1972

An approach to software reliability prediction and quality control

Schneidewind, Norman F.

An approach to software reliability prediction and quality control*

by NORMAN F. SCHNEIDEWIND

Naval Postgraduate School
Monterey, California

INTRODUCTION

The increase in importance of software in command and control and other complex systems has not been accompanied by commensurate progress in the development of analytical techniques for the measurement of software quality and the prediction of software reliability. This paper presents a rationale for implementing software reliability programs; defines software reliability; and describes some of the problems of performing software reliability analysis. A software reliability program is outlined and a methodology for reliability prediction and quality control is presented. The results of initial efforts to develop a software reliability methodology at the Naval Electronics Laboratory Center are reported.

RATIONALE OF SOFTWARE RELIABILITY

The purpose of a software reliability and quality control program is to provide a means for establishing quantitative criteria for the acceptance or rejection of software and to provide a method for predicting the reliability of software under operating conditions. A computer system consists of hardware, software and human operators. Within the software sub-system, there may exist a number of modules or programs. A total reliability analysis would address the reliability requirements of each major sub-system: hardware, software and operators and for each component within a sub-system. Within the hardware sub-system, reliability estimates should be provided for the central processing unit, discs, magnetic tapes, and other peripheral units. Within the software sub-system, reliability estimates should be provided for each module or program.

Relatively little work has been done in the areas of software and human operator reliabilities, despite the fact that these sub-systems are as important as hardware in determining total system reliability. This research effort is directed toward the goals of developing methodologies and programs for software reliability prediction and quality control.

SOFTWARE RELIABILITY PROGRAM

A description of the elements of a software reliability program follows.

Reliability specification

Reliability specifications are established in advance of software production in order to provide quantitative criteria for the acceptance or rejection of software products. Without such a specification there is no objective criteria on which to judge the quality of a program. Software reliability specifications would be determined from an analysis of total computer system reliability requirements. Individual program or module specifications would be determined by allocating to a program the reliability necessary to achieve the desired total computer system reliability, when all hardware, software and operator reliabilities are considered. Prevailing practice is to consider only hardware when establishing reliability specifications. The matter of establishing software reliability specifications has been largely ignored. The reliability program described here would make explicit provision for software reliability.

* This work was supported by the Computer Sciences Department of the Naval Electronics Laboratory Center under Project P509001. The opinions and assertions contained herein are the private ones of the writer and are not to be construed as official, or as reflecting the views of the Department of the Navy or the Naval service at large.

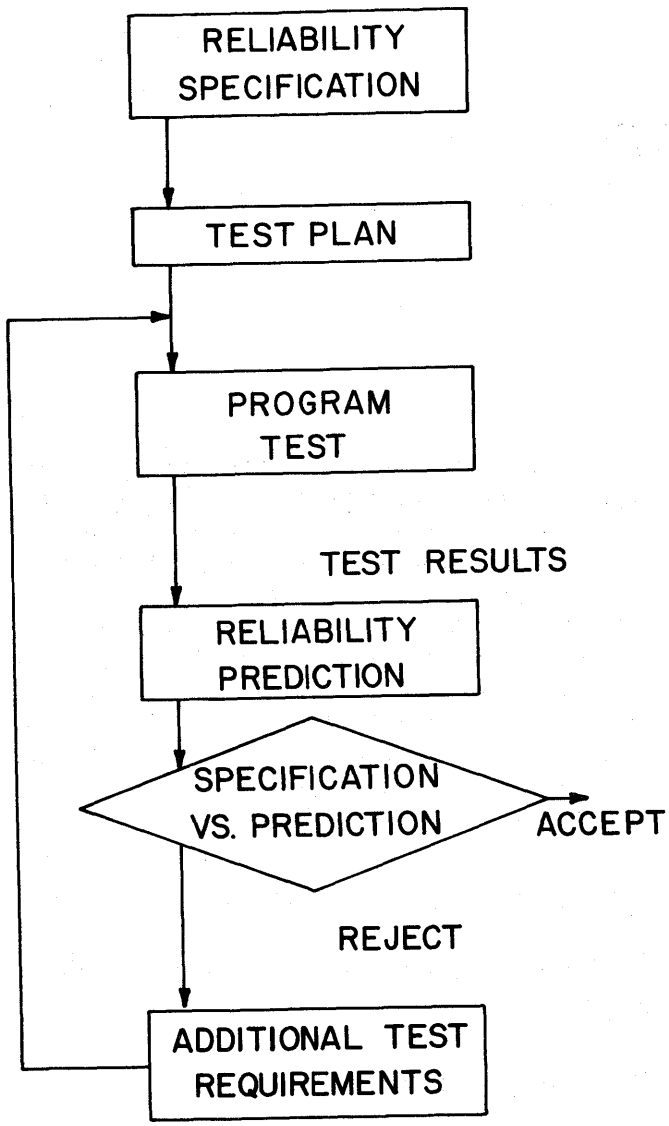

Figure 1—Reliability sequence

Reliability specifications are also used to establish initial test performance requirements in terms of test time and number of troubles. These requirements pertain to the formal test period which starts after the program has been released by the programmer and submitted for independent test. This period also constitutes the reliability demonstration period. During the formal test period, a program must operate for a period of time with less than a specified number of troubles. Satisfying this requirement constitutes meeting the reliability specification. Testing and computation of test requirements proceeds in stages. A stage is a test period during which an attempt is made to qualify a program. The number of stages is governed by the number of test periods which are required to demonstrate reliability.

