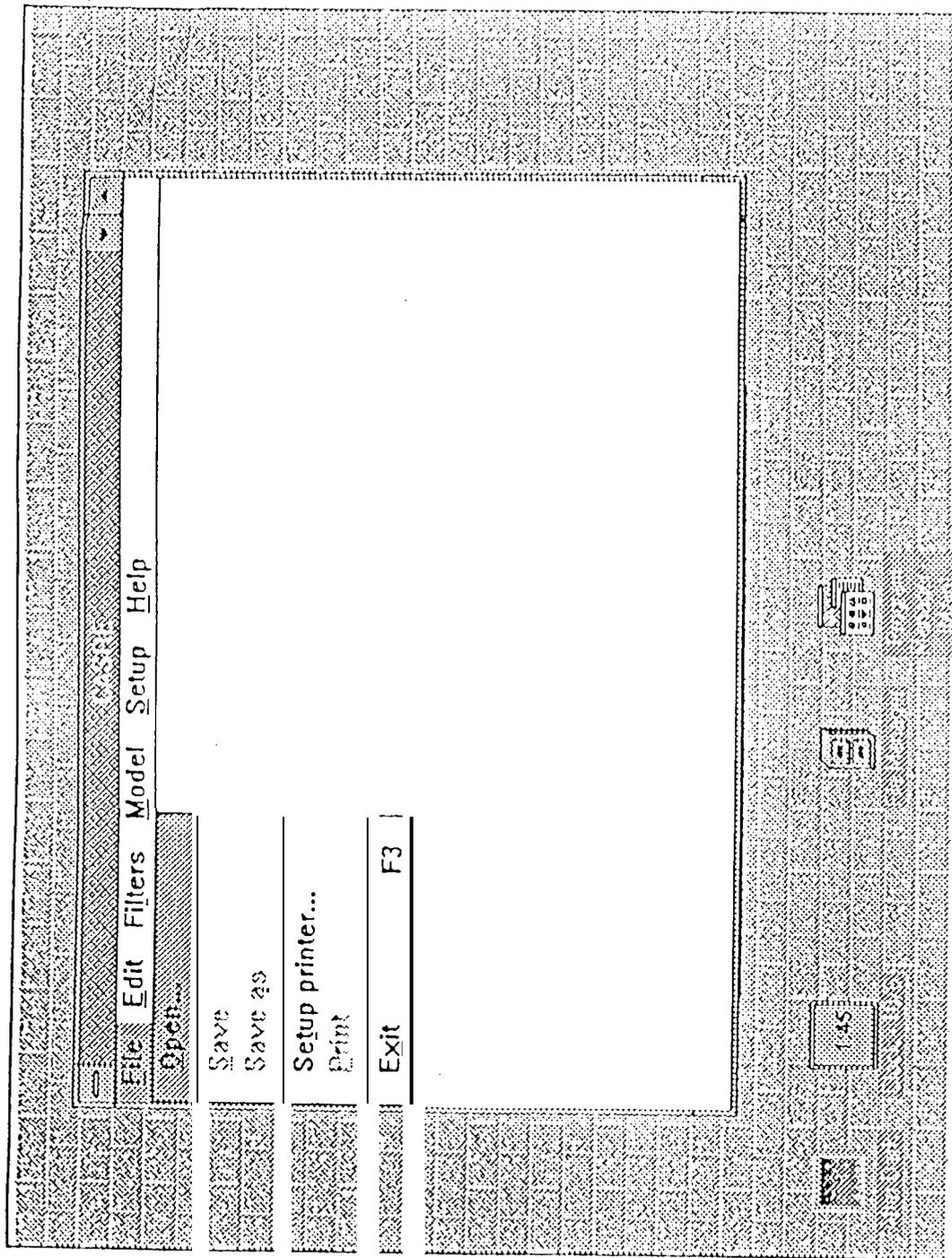
- (1) Accuracy ("Prequential Likelihood")
- (2) Bias ("U-Plot")
- (3) Trend ("Y-Plot")
- (4) Noise

# CASRE High-Level Architecture

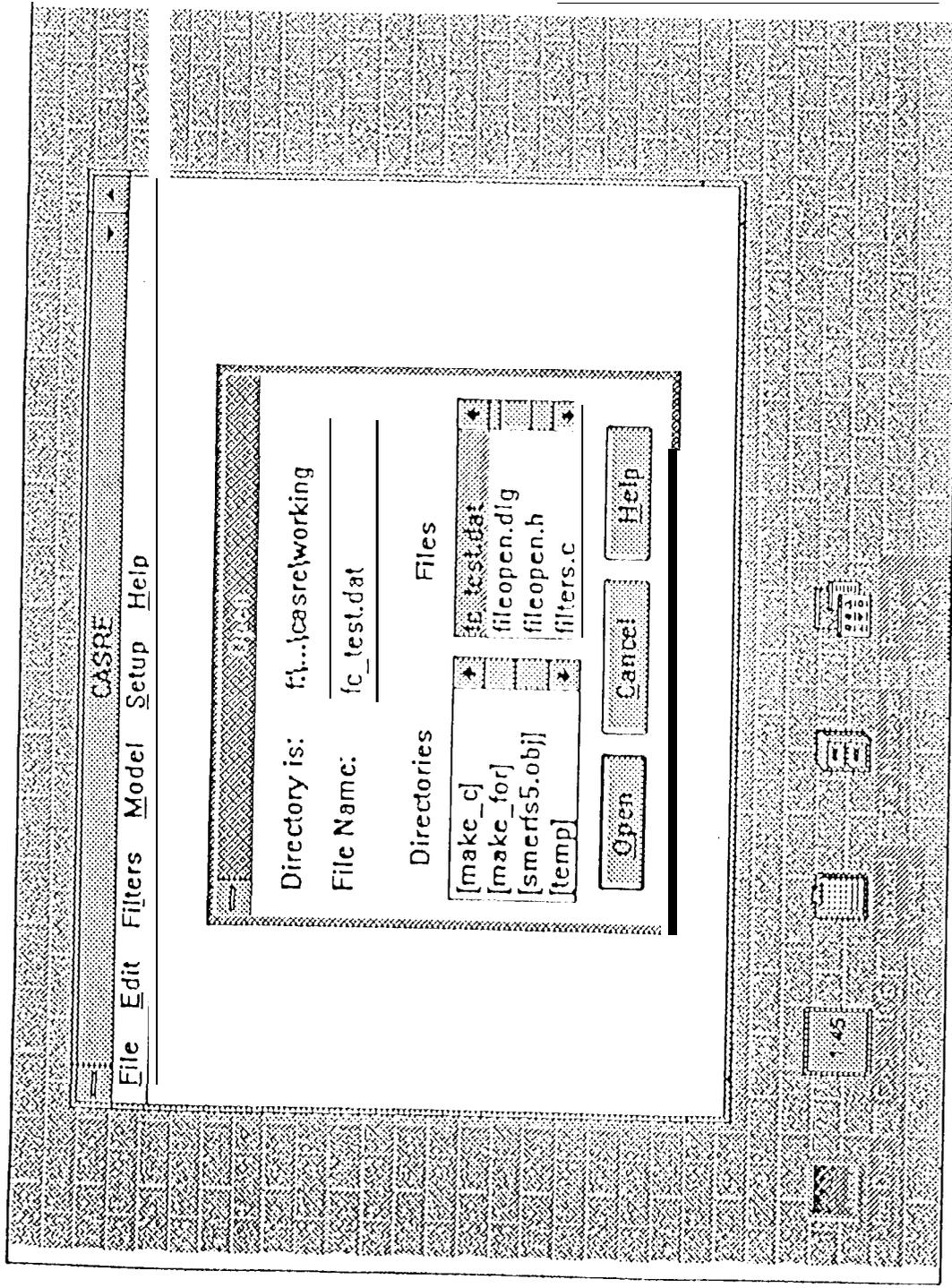


(\*) PL -- Prequential Likelihood  
AIC - Akaike Information Criterion

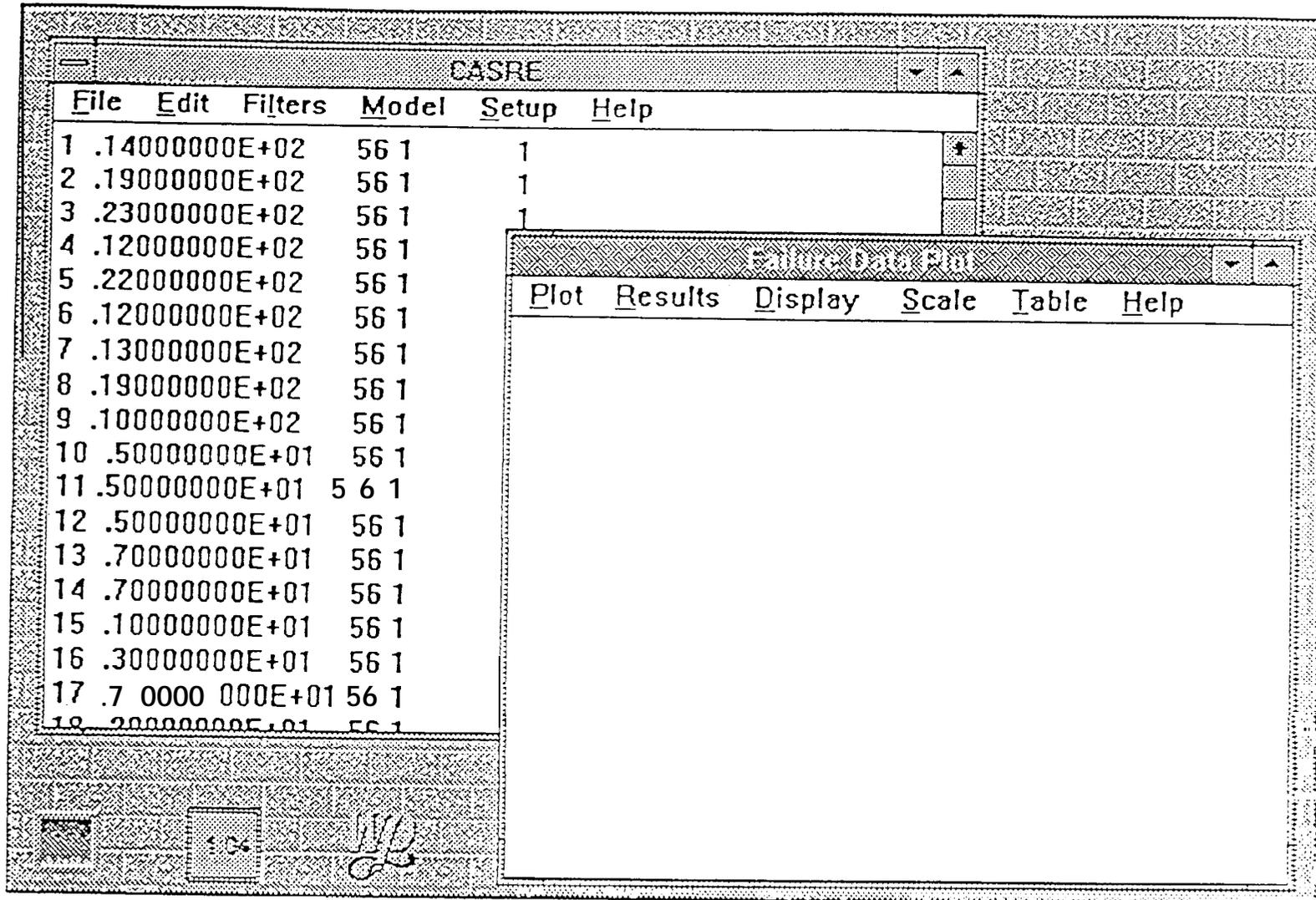
# CASRE Main Window File Menu



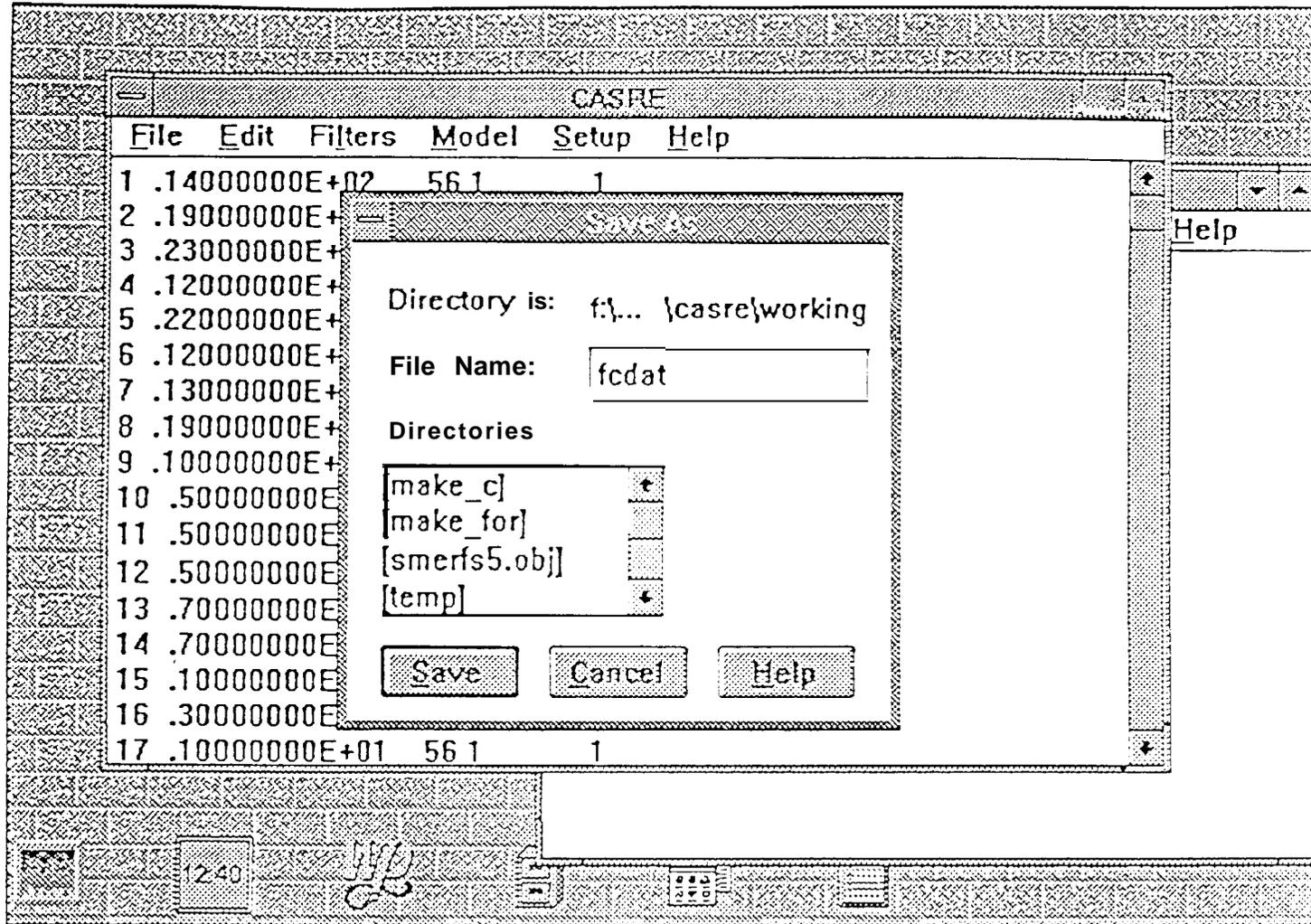
# Opening a Failure Data File



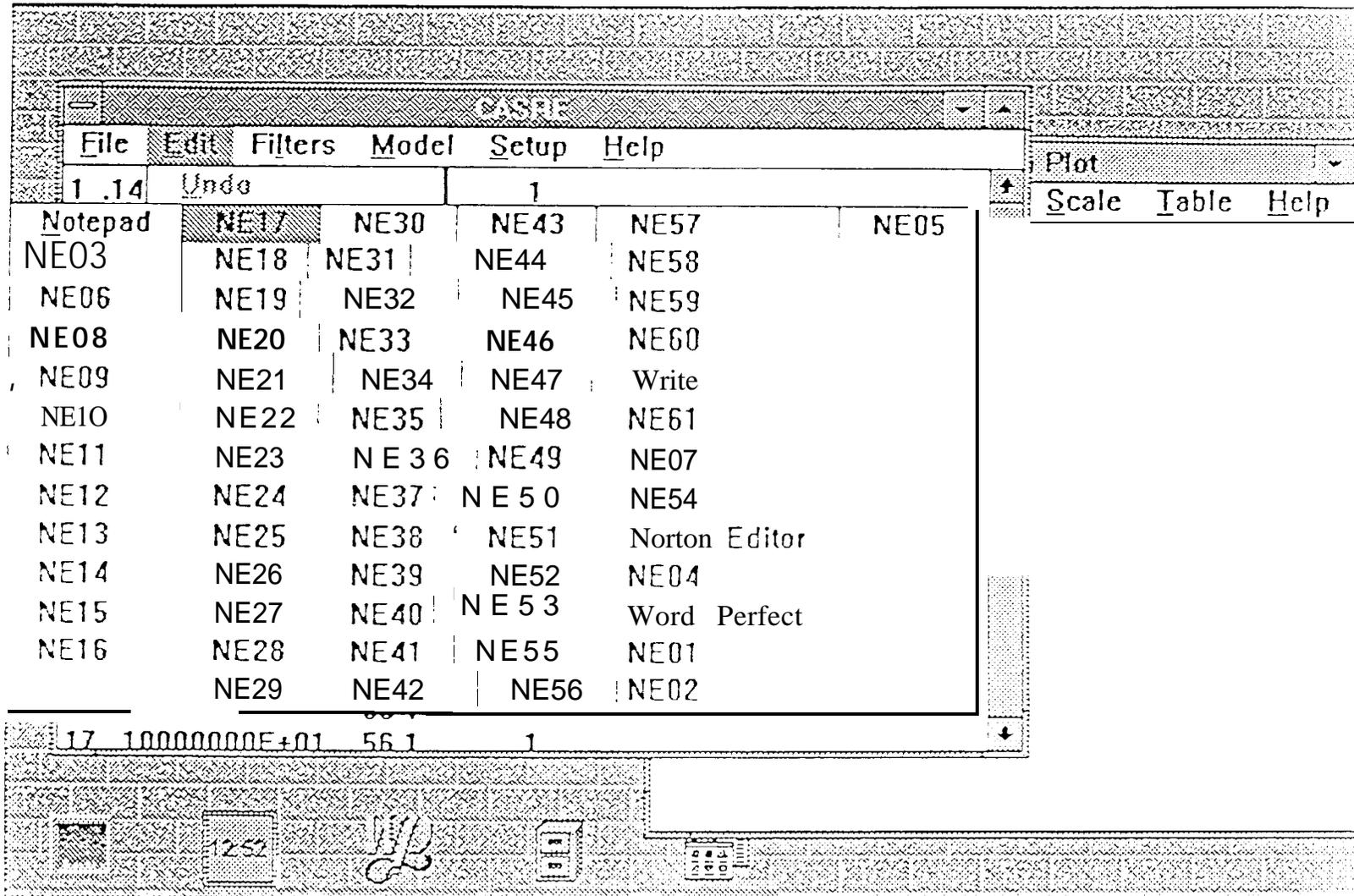
# CASRE Graphics Display Window



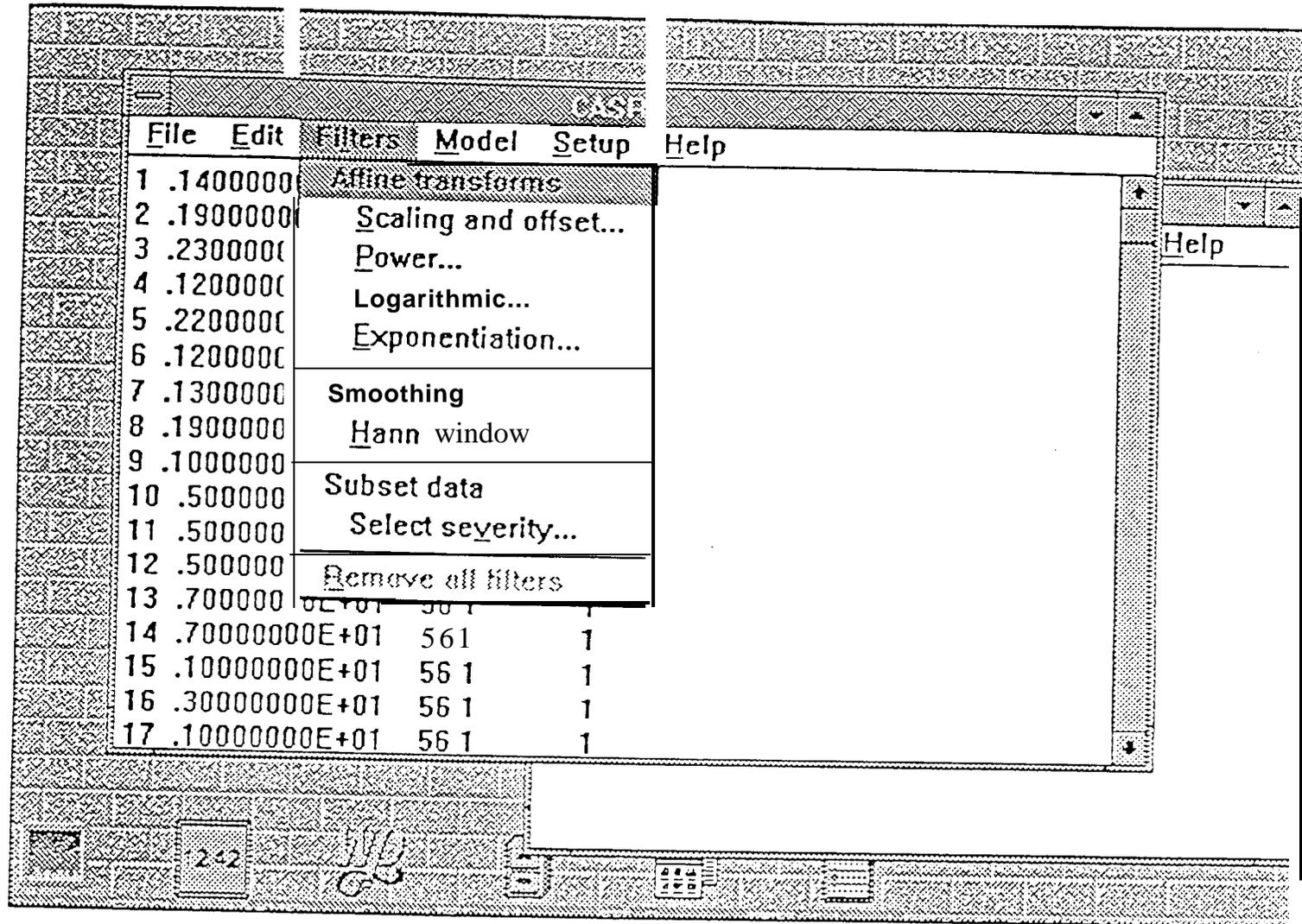
# Saving the Work Space Contents



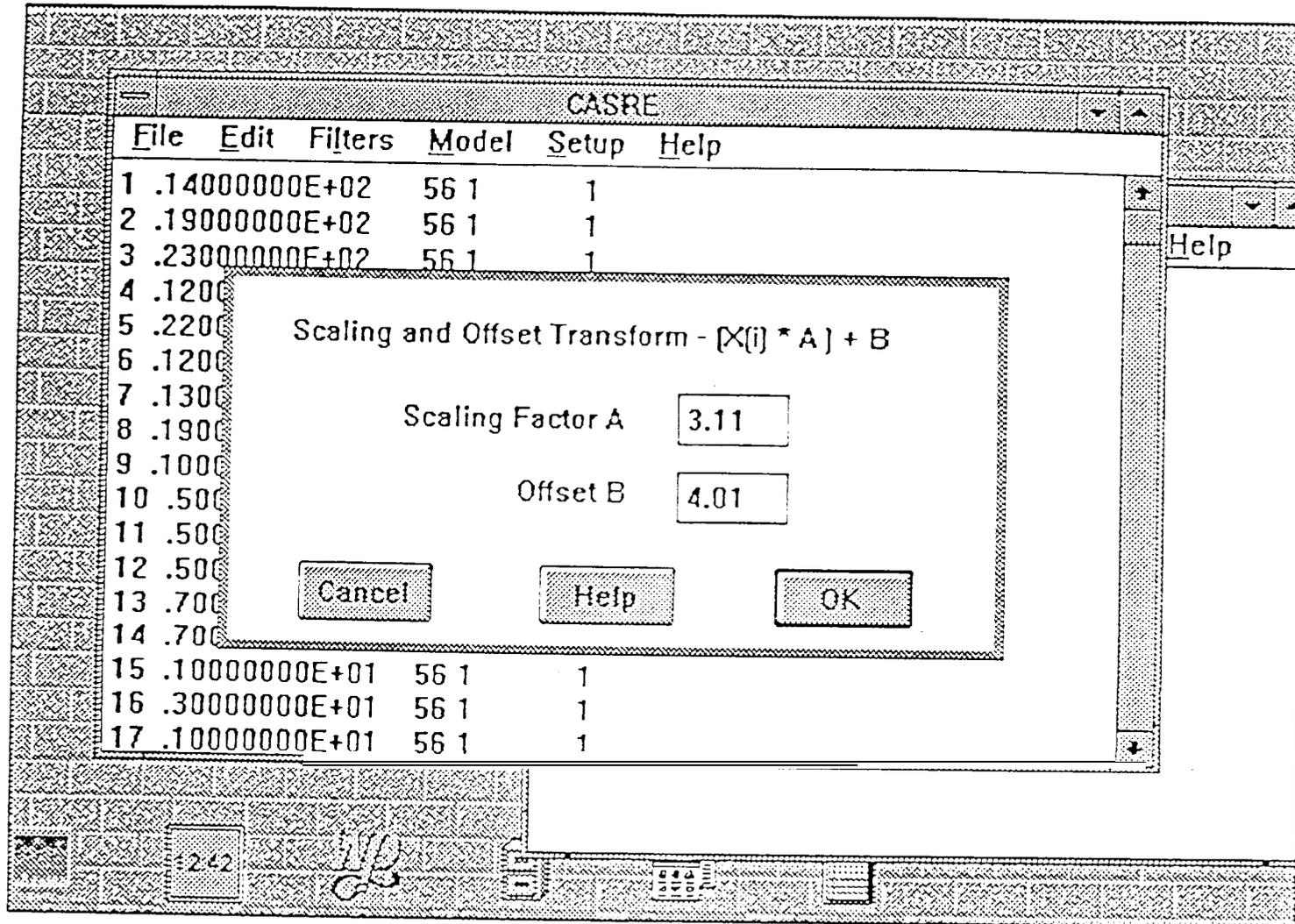
# CASRE Edit Menu and Submenu of External Applications



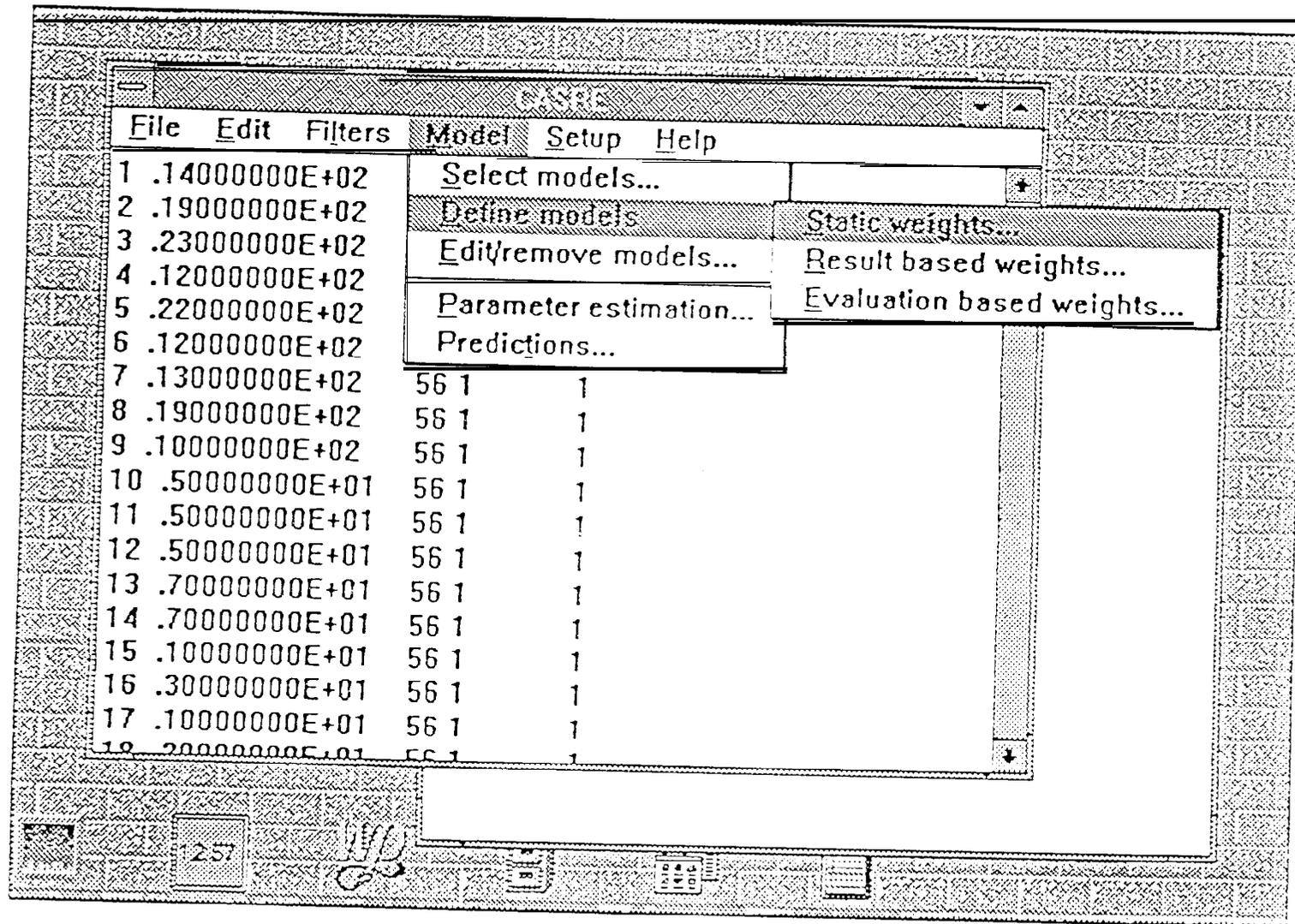
# Transformations and Smoothing Operations



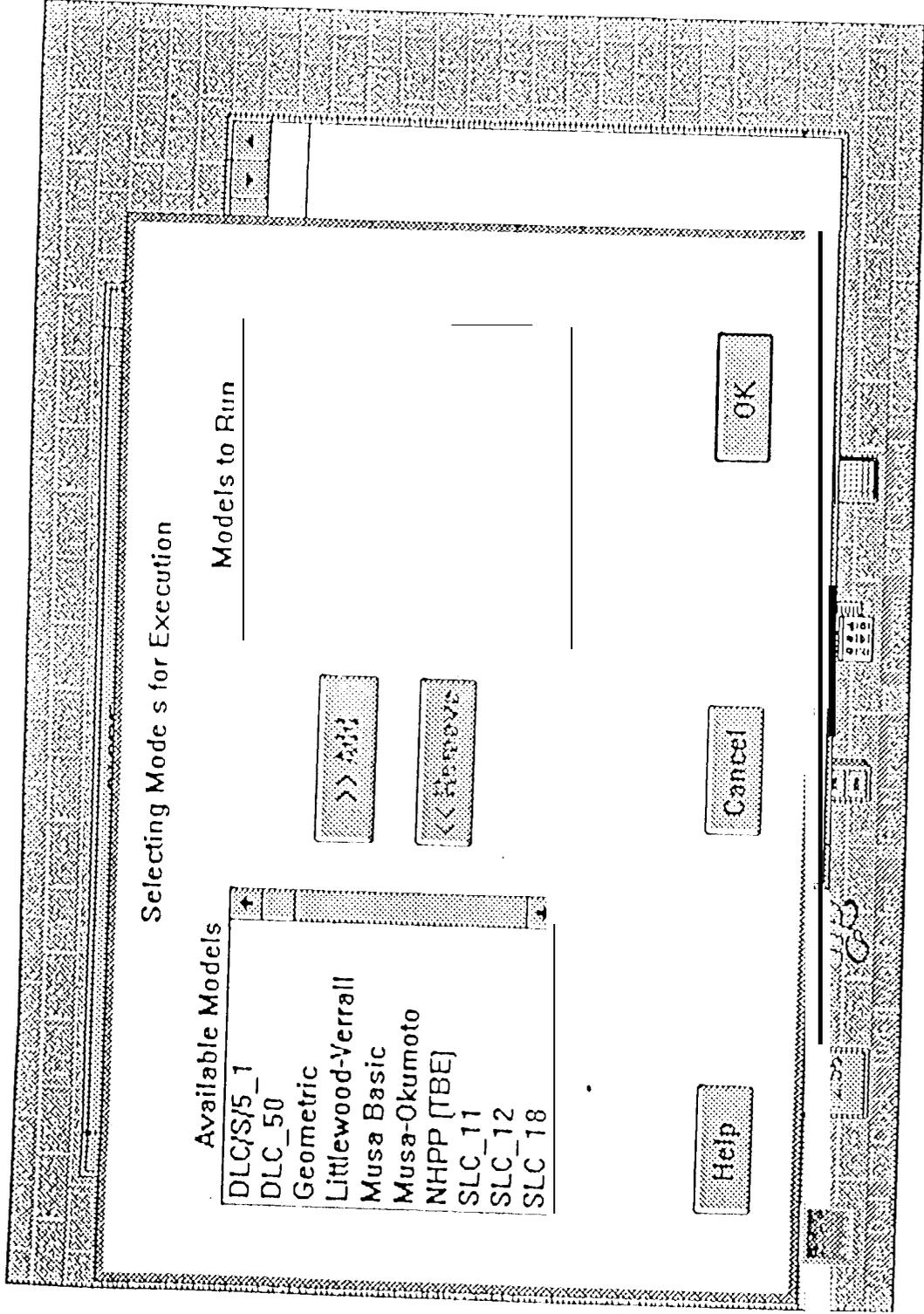
# Transformations and Smoothing Operations (cent'd)



# Model Selection and Definition



# Selecting Models for Execution



# Defining a Statically-Weighted Combination

**Statically Weighted Linear Model Combination**

Combination Name

**Available Models**

DLC/S/5_1	↑
Geometric	
Muss Basic	
SLC_11	
SLC_12	
SLC_18	
SLC_19	
SLC_20	
SLC_21	
SLC_22	↓

>> Add

<< Remove

**Current Combination**

DLC_50	1
Littlewood-Verrall	3
Muss-Okumoto	2
NHPP (TBE)	1

**View Weights**

Relative

Normalized

**Input Data Type**

Interfailure times

Failure counts

**Relative Weight:**

Set Weight:

Help Cancel OK

# Weighting Components by Modeling Results

Result-Based Linear Model Combination

Combination Name

Available Models

- DLC/S/5\_1
- Geometric
- Musa Basic
- NHPP (TBE)
- SLC\_12
- SLC\_18
- SLC\_19
- SLC\_20
- SLC\_21
- SLC\_22

Current Combination

- DLC\_50
- Littlewood-Verrall
- Musa-Okumoto
- SLC\_11

Weight by Rank

1	1
2	3
3	3
4	1

View Weights

- Relative
- Normalized

Evaluation window type

- Sliding
- Fixed

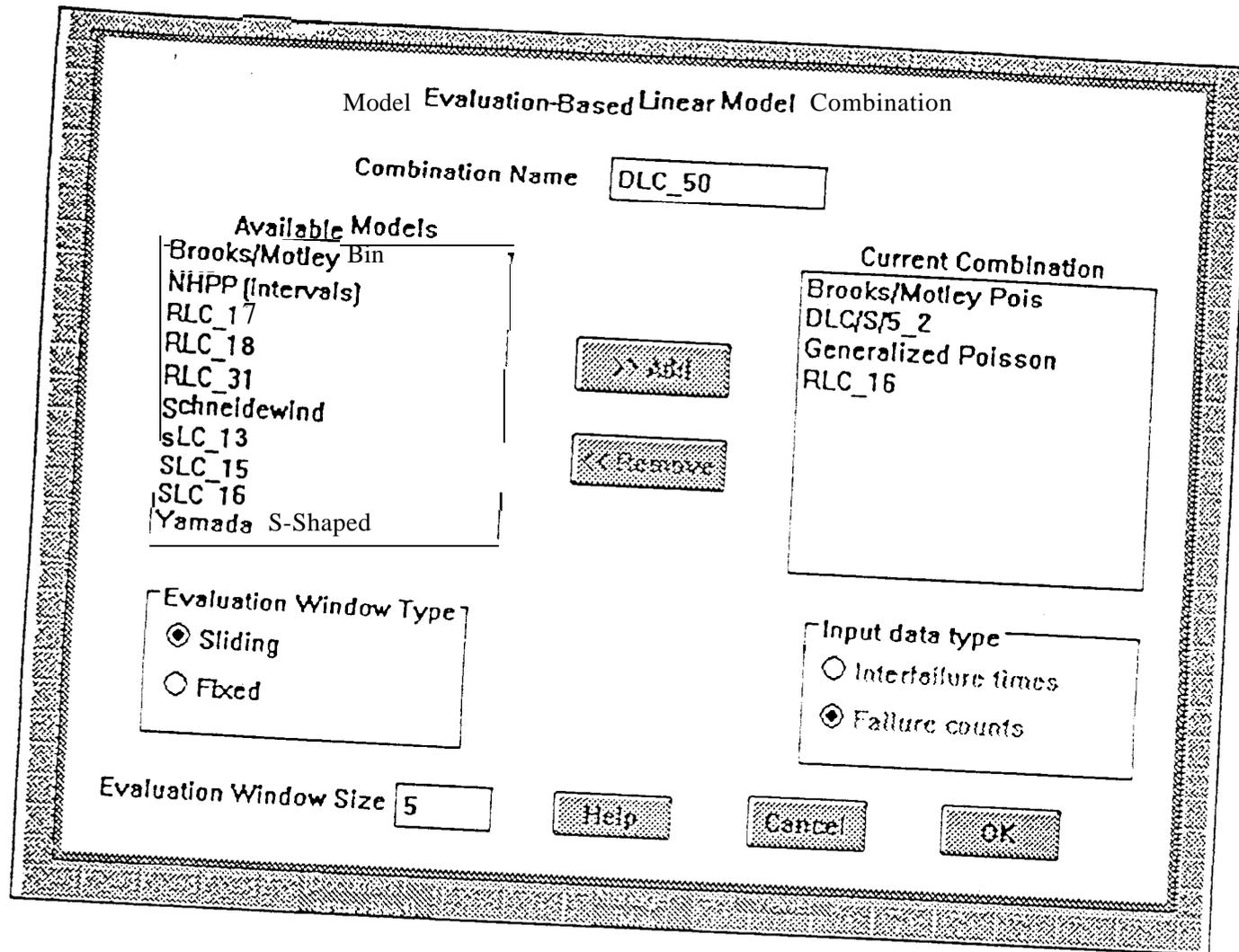
Relative Weight

Failure data type

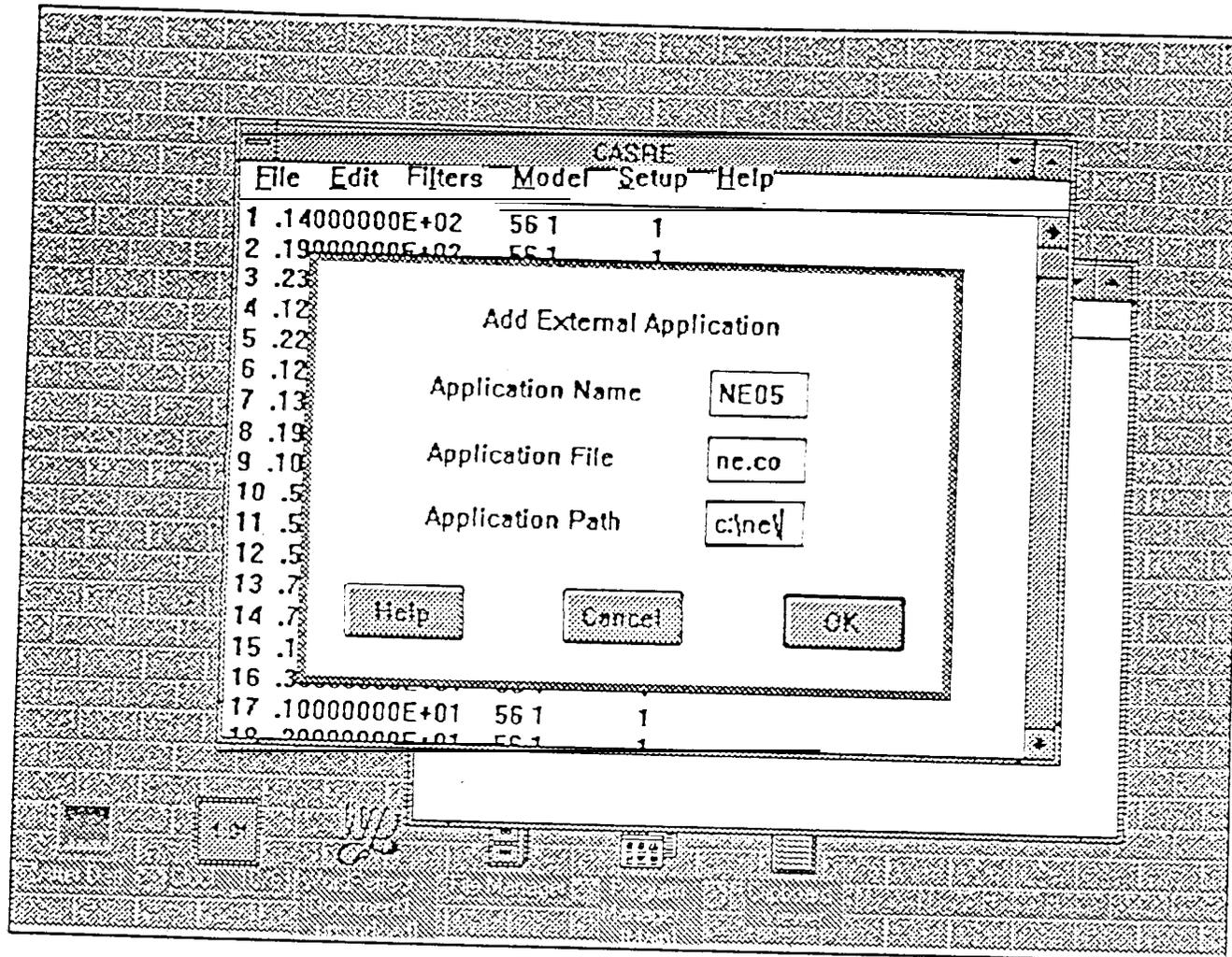
- Interfailure times
- Failure counts

Evaluation window size

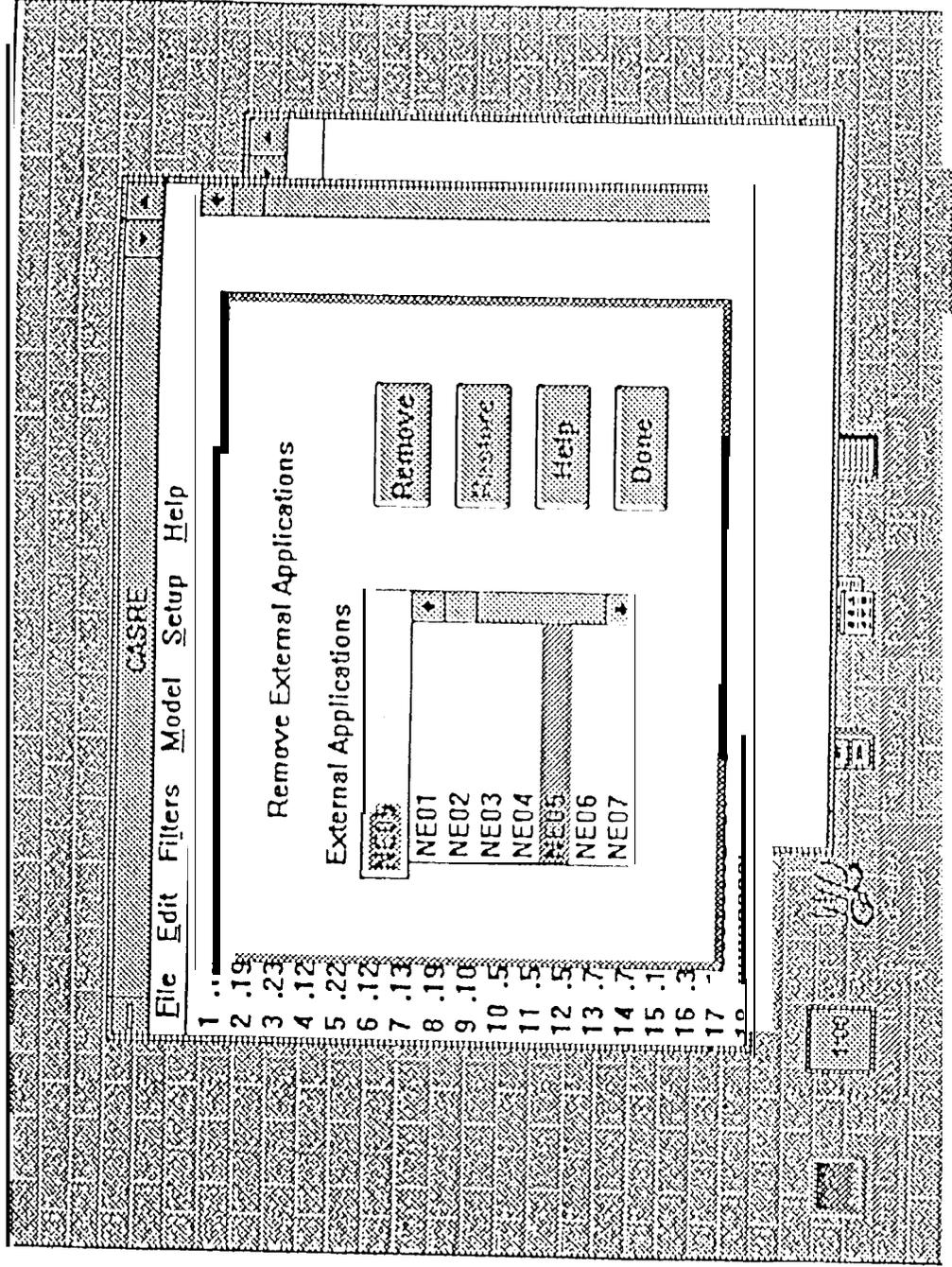
# Using Applicability Values to Weight Components



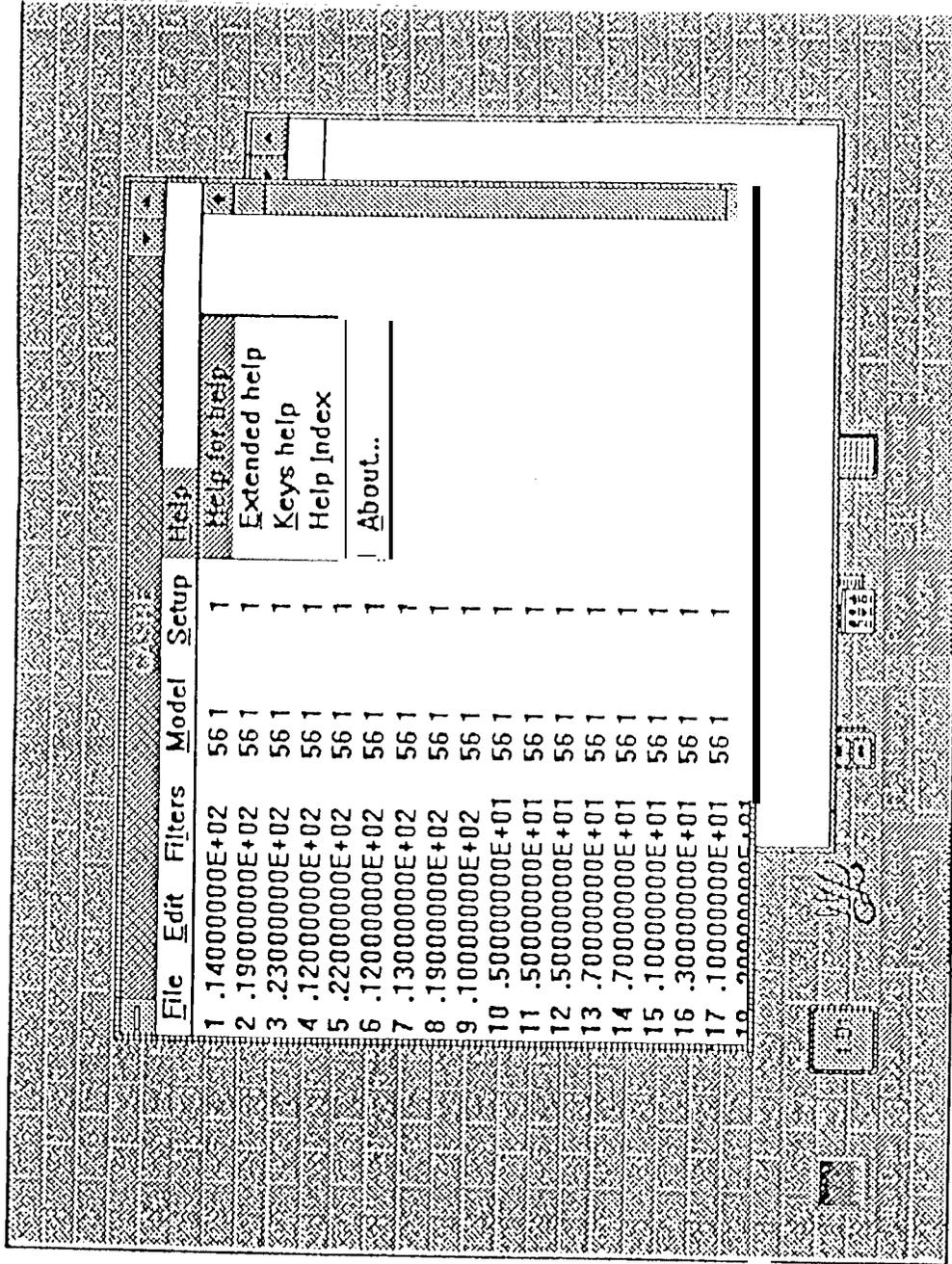
# Adding and Removing External Applications



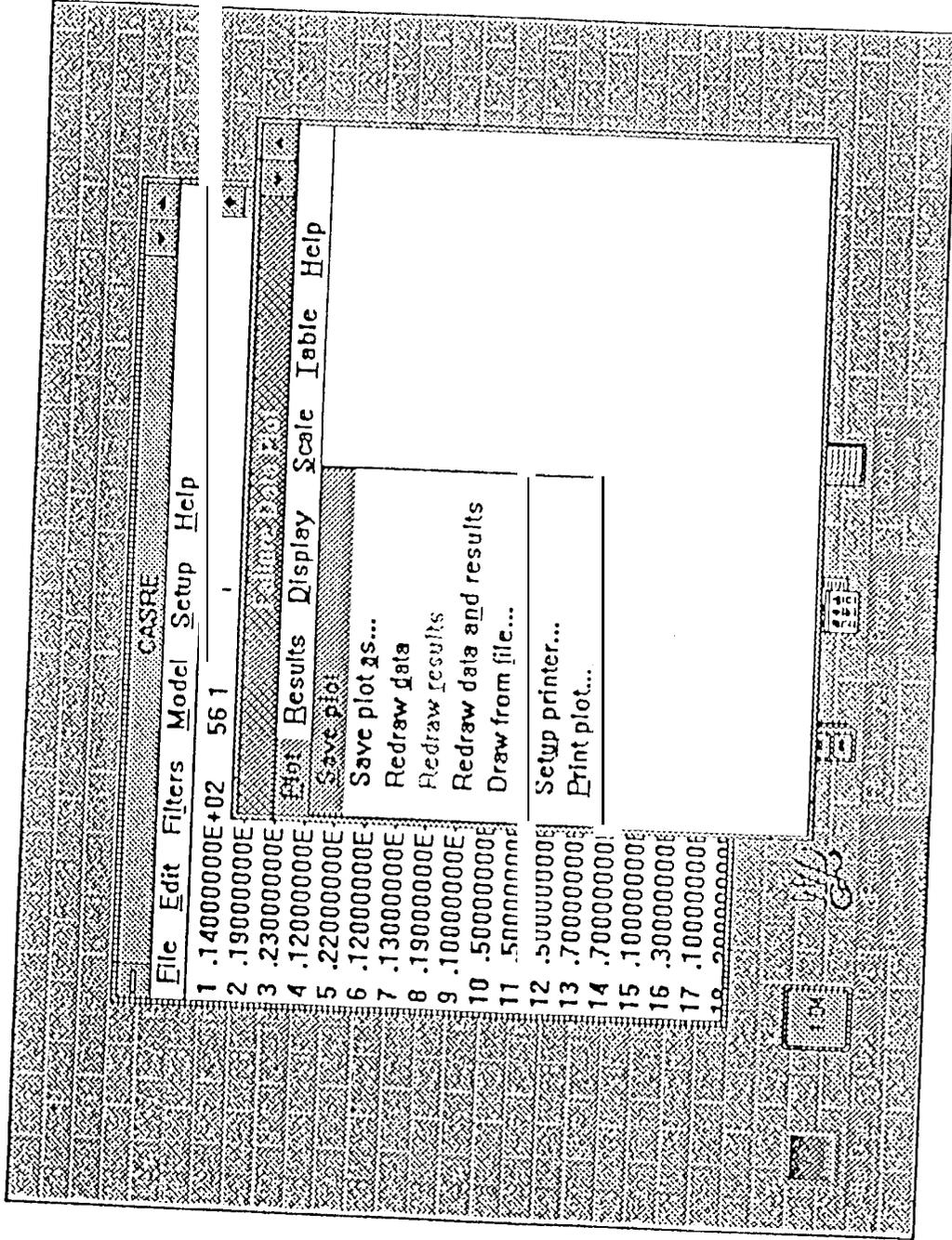
# Adding and Removing External Applications (cont'd)



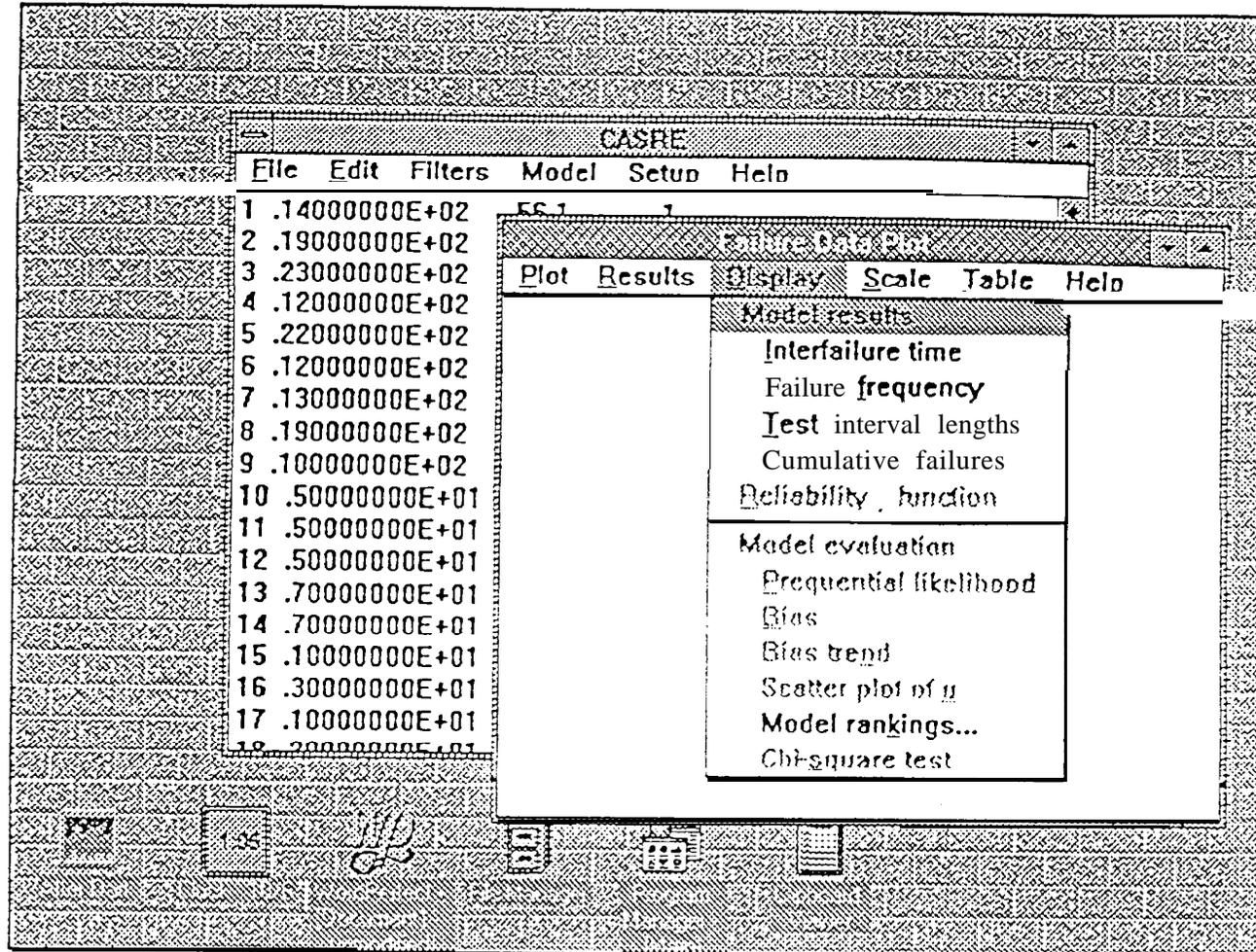
# xELP for Man Windo



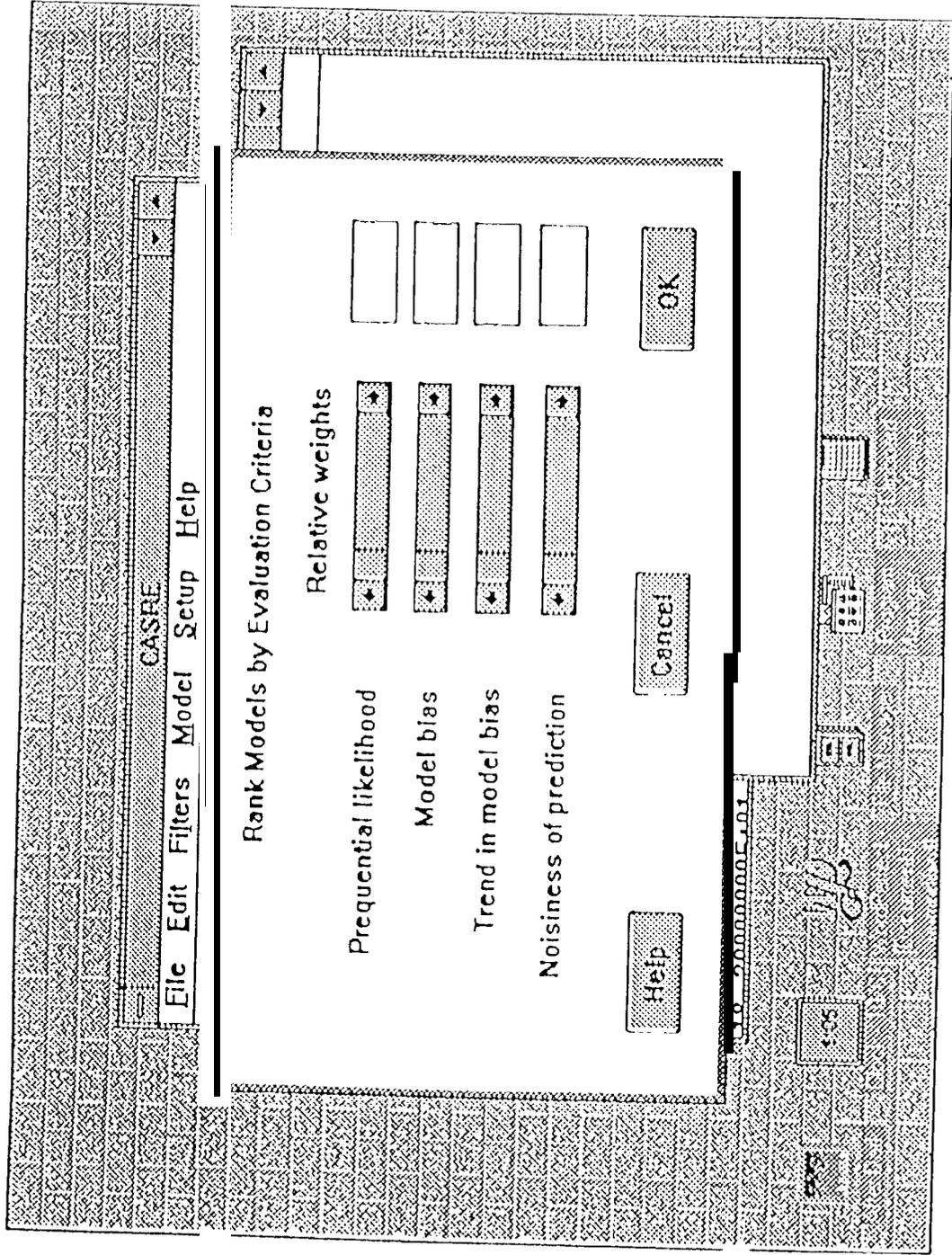
# Graphics Display Window "Plot" Menu



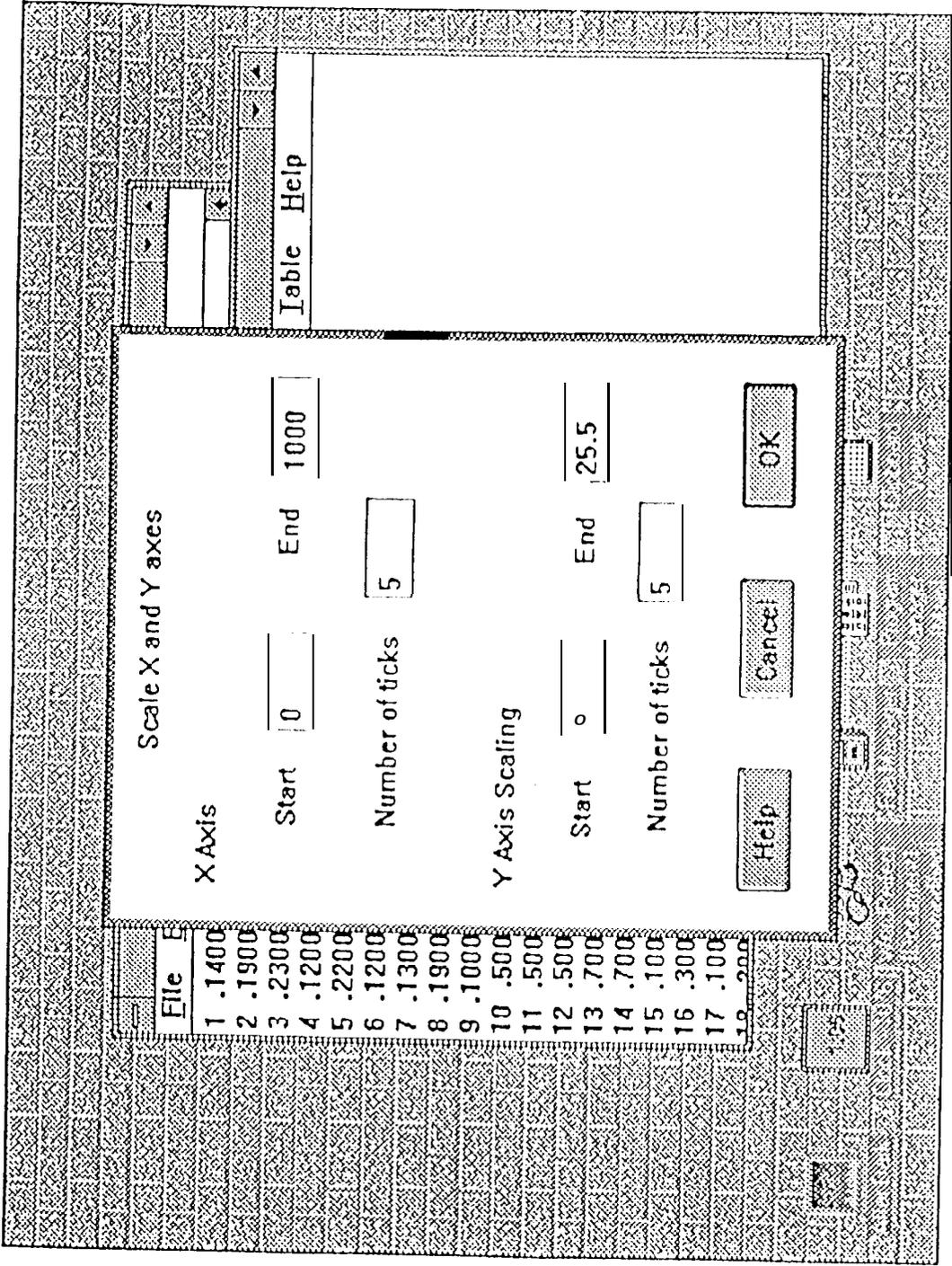
# Graphics Display Window "Display" Menu



# Determination of Mode Applicability



# Ax's Scaling



Acknowledgements:

The research presented in this tutorial was done at the University of Iowa under a faculty starting fund, at Bellcore, and at the Jet Propulsion Laboratory, California Institute of Technology, under a NASA contract through the Director's Discretionary Fund. CASRE's implementation is being supported by the Air Force Operational Test and Evaluation Center under Task Order RE-182, Amendment 655, Proposal 80-3417. We wish to thank Dr. William Farr of the Naval Surface Warfare Center for allowing SMERFS to be included in this tutorial. We also wish to thank Dr. Bev Littlewood of the City University of London for allowing SRMP to be included in this tutorial. Finally, we express our appreciation to AT&T for allowing the AT&T Toolkit to be included in this tutorial.