A Theory of Software Reliability and Its Application

JOHN D. MUSA, MEMBER, IEEE

Abstract—An approach to a theory of software reliability based on *execution time* is derived. This approach provides a model that is simple, intuitively appealing, and immediately useful.

The theory permits the estimation, in advance of a project, of the amount of testing in terms of execution time required to achieve a specified reliability goal [stated as a mean time to failure (MTTF)]. Execution time can then be related to calendar time, permitting a schedule to be developed. Estimates of execution time and calendar time remaining until the reliability goal is attained can be continually remade as testing proceeds, based only on the length of the execution time intervals between failures. The current MTTF and the number of errors remaining can also be estimated. Maximum likelihood estimation is employed, and confidence intervals are also established. The foregoing information is obviously very valuable in scheduling and monitoring the progress of program testing. A program has been implemented to compute the foregoing quantities.

The reliability model that has been developed can be used in making system tradeoffs involving software or software and hardware components. It also provides a soundly based unit of measure for the comparative evaluation of various programming techniques that are expected to enhance reliability.

The model has been applied to four medium-sized software development projects, all of which have completed their life cycles. Measurements taken of MTTF during operation agree well with the predictions made at the end of system test. As far as the author can determine, these are the first times that a software reliability model has been used *during* software development projects. The paper reflects and incorporates the practical experience gained.

Index Terms—Debugging, error prediction, program testing, reliability prediction, software project monitoring, software project scheduling, software reliability.

I. INTRODUCTION

THE application of the concepts of reliability to the description of the performance of computer programs is in its infancy. Obviously, however, reliability is an extremely important performance parameter whose quantification is desperately needed. Potential uses of software reliability figures include the following:

1) Making intelligent system design tradeoffs between reliability, performance, cost, schedules, and other factors for and between the programs themselves and other system elements as well. These tradeoffs may be made both at the start of the project and as it proceeds. They may involve combination of software reliability parameters with those of other system components to obtain system reliability estimates or allocation of system reliability goals among subsystems, one or more of the subsystems being computer programs.[1]

Manuscript received April 7, 1975; revised June 12, 1975.
The author is with Bell Laboratories, Whippany, N.J. 07981.

[1] Littlewood [1] has shown how mean time to failure (MTTF) of a large program may be computed from the failure rates of its modules under a fairly broadly applicable set of conditions (software components generally do not combine with each other in simple series–parallel fashion).

2) Scheduling and monitoring progress of a testing effort by using continually updated estimates of current reliability. Included in the foregoing is determining when to terminate a testing effort.

3) Comparatively evaluating the effects on reliability of different design techniques, coding techniques, testing techniques, and documentation approaches.

The concept of software reliability differs from that of hardware reliability in that failure is not due to a "wearing out" process. Once a software defect is properly fixed, it is in general fixed for all time. Failure usually occurs only when a program is exposed to an environment that it was not designed or tested for. The large number of possible states of a program and its inputs make perfect comprehension of the program requirements and implementation and complete testing of the program generally impossible. Thus, software reliability is essentially a measure of the confidence we have in the design and its ability to function properly in all environments it is expected to be subjected to. In the life cycle of software there are generally one or more test phases during which reliability improves as errors are identified and corrected, typically followed by a nongrowth operational phase during which further corrections are not made (for practical and economic reasons) and reliability is constant.

The "design reliability" concept described above could be applied to hardware systems; the reason it has not been to date is most likely because the probability of failure due to wearing out has usually been much greater than the probability of failure due to an unrecognized design problem. Hardware failures due to design problems could be kept low because hardware was generally less complex logically than software and they had to be kept low because retrofitting of manufactured items in the field was a very expensive process.

Despite the foregoing differences, the software reliability model derived in this paper is compatible with hardware reliability models and may be combined with them, using standard techniques, to compute system reliability figures.

Jelinski and Moranda [2] developed one of the earliest software reliability models, describing what they call a "deeutrophication process" in which the error detection rate is proportional to the number of errors remaining (Miyamoto [3] has recently published data confirming this relationship), and applying maximum likelihood estimation to its parameters. Shooman [4], [5] used a similar model, relating error detection rate also to the program size and instruction processing rate. He investigated various models for error correction and proposed a two-point parameter estimation approach. Schneidewind [6], [7] took an empirical approach and suggested a reliability

prediction scheme based on fitting failure intervals with an appropriate reliability function. He has developed models for error detection and correction processes and applied maximum likelihood estimation to the determination of the parameters of these processes [8]. Littlewood and Verrall [9], [10] developed a very general Bayesian reliability growth model for software in which repair actions diminish the failure rate in a probabilistic rather than deterministic fashion.

After considerable study, based partly on experience with software development projects, the author concluded that a superior reliability model, incorporating some of the concepts of the foregoing models, could be constructed if one first thought in terms of *execution time*, i.e., the actual processor time utilized in executing the program, and then determined how execution time and calendar time were related. Furthermore, one could relate the error correction rate to the instantaneous failure rate during test, eliminating the need for developing an error correction model. The correction rate curve will be shown to be exponential with *execution time* and it can take various forms in calendar time, depending on the personnel and computer resources available. The *execution time* model therefore explains the error correction profiles observed by Shooman [4] in a very simple and intuitively appealing fashion.

This reliability model was then applied during the system integration test phases of four medium-sized software development projects (19 500, 6600, 11 600, and 9000 instructions written or modified in programs of total size of 21 700, 27 700, 23 400, and 33 500 instructions, respectively). The projects all involved the development of real-time interactive systems. Each system utilized one, two, or three display consoles with pushbuttons and light pens and repertoires of displays, one or two interactive teletypewriter positions, and some wall displays. The applications primarily involved complex logic and control, with some command interpretation but with only a small amount of mathematical processing. There was extensive hardware–software interaction. Time response was critical, and there was a fairly elaborate set of processing priorities. Program overlay was very frequent.

Programming was done in assembly language. Nine, five, six, and seven programmers were employed, respectively, and the total project lengths were 12, 11, 12, and 10 months. In each case some of the routines were new and others were modifications of existing ones. Debugging was performed partially on-line and partially off-line, in the latter case using dumps and event records. Thus, the debugging environment was somewhere between the typical minicomputer operation and large computation center batch processing. The projects have completed their life cycles, after use periods of about 7, 3, 2, and 2 months, respectively.

The principal part of this paper will be devoted to an exposition of the execution time and calendar time components of the model for the phases of reliability growth (referred to as "test" phases). The number of errors remaining and the mean time to failure (MTTF) for the operational phase are constant and equal to those at the end of the last test phase (unless errors are corrected, in which case the operational phase should be considered as a "test" phase or phase of reliability growth). Following a discussion of how to estimate model parameters, use of the model in monitoring testing progress and in computing cost for various purposes will be covered. Finally, data bearing on the validity of the model will be presented.

II. EXECUTION TIME MODEL—ASSUMPTIONS

The execution time software reliability model can be applied theoretically to any phase[2] of test of any executable[3] program or program unit (or release of a multiple-release series) and test environment that satisfies a few simple requirements.

Requirement 1: Any errors in the program are independent of each other and are distributed at any time with a constant average occurrence rate (per instruction) throughout the program.

Requirement 2: Types of instructions are reasonably well mixed and execution time between failures is large compared to average instruction execution time.

Requirement 3: The potential test space for the program "covers" its use space.

Requirement 4: The set of inputs for each run of the program, whether during a test or operational phase, is selected randomly.

Requirement 5: All failures are observed.

Requirement 6: The error causing each failure is fixed before testing resumes or its rediscovery is not counted again.

Most programs and their test environments will satisfy the above requirements in at least approximate fashion. Although a constant average error occurrence rate in a program unit is disturbed somewhat whenever an error is found and removed, it is approximated satisfactorily as long as testing (and hence exposure and correction of errors) is well distributed across all parts of the program unit at all times. Consequently, for the best results *in applying the model* to a program unit, integration of that unit should be complete (i.e., do not apply the model until the entire unit is present).

The effect of the second assumption is to make the set of instructions between failures a large random sample so that the average instruction execution rate of the sample will be very close to the overall average instruction execution rate.

The third requirement insures that testing is *representative* of the eventual use of the program. It does not mean that testing must be complete, only that tests must be selected from a complete set of use input sets. A test selection technique that was applied to the four projects described in this paper was to directly plan tests at the extremes of system performance and then randomly pick tests at intermediate points. In any event, test planning requires care and a clear understanding of the operating environment the program must work in.

The effect of the fourth assumption is to prevent arti-

[2] Applying the model to the system integration test phase is particularly useful; since it occurs just before release of the program, MTTF during use can be predicted.

[3] Successfully compiled and linked.

ficial selection of sequences of input sets that might have a higher or lower proportion of errors associated with them than the average, thus causing biased results.

Requirement 5 is necessary because a "failure" (defined as a departure of program operation from program requirements) can sometimes be subtle. Observation may occur at the time of program operation or when analyzing program output afterwards. If some failures are missed, estimation of the program's reliability will be optimistic. The author found that one is more likely to miss failures in the use period of the program than the test period (heightened attention to failures is characteristic of most test periods). Hence, the estimate of the program's reliability is still *relatively* pessimistic, which is satisfactory from a practical viewpoint.

The sixth assumption can always be met.

An "error" is defined as the defect or set of defects that is causing the failure. The model is not dependent on the nature of the errors; they may result from single or multiinstruction defects, from wrong code, missing code, superfluous code, or misplaced code. Neither is it dependent on the cause of the errors (requirements misunderstanding, poor interface definition or communication, poor algorithm, "stupid" error, etc.). If one wishes, failures can be classified as to severity and the theory developed in this paper applied to each class. This will generally degrade the estimates that are made, however, due to the smaller sample sizes. Further, the estimation of calendar times will be very difficult.

Note that Requirement 1 implies that the number of failures occurring in a given execution time has a Poisson distribution (the parameter changes whenever an error is corrected). The execution time between the occurrence of failures will be distributed exponentially.

It should be emphasized that the model is a statistical one; it does not predict, for example, precisely when the next failure will occur but only the most likely time and a set of confidence intervals.

III. DERIVATION OF EXECUTION TIME COMPONENT OF MODEL

A. Basic Derivation

Let τ be the execution time or actual processor [central processing unit (CPU)] time utilized in executing the program up to the present and let τ' represent execution time projected from the present into the future. Let N_0 be the number of inherent (existing before the test phase) errors in the program. We will consider N_0 as constant.[4] Let n be the net number of errors corrected and $\mathfrak{N}$ the number of errors remaining (both are functions of τ). Then

$$\mathfrak{N} = N_0 - n. \tag{1}$$

With Requirements 1 and 2 of Section II satisfied, the instantaneous failure rate or hazard function z of reliability theory [11] will be proportional to $\mathfrak{N}$ and the linear execution frequency f (average instruction execution rate divided by number of instructions in the program). Thus

$$z(\tau) = Kf\mathfrak{N}, \tag{2}$$

where the proportionality constant K is an error exposure ratio which relates error exposure frequency to linear execution frequency.[5] Note that the hazard function is independent of τ' as long as we do not let τ' range beyond the time of correction of the next error. Hence, we have a piecewise-constant failure rate model, with the rate parameter changing at discrete intervals. By using (1), (2) may be written as

$$z(\tau) = fKN_0 - fKn. \tag{3}$$

Now, Requirements 5 and 6 plus the assumption (for the moment) that we do not spawn any new errors in the process of error correction imply that the error correction rate $dn/d\tau$ will be equal to the error exposure rate. This deceptively simple result (in hindsight) is one of the keys to the development of the execution time model. Thus

$$\frac{dn}{d\tau} = z(\tau). \tag{4}$$

If we combine (3) and (4) we obtain the differential equation

$$\frac{dn}{d\tau} + fKn = fKN_0. \tag{5}$$

Since $n = 0$ at $\tau = 0$, the solution of (5) is given by

$$n = N_0[1 - \exp(-fK\tau)]. \tag{6}$$

The reliability of the program is conveniently characterized by its MTTF T, given by

$$T = \frac{1}{z(\tau)}. \tag{7}$$

By using (3), (6), and (7) we obtain

$$T = \frac{1}{fKN_0} \exp(fK\tau). \tag{8}$$

[4] The theory developed in this paper can be generalized by considering the number of inherent errors to be a variable. Such an approach can account for new inherent errors resulting from the introduction of new or modified code (such as that resulting from requirements or design changes). The approach was not taken because of the substantial complexity that is added.

[5] Shooman [4], [5] uses an expression for the hazard function that is based on average instruction processing rate and error density (errors per instruction); it and the above expression are essentially equivalent. Equation (2) has been used in this paper because it permits simple derivations to be made without the need to assume that the number of instructions in the program is constant during the test period, and because it was found more convenient to work with than error density. The coefficient K takes account of the fact that each instruction can be executed in many different machine states, with an error usually occurring for only a few of them, and these few often happening rather infrequently. Note that K represents an *average* error exposure frequency to linear execution frequency ratio, taken over that set of program executions that will occur in use.

Let T_0 represent the initial MTTF before testing. Then

$$T_0 = \frac{1}{fKN_0}. \tag{9}$$

We may rewrite (8) as

$$T = T_0 \exp\left(\frac{\tau}{N_0 T_0}\right). \tag{10}$$

Note that MTTF exhibits "growth" as testing proceeds.

It is easy to express results in terms of reliability by using a basic relationship between reliability and the hazard function and (7):

$$R(\tau,\tau') = \exp\left[-\int_0^{\tau'} z(\tau)\,dx\right] = \exp\left[-\tau' z(\tau)\right]$$

$$= \exp\left(-\frac{\tau'}{T}\right). \tag{11}$$

B. Generalizations of Model

It will be noted that in the derivation we temporarily assumed that no new errors were spawned in the process of error correction. This is not a bad approximation; however, greater generality is readily obtained. Let us define an error reduction factor B as the average ratio of the rate of reduction of errors to the rate of failure occurrence.[6] This factor can be measured for a similar project and environment. For large projects, it appears likely that B may not vary from project to project because of large-sample averaging effects. Now the error correction rate becomes

$$\frac{dn}{d\tau} = Bz(\tau). \tag{12}$$

We have ignored one other consideration which is easily incorporated in the model. Failures generally occur more frequently during testing than during use of a program because the test cases are usually planned to "compress reality" somewhat.

To visualize the relationship of testing to use, consider the space of the A possible different sets of inputs expected during use (Fig. 1). The point represents the ath set; assume it will occur a total of r_a' times during the life of the system. If m_a is the number of failures and τ_a the execution time associated with a "run" involving the ath input set, then the failure rate z' during use is given by

$$z' = \sum_{a=1}^{A} \frac{m_a}{r_a' \tau_a}. \tag{13}$$

Let r_a be the number of occurrences of the ath input set during test. Since the potential test space must cover the

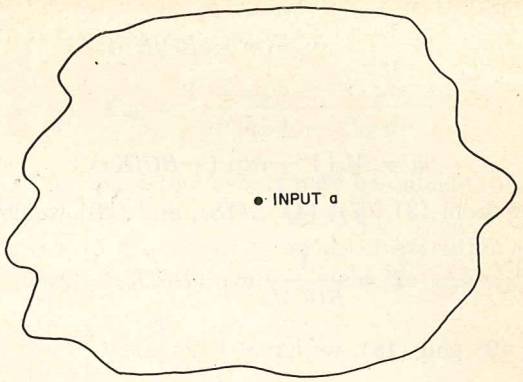

Fig. 1. Use space of A inputs.

input space, the average failure rate z during test is

$$z = \sum_{a=1}^{A} \frac{m_a}{r_a \tau_a} \tag{14}$$

where $r_a \geq 1$. We will define a testing compression factor C as the average ratio of rate of detection of failures (with execution time) during test to that during use. Thus

$$C = \frac{\sum_{a=1}^{A} \frac{m_a}{r_a \tau_a}}{\sum_{a=1}^{A} \frac{m_a}{r_a' \tau_a}}. \tag{15}$$

If $r_a = r_a'$ then $C = 1$. However, usually $r_a < r_a'$ and $C > 1$. If $r_a = 1$ (no tests are repeated), C is maximized.[7]

It is not generally feasible to compute C. However, this factor may be measured for similar testing environments or estimated from the test length and the use period the test is believed to represent.

With this added consideration, the error correction rate now becomes

$$\frac{dn}{d\tau} = BCz(\tau). \tag{16}$$

If we let m represent the number of failures experienced in correcting n errors and M_0, the number of failures required to expose and remove the N_0 inherent errors, we will have

$$m = \frac{n}{B} \tag{17}$$

and

$$M_0 = \frac{N_0}{B}. \tag{18}$$

We may recast (5), using (3), (16), (17), and (18), as

[6] The factor is usually positive and less than one although it could conceivably be negative or zero (in the latter case some equations will have to be specially reinterpreted). It could also be greater than one in the case that finding the error that produced a particular failure causes other errors to be found and fixed as well.

[7] This extreme situation is probably undesirable from a practical viewpoint; some repetition of tests is necessary to prevent errors that may be masked by other errors in early tests from being lost.

$$\frac{dm}{d\tau} + BCfKm = BCfKM_0. \tag{19}$$

Then

$$m = M_0[1 - \exp(-BCfK\tau)]. \tag{20}$$

Now from (3), (7), (17), (18), and (20) we obtain

$$T = \frac{1}{BfKM_0} \exp(BCfK\tau). \tag{21}$$

Using (9) and (18), we have

$$T = T_0 \exp\left(\frac{C}{M_0 T_0}\tau\right). \tag{22}$$

C. Related Useful Results

The number of failures Δm that must be detected and corrected to achieve an increase in MTTF from T_1 to T_2 may be obtained by combining (9), (18), (20), and (21) to yield

$$m = M_0\left[1 - \frac{T_0}{T}\right] \tag{23}$$

and then forming

$$\Delta m = m_2 - m_1 = M_0 T_0\left[\frac{1}{T_1} - \frac{1}{T_2}\right]. \tag{24}$$

The additional execution time required to attain the MTTF increase is readily obtained by manipulating (22) to yield

$$\Delta\tau = \frac{M_0 T_0}{C} \ln\left(\frac{T_2}{T_1}\right). \tag{25}$$

IV. CALENDAR TIME COMPONENT OF MODEL

A. Derivation

We shall now relate execution time τ to calendar time t. The calendar time component of the model is most practically applied to the system test phase of a project. The pace of testing is constrained by three limited resources: failure identification personnel, failure correction personnel, and computer time.[8] Although the quantities of these resources to be used may be more or less freely established in the early stages of a project, increases are generally not feasible by the time one has reached the system test phase because of the long lead times required for training and computer procurement. At any given value of τ, one of the constraints will be limiting and the other two resources will not be fully utilized. Hence, if we let $dt_I/d\tau$, $dt_F/d\tau$, and $dt_C/d\tau$ represent the instantaneous calendar time to execution time ratios that result from the effects of each of the resource constraints taken alone, respectively, then we have

$$\Delta t = \int_{\tau_1}^{\tau_2} \max\left(\frac{dt_I}{d\tau}, \frac{dt_F}{d\tau}, \frac{dt_C}{d\tau}\right) d\tau. \tag{26}$$

The integration will be divided into segments, each of which is associated with a constraining resource.

The calendar time model we will develop depends on various *average* resource expenditure statistics determined for similar program development environments. It must be applied in a statistical sense; one can predict "most likely" values and determine confidence intervals but precise prediction is not possible. It is assumed that the project for which calendar time predictions are being made is large enough so that small-sample effects of different programmer skill levels and levels of task difficulty may be ignored (unless they are known and can be accounted for). We also assume that computer turnaround time is kept short enough so that total wait time involved in correcting each failure is negligible in comparison with average failure correction work time required per failure (note that there is usually only one approach to the machine for each failure identification; hence turnaround time has little effect on that process).[9]

Experience has indicated that all three resource requirements may be closely approximated by a model of the form

$$\chi = \theta\Delta\tau + \mu\Delta m, \tag{27}$$

where χ is the resource requirement, θ is the average resource expenditure rate with execution time, and μ is the average resource expenditure per failure. Identification of failures requires work and computer time that increases with the amount of execution time used and the related amount of test output generated.[10] Also, each failure requires work for verification and determination of its specific nature. Correction of failures requires work and computer time that is proportional to the number of failures (note that m failures are worked on even if only n errors are effectively fixed).[11] Correction is considered to include

[8] "Computer time" is not necessarily the same as execution time. It is the measure that is used for allocating computer resources. Frequently, it is the residence time of the program in the machine (perhaps divided by the number of programs that can be concurrently resident in the case of multiprogramming). Failure identification personnel and failure correction personnel are considered as separate resources since testing and failure correction are usually performed by different people in the case where this model will be most frequently applied, the system test phase of a project of reasonable size. Note that failure identification involves *only* the determination that the program is not performing in accordance with its requirements in some specific way; it does not involve finding the error that is causing the failure.

[9] Shooman and Bolsky [12] have collected data that indicate an average of 0.61 runs per failure for failure identification and 1.35 runs per failure for failure correction.

[10] Work associated with test planning and test case development is assumed to have been completed prior to the testing effort and is not included in this model.

[11] One might think that failures occurring late in a test phase would be more complex and hence require more effort to correct. Data taken by Shooman and Bolsky [12, p. 351] do not demonstrate such a pattern with any significance, and this finding was verified in experience with the four projects reported in this paper. It may be that increasing programmer experience with the system is a countervailing factor. In any case, if the relationship does exist, it is apparently so weak that it does not justify departing from the simpler constant model. Hence, for failure correction work, θ is taken to be zero.

repeating the test that previously caused the failure and all change documentation prepared during (not after) the test period.

Now, using (9), (18), and (20) in (27) we obtain

$$\chi = \theta\Delta\tau + \mu M_0\left[\exp\left(-\frac{C}{M_0T_0}\tau_1\right) - \exp\left(-\frac{C}{M_0T_0}\tau_2\right)\right]. \tag{28}$$

Let P represent the number of available personnel or the available computer shifts. The latter quantity is measured in terms of prescribed work periods (customary work period including average overtime) to put it on the same basis as personnel. For example, if the prescribed work week is 40 h and the computer is available 120 h/week, $P = 3$ for the computer. It is assumed that both the failure identification and failure correction personnel are *available* to work the same prescribed work week (which may include overtime). This does not mean that they actually do so. Let ρ be a utilization factor (discussed later). Then the effective available personnel or computer shifts is ρP.

The calendar time requirement associated with each resource is

$$\Delta t = \frac{\theta}{P\rho}\Delta\tau + \frac{M_0\mu}{P\rho}\left[\exp\left(-\frac{C}{M_0T_0}\tau_1\right) - \exp\left(-\frac{C}{M_0T_0}\tau_2\right)\right]. \tag{29}$$

Differentiating (29), we obtain

$$\frac{dt}{d\tau} = \frac{\theta}{P\rho} + \frac{C\mu}{PT_0\rho}\exp\left(-\frac{C}{M_0T_0}\tau\right). \tag{30}$$

Using (22), we obtain

$$\frac{dt}{d\tau} = \frac{\theta T + C\mu}{P\rho T}. \tag{31}$$

For convenience, we will change the variable of integration in (26), using (22):

$$\Delta t = \frac{M_0T_0}{C}\int_{T_1}^{T_2}\frac{1}{T}\max_k\left[\frac{\theta_kT + C\mu_k}{P_k\rho_kT}\right]dT, \tag{32}$$

where k may be C, F, or I. Now for each segment

$$\Delta t_k = \frac{M_0T_0}{P_k\rho_k}\left[\mu_k\left(\frac{1}{T_{k_1}} - \frac{1}{T_{k_2}}\right) + \frac{\theta_k}{C}\ln\left(\frac{T_{k_2}}{T_{k_1}}\right)\right], \tag{33}$$

where k_1, k_2 are the integration limits for that segment and k is chosen to maximize

$$\frac{\theta_kT + C\mu_k}{P_k\rho_kT}$$

for any value of T in the segment.

The segment integration limits are a subset of T_1, T_2 and three transition points obtained by equating $dt_k/d\tau$ and $dt_{k'}/d\tau$ for $k = C, F, I$, and $k' = F, I, C$, respectively. The transition points are given by

$$T_{kk'} = \frac{C(P_k\mu_{k'}\rho_k - P_{k'}\mu_k\rho_{k'})}{P_{k'}\rho_{k'}\theta_k - P_k\rho_k\theta_{k'}}. \tag{34}$$

Some of the transition points may be outside the interval $[T_1,T_2]$; they are, of course, ignored.

In order to express Δt in terms of execution time, we may substitute (25) in (33) to yield for each segment

$$\Delta t_K = \frac{1}{P_k\rho_k}\left\{M_0\mu_k\left[\exp\left(-\frac{C\tau_{k_1}}{M_0T_0}\right) - \exp\left(-\frac{C\tau_{k_2}}{M_0T_0}\right)\right] + \theta_k\Delta\tau\right\}. \tag{35}$$

In general, it is difficult to express $\Delta\tau$ explicitly in terms of Δt except in the case of a test phase which is completely failure-correction-personnel-limited (and with $\theta_F = 0$) and for the region $\tau \ll M_0T_0/C$. In this situation (35) simplifies to

$$\Delta\tau = \frac{P_FT_0\rho_F}{C\mu_F}\Delta t. \tag{36}$$

This expression may be useful for tracking the amount of testing that should be completed at any point in calendar time.

The existence of "repair data" may stimulate one to consider applying the concept of availability to operational software. The concept can be applied, of course, only if defects *are* fixed during the operational period of the program, a situation that does not necessarily obtain. Availability is more difficult to compute for software than hardware, since the failure rate changes in stepwise fashion as each error is corrected. However, Trivedi and Shooman have developed an approach based on failure and correction rates expressed in calendar time [13].

B. Utilization Factors

During the failure-identification-personnel-limited period, there is usually no reason why available failure identification personnel cannot be fully used; hence, $\rho_I = 1$.

In the failure-correction-personnel-limited period, available failure correction personnel cannot be fully employed at all times because of the unpredictable identification times of failures and inequality in load among the debuggers.

The identification of failures is a time-invariant Poisson process in calendar time (mutual independence of failures has already been postulated and the fact that the failure rate with respect to calendar time is constant during the failure-correction-personnel-limited period will be demonstrated shortly). Let λ be the parameter of the process. As failures are identified, they are apportioned among P_C debuggers for correction in accordance with their assigned areas of program responsibility. Since one of the

postulates of the execution time model implies that testing must be well distributed across different parts of the program, this assignment of failures may be viewed as resulting in a random selection of debuggers from a calendar time point of view. Hence, the input rate to each debugger is λ/P_F.

It will be assumed that the correction of failures is also a time-invariant Poisson process. This implies that the correction of a failure is independent of the correction of other failures and that the probability that an error is fixed during any time period of debugging effort is equal to the probability that it is fixed in any other such time period. This is a fairly good assumption, although it may somewhat overestimate the proportion of errors with short fix times. The service rate of each debugger for correction is seen to be $1/\mu_F$.

The utilization factor ρ_F for each debugger is defined (in queueing theory) as the ratio of input rate to service rate or

$$\rho_F = \frac{\lambda \mu_F}{P_F} . \tag{37}$$

Experience has indicated that testing is stopped whenever an excessive backlog[12] of failures is building up for any one debugger and hence impeding the identification and correction of other failures in his program area.

The probability that a particular debugger has a queue of m_Q or more failures awaiting correction or being corrected is $\rho_F{}^{m_Q}$ in the steady state, assuming $\rho_F < 1$ [14]. The probability $\mathcal{P}_{m_Q}$ that no debugger has m_Q or more failures queued is

$$\mathcal{P}_{m_Q} = (1 - \rho_F{}^{m_Q})^{P_F}. \tag{38}$$

Therefore, we may assure with probability $\mathcal{P}_{m_Q}$ that no debugger has m_Q or more failures to work on at any given time by limiting ρ_F to

$$\rho_F = (1 - \mathcal{P}_{m_Q}{}^{1/P_F})^{1/m_Q}. \tag{39}$$

Note that the restriction $\rho_F < 1$ is satisfied for (39). If we control ρ_F *at* the value indicated in (39), we will maximize effective manpower and prevent excessive backlogs. Note that we will also maintain λ constant, justifying a previous assumption.

Use of the steady-state value of probability for queue length results in a small overestimate of time expended, since queues are actually shorter during the buildup transient. However, we also have assumed that identification and correction of failures are parallel processes. Correction, of course, always follows identification, so that there are intervals at the start and at the end of the test phase during which only one process is occurring. Thus our assumption results in a small underestimate of calendar time expended. It will be assumed that the two foregoing factors approximately cancel.

[12] In measuring backlog we do not count errors whose correction is deferred for reasons other than availability of personnel to fix them.

TABLE I
PARAMETERS OF RELIABILITY MODEL

Parameter	Source	Accuracy	Average Value (weighted) over Validating Projects	Predictions Affected
B	collected data	medium	0.96	T,τ,t
C	previous data - similar project	low	10.9	T,τ,t
f	computed from average instruction execution rate and program size	high		T,τ,t
K	previous data - similar project initially, reestimated during test	low to medium	1.31×10^{-6}	T,τ,t
N_o	computed from collected data and instructions to be coded estimate initially, reestimated during test	low to medium	error rate = 5.88 errors/K inst	T,τ,t
P_C	computer center	high		t
P_F	project plans	high		t
P_I	project plans	high		t
T_F	established with customer	"perfect"		τ,t
θ_C	collected data	high	1.11	t
θ_I	collected data	high	2.15	t
μ_C	collected data	high	1.67 hr	t
μ_F	collected data	high	6.93 hr	t
μ_I	collected data	high	5.28 hr	t
ρ_C	computer center	high	0.8	t
ρ_F	computed [for validating projects, assumed that no debugger should have backlog of 3 or more failures (with 90% probability]	medium		t

The computer utilization factor ρ_C is basically determined by the need to control turnaround time such that wait time is negligible (as previously mentioned). If turnaround time cannot be controlled (e.g., in the case of a small project using a general computation facility), then ρ_C should be set to its actual value and μ_F should be increased by the ratio of correction work time plus wait time to correction work time.

V. ESTIMATION OF MODEL PARAMETERS

A number of parameters have been defined in preceding sections that must be evaluated in order to specify the reliability model completely. The essential ones are listed in Table I, along with notes on how well they can be estimated and which prediction they affect. The average of parameter values used for the four validating projects is given where they have any possible significance for other projects. Presently, previous experience with at least one similar application in a similar development environment (this was generally the case with the projects described in this paper) is necessary to obtain good parameter esti-

mates. However, it appears probable that, as further data are taken, some of the parameters (most likely B, μ_C, μ_F, μ_I, θ_C, θ_I, and errors/inst) will prove to be relatively constant for a wide variety of projects and environments or perhaps dependent on a small number of characterizing variables. For example, the computer time/execution time ratio θ_C will probably be related to the proportion of input/output in the application.

The parameters μ_C, μ_F, μ_I, θ_C, and θ_I may be evaluated by collecting data on failure identification and correction work and computer usage profiles (with execution time) for a similar program development environment and fitting (28) to them, using a weighted least squares criterion.[13] The parameter B can be determined by taking data on the number of errors spawned while fixing other errors. Data taken by Miyamoto [3, p. 199] and his associates, interpreted in terms of this author's formulation of B, would indicate values of B from 0.91 to 0.95 over a number of cases of development of general purpose software systems. This compares with values from 0.94 to 1.00 over the four projects reported in this paper. Data can be developed on average error rates (errors per instruction) at the start of various phases of testing; these data will permit N_0 (and hence M_0) to be estimated. Data taken in this study and data taken by Akiyama [15] and Endres [16] give a range of 3.36 to 7.98 errors per thousand instructions for assembly language programs at the start of system test; the weighted (by number of instructions) mean is 5.43 errors/K inst.

The value of C must be obtained from measurements taken in a similar testing environment [see (52)] or estimated (if there is no basis for estimation, it is probably best to be conservative and take $C = 1$). The parameter K is initially determined from data for a similar program. It may be possible in the future to relate K to program structure in some way.

The parameters K and M_0 can be refined by reestimation as testing progresses. We will actually reestimate T_0 rather than K because of its greater physical significance. However, K may be readily determined by using (9).

We will use maximum likelihood estimation (see Appendix B for derivations) to make the reestimates. We find $\hat{M}_0$ implicitly from

$$\hat{\phi} = \phi(\hat{M}_0, m), \tag{40}$$

where $\hat{\phi}$ is the failure moment statistic

$$\hat{\phi} = \frac{1}{m\tau_m} \sum_{i=1}^{m} (i-1)\tau_i', \tag{41}$$

in which the τ_i' are the execution time intervals between failures and τ_m is the total execution time. Now

$$\phi(\hat{M}_0, m) = \frac{M_0}{m} - \frac{1}{\Delta\psi}, \tag{42}$$

where

$$\Delta\psi = \psi(M_0 + 1) - \psi(M_0 + 1 - m) \tag{43}$$

and ψ is the psi (digamma) function [17].

Now $\hat{T}_0$ is given by

$$\hat{T}_0 = C\bar{\tau}_m \left(1 - \frac{m}{M_0}\hat{\phi}\right), \tag{44}$$

where $\bar{\tau}_m$ is the sample mean of time to failure during test.

The variance of $\hat{\phi}$ is given by

$$\text{var}\,(\hat{\phi}) = -\frac{1}{(\Delta\psi)^2}\left[\frac{\Delta\psi'}{(\Delta\psi)^2} + \frac{1}{m}\right], \tag{45}$$

where

$$\Delta\psi' = \psi'(M_0 + 1) - \psi'(M_0 + 1 - m) \tag{46}$$

and ψ' is the trigamma function [17, p. 44]. Confidence intervals may be established for M_0 by determining the values of M_0 that correspond to values of ϕ chosen so that

$$\phi = \hat{\phi} \pm \left(\frac{1}{1 - \mathcal{P}}\right)^{1/2} S.D.(\hat{\phi}), \tag{47}$$

where

$$S.D.(\hat{\phi}) = [\text{var}\,(\hat{\phi})]^{1/2} \tag{48}$$

and $\mathcal{P}$ is the confidence level. The intervals are based on the application of Chebyshev's inequality. Although this inequality tends to produce conservatively large intervals, it is probably best to employ it until more experience has been gained with the reliability model of this paper.

The variance of $\hat{T}_0$ is given by

$$\text{var}\,(\hat{T}_0) = \frac{T_0^2}{m}. \tag{49}$$

In the case of the operational phase (after release to the user), if we assume that errors are not corrected, the hazard function $z(\tau)$ is constant, and use of the maximum likelihood method does not yield two independent equations in $\hat{M}_0$ and $\hat{T}_0$. However, the last estimate made of M_0 during the immediately previous system integration test phase may be used to yield an estimate of T_0. From (23) we have

$$T_0 = T\left(1 - \frac{m}{M_0}\right). \tag{50}$$

Now the MTTF at the end of the system integration test phase will be equal to the average failure interval $\bar{\tau}_{m'}$ during the use phase. Hence

$$\hat{T}_0 = \left(1 - \frac{m}{M_0}\right)\bar{\tau}_{m'}. \tag{51}$$

[13] Note that μ_F, μ_I, and θ_I are defined as gross figures that incorporate overhead such as vacations and absences, training, and adminstrative activities. Usually measurements made of the parameters will represent net times; they must be increased by a suitable overhead factor. A weighting factor equal to the square root of the number of failures that have occurred should be applied to each data point in the least squares fit to account for the greater confidence one has in the later data.

It is readily shown from (51) and the known mean and variance of the probability distribution of τ' that the coefficient of variation of $\hat{T}_0$ is $1/(m')^{1/2}$. Note that T is constant and equal to $\bar{\tau}_{m'}$.

If failures are corrected during any part of the operational phase, the period during which this occurs should be treated just like an extension of the test phase, with intervals between failures reduced by a factor of C.

Experience indicates that the quality of estimation can be considerably improved for small sample sizes by smoothing $\hat{\phi}$. On the other hand, the smoothing should eventually be removed because it impairs the responsiveness of the estimation algorithms as the number of remaining errors becomes small. After trying a number of different smoothers, a variable length smoother which builds up to a length of 40 samples at $m = 40$ and then drops to a length of 1 sample (i.e., no smoothing) at $m = 79$ was adopted. The smoother weights each value of $\hat{\phi}$ by the corresponding $m^{1/2}$.

It proved most convenient to allow one or two days to elapse after a test run before using the failure interval data, so that false failures could be weeded out and the results thoroughly analyzed to minimize missed failures. Experience indicates that, after analysis, one is more likely to miss failures than to report false ones.

Although our experience indicated that perhaps 95 percent or more of failures could be readily classified on examination as software or nonsoftware (hardware, operator, etc.), occasionally the choice became uncertain. The criterion that was followed was to classify the failure as nonsoftware if it could not be made to recur when the program was rerun with all software and with known nonsoftware inputs exactly the same.

Almost all failures could be readily associated with particular test times; those that could not, did relate to a fairly short time interval. The midpoint of the interval was taken as the time of failure in those cases.

If there are very few failures (e.g., 40 or less) during the test phase, the confidence intervals may be very broad due to the small sample size. Somewhat narrower intervals but a worst case prediction may be obtained by assuming that a failure occurs right at the end of testing.

The testing compression factor may be reestimated after the operational phase of a program from

$$C = \frac{M_0 - m}{M_0 - m\hat{\phi}} \frac{\tau_{m'}}{\bar{\tau}_m}, \tag{52}$$

which is obtained by equating (44) and (51).

Since the reliability model described in this paper has been applied to only four projects so far, there will probably be improvement in the determination of the values of the parameters of Table I as more data are collected. It therefore appears likely that the quality of initial estimation and reestimation can ultimately be substantially improved.

The behavior of M_0 and T_0 reestimates and the 75 percent confidence band as testing progressed for the first of the four projects is shown in Figs. 2 and 3. The first project was chosen for these plots and the ones to follow, because the larger number of failures experienced results in a larger range of sample sizes over which the behavior of the reestimates may be observed. The plots shown are representative of all the projects, although some variations in behavior occur. Note that for these and following plots, it is the behavior of the confidence band (and not the maximum likelihood estimate) that is most significant. It should also be noted that estimates based on 20 failures or less have not been shown; the sample sizes in these cases are too small to yield meaningful results.

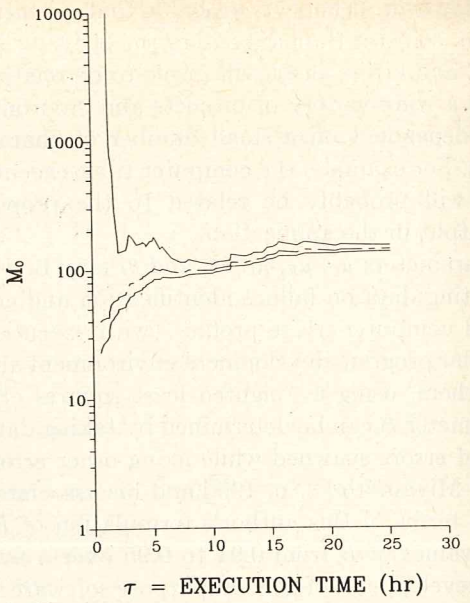

Fig. 2. Estimates of total failures M_0 for Project 1 (with 75 percent confidence band).

The increase in T_0 with execution time observed in Fig. 3 appears to be partly the result of system testing starting before system integration was complete and partly the result of testing not being well distributed across all areas of the program at each point in time. Note that the effect of starting test before the full program size is reached is to cause the linear execution frequency f to decrease as testing proceeds. Then, from (9), it will be seen that T_0 will increase if K and N_0 are constant. When a "well distributed" test phase was "constructed" by simulating cycling through the set of tests placed in randomized order (which also would "remove" the program size trend), the plot of T_0 versus τ became level.

VI. MONITORING TESTING PROGRESS

Since M_0 and T_0 are continually refined by reestimation during test, it is easy to continually reestimate:

1) Present MTTF T_P, using (22) with τ set equal to execution time used to date.

2) Remaining execution time $\Delta\tau$ required for the maximum likelihood estimate of T to reach an established

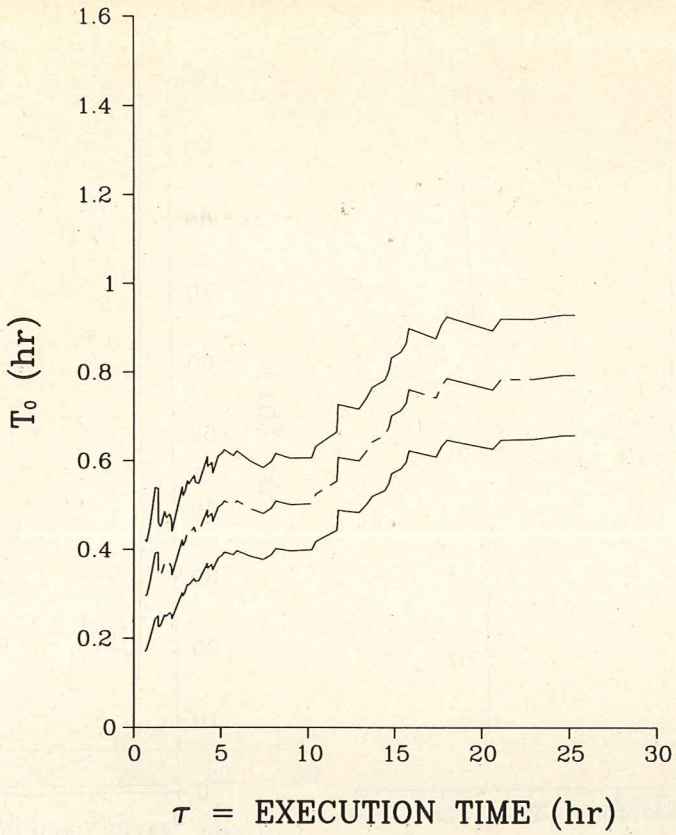

Fig. 3. Estimates of mean time to failure at start of testing T_0 for Project 1 (with 75 percent confidence band).

MTTF objective T_F, using (25) with $T_2 = T_F$ and $T_1 = T_P$.

3) Remaining calendar time Δt required to reach T_F, evaluating (32) over the interval $[T_P, T_F]$ by use of segment values taken from (33).

4) Remaining failures Δm required to be experienced and corrected to reach T_F, using (24) with $T_1 = T_P$ and $T_2 = T_F$.

Confidence limits may be established (in addition to the "most likely" values) by carrying out the computations for corresponding confidence limits of M_0 and T_0 (e.g., the confidence limits for $\Delta\tau$ are the remaining execution times required for the confidence bounds of T to reach T_F). The total set of results is useful to managers for planning schedules, estimating status and progress, and determining when to terminate testing. The 75 percent confidence level estimate was found from experience to be the most useful from the manager's viewpoint, although the 50 percent, 90 percent, and 95 percent confidence levels contribute some added insight.

If testing starts before integration is complete, then there may be periods of inactivity in testing and debugging due to waits for programs not yet ready. These periods must be estimated separately and added to estimates of remaining t or else t must not be viewed as "running" when they occur. Note that severe computer outages (one day or more) and design changes have also not been included in the estimate of remaining test phase length and must be added if expected.

Prediction of calendar time is fairly sensitive to the parameter P_F, the number of debuggers. It appears that the best estimate of P_F is given by the number of programmers available who were occupied full time in coding the various new or extensively modified units of the system under test.

A program is available that computes estimates of M_0, T_0, T_P, Δm, $\Delta\tau$, and Δt from a set of intervals between failures and certain required model parameters [18], [19].

An example of the continual reestimation of T_P, $\Delta\tau$, Δt, and Δm is shown in Figs. 4–7. The 75 percent confidence band is used. These reestimates were made for the first project. It should be noted that testing was terminated because of schedule constraints before the MTTF objective of 27.8 h was reached.

VII. COST AND RELIABILITY

The resource parameters defined in the derivation of the calendar time model pave the way for relating reliability requirements and cost, since each of the resources has an associated cost that is readily determined. An expression for cost as a function of reliability for the identification and correction of failures during the system test phase of a project is easily written. If program design, coding, unit testing, test planning, and test preparation

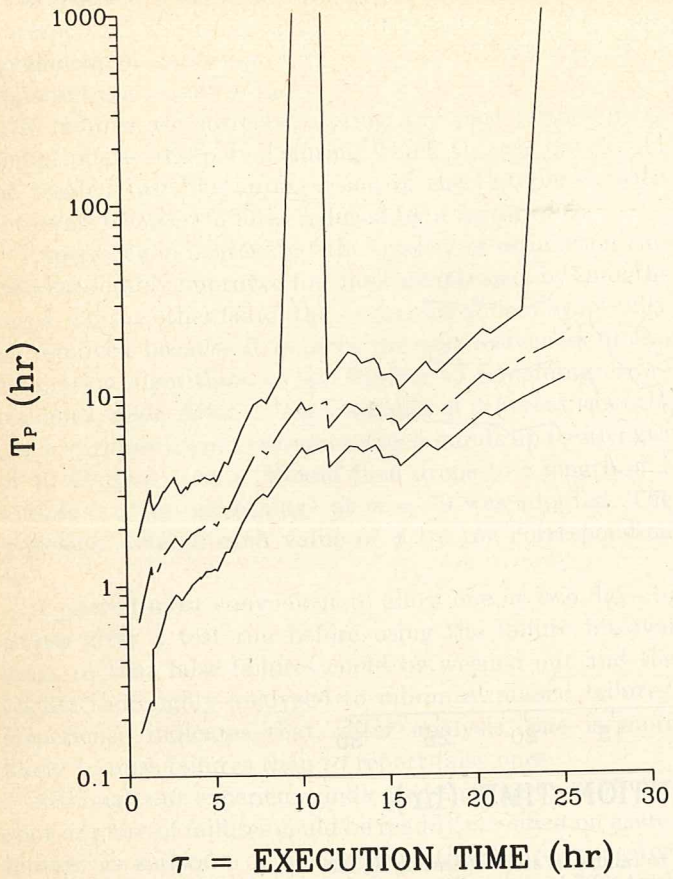

Fig. 4. Estimates of present mean time to failure for Project 1 (with 75 percent confidence band).

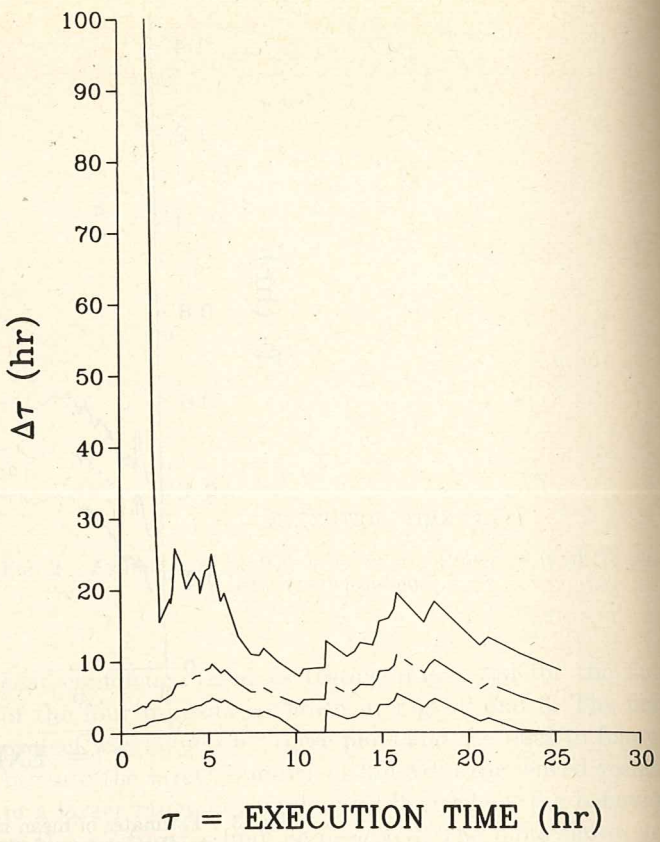

Fig. 5. Predictions of remaining execution time $\Delta\tau$ for Project 1 (with 75 percent confidence band).

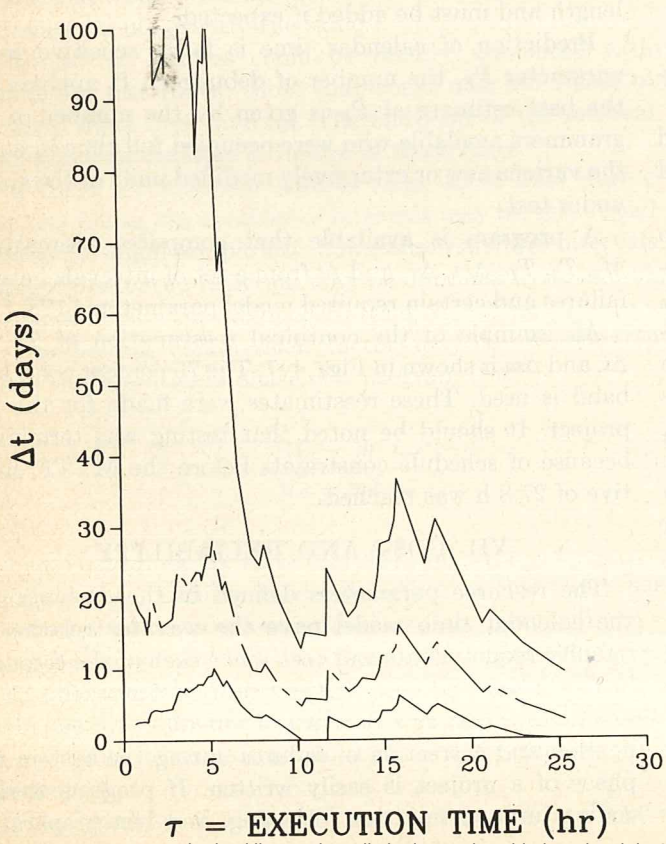

Fig. 6. Predictions of remaining calendar time Δt for Project 1 (with 75 percent confidence band).

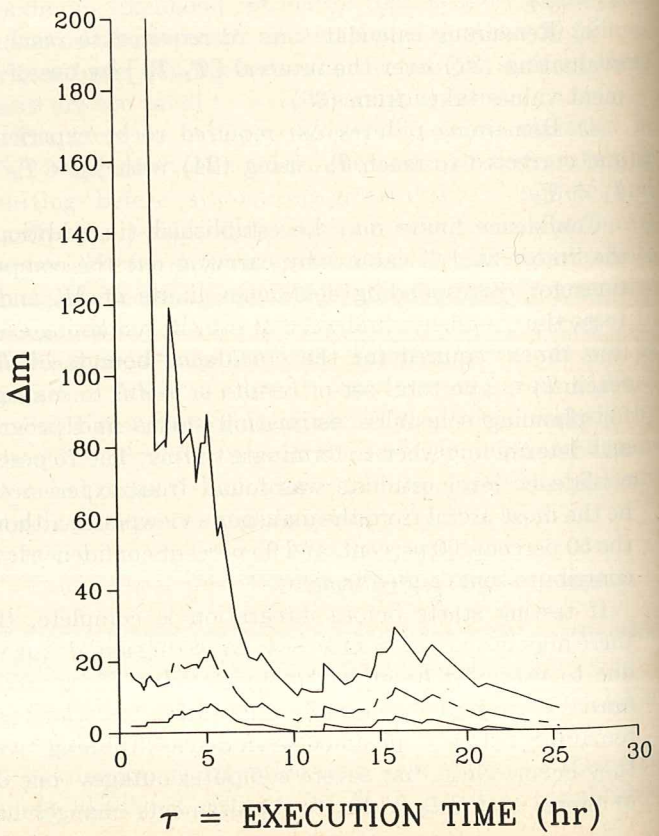

Fig. 7. Predictions of remaining failures Δm for Project 1 (with 75 percent confidence band).

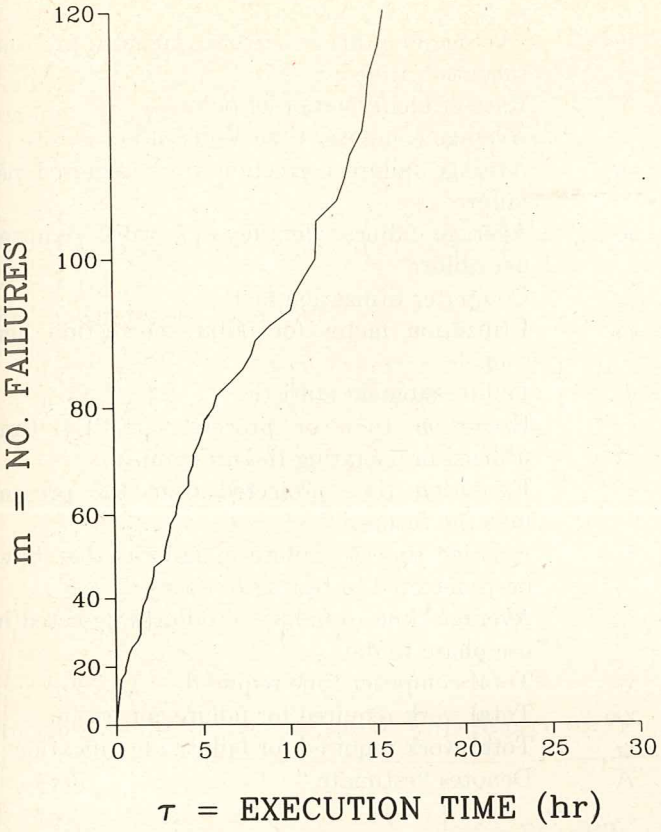

Fig. 8. Failures experienced m for Project 1.

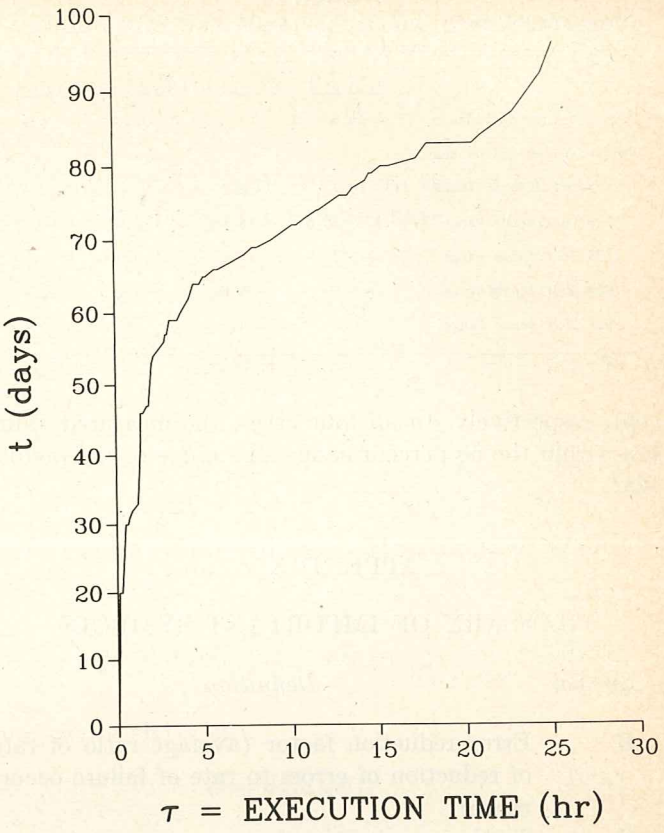

Fig. 9. Relationship between calendar time t and execution time τ for Project 1.

costs can be considered constant[14] then the foregoing function will completely determine the variation in cost with reliability objective. If a benefit function can be established that indicates the *value* of reliability for the specific application, then maximization of the benefit-cost difference can be used to set the reliability goal.

The effects on reliability of various program development techniques such as structured programming, better documentation, code reading, and better test design can be determined by first finding the effects of the techniques on the parameters C, M_0, T_0, μ_C, μ_F, μ_I, θ_C, and θ_I and then investigating the impact of these parameters on reliability. This two-stage approach appears to be desirable because the parameters can be more easily related to the techniques than reliability can. The effect on cost of the techniques can be computed by holding the reliability goal constant and subtracting the incremental cost of the techniques (such as additional program design, test planning, or documentation costs) from the incremental savings achieved in failure identification and correction.

VIII. VALIDATION OF MODEL

We will now look at some data that bear on the validity of the model. Fig. 8 shows the number of failures experienced plotted against execution time for the first project. The ordinate axis has been transformed by the function $-\ln(1 - (m/M_0))$ so that noise-free data that followed the execution time component of the model postulated in (20) would plot as a straight line. The estimate of M_0 used in the transformation was the one made with the largest sample (estimate at the end of test) and hence probably the most accurate one. The data follow the model quite well; results for the other three projects are similar.

Data taken on calendar time for the first project are plotted against execution time in Fig. 9. Note from (35) that at the start of testing ($\tau_{k_1} = 0$), t is ordinarily related exponentially to τ, since the number of failure correction personnel is usually limiting at this time and $\theta_F = 0$. There is normally a region of appreciable size in which $C\tau/M_0T_0 \ll 1$ and the exponential may be approximated by a straight line. When τ is large, computer time or the number of failure identification personnel is usually limiting and the relationship between t and τ approaches linearity. It will be noted that the plot in Fig. 9 approximates a straight line for low τ and for high τ, with a transition region in between. This is thus in good agreement with the predictions of the calendar time component of the model.

The foregoing two comparisons are sufficient to establish the validity of the model during the test phase, since all other quantities are derived.

The usefulness of the model in predicting MTTF to be expected during use of the program is indicated in Table II. Note that the predicted maximum likelihood values differ from the measured values by 31, 12, 19, and 34 per-

[14] In some cases this may be an oversimplification, since test planning and test preparation costs, in particular, may increase due to the increased testing required to meet a higher reliability objective. However, the cost function is readily modified.

TABLE II
COMPARISON OF MEASURED AND PREDICTED MTTF (HOURS)

	Project 1	Project 2	Project 3	Project 4
Measured (use period)	14.6	31.4	30.3	9.2
Predicted (at end of test)				
Max. Likelihood Value	19.1	35.2	24.4	12.3
50% Confidence Range	13.5 - 28.8	>19.8	>12.9	6.4 - 23.6
75% Confidence Range	>11.7	>16.0	> 9.5	4.8 - 33.3
90% Confidence Range	> 8.9	> 9.8	> 4.7	>2.4
95% Confidence Range	> 6.6	> 4.7	> 0.6	>0.6

cent, respectively. In all four cases, the measured value lies within the 50 percent confidence range of the prediction.

APPENDIX A

GLOSSARY OF IMPORTANT SYMBOLS

Symbol	*Definition*
B	Error reduction factor (average ratio of rate of reduction of errors to rate of failure occurrence).
C	Testing compression factor.
f	Linear execution frequency of program.
i	Failure index, i.e., sequential number of failure.
K	Error exposure ratio.
M_0	Number of failures required to expose and remove the inherent errors (N_0).
m	Failures experienced to date. Hence, also size of sample of failures used in estimating model parameters.
m_Q	Limit on number of failures any debugger may work on at one time (no one may equal or exceed this figure).
N_0	Number of inherent errors.
n	Net number of errors corrected.
$\mathfrak{N}$	Number of remaining errors.
P_C	Available computer shifts (measured in terms of prescribed work periods).
P_F	Number of available failure correction personnel.
P_I	Number of available failure identification personnel.
$\mathcal{O}_{m_Q}$	Probability that no debugger has m_Q or more failures to work on at any given time.
$R(\tau,\tau')$	Reliability function.
T	Mean time to failure (MTTF).
T_0	MTTF at start of testing.
T_F	Final (objective) MTTF.
T_P	Present MTTF.
t	Calendar time.
Δt	Calendar time increment related to an execution time increment.
$z(\tau)$	Hazard function or instantaneous failure rate.
θ_C	Average computer time expended per unit execution time.
θ_I	Average identification work expended per unit execution time.
λ	Rate of identification of failures.
μ_C	Average computer time required per failure.
μ_F	Average failure correction work required per failure.
μ_I	Average failure identification work required per failure.
ρ_C	Computer utilization factor.
ρ_F	Utilization factor for failure correction personnel.
$\hat{\phi}$	Failure moment statistic.
τ	Execution time or processor (CPU) time utilized in operating the program.
τ'	Execution time projected from the present into the future.
$\bar{\tau}_m$	Average time to failure of failures that have been detected in testing to date.
$\bar{\tau}_{m'}$	Average time to failure of failures detected in use phase to date.
χ_C	Total computer time required.
χ_F	Total work required for failure correction,
χ_I	Total work required for failure identification.
$\wedge$	Denotes "estimate."

APPENDIX B

DERIVATION OF MAXIMUM LIKELIHOOD ESTIMATES

A. Formulation of Likelihood Function

We will estimate M_0 and T_0 based on the execution time intervals between detections of the first m failures. Let these values be $\tau_1',\tau_2',\cdots,\tau_m'$. We will find the maximum likelihood estimators. The probability density function of failure during test $f_T(\tau,\tau')$ is given by

$$f_T(\tau,\tau') = R_T(\tau,\tau')z_T(\tau), \tag{B1}$$

where

$$z_T = Cz(\tau). \tag{B2}$$

Hence, using (11), we obtain

$$f_T(\tau,\tau') = Cz(\tau)\exp[-\tau' Cz(\tau)]. \tag{B3}$$

Note that τ' is exponentially distributed with mean $1/Cz(\tau)$ and variance $1/C^2z^2(\tau)$.

Since the failures are independent of each other, the likelihood function is given by

$$L(\tau_1',\cdots,\tau_m') = \prod_{i=1}^{m} f_T(\tau_{i-1},\tau_i')$$

$$= \prod_{i=1}^{m} Cz(\tau_{i-1})\exp[-\tau_i' Cz(\tau_{i-1})]. \tag{B4}$$

Now note that at τ_{i-1}

$$\mathfrak{N}(\tau_{i-1}) = N_0 - B(i-1) = B[M_0 - (i-1)]. \tag{B5}$$

Hence, using (2), (9), (18), and (B5), we have

$$z(\tau_{i-1}) = \frac{\mathfrak{R}(\tau_{i-1})}{BM_0T_0} = \frac{1}{T_0}\left[1 - \frac{i-1}{M_0}\right]. \quad \text{(B6)}$$

Therefore

$$L(\tau_1', \cdots, \tau_m')$$

$$= \prod_{i=1}^{m} \frac{C}{T_0}\left[1 - \frac{i-1}{M_0}\right] \exp\left[-\tau_i' \frac{C}{T_0}\left(1 - \frac{i-1}{M_0}\right)\right]. \quad \text{(B7)}$$

We will maximize the logarithm

$$\ln L = \sum_{i=1}^{m} \ln\left\{\frac{C}{T_0}\left[1 - \frac{i-1}{M_0}\right]\right\} - \sum_{i=1}^{m} \frac{C}{T_0}\left[1 - \frac{i-1}{M_0}\right]\tau_i'$$

$$= m \ln \frac{C}{T_0} + \sum_{i=1}^{m} \ln\left[1 - \frac{i-1}{M_0}\right]$$

$$- \frac{C}{T_0} \sum_{i=1}^{m} \left[1 - \frac{i-1}{M_0}\right]\tau_i'. \quad \text{(B8)}$$

B. Estimate of T_0

Now setting the partial derivative with respect to T_0 equal to zero, we have

$$\frac{\partial(\ln L)}{\partial T_0} = -\frac{m}{T_0} + \frac{C}{T_0^2} \sum_{i=1}^{m} \left[1 - \frac{i-1}{M_0}\right]\tau_i' = 0. \quad \text{(B9)}$$

Solving (B9) we obtain

$$\hat{T}_0 = \frac{C}{m} \sum_{i=1}^{m} \left[1 - \frac{i-1}{M_0}\right]\tau_i'. \quad \text{(B10)}$$

Equation (B10) may be written as

$$\hat{T}_0 = \frac{C}{m}\left\{\sum_{i=1}^{m} \tau_i' - \frac{1}{M_0} \sum_{i=1}^{m} (i-1)\tau_i'\right\}. \quad \text{(B11)}$$

Note that

$$\tau_m = \sum_{i=1}^{m} \tau_i'. \quad \text{(B12)}$$

Define a failure moment statistic $\hat{\phi}$ as

$$\hat{\phi} = \frac{1}{m\tau_m} \sum_{i=1}^{m} (i-1)\tau_i'. \quad \text{(B13)}$$

Since $0 \leq i - 1 < m$ for all i, it may be noted, using (B12), that $0 \leq \hat{\phi} < 1$. Now (B11) becomes

$$\hat{T}_0 = C\bar{\tau}_m\left[1 - \frac{m}{M_0}\hat{\phi}\right], \quad \text{(B14)}$$

where $\bar{\tau}_m$ is the sample MTTF measured during test, τ_m/m. It is readily shown from (B10) and (B26) that $\hat{T}_0$ is an unbiased estimator of T_0.

Note that the estimate of T_0 is dependent on an estimate of M_0. The quantity $1 - (m/M_0)\hat{\phi}$ may be viewed as a "correction factor" that is applied to the sample mean of failure intervals to account for the growth of these intervals with execution time τ.

C. Estimate of M_0

Forming the partial derivative with respect to M_0 of (B8) and setting it equal to zero, we have

$$\frac{\partial(\ln L)}{\partial M_0} = \sum_{i=1}^{m} \frac{1}{M_0 - (i-1)} - \frac{C}{M_0T_0} \sum_{i=1}^{m} \tau_i' = 0. \quad \text{(B15)}$$

Substituting (B12) and (B14) and letting $j = M_0 - (i-1)$ we obtain

$$\sum_{j=M_0-m+1}^{M_0} \frac{1}{j} = \frac{m}{M_0 - m\hat{\phi}}. \quad \text{(B16)}$$

The sum may be expressed in closed form using psi (digamma) functions [17]:

$$\psi(M_0 + 1) - \psi(M_0 + 1 - m) = \frac{m}{M_0 - m\hat{\phi}}. \quad \text{(B17)}$$

We may write (B17) as

$$\hat{\phi} = \phi(M_0, m), \quad \text{(B18)}$$

where

$$\phi(M_0, m) = \frac{M_0}{m} - \frac{1}{\psi(M_0 + 1) - \psi(M_0 + 1 - m)}. \quad \text{(B19)}$$

Note that (B18) yields an implicit estimate of M_0. We may consider $\hat{\phi}$ as an unbiased estimator of the function $\phi(M_0, m)$. Note that the right-hand term of (B19) is equal to $1/m$ times the harmonic mean of the "final" m failure numbers.

D. Accuracy of T_0 Estimate

In order to determine the accuracy of the estimate of T_0, note from (B10) that

$$\text{var}\,(\hat{T}_0) = \frac{C^2}{m^2} \sum_{i=1}^{m} \left[1 - \frac{i-1}{M_0}\right]^2 \text{var}\,(\tau_i'), \quad \text{(B20)}$$

since the τ_i' are independent of each other.

Since τ_i' is exponentially distributed, from (B3) and (B6) we obtain

$$\text{var}\,(\tau_i') = \frac{1}{C^2 z^2(\tau_{i-1})} = \frac{T_0^2}{C^2}\left[1 - \frac{i-1}{M_0}\right]^{-2}. \quad \text{(B21)}$$

Hence

$$\text{var}\,(\hat{T}_0) = \frac{T_0^2}{m}. \quad \text{(B22)}$$

The coefficient of variation of $\hat{T}_0$ is then $1/m^{1/2}$. For a sample of 25 failures, the error in estimating T_0 is 20 percent, a reasonable figure.

E. Accuracy of M_0 Estimate

We cannot readily and satisfactorily determine, in the general case, the accuracy of the M_0 estimate in percentage terms but we can establish confidence intervals for M_0.

Consider the variance of the statistic $\hat{\phi}$. Let $\hat{\phi} = x/y$, where

$$x = \sum_{i=1}^{m} (i-1)\tau_i' \tag{B23}$$

and

$$y = m\tau_m = m\sum_{i=1}^{m} \tau_i'. \tag{B24}$$

Now the variance of a ratio of random variables may be found from [20]:

$$\operatorname{var}\left(\frac{x}{y}\right) = \left[\frac{E(x)}{E(y)}\right]^2 \left[\frac{\operatorname{var}(x)}{E^2(x)} + \frac{\operatorname{var}(y)}{E^2(y)} - \frac{2\operatorname{cov}(x,y)}{E(x)E(y)}\right]. \tag{B25}$$

From (B3) and (B6) we have

$$E(\tau_i') = \frac{1}{Cz(\tau_{i-1})} = \frac{T_0}{C}\left[1 - \frac{i-1}{M_0}\right]^{-1}. \tag{B26}$$

If we use (B21) and (B26) and the fact that τ_i' and τ_j', $i \neq j$, are independent, we may write

$$E(x) = \sum_{i=1}^{m} (i-1)E(\tau_i') = \frac{M_0T_0}{C}\sum_{i=1}^{m} \frac{i-1}{M_0-(i-1)}, \tag{B27}$$

$$E(y) = m\sum_{i=1}^{m} E(\tau_i') = \frac{mM_0T_0}{C}\sum_{i=1}^{m} \frac{1}{M_0-(i-1)}, \tag{B28}$$

$$\operatorname{var}(x) = \sum_{i=1}^{m} (i-1)^2 \operatorname{var}(\tau_i') = \frac{M_0^2T_0^2}{C^2}\sum_{i=1}^{m} \frac{(i-1)^2}{[M_0-(i-1)]^2}, \tag{B29}$$

$$\operatorname{var}(y) = m^2\sum_{i=1}^{m} \operatorname{var}(\tau_i') = \frac{m^2M_0^2T_0^2}{C^2}\sum_{i=1}^{m} \frac{1}{[M_0-(i-1)]^2}, \tag{B30}$$

and

$$\operatorname{cov}(x,y) = m\sum_{i=1}^{m} (i-1)\operatorname{var}(\tau_i') = \frac{mM_0^2T_0^2}{C^2}\sum_{i=1}^{m} \frac{(i-1)}{[M_0-(i-1)]^2}. \tag{B31}$$

Make the change of variable $j = M_0 - (i-1)$. The sums may be expressed in terms of psi or digamma [17] and trigamma [17, p. 44] functions. Let

$$\Delta\psi = \psi(M_0+1) - \psi(M_0+1-m) \tag{B32}$$

and

$$\Delta\psi' = \psi'(M_0+1) - \psi'(M_0+1-m). \tag{B33}$$

We have

$$E(x) = \frac{M_0T_0}{C}\sum_{j=M_0+1-m}^{M_0} \frac{M_0-j}{j} = \frac{M_0T_0}{C}(M_0\Delta\psi - m), \tag{B34}$$

$$E(y) = \frac{mM_0T_0}{C}\sum_{j=M_0+1-m}^{M_0} \frac{1}{j} = \frac{mM_0T_0}{C}\Delta\psi, \tag{B35}$$

$$\operatorname{var}(x) = \frac{M_0^2T_0^2}{C^2}\sum_{j=M_0+1-m}^{M_0} \frac{(M_0-j)^2}{j^2} = \frac{M_0^2T_0^2}{C^2}(-M_0^2\Delta\psi' - 2M_0\Delta\psi + m), \tag{B36}$$

$$\operatorname{var}(y) = \frac{m^2M_0^2T_0^2}{C^2}\sum_{j=M_0+1-m}^{M_0} \frac{1}{j^2} = -\frac{m^2M_0^2T_0^2}{C^2}\Delta\psi', \tag{B37}$$

and

$$\operatorname{cov}(x,y) = \frac{mM_0^2T_0^2}{C^2}\sum_{j=M_0+1-m}^{M_0} \frac{M_0-j}{j^2} = \frac{mM_0^2T_0^2}{C^2}(-M_0\Delta\psi' - \Delta\psi). \tag{B38}$$

After substitution in (B25) and simplification, we obtain

$$\operatorname{var}(\hat{\phi}) = -\frac{1}{(\Delta\psi)^2}\left[\frac{\Delta\psi'}{(\Delta\psi)^2} + \frac{1}{m}\right]. \tag{B39}$$

An alternative form, obtained by differentiating (B19), is

$$\operatorname{var}(\hat{\phi}) = -\frac{1}{(\Delta\psi)^2}\frac{d\phi}{dM_0}. \tag{B40}$$

Applying Chebyshev's inequality, it can be shown that since

$$E(\hat{\phi}) = \hat{\phi}, \tag{B41}$$

$\hat{\phi}$ lies within k standard deviations of ϕ with probability greater than $1 - (1/k^2)$. Hence a confidence interval for M_0 can be established by determining the values of M_0 that correspond to values of ϕ chosen so that

$$\hat{\phi} = \phi_{\text{low}} + k\,S.D.(\hat{\phi}) \tag{B42}$$

and

$$\hat{\phi} = \phi_{\text{hi}} - k\,S.D.(\hat{\phi}), \tag{B43}$$

where

$$k = \left(\frac{1}{1 - \mathcal{P}}\right)^{1/2} \tag{B44}$$

and $\mathcal{P}$ is the confidence level.

ACKNOWLEDGMENT

The data used to validate the model were taken by R. S. Ball, J. D. Heacock, D. J. Huber, L. M. Nye and W. J. Wesner. D. J. Huber assisted in fitting some of the data. The author is indebted to S. J. Amster, A. S. Kamlet, M. L. Shooman, and W. J. Wesner for their helpful comments and suggestions.

REFERENCES

[1] B. Littlewood, "A reliability model for Markov structured software," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, pp. 204–207.
[2] Z. Jelinski and P. B. Moranda, "Software reliability research," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 465–484.
[3] I. Miyamoto, "Software reliability in online real time environment," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, p. 198.
[4] M. Shooman, "Probabilistic models for software reliability prediction," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 485–502; also in *1972 Int. Symp. Fault-Tolerant Computing*, Newton, Mass., June 21, 1972, pp. 211–215.
[5] ——, "Operational testing and software reliability estimation during program development," in *1973 IEEE Symp. Computer Software Reliability*, New York, N. Y., Apr. 30–May 2, 1973, pp. 51–57.
[6] N. F. Schneidewind, "An approach to software reliability prediction and quality control," in *1972 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 41. Montvale, N. J.: AFIPS Press, pp. 837–847.
[7] ——, "Methodology for software reliability prediction and quality control," NTIS Rep. AD 754377.
[8] ——, "Analysis of error processes in computer software," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, pp. 337–346.
[9] B. Littlewood and J. L. Verrall, "A Bayesian reliability growth model for computer software," in *1973 IEEE Symp. Computer Software Reliability*, New York, N. Y., Apr. 30–May 2, 1973, pp. 70–77.
[10] ——, "A Bayesian reliability growth model for computer software," *J. Roy. Statist. Soc.* (Series C, Applied Statistics), vol. 22, no. 3, pp. 332–346, 1973.
[11] R. E. Barlow and F. Proschan, *Mathematical Theory of Reliability*. New York: Wiley, 1965, p. 10.
[12] M. L. Shooman and M. I. Bolsky, "Types, distribution, and test and correction times for programming errors," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, pp. 350–351.
[13] A. K. Trivedi and M. L. Shooman, "A many-state Markov model for the estimation and prediction of computer software performance parameters," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, pp. 208–220.
[14] R. E. Machol, *System Engineering Handbook*. New York: McGraw-Hill, 1965, p. 28-21.
[15] F. Akiyama, "An example of software system debugging," in *Proc. 1971 IFIPS Conf.*, p. 359.
[16] A. Endres, "An analysis of errors and their causes in system programs," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21–23, 1975, pp. 328–329.
[17] H. T. Davis, *The Summation of Series*. San Antonio: Principia Press of Trinity Univ., 1962, p. 36.
[18] J. D. Musa, *Program for Estimating Software Reliability (User's Guide for PL/1 Version)*, available from IEEE Computer Society Repository.
[19] ——, *Program for Estimating Software Reliability (Program Documentation for PL/1 Version)*, available from IEEE Computer Society Repository.
[20] M. G. Kendall and A. Stuart, *The Advanced Theory of Statistics*, vol. 1. New York: Hafner, 1952, p. 232.

John D. Musa (M'65) received the B.A. and M.S. degrees summa cum laude in electrical engineering in 1954 and 1955, respectively, from Dartmouth College, Hanover, N. H., where he was a holder of General Electric and National Science Foundation Fellowships.

In 1958 he joined Bell Laboratories, Whippany, N. J., where he has held a variety of assignments in program design. He has also worked in the areas of analysis, simulation, systems engineering, and human factors engineering. He is currently Supervisor of the Data Management and Graphics Systems Group. Two of his particular interests are the improvement of program design techniques and the improvement of software project management.

Mr. Musa is a member of the IEEE Computer Society and Phi Beta Kappa.