Reliability prediction

The main purpose of reliability prediction is to provide an estimate of *future* probability of successful program operation. As in any reliability program, this estimate is based on historical and current test results. The reliability prediction is updated with new test data at the end of each stage of testing. In addition to prediction, reliability is used for quality control. Predicted reliability is compared with specified reliability. If a program does not qualify at a given stage, test requirements are computed for the next stage which, if satisfied, will result in the program satisfying the reliability specification. For example, assume on the basis of the reliability specification, that a program must operate for 100 hours during the first stage with no more than one trouble of a specified severity level in order to qualify. Assume that three troubles occur during the first stage; the program fails the first test. The test requirements for stage two which are necessary in order for the predicted reliability to equal or exceed the specified reliability would be determined. In this case, the additional requirement might be to operate another 200 hours without trouble. This process is repeated until the required reliability has been demonstrated. When a program does qualify, the final estimate of predicted reliability (applicable to actual operation) is computed. The process is summarized in Figure 1.

PROBLEMS IN SOFTWARE RELIABILITY ANALYSIS

There are many conceptual and definitional problems associated with software reliability. Some of these problems are described below.

Classification of troubles

Frequently the source of an error—whether it be hardware or software—cannot be definitely established. For example, a memory-to-memory transfer may produce incorrect data at the destination locations. Was this error due to a marginal memory unit or to a defective program? It may require days or weeks before the cause of the error can be positively identified. Another difficulty is that software errors can

result from either an operating system or applications programs. It may be difficult to establish whether:

1. The error was actually an error in the application program, or
2. was a violation of operating system protocol (Job Control Language), or
3. was the result of an error in the operating system itself.

Severity and type of trouble

Software troubles must be defined and classified in terms of severity and type. Software troubles differ as to their impact on system operation. A few incorrect characters in textual information which can still be deciphered is much less serious than a transfer outside the memory bounds of a program. Reliability predictions should specify the severity of troubles that are included in the prediction. A reliability predictor may involve one, more than one, or all severity levels. It may be appropriate to have a reliability predictor for each severity level and one which includes all severity levels.

Difference between test and operating environments

It may be difficult to completely duplicate the actual software operating environment during a test. The operating environment may comprise certain inputs, system load or operator actions which cannot be simulated during test. The inability under test conditions to fully duplicate the influence of inputs and stresses placed on the system by uncontrollable external inputs, operator performance, equipment reliability and equipment maintenance practice means that a reliability prediction is only an estimate of the actual reliability which will be obtained in operation. The accuracy of the reliability prediction will improve as the test environment approaches conditions of actual operation.

Adequacy of detail in trouble reporting

It is reasonable to assume that lower levels of program structure will provide greater accuracy of prediction than higher levels. For example, a subroutine may be critical with respect to the operation of a program. Trouble report data at the subroutine level may be more useful than data at the program level. However, in practice, software troubles may not be documented at the level which is most desirable for analysis purposes. Assuming an adequate sample size,

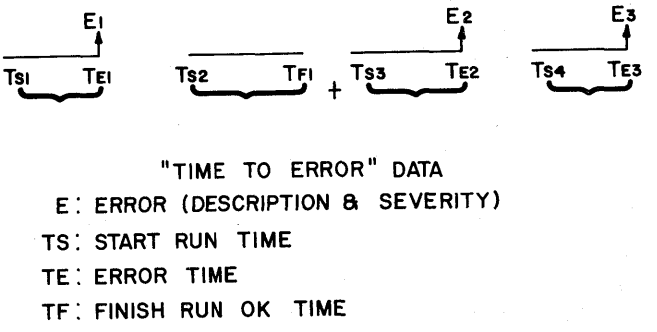

Figure 2—Data collection requirements

the smaller the unit of programming for which error information is obtained, the more accurate the reliability prediction, since many detailed reliability analyses can be combined for the purpose of system reliability estimation. However, a counterbalancing effect is that sample size, in terms of number of troubles, decreases as the unit of programming is decreased. Also, if the unit is too small, the number of program units which must be analyzed in order to compute total system reliability becomes excessive. In practice, the analyst seldom has a choice of levels of program error documentation. The problem is more one of uncovering *any* usable data!

Another problem is the absence of data which records the start time of each program test and the times at which trouble occurs, in order that the distribution of time between troubles can be determined. What is required is a time trace of program testing and trouble reporting such as that depicted in Figure 2.

Selection of test sample

Another problem is the possible nonrepresentativeness of trouble report data. The selection of program functions for testing should be proportional to the criticality of a function to mission success and also to the frequency of occurrence of the function during program operation. Frequency of program function testing based on the product of criticality and frequency of use in operation appears to be a reasonable basis for selecting functions for test. To the extent that testing does not occur in accordance with criterion, it will be nonrepresentative of the importance and frequency of use of functions in actual operation, thus causing a bias in reliability prediction. The problem posed by a nonrepresentative sample is difficult to counteract due to the following reasons:

- The selection of the sample is under the control of the test group—not the reliability analyst.

- It may be difficult to obtain information regarding the criticality and frequency of occurrence of program functions.
- Attempts at selecting a representative sample from the available test report data may be infeasible due to existing small samples.

A partial solution to these problems can be obtained by proper design of test procedures or by changes to existing procedures. Software troubles can be classified in detail, test environments can be made realistic to the limit of economic feasibility and representative program functions can be selected for test. The most important and least expensive improvement would be to make test reporting responsive to the needs of reliability analysis. This involves reporting software troubles at least down to the level of the program entity used for reliability prediction and of providing time histories of program troubles. However, even if administrative procedures are changed, the problem of identifying the source of a trouble as to hardware, software or human action will remain. This situation illustrates the need for extensive documentation of program troubles at the time of their occurrence.

DEFINITIONS

Software trouble

In order to provide a method for predicting software reliability, it is first necessary to define software errors or troubles. The following definition will be utilized:

> *A software trouble is any logical or clerical error made by a programmer in creating or coding an algorithm which causes the algorithm to produce an incorrect result when the algorithm is presented with a correct input.*

The above definition excludes errors due to hardware, input or operator action. In addition, it will be understood to exclude compilation errors and errors caused by the operating system. In short, the errors considered in this study are application program errors.

Software reliability

Software reliability $R(t)$ is the probability that a program will operate without a single occurrence of a specified severity of trouble, or worse, for a specified length of time t, and with a specified input load. This is equivalent to the probability that a program will operate successfully for at least time t.

Software probability density function

A probability law $f(t)$ which governs the occurrence of troubles in the operating time domain (distribution of time between troubles). $f(t)dt$ is the probability of trouble in the interval dt. It is the time rate of change of probability of trouble.

Hazard rate

The instantaneous trouble rate $z(t)$. $z(t)$ is the time rate of change of probability of trouble, given that there has been no trouble in the time interval 0 to t. Thus, this conditional time rate of change of trouble is given by $z(t)=f(t)/R(t)$.

RELIABILITY PREDICTION AND QUALITY CONTROL

Background

Much of reliability theory is based on probability concepts which are independent of the physical form or characteristics of systems or devices. Since hardware has a long history relative to software, it is natural that almost all reliability literature and experience is based on the application of theory to hardware. Although certain modifications are necessary, it appears that important elements of reliability theory can validly be applied to software.

The classical model of reliability as applied to hardware involves three distinct periods in the life of equipment. During the burn-in period, when major bugs in equipment are identified and corrected or marginal components are forced to fail, equipment experiences a decreasing hazard rate. During this period the hazard rate is a function of the equipment operating time. In this period, the occurrence of failures *is dependent* on the failure history. If failures occur and are corrected prior to time t, this will have the effect of reducing the hazard rate at time t.

According to the classical model, following the burn-in period, failures are assumed to occur at a constant rate. This means that the occurrence of failures is *independent* of the age or operating time of the equipment. The expected number of failures during a given time interval is the same regardless of when the time interval starts, provided the equipment is operating within the constant hazard rate region. Failures within this region are said to occur suddenly or by chance, for example, when operating requirements or environmental requirements exceed the capabilities of the equipment. Another way of viewing

the constant hazard rate region is that there is no preferred time of failure or time about which failures cluster.

The third region occurs when equipment is subject to rapid deterioration and wearout. This is the region of increasing hazard rate. During this period, the hazard rate is a function of operating time; hazard rate increases with time.

It has been found that certain probability density functions are appropriate for representing the distribution of time between failures or time to failure during the three regions of equipment life. For example, the two parameter (amplitude and shape) Weibull probability density function $f(t)=\alpha\beta t^{\beta-1}e^{-\alpha t^{\beta}}$ has a decreasing hazard rate function $z(t)=\alpha\beta t^{\beta-1}$, when $\beta<1$, where t is the time to failure or time between failures, α is an amplitude parameter and β is a shape parameter. The probability density and hazard rate functions can, in certain situations, be employed to represent the distribution of time between failures and the hazard rate, respectively, during the burn-in period.

During the operational, or constant hazard rate period, the time between failures is exponentially distributed. The exponential distribution corresponds to a Weibull distribution with $\beta=1$. Then, the probability density function and hazard rate are given by $f(t)=\alpha e^{-\alpha t}$ and $z=\alpha$, where α is the constant hazard rate and $1/\alpha$ is the mean time between failures.

Finally, during the wearout stage, when the hazard rate is increasing, a Weibull distribution with $\beta>1$ may be the appropriate distribution for