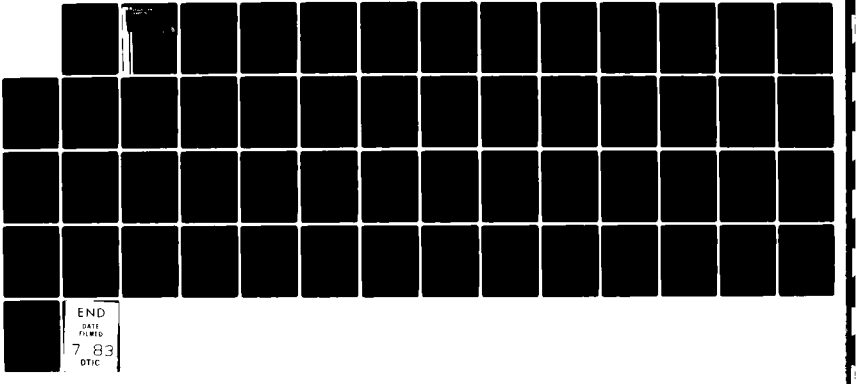
AD-A129 556 FITTING AN EXPONENTIAL SOFTWARE RELIABILITY MODEL TO FIELD FAILURE DATA..(U) POLYTECHNIC INST OF NEW YORK BROOKLYN DEPT OF ELECTRICAL ENGI.. R W SCHMIDT MAY 82 1/1

UNCLASSIFIED POLY-EE/CC-82-002 N00014-75-C-0858 F/G 12/1 NL

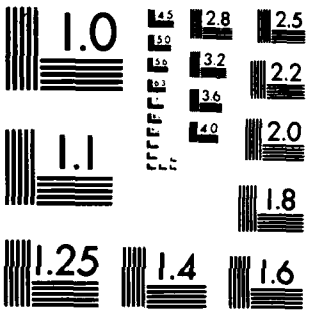


END

DATE FILMED

7-83

DTIC



Polytechnic Institute of New York

(12)

FITTING AN EXPONENTIAL SOFTWARE RELIABILITY MODEL TO FIELD FAILURE DATA

BY

R. W. SCHMIDT

Prepared for
Office of Naval Research
Contract N00014-75-C-0858

Report No. POLY EE/CS 82-002

Polytechnic Institute of New York
Department of Electrical Engineering
and Computer Science
333 Jay Street
Brooklyn, New York 11201

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A129556 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Fitting An Exponential Software Reliability Model to Field Failure Data	5. TYPE OF REPORT & PERIOD COVERED Technical
	6. PERFORMING ORG. REPORT NUMBER POLY EE-82-002
7. AUTHOR(s) R. W. Schmidt	8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0858
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Elec. Engineering & Computer Science Polytechnic Institute of New York 333 Jay Street, Brooklyn, New York 11201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, Virginia 22217	12. REPORT DATE May, 1982
	13. NUMBER OF PAGES 51
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for any purpose of the United States Government.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

- - - -

18. SUPPLEMENTARY NOTES

- - - -

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

software reliability, model, model construction, field failure data

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The quantitative prediction and measurement of software reliability is of vital importance in the development of high quality cost effective software. Many software reliability models have been postulated in the literature, however few have been applied to field data. A model based upon the assumption that the failure rate of the software is proportional to the number of residual software errors leads to a constant failure rate and an exponential reliability function. The model contains two constants: the proportionality constant K and the initial (total) number of errors E_T. $E_{sub\ T}$.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The constants K and E_T can be estimated during early design by comparison of the present project with historical data. During the intergration test phase, a more accurate determination of the model parameters can be obtained by using simulator test data as if it were operational failure data. The simulator data is collected at two different points in the integration test phase and the two parameters can be determined from moment estimator formulas. The more powerful maximum likelihood method can also be employed to obtain point and interval estimates. It is also possible to use least squares methods to obtain parameter estimates which is the simplest method and provides insight into the analysis of the data.

This report utilizes a set of software development and field data taken by John D. Musa as a vehicle to study the ease of calculation and the correspondence of the three methods of parameter estimation. The sensitivity of the reliability predictions to parameter changes are studied and compared with field results.

The results show if data is carefully collected, software reliability models are practical and yield useful results. These can serve as one measure to help in choosing among competitive designs and as a guage of when to terminate the integration test phase.

FITTING AN EXPONENTIAL SOFTWARE RELIABILITY MODEL TO FIELD FAILURE DATA

BY

R. W. SCHMIDT

Prepared for
Office of Naval Research
Contract N00014-75-C-0858

Report No. POLY EE/CS 82-002

Polytechnic Institute of New York
Department of Electrical Engineering
and Computer Science
333 Jay Street
Brooklyn, New York 11201

ABSTRACT

The quantitative prediction and measurement of software reliability is of vital importance in the development of high quality cost effective software. Many software reliability models have been postulated in the literature (Ref. 11), however few have been applied to field data. A model based upon the assumption that the failure rate of the software is proportional to the number of residual software errors leads to a constant failure rate and an exponential reliability function, (Ref. 1). The model contains two constants: the proportionality constant K and the initial (total) number of errors E_T.

The constants K and E_T can be estimated during early design by comparison of the present project with historical data. During the integration test phase, a more accurate determination of the model parameters can be obtained by using simulator test data as if it were operational failure data. The simulator data is collected at two different points in the integration test phase and the two parameters can be determined from moment estimator formulas (Ref. 9). The more powerful maximum likelihood method can also be employed to obtain point and interval estimates (Ref. 3). It is also possible to use least squares methods to obtain parameter estimates which is the simplest method and provides insight into the analysis of the data (Ref. 12).

This thesis utilizes a set of software development and field data taken by John D. Musa (Ref. 10) as a vehicle to study the ease of calculation and the correspondence of the three methods of parameter estimation. The sensitivity of the reliability predictions to parameter changes are studied and compared with field results.

This thesis is based in part on a joint paper written by the author and Professor Martin L. Shooman, presented at the ORSA-TIMS Conference in Florida, January 1981. (14)

The results show that if data is carefully collected, software reliability models are practical and yield useful results. These can serve as one measure to help in choosing among competitive designs and as a guage of when to terminate the integration test phase.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1.0 Introduction

Software presently represents the highest cost item in the development of computer systems. There is a paucity of quantitative measures to judge the quality of the final software and use as a measure of progress during the test and debugging phase. The reliability and meantime to failure (MTTF) of the software is a most useful metric for both the above purposes.

An important class of software reliability models (see Refs. 1, 8) make the assumption that the operational software failure rate is proportional to the remaining number of errors. Thus the failure rate is dependent on development time τ, but not on operating time t. This leads to a constant hazard and exponential reliability model, with two unknown parameters K and E_T.

A major focus of this thesis is to investigate and provide insight into a number of issues related to the estimation of these two parameters:

1. The accuracy one can obtain by using historical data to determine K and E_T. (Musa's);
2. A comparison of the accuracy obtained using three different methods of parameter estimation: the maximum likelihood, moments, least squares;
3. A comparison of predicted values of MTTF and observed MTTF values from field failure data;
4. Model parameter sensitivity; and,
5. Model accuracy/parameters related to the practical application by the software manager.

The conclusions reached in this study clearly indicate that parameter estimation during system development is a highly practical tool which can be used to successfully predict software MTTF and the 'debugging' time required to achieve that goal.

2.0 Development of Error and Reliability Models

2.1 An Error Removal Model

The reliability model used in this paper has been described in detail in a number of references (1, 2, 3). In brief, the model assumes that the program enters the integration test phase with E_T total errors remaining in the software. As integration testing proceeds, all detected errors are promptly corrected, and at any point in the development cycle (after τ months of development time)*, a total of $E_c(\tau)$ errors have been corrected, and the remaining number of errors, E_r is

$$E_r(\tau) = E_T - E_c(\tau) \tag{1}$$

In a more advanced model (4) it is assumed in addition that new errors are generated during development. One can often normalize the above equation through division by the number of object code instructions I_T to yield

$$\frac{E_r(\tau)}{I_T} = \frac{E_T}{I_T} - \frac{E_c(\tau)}{I_T} \tag{2a}$$

$$\epsilon_r(\tau) = \frac{E_T}{I_T} - \epsilon_c(\tau) \tag{2b}$$

Where $\epsilon_r = \frac{E_r}{I_T}$ and $\epsilon_c = \frac{E_c}{I_T}$

Basically the error model used in this paper assumes that the total number of errors in the program is fixed and that if we record the

---

* In some cases the actual number of test hours is estimated and is used as the development time variable rather than the cruder calendar days.

cumulative number of errors corrected during debugging, then the difference represents the remaining errors. We can define error removal rates as:

$$\frac{d\epsilon_r \tau}{d\tau} = r_r(\tau) \tag{3}$$

which can be normalized to yield

$$\frac{d\epsilon_r(\tau)}{d\tau} = \rho_r(\tau) \tag{4}$$

where

$\rho_r(\tau)$ = errors removed/total number of instructions/test hours (5)

$\epsilon_r(\tau) = \int_0^\tau \rho(x)dx$ = cumulative errors/total number of instructions. (6)

In Reference 5 error data are reported for seven large supervisory programs and applications programs. In Fig. 1 the normalized error rate $\rho(\tau)$ calculated from this data is plotted as a function of τ, the number of months of debugging after release for three of the seven systems. Although several curve shapes might be fitted to this data, one characteristic is common for all curves. The normalized error rate decreases over the entire curve or at least over the latter two-thirds or half of the curve; whereas initial behavior of $\rho(\tau)$ differs from example to example. A curve of the cumulative error data for the supervisory system A of Fig. 1 is shown in Fig. 2. Similar curves of $\epsilon(\tau)$ drawn for the other examples of Fig. 1 all build up initially with a constant or increasing slope and then exhibit a decreasing curvature appearing to become asymptotic. The smoothing nature of integration makes all the $\epsilon(\tau)$ curves look more alike than the $\rho(\tau)$ curves do. (Figs. 2A & B). Both are needed for a detailed study.

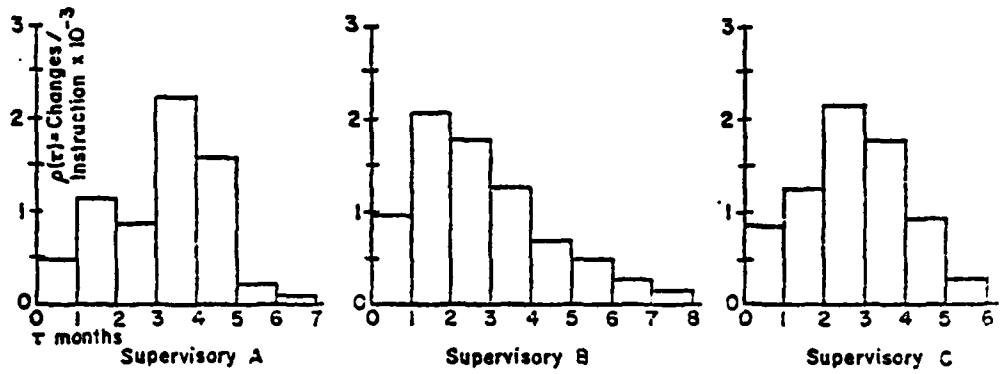


FIGURE 1. NORMALIZED ERROR RATE vs DEBUGGING TIME FOR THREE SUPERVISORY PROGRAMS

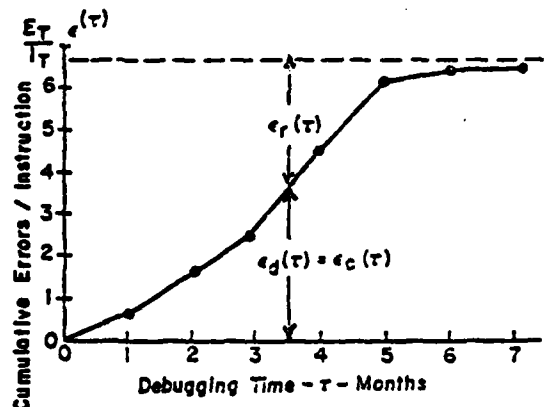


FIGURE 2. Cumulative Error Curve for Supervisory System A Given in Figure 1.

FIGURE 2A

CUMULATIVE ERROR CURVE FOR SUPERVISORY SYSTEM B

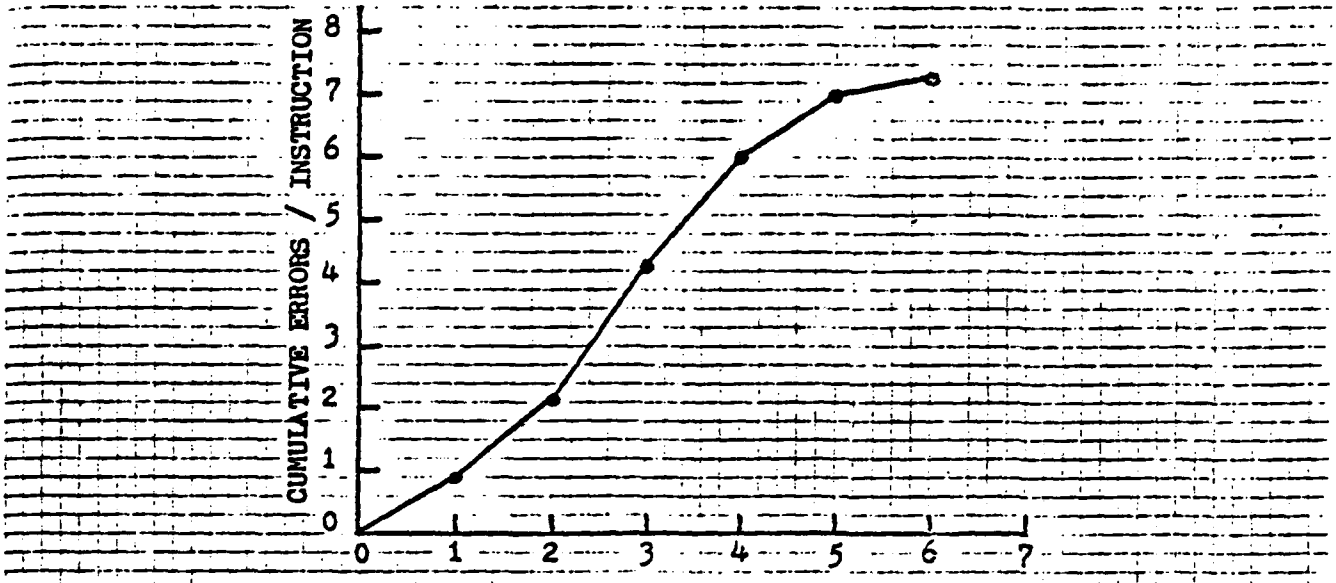
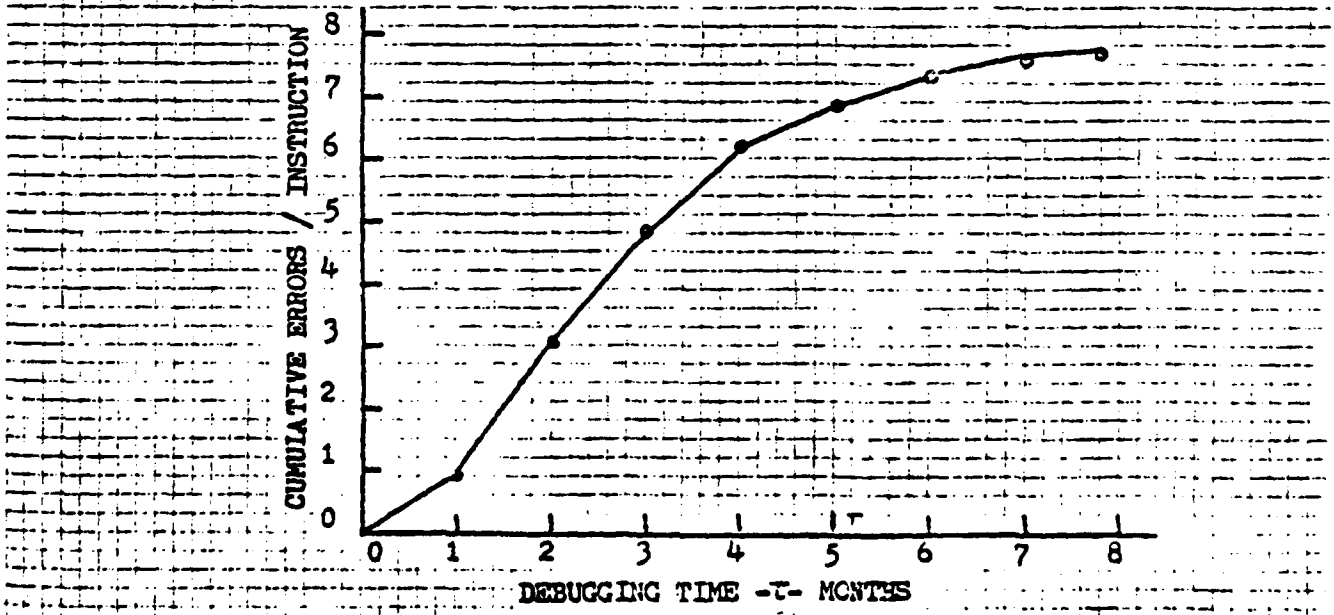


FIGURE 2B

CUMULATIVE ERROR CURVE FOR SUPERVISORY SYSTEM C



If we assume that the total number of errors in the program (E_T) is constant and that the program contains L_T instructions, then the asymptote which the $\epsilon(\tau)$ curves approach is E_T/L_T. (Figs. 2, 2A, 2B).

The error model developed above will be used in the following section to formulate a reliability model.

2.2 Development of the Exponential Reliability Model

We assume that all operational software errors occur due to the occasional traversing of a portion of the program in which a hidden software error is lurking. We begin by writing an expression for the probability that an error is encountered in the time interval Δt after t successful hours of operation. We make the assumption that this probability is proportional to the number of errors remaining in the program. (See Ref. 6 for data substantiating this assumption).

From a study of basic probability and reliability theory we know that the probability of failure in time interval t to $t + \Delta t$, given that no failures have occurred up till time Δt, is proportional to the failure rate (hazard function) z(t). Thus, we obtain

$$P(t<t_f<t+\Delta t \mid t_f>t) = z(t)\Delta t = K\epsilon_r(\tau)\Delta t \qquad (7)$$

where t_f = time to failure, (occurrence of a software error)

$P(t<t_f<t+\Delta t \mid t_f>t)$ = probability of failure in interval Δt, given no previous failure.

K = an arbitrary constant

From reliability theory (7) we can show that the probability of no system failures in the interval 0 to t is the reliability function which is related to the hazard, z(t), by:

$$R(t) = e^{-\int_0^t z(x)\,dx} \tag{8}$$

If we substitute our expression for z(t) from Eq. 7 into Eq. 8 and assume K, and $\epsilon_r(\tau)$, are independent of operating time, we obtain

$$R(t) = e^{-K\epsilon_r(\tau)t} = e^{-\gamma t} \tag{9}$$

Basically the above equation states that the probability of successful operation without software errors is an exponential function of operating time. When the system is first turned on, t = 0 and R(0) = 1. As operating time increases the reliability monotonically decreases as shown in Fig. 3. We depict the reliability function for three values of debugging time, $\tau_0 < \tau_1 < \tau_2$. From this curve we may make various predictions about the system reliability. For example, looking along the vertical line $t = 1/\gamma$ we may state:

1. If we spend τ_0 hours of debugging then $R(1/\gamma) = 0.35$
2. If we spend τ_1 hours of debugging then $R(1/\gamma) = 0.50$
3. If we spend τ_2 hours of debugging then $R(1/\gamma) = 0.75$

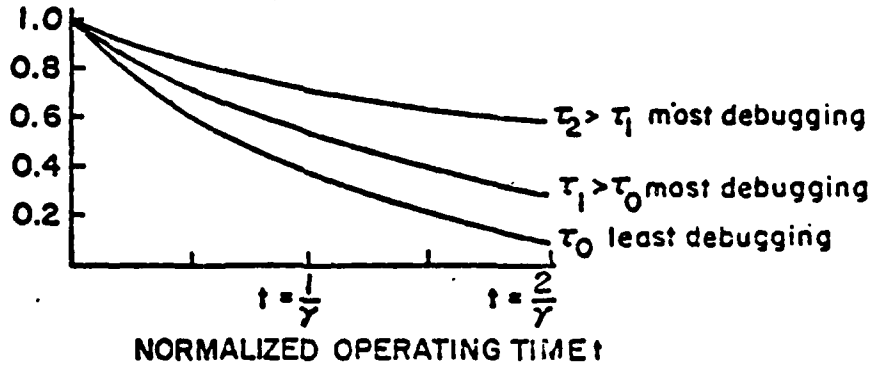
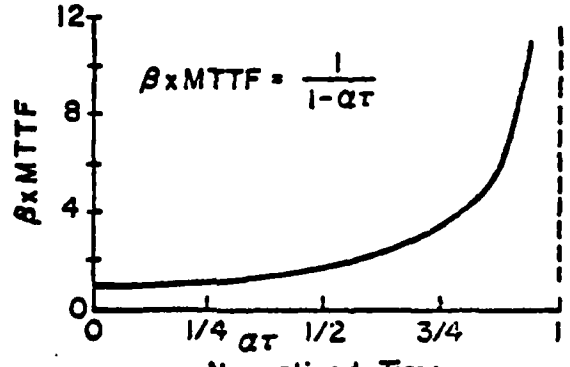


FIG.3 Variation of Reliability Function R(t) with Debugging Time τ.



a. Constant Error Debug Rate

FIGURE 4. Comparison of MTTF vs Debug Time for The Error Model.

2.3 Meantime to Software Failure

A simpler way to summarize the results of the reliability model is to compute the mean time to (software) failure, MTTF, for the system.

$$MTTF = \int_0^\infty R(t)\, dt \tag{10}$$

For purposes of illustration we let $\rho(\tau)$ be modeled by a constant rate of error correction ρ_o. Solution of Eqs. 2 and 10 then yields

$$MTTF = \frac{1}{K\left[\frac{E_T}{I_T} - \rho_o \tau\right]} = \frac{1}{\beta(1 - \alpha\tau)} \tag{11}$$

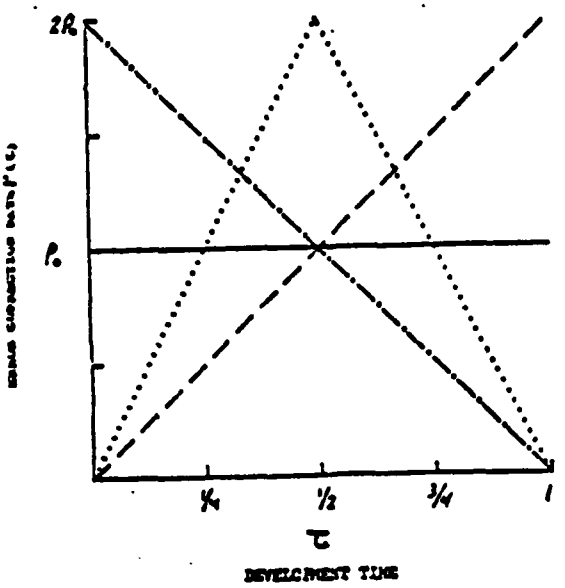
where $\beta = K E_T/I_T$ and $\alpha = \rho_o I_T/E_T$

This is depicted in Fig. 4 where β x MTTF is plotted vs $\alpha\tau$, and we note that the greatest improvement in MTTF occurs during the last $\frac{1}{4}$ of the debugging stage.

Reference 5 describes other error correction rate models, as illustrated in Fig. 4A. In order to compare the relative effects on system MTTF by assuming a varying $\rho(\tau)$, it is essential that we integrate the latter over the same range. This is readily verified by noting that the total area under each curve is identical. Integration results are plotted in Fig. 4B for each model to yield the cumulative number of expected errors. The effect on system MTTF is depicted in Fig. 4C.

FIGURE 4A

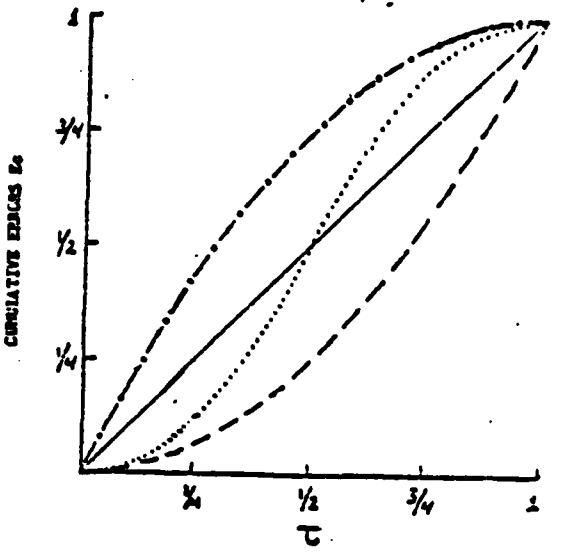
ERROR CORRECTION PATH MODELS



DEVELOPMENT TIME

FIGURE 4B

GROWTH OF $E_c(\tau)$ ASSOCIATED WITH $\rho(\tau)$ OF FIG. 4A

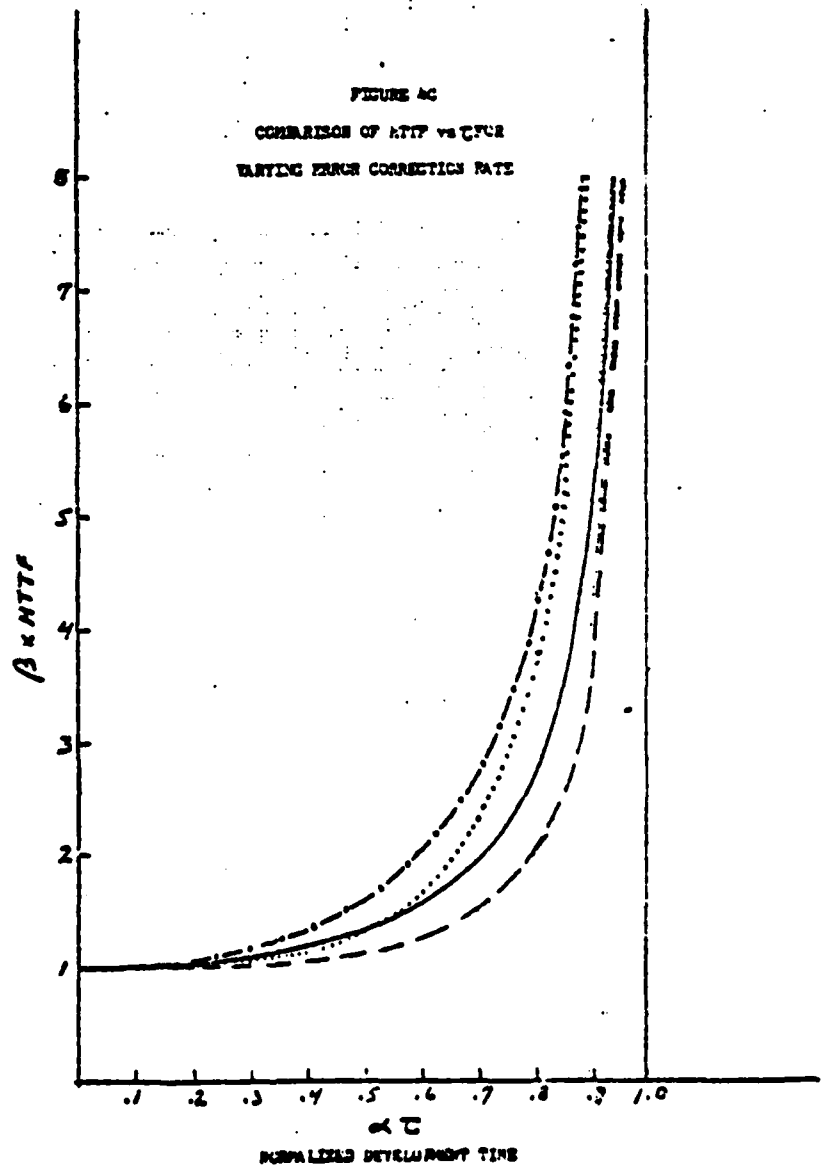


DEVELOPMENT TIME

FIGURE 4C

COMPARISON OF NTTF vs τ FOR

VARYING ERROR CORRECTION RATE



NORMALIZED DEVELOPMENT TIME

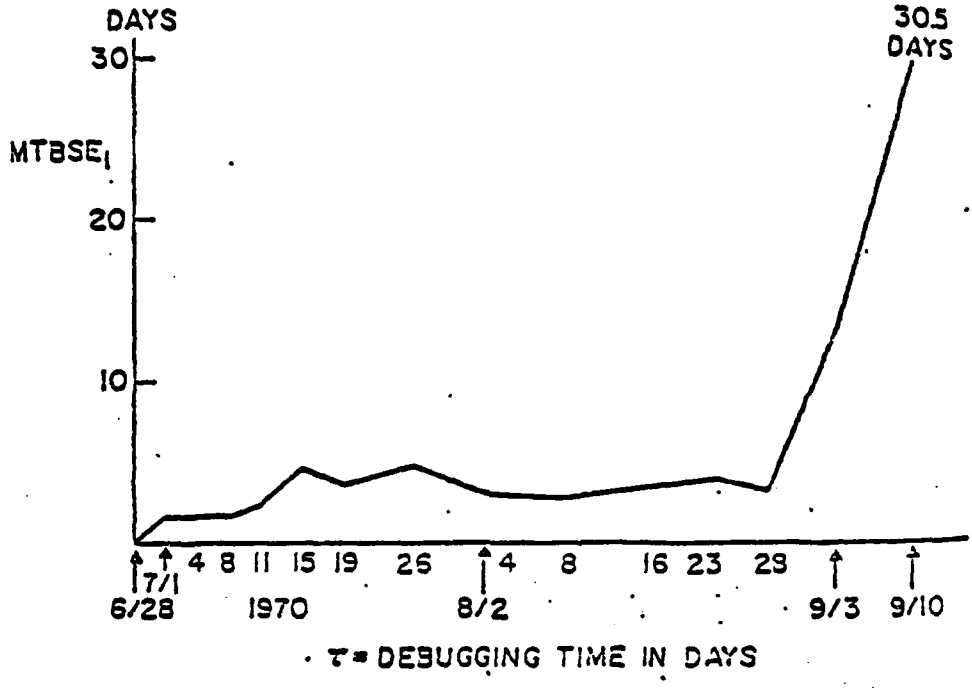


Figure 5. Growth Curve of Software Reliability Mean Time Between Software Errors. (From L. Miyamoto, Ref. 13).

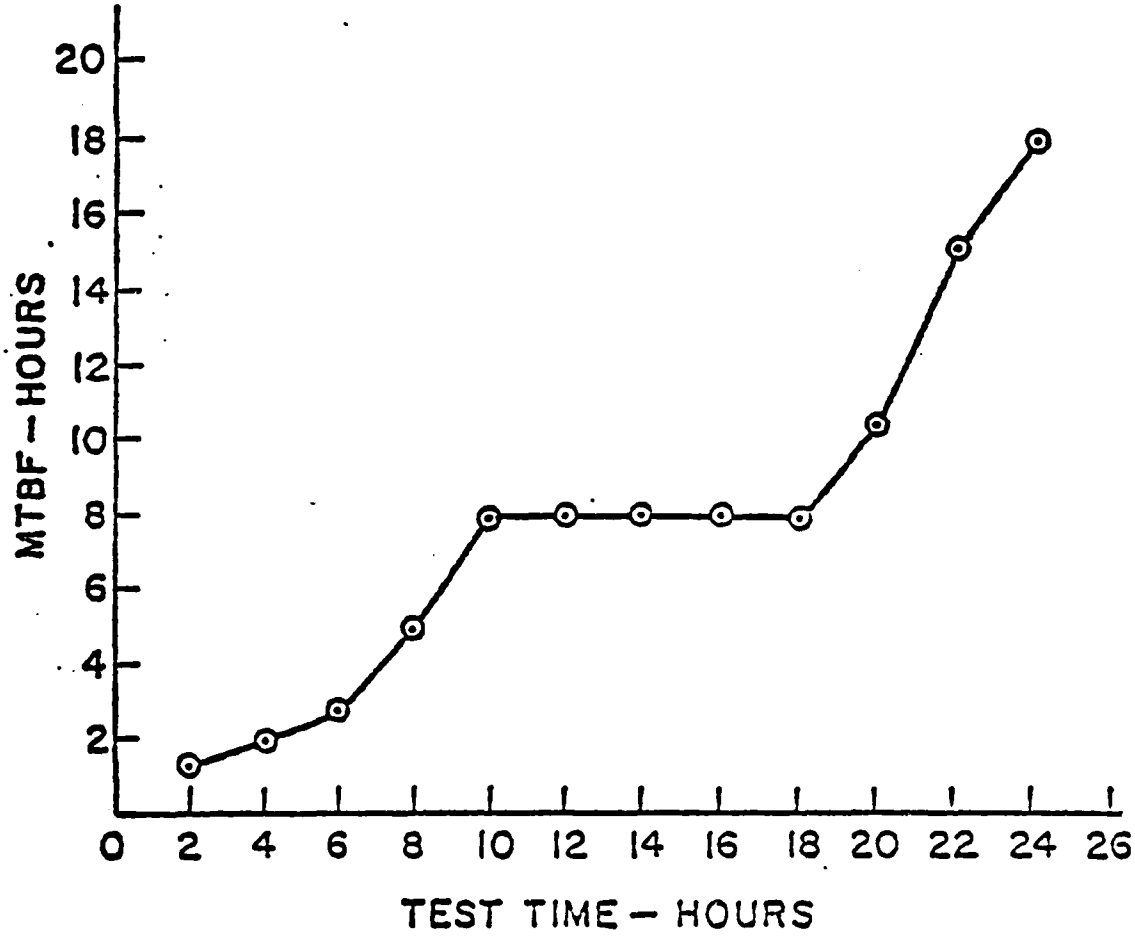


Figure 6 MTBF vs. Test Time for Project 1. (Replotted from the data of Fig. 3, J. Musa, Ref. 8)

The rapid rise in MTTF toward the end of the integration phase is of considerable importance in software project management. Field data by others (see Figs. 5 & 6) confirms this basic shape. If a manager is pressed to release a system to the field at an early date, he may accept the current reliability and deliver the software. If we believe Figs. 4 thru 6, however, then a few more weeks could yield a big improvement in MTTF. The model predicts such behaviour, but if one only had test data for $\alpha\tau/\dot{\tau}$, then it would be difficult to predict the sharp rise near $\alpha\tau=1$. The model is thus of great use in managing a project and setting its release date.

2.4 Musa's Model

Musa has developed a model similar to that given in Sec. 2.2. However, instead of basing his model on development time as the resource measure during integration, he utilized actual CPU time. Musa also used regular test data rather than simulator test data. His model is given by (8):

$$R(\tau') = e^{-\tau'/T} \tag{12}$$

$$T = T_o e^{-(C\tau/M_o T_o)} \tag{13}$$

where

τ' = hours of program operation = t

T = mean time to failure in operating hours = MTTF

T_o = MTTF at the start of test (t = 0)

C = ratio of equivalent operating time/test time

M_o = number of failures which must occur to uncover all errors = E_T

τ = the CPU time in hours during testing

Since we will be using Musa's data and some of his results in Sec. 5 of this thesis, we must carefully account for the different definitions of time between Musa's model and the exponential model during the analysis of his data.

3.0 Estimation of Model Parameters

3.1 Introduction

The exponential reliability model given in Eq. 9 contains the parameters K and $\epsilon_r(t)$ which must be estimated. In many cases one wishes to use such a model to roughly predict MTTF during a proposal phase or early design of the project. In such a case the only available technique for determining values of the parameters is to use historical data. Presently Rome Air Development Center is developing a handbook and database on Software Reliability for just such a purpose.

3.2 Moment Estimates

In Ref. 9, a method is discussed for measurement of the two needed parameters based on simulation testing of the software. A program simulating the field environment is generally available for all real live computer programs. It is necessary that this program be run for a total of H_1 hours following τ_1 months of development. It is assumed also that the testing will produce r_1 failures. Similarly after τ_2 months of testing, the simulator is run and H_2 hours and r_2 failures are obtained. The MTTF for the data is given by the ratio H_2/r_2 and is equated to the MTTF expression obtained by substituting Eq. 9 into Eq. 10. The two equations (for τ_1 and τ_2) allow us to solve for the constants K and E_T:

$$\frac{H_1}{r_1} = \frac{1}{K\left[E_T - E_c(\tau_1)\right]} \tag{14}$$

$$\frac{H_2}{r_2} = \frac{1}{K\left[E_T - E_c(\tau_2)\right]} \tag{15}$$

Simultaneous solution of Eqs. 14 and 15 yields the desired values of $\hat{E}_T$ and $\hat{K}$:

$$\hat{E}_T = \frac{\left[\lambda_2/\lambda_1 \times E_c(\tau_1)\right] - E_c(\tau_2)}{\left(\lambda_2/\lambda_1 - 1\right)} \tag{16}$$

$$\hat{K} = \frac{\lambda_1}{\hat{E}_T - E_c(\tau_1)} \tag{17}$$

Note that in deriving the above results we have assumed that the failure rate is constant and have used the common notation and well known results for constant failure rates:

$$z(t) = \lambda = \frac{r}{H} = \frac{1}{MTTF}$$

3.3 Least Square Estimates

Another method of estimating model parameters is to rewrite equation 11 in the form

$$\lambda_i = \hat{K}\left[\hat{E}_T - E_c(\tau_i)\right] \tag{18}$$

for λ = failure rate

to yield $E_c(\tau_i) = E_T - \frac{1}{\hat{K}}\lambda_i$ (19)

This equation represents a straight line whose parameters can be determined from the slope and intercept of a least squares fit of the data, where

$$\hat{K} = \frac{-1}{\text{slope}} \tag{20}$$

$$\hat{E}_T = \text{intercept} \tag{21}$$

Naturally, the largerand the more accurate the data set, the more precisely can the model parameters be predicted.

3.4 Maximum Likelihood Estimates

Another method which can also be used to estimate the values of $\hat{K}$ and $\hat{E}_t$ is known as the Maximum Likelihood Estimation technique (MLE). The likelihood function, L is the joint probability of occurrence for the observed set of test values. If during simulation testing we observe r_1 failure times ($t_1, t_2, \ldots t_{r_1}$) and n_1-r_1 successful runs testing ($T_1, T_2, \ldots, T_{n_1-r_1}$) then the likelihood function for a single test after $E_c(\tau_1)$ errors have been removed is given by

$$L(K,E_T) = f(t_1)f(t_2)\ldots f(t_{r_1})R(t_1)R(T_2)\ldots R(T_{n_1-r_1})$$

where

$f(t_i)$ = the density function $KE_r(\tau_1)e^{-K\epsilon_r(\tau_1)t_i}$

$R(T_i)$ = the reliability function $e^{-K\epsilon_r(\tau_1)T_i}$

To maximize the likelihood function we take partial derivitives with respect to K and E_T and set them equal to zero. To solve for the two operations we need a second equation obtained from another likelihood equation based on a second set of test data at time τ_2. Applying MLE

theory to two tests with r_1 and r_2 failures over H_1 and H_2 total hours, we obtain (Appendix A, Ref. 3)

$$\hat{K} = \frac{r_1 + r_2}{H_1\left[\hat{E}_T - E_c(\tau_1)\right] + H_2\left[\hat{E}_T - E_c(\tau_2)\right]} \tag{22}$$

$$\hat{K} = \frac{1}{H_1 + H_2}\left[\frac{r_1}{\hat{E}_T - E_c(\tau_1)} + \frac{r_2}{\hat{E}_T - E_c(\tau_2)}\right] \tag{23}$$

As is often the case with MLE, the above equations require numerical solution; however, most statisticians believe them to yield superior results to moment estimates. An iterative computer solution of Eqs. 22 and 23 is easily implemented, yet a graphical solution using a simple calculator suffices in most cases. The first step is to obtain starting values for $\hat{E}_T$ and $\hat{K}$ using some other method, such as Method of Moments. Values of $\hat{E}_T$ above and below the starting value are substituted into Eqs. 22, 23 and the curves of K vs E_T plotted on the same axes. Their intersection determines the value of $\hat{E}_T$ and $\hat{K}$.

4.0 Musa's Data

4.1 Introduction

The software reliability data used in this paper was compiled by John D. Musa (10) over a period of time on a variety of large software systems, ranging from tens to hundreds of thousands of object code instructions. His objective was "to present in detail a substantial body of data that has been gathered in the application of the execution time theory of software reliability"; the end product is ideally suited for the purpose of this paper, as it presents a wealth of precise software failure data obtained under carefully controlled circumstances.

4.2 Description of Raw Data

Software failure interval data was presented in the following format:

Failure number Failure interval Day of failure

Failure interval was measured in seconds, and represents either running clock time, (operating time on the computer), or, in one case, actual CPU time. 'Day of Failure' is the working day counted from the start of project on which the failure occurred. (See Table 1).

TABLE 1

FAILURE INTERVALS - SYSTEM 3 TEST PHASE

Failure Number	Failure Interval	Day of Failure	Failure Number	Failure Interval	Day of Failure
1	115	1	21	15	26
2	0	1	22	390	26
3	83	3	23	1863	27
4	178	3	24	1337	30
5	194	3	25	4508	36
6	136	3	26	834	38
7	1077	3	27	3400	40
8	15	3	28	6	40
9	15	3	29	4561	42
10	92	3	30	3186	44
11	50	3	31	10571	47
12	71	3	32	563	47
13	606	6	33	2770	47
14	1189	8	34	652	48
15	40	8	35	5593	50
16	788	18	36	11696	54
17	222	18	37	6724	54
18	72	18	38	2546	55
19	615	18	39	10175*	56
20	589	26			

* Note: If the last interval is followed by an asterisk, there was no failure at the end of the period and the time represents the interval between the last failure and the end of the period. (10).

For the purpose of this paper, failures occurring on the same working day were summed together to form one statistical data point, and were tabulated under the following format: (See Table 2)

S	WD	E	Ec	z	T	Tc

where:

S = sequential serial number assigned to each statistical data point

WD = working day on which failure(s) occurred

E = number of errors occurring that working day

Ec = cumulative errors to date

T = total operating time (failure interval time) for that working day

Tc = cumulative failure interval time to date

z = failure rate (E/T) for that working day

Presentation of Musa's data in this format had a two-fold purpose:

a) reduce the sheer bulk of the raw data without affecting its statistical significance by grouping the occurrence of software failures by working day; and,

b) tabulate the data in a format more suitable for subsequent calculations.

4.3 System Characteristics

Systems 1-4 studied by Musa are real-time command and control software packages consisting of 21,700 to 33,500 object instructions and a failure sample size of 38 to 136.

System 5 is a realtime commercial application, consisting of 2,445,000 object instructions and a failure sample size of 831. Musa notes that for system 5, design changes involving 21% of the source code were introduced after approximately 30% of total testing time had elapsed.

TABLE 2

SYSTEM #3 FAILURE RATE DATA

S	WD	E	Ec	z	T	Tc
1	1	2	2	.01739	115	115
2	3	10	12	.00523	1911	2026
3	6	1	13	.00165	606	2632
4	8	2	15	.00163	1229	3861
5	18	4	19	.00236	1697	5558
6	26	3	22	.00302	994	6552
7	27	1	23	.00054	1863	8415
8	30	1	24	.00075	1337	9752
9	36	1	25	.00022	4508	14,260
10	38	1	26	.00119	834	15,094
11	40	2	28	.00059	3406	18,500
12	42	1	29	.00022	4561	23,061
13	44	1	30	.00031	3186	26,247
14	47	3	33	.00022	13,904	40,151
15	48	1	34	.00153	652	40,203
16	50	1	35	.00018	5593	46,396
17	54	2	37	.00011	18,420	64,816
18	55	1	38	.00039	2546	67,362

5.0 Estimation of Model Constants

In the following section, we will be estimating the model constants $\hat{K}$ and $\hat{E}_T$ of several system, using system #3 as a working example to demonstrate the calculations used for Least Squares, Method of Moments, and Maximum Likelihood Estimates. These constants will then be used to estimate system MTTF using equation 11.

As indicated in Sec. 4.2, Musa's raw data was condensed to form one statistical data point per working day during which one or more errors were uncovered.

In the following section, this condensed data was reduced even further to allow 2, 3, 4, 6, 8... points to represent an entire system as required. For example, if it were desired to represent system 3 (see Table 2) consisting of 18 entries, by two points (Fig. 7), the first nine entries were averaged statistically to yield the first point and the last 9 entries to yield the second point.

5.1 Method of Moments

The data reported by Musa for system #3 has been processed and grouped by working day as described in Sec. 4.2 (see Table 2). Using data points for the 9th and 18th group of failures, (Ec 9, Tc 9), (Ec 18, Tc 18), we obtain the average failure rate for the interval 0-9:

$$\lambda_1 = \frac{Ec9}{Tc9} = \frac{25 \text{ failures}}{14{,}260 \text{ sec.}} = 0.00175 \text{ failures/sec.} = 6.3 \text{ failures/hr.}$$

and similarly the average failure rate for the interval 10-18 is:

$$\lambda_2 = \frac{Ec18 - Ec9}{Tc18 - Tc9} = \frac{38 - 25}{67{,}362 - 14{,}260}$$

$$= 2.448 \times 10^{-4} \text{ failures/sec.} = 0.881 \text{ failures/hr.}$$

$$\lambda_2/\lambda_1 = 0.1396$$

$$\therefore \hat{E}_T = \frac{(\lambda_2/\lambda_1 \cdot Ec9) - Ec19}{(\lambda_2/\lambda_1 - 1)} = \frac{(0.1396 \cdot 25) - 38}{(0.1396 - 1)}$$

$$= 40.110$$

$$\text{and } K = \frac{\lambda_1}{\hat{E}_T - Ec9} = \frac{0.00175}{40.110 - 25} = 1.160 \times 10^{-4}$$

See Table 3 for a summary of all systems.

5.2 Least Squares Linear Regression

In this method, we plot the failure rate z vs cumulative errors Ec. Theoretically the best fit y-intercept yields $\hat{E}_T$ and the negative reciprocal of the slope equals $\hat{K}$.

The primary equations used in linear regression are:

$$\text{slope } m = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sum X^2 - (\sum X)^2/N} \tag{24}$$

$$\text{intercept } y' = \frac{\sum Y - m\sum X}{N} \tag{25}$$

for the straight line given by:

$$y = mx + y' \tag{26}$$

Applying the above to equation 19, and referring back to Table 2,

TABLE 3

METHOD OF MOMENTS ESTIMATION OF $\hat{E}_T$, $\hat{K}$

System				
1	#Pts	2	4	8
Ec = 136	$\hat{E}_T$	151.6	153.6	126.4
	$\hat{K}$ *	5.514	4.576	13.56
2	#Pts	2	4	8
Ec = 54	$\hat{E}_T$	61.6	52.6	38.7
	$\hat{K}$ *	3.909	5.688	7.731
3	#Pts	2	3	6
Ec = 38	$\hat{E}_T$	40.1	38.2	28.3
	$\hat{K}$ *	11.60	21.40	19.61
4	#Pts	2	3	6
Ec = 53	$\hat{E}_T$	54.9	54.9	22.2
	$\hat{K}$ *	19.88	13.40	26.20
6	#Pts	2	3	
Ec = 74	$\hat{E}_T$	111.7	117.7	
	$\hat{K}$ *	28.345	27.639	

*N.B. $\hat{K}$ values x 10^{-5}

$E_{c9} = 25$ $\qquad$ $E_{c18} = 38$

$T_{c9} = 14{,}260$ $\qquad$ $T_{c18} = 67{,}362$

We plot

$X_1 = \frac{25}{14{,}260} = 17.53 \times 10^{-4}$ $\qquad$ $Y_1 = E_{c9} = 25$

$X_2 = Z_2 = \frac{38-25}{67{,}362-14{,}260} = 2.45 \times 10^{-4}$ $\qquad$ $Y_2 = E_{c18} = 38$

to yield

$\hat{E}_T = 40.1$ $\qquad$ $\hat{K} = 1.1603 \times 10^{-4}$

The above calculations are depicted in Fig. 7 for system 3 and summarized for all systems in Table 4.

5.3 Maximum Likelihood Estimates (MLE)

The primary equations used (see Sec. 3.4) are:

$$\hat{K}_1 = \frac{r_1 + r_2}{H_1(E_T - E_{c1}) + H_2(E_T - E_{c2})} \tag{22}$$

$$\hat{K}_2 = \frac{1}{H_1 + H_2} \left[\frac{r_1}{E_T - E_{c1}} + \frac{r_2}{E_T - E_{c2}} \right] \tag{23}$$

FIGURE 7

CUMULATIVE ERRORS vs FAILURE RATE

SYSTEM 3

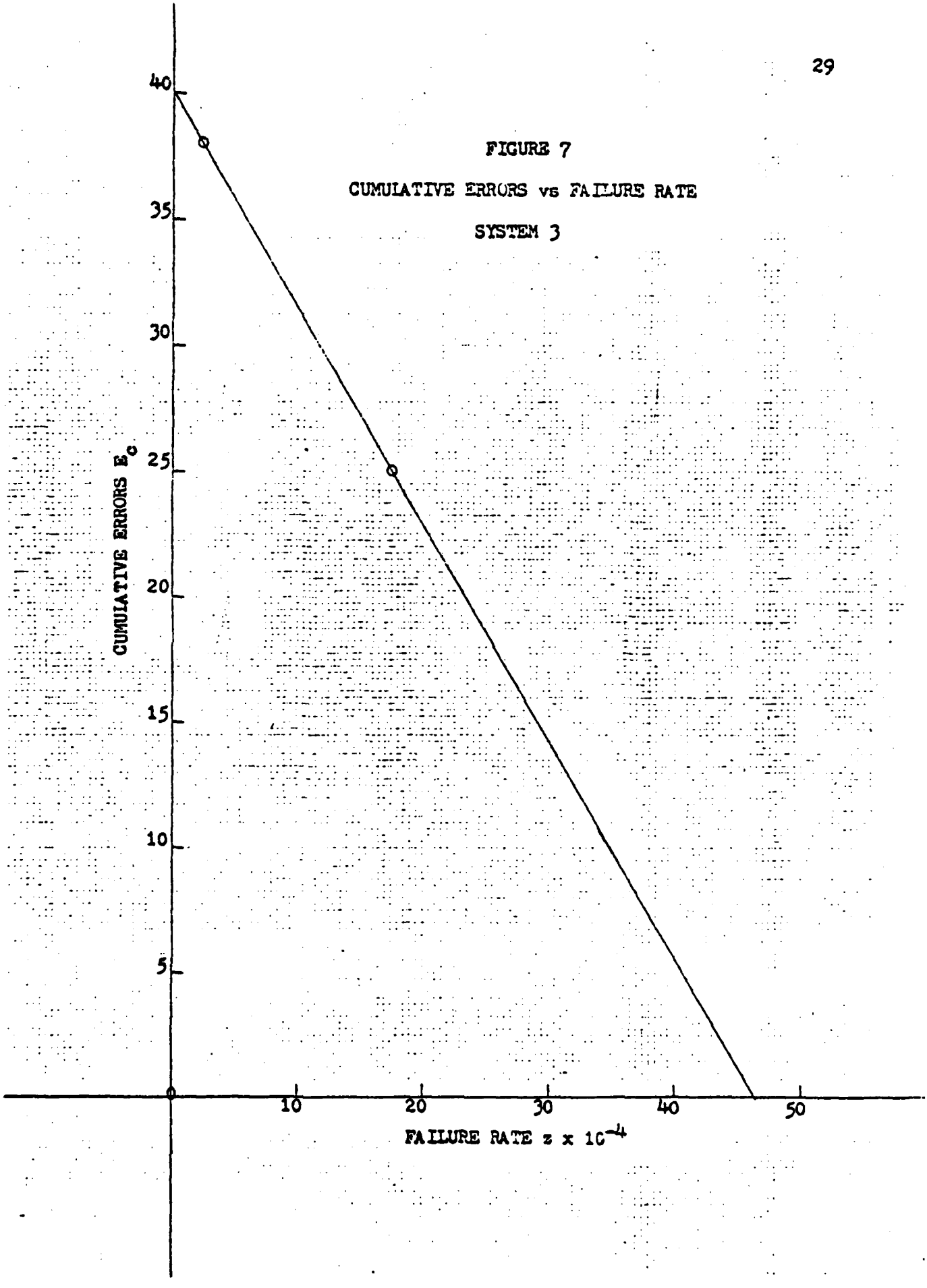


TABLE 4

LEASE SQUARES REGRESSION ESTIMATES OF $\hat{E}_T$, $\hat{K}$

System						
	#Pts	2	4	8	47*	
1	$\hat{E}_T$	151.6	140.1	103.3	74.6	
	$\hat{K}$	5.515	5.245	17.609	543.48	
	#Pts	2	4	8	12	24*
2	$\hat{E}_T$	61.6	51.7	45.5	44.9	42.9
	$\hat{K}$	3.909	6.253	8.333	9.395	10.279
	#Pts	2	3	6	9	18*
3	$\hat{E}_T$	40.1	34.9	32.8	30.6	28.4
	$\hat{K}$	11.603	25.490	23.928	27.942	56.593
	#Pts	2	4	10	19*	
4	$\hat{E}_T$	54.9	56.8	49.8	42.4	
	$\hat{K}$	19.885	14.029	20.400	40.601	
	#Pts	2	4	8	16	32
5	$\hat{E}_T$	2390	998.9	805.2	742.9	497.6
	$\hat{K}$	8.089	30.675	44.983	51.475	94.518

N.B. All $\hat{K}$ values x 10^{-5}

* Number of points used represents total system data without interval grouping.

A numerical solution of the above equations can be readily obtained, since we have two equations in two unknown, $\hat{K}$ and $\hat{E}_T$. A simpler approach assuming we only have a few estimates to make, is to effect a graphical solution:

Step 1: obtain a starting value for E_T by any method of your choice: Method of Moments, Least Squares Regression, Non-deterministic (ie... guess), ...;

Step 2: substitute a value E_T obtained in Step 1 into equations 22 and 23. Try other values of E_T above and below this value and repeat the calculations, plotting them on the same axis of $\hat{K}$ vs $\hat{E}_T$ for the two equations; and,

Step 3: the intersection of these two curves yields the required parameters.

The method is illustrated using system 3 as an example:

Step 1: $E_T = 40.1$ using the starting value obtained by the Method of Moments (MOM) Table 3.

Step 2: For $E_{T1} = 39$, $K_1 = 1.504 \times 10^{-4}$ (from 22)

$K_2 = 2.195 \times 10^{-4}$ (from 23)

For $E_{T2} = 41$, $K_1 = 8.355 \times 10^{-5}$ (from 22)

$K_2 = 7.008 \times 10^{-5}$ (from 23)

Step 3: The above results are plotted on Fig. 8 to yield

$\hat{E}_T = 40.1$

$\hat{K} = 11.6 \times 10^{-5}$

Once an approximate value is obtained from the curves' intersection, iterative methods can be used to obtain the required degree

of accuracy.

See Table 5 for a summary of the MLE estimates for all four systems.

FIGURE 8

ESTIMATED $\hat{K}$ vs TOTAL ERRORS $\hat{E}_T$

SYSTEM 3

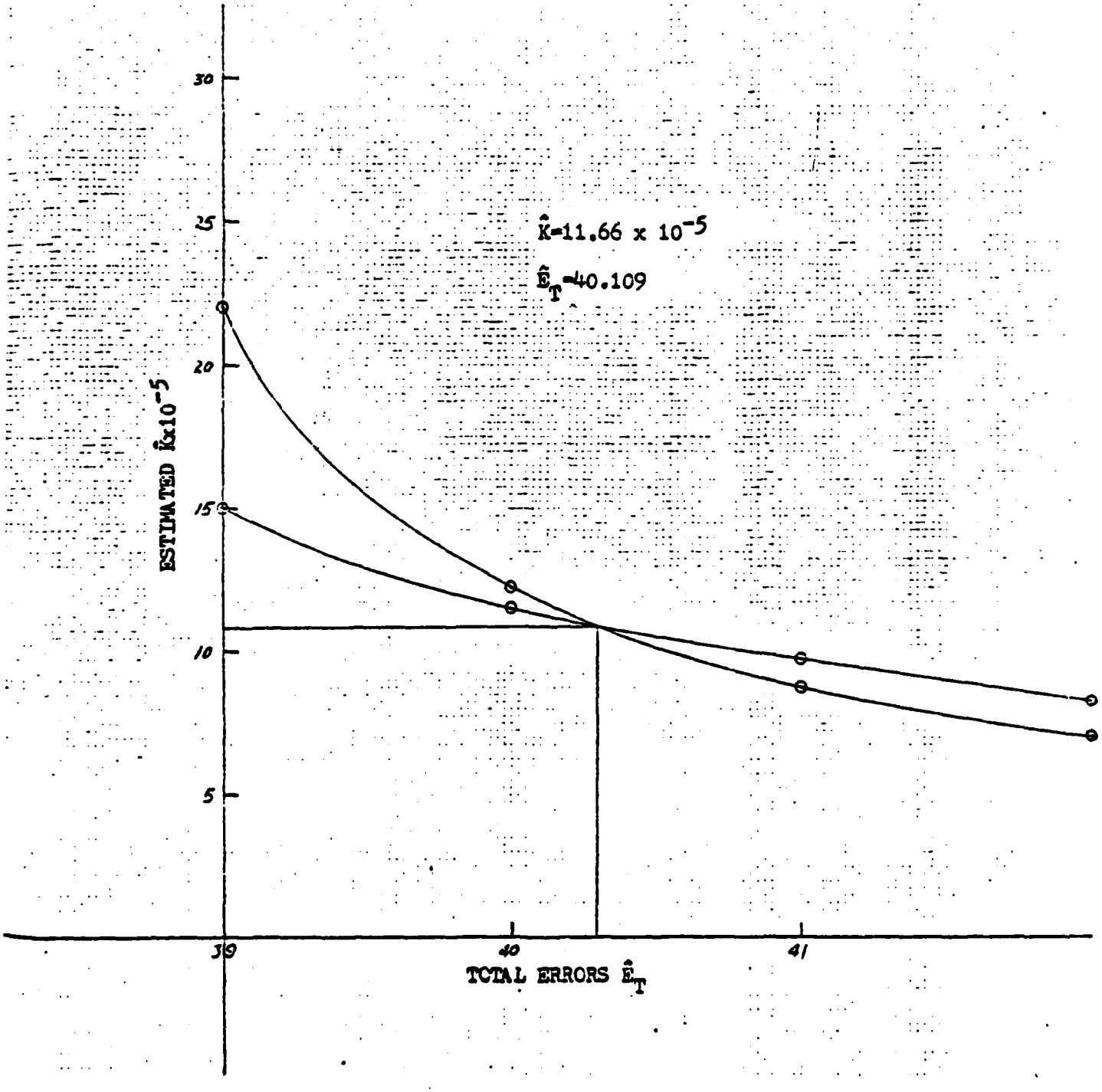


TABLE 5

MAXIMUM LIKELIHOOD ESTIMATES OF $\hat{E}_T$, $\hat{K}$

System	$\hat{E}_T$	$\hat{K}$
1	151.59	5.141×10^{-5}
2	61.61	3.909×10^{-5}
3	40.11	1.160×10^{-4}
4	54.80	2.249×10^{-4}

6.0 Model Parameter Accuracy and Sensitivity

6.1 Introduction

Several additional factors must be considered when using this exponential model - or any other model for that matter - in predicting software system performance.

Two aspects considered in this thesis are:

1. Model Accuracy: The more data one has on a particular system, the more confident one can be that the particular model predictions will approximate reality;

2. Model Sensitivity: When software field failure data is available to compare with model prediction estimates, it seems - initially that slight changes in model parameters are not proportionally reflected in model predictions for system MTTF;

6.2 Model Accuracy

Intuitively, one feels that the more data one has about a particular system, the more accurate will be our predictions about system performance. To a certain extent this is true. This implies, however, that model prediction accuracy only approaches reality as we near the final debugging stage, whereas the 'software manager' is quite interested in estimating software reliability during early testing stages to determine future trends. As noted in Section 2.3, this is strongly implied by Fig. 4 which indicates that the maximum return for

one's efforts is obtained only during the latter 25% of the debugging phase.

Hindsight will usually enable us to look back at a particular software development project and to announce that 'x' time rather than 'y' time should have been spent on development and testing.

The onset of the release point can be determined in several ways, two of which are to:

a) closely monitor the slop of our MTTF curve plotted versus time spent on debugging and to watch for a sharp increase as it nears a vertical asymptote (Fig. 4, 4C) or,

b) since the system MTTF is directly related to the number of software errors remaining $(E_T - E_c)$, look for the point in time when model predictions of $\hat{E}_T$ applied to the system in question approach a horizontal asymptote. This aspect is depicted in Fig. 9 by the shape of expected value and 2σ confidence bands around E_T as testing nears completion.

6.3 Model Sensitivity

The un-normalized equation used to calculate system MTTF is

$$MTTF = \frac{1}{K(E_T - E_c(\tau))} \qquad (27)$$

and the derivative with respect to $\hat{K}$ is

$$\frac{\partial MTTF}{\partial K} = \frac{-1}{K^2(E_T - E_c(\tau))} = -\frac{MTTF}{K} \tag{28}$$

This indicates that variations in $\hat{K}$ result in a quasi-linear effect on MTTF. For small variations in $\hat{K}$ this variation can be linearized as shown in Fig. 10 where the percentage change in $\hat{K}$ is plotted vs resulting percentage change in MTTF.

Slight variations of $\hat{E}_T$, however, result in dramatic changes in MTTF, especially as the former approaches the value of $E_c(\tau)$. Calculating $\frac{\partial MTTF}{\partial E_T}$ yields:

$$\frac{\partial MTTF}{\partial E_T} = \frac{-1}{K(E_T - E_c(\tau))^2} = -\frac{MTTF}{(E_T - E_c(\tau))} \tag{29}$$

Thus, as $E_c(\tau) \rightarrow E_T$, the sensitivity becomes quite large. It can be seen from Fig. 11 that a 10% change in $\hat{E}_T$ results in 70%+ variation of system MTTF. This factor alone emphasizes the requirement of high-quality failure data if accurate model predictions are to ensue.

FIGURE 9

TOTAL ERRORS vs TESTING TIME

SYSTEM 3

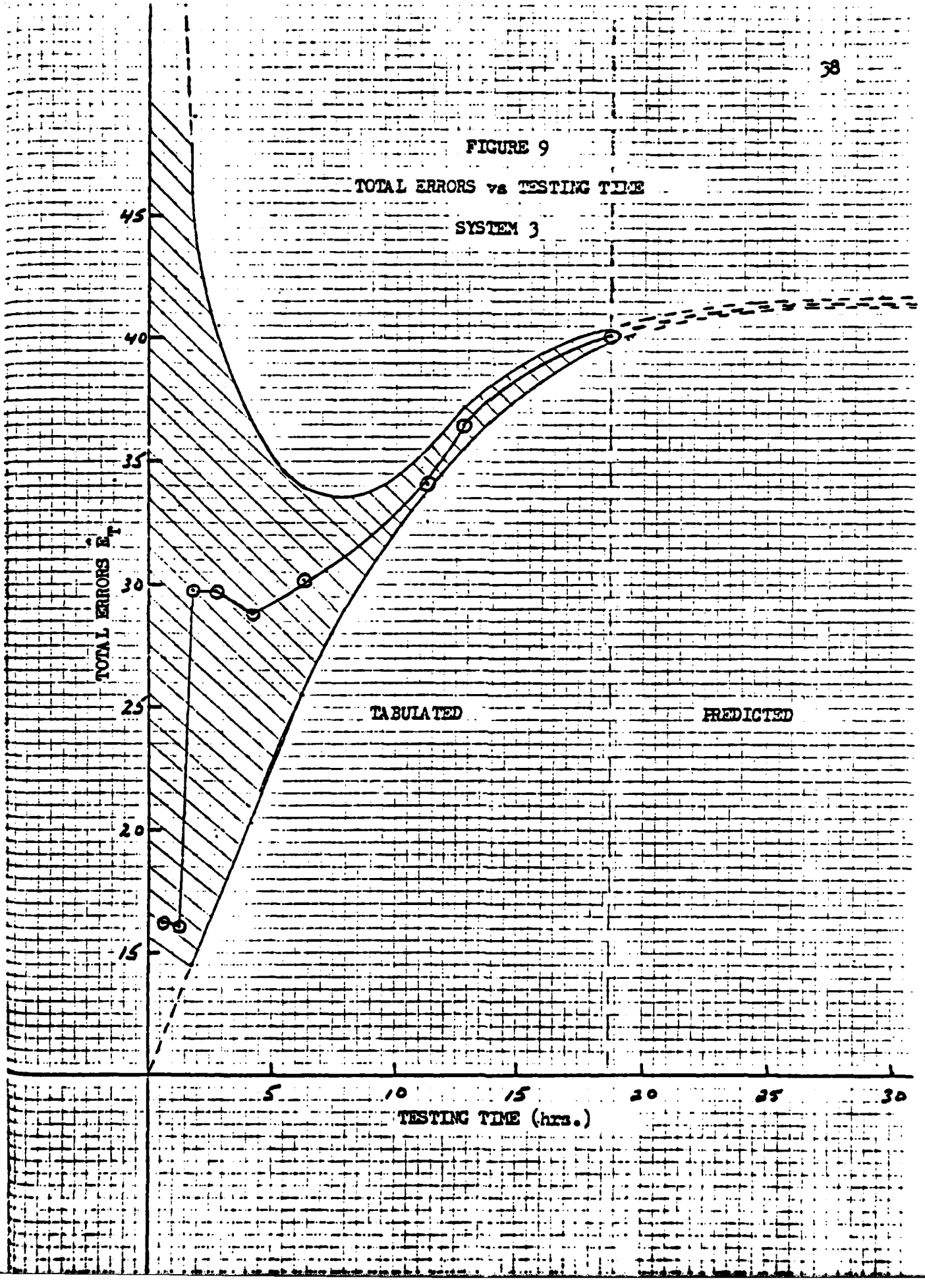


FIGURE 10

CHANGE IN MTTF BY VARYING R

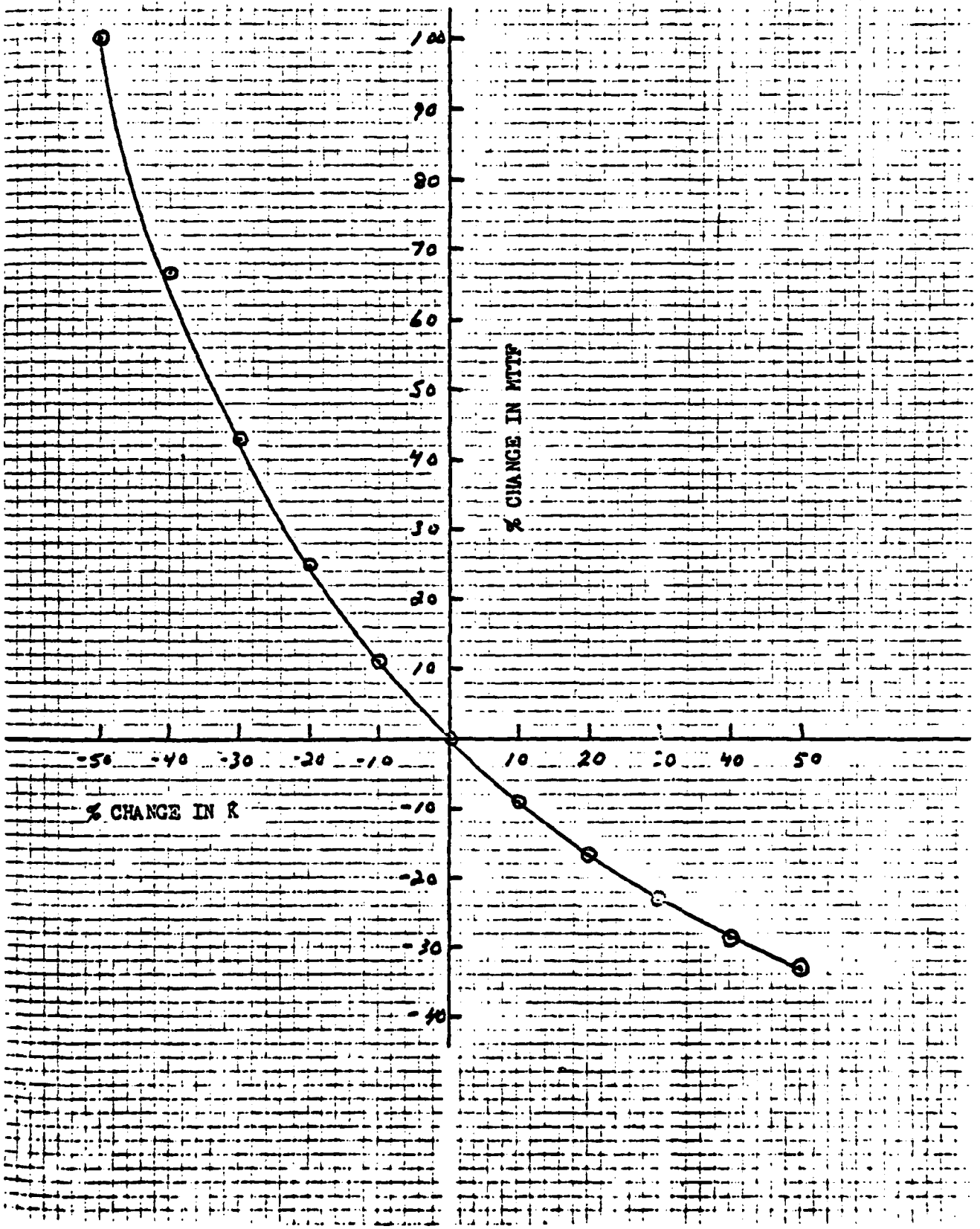
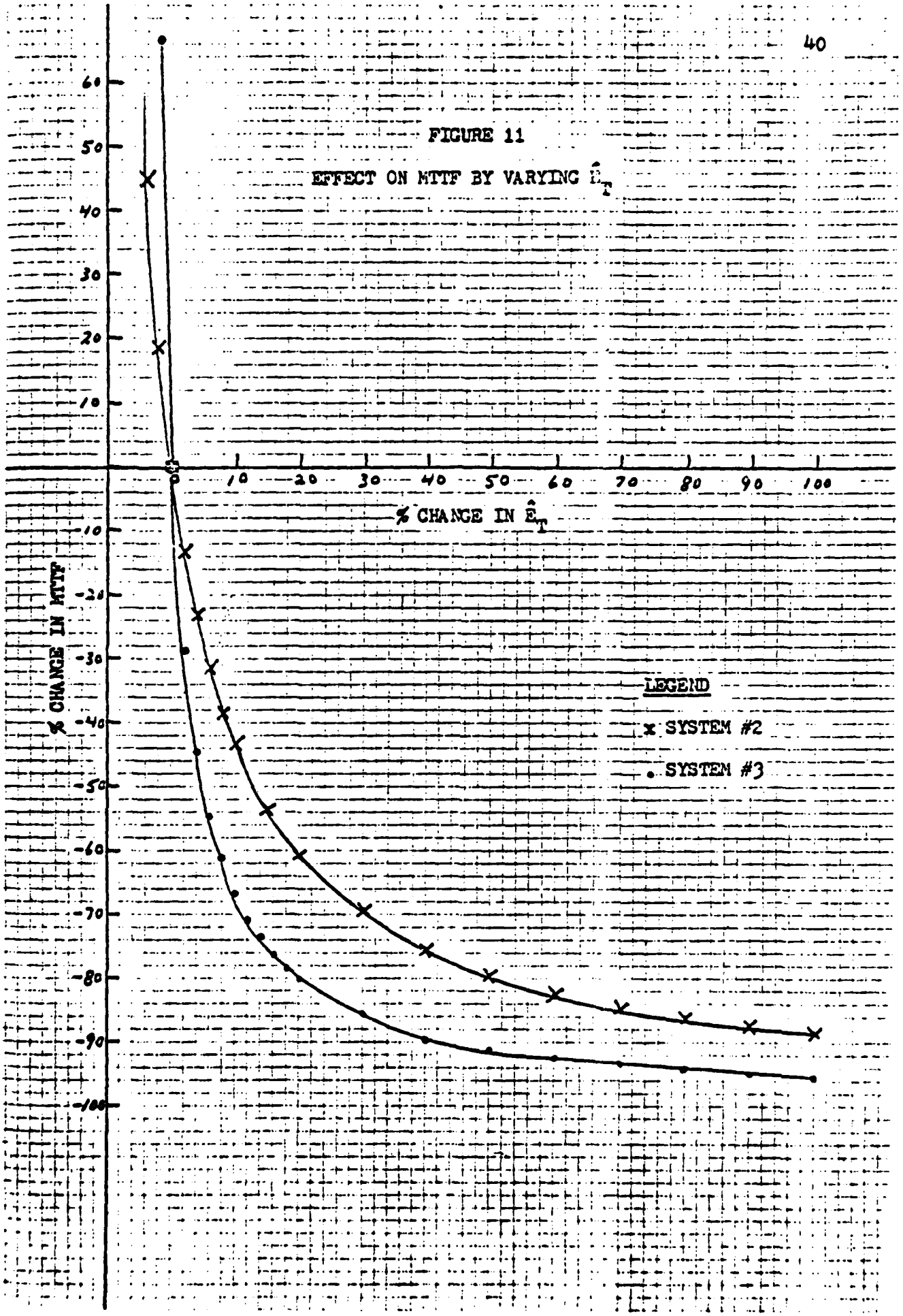


FIGURE 11

EFFECT ON MTTF BY VARYING $\hat{E}_T$



7.0 Conclusion

The primary aim of this thesis was to present a model of software system parameter estimation, to subsequently compare predicted system characteristics with actual field data, and to guide the software manager in the practical application thereof to software systems under his development.

Table 6 summarizes model parameters by system and method of calculation. It can be seen that MTTF prediction accuracy varied from 2% to 67%, depending on the system, with an overall average of 45%.

Surprisingly enough, there was less than $\frac{1}{2}$% difference in prediction accuracy between the three different methods of model parameter calculation Method of Moments, Least Squares, and Maximum Likelihood Estimates. Although most statisticians consider MLE to yield superior results to other methods, the findings of this thesis would indicate that parameter estimation using Least Squares is preferable for the software manager due to the less cumbersome calculations required.

Model parameter sensitivity was explored and indicated that a variation in $\hat{K}$ resulted in a linear variation in estimated MTTF. This was not the case with variations in total system estimated errors ($\hat{E}_T$), however, as minute variations in the latter resulted in drastic changes in predicted MTTF.

Once model parameters and system MTTF have been calculated, it is of great interest to the software manager to use these calculations to determine/predict the optimal release date for the system under development. Sections 2.3 and 6.2 suggest several ways of doing so.

TABLE 6

COMPARISON OF MODEL PREDICTIONS WITH FIELD EXPERIENCE

Syst. No.		$K \times 10^{-5}$	E_T	E_c	$MTTF_T$	C	$MTTF_O$ ($C \times MTTF_T$)	$MTTF_f$	$\%\Delta_1$	$MTTF_C$ (MUSA)	$\%\Delta_2$
1	MCM	5.514	152	136	.323	15.1	4.9	14.6	66.6	20.4	39.7
	LS	5.515	151		.323		4.9		66.6		
	MLE	5.141	152		.346		5.2		64.2		
2		3.909	62	54	.935	13.6	12.7	31.4	59.5	43.5	38.5
		3.909	62		.935		12.7		59.5		
		3.909	62		.934		12.7		59.6		
3		11.60	40	38	1.14	13.2	15.0	30.3	50.4	30.4	0.33
		11.603	40		1.14		15.0		50.3		
		11.60	40		1.13		15.0		50.6		
4		19.88	55	53	.735	13.1	9.63	9.17	5.1	14.5	58.1
		19.89	55		.735		9.63		5.0		
		22.49	55		.686		8.98		2.0		

LEGEND

$MTTF_T$ - Predicted MTBF based on testing time

$MTTF_O$ - Predicted operational (field) MTBF

$MTTF_f$ - Actual operational (field) MTBF

$\%\Delta_1$ - % difference between actual and predicted MTBF (exponential model)

$\%\Delta_2$ - % difference between actual and predicted MTBF (MUSA'S model)

MCM - Method of Moments

LS - Least Squares

MLE - Maximum Likelihood Estimate

REFERENCES

1. Martin L. Shooman, "Probabilistic Models for Software Reliability Prediction", published in "Statistical Computer Performance Evaluation", Walter Freiberger editor, Academic Press, New York, 1972, pp. 485-497.

2. _____, "Software Reliability", published in "Computing Systems Reliability", T. Anderson and B. Randell editors, Cambridge University Press, New York, 1979.

3. _____, "Software Engineering: Design, Reliability, Management", McGraw-Hill Book Co., New York, 1980.

4. M.L. Shooman and S. Natarajan, "Effect of Manpower Deployment and Bug Generation on Software Error Models", Proceedings of the 1976 MRI Symposium on Computer Software Engineering, Polytechnic Press, 1976.

5. J. Dickson, J. Hesse, A. Kientz, M. Shooman, "Quantitative Analysis of Software Reliability", 1972 Annual Reliability Symposium Proceedings, IEEE, Jan. 1972.

6. John D. Musa, "Validity of Execution-Time Theory of Software Reliability", IEEE Transactions on Reliability, Specil Issue on Software Reliability, August 1979, vol. R-28, No. 3, p. 181-191.

REFERENCES

7. Martin L. Shooman, "Probabilistic Reliability: An Engineering Approach", McGraw-Hill Book Co., New York, 1968.

8. John D. Musa, "A Theory of Software Reliability and Its Application", IEEE Transactions on Software Engineering, Vol. SE-1, No. 3, Sept. 75, pp. 312-327.

9. M.L. Shooman, "Operational Testing and Software Reliability Estimation During Program Development", Record 1973, IEEE Symposium on Computer Software Reliability, April 30 - May 2, 1973 IEEE, New York, pp. 51-57.

10. John D. Musa, "Software Reliability Data", Data and Analysis Center for Software (DACS), Rome Air Development Center, Griffiss AFB, N.Y. 13441, 1979.

11. "Software Engineering Research Review-Quantitative Software Models", Data and Analysis Center for Software (DACS), Rome Air Development Center, Griffiss AFB, N.Y. 13441, March 1979.

12. M.L. Shooman, "Software Reliability Data Analysis and Model Fitting", Proceedings of Workshop on Quantitative Software Models, IEEE, N.Y., Oct. 1979, pp. 182-188.

REFERENCES

13. I. Miyamoto, "Software Reliability in Online Real Time Environment", Proceedings, International Conference on Reliable Software, April 1975, IEEE Cat. No. 75CH0940-7CSR.

14. M.L. Shooman and R.W. Schmidt, "Fitting of Software Error and Reliability Models to Field Failure Data", January 1981, ORSA-TIMS Conference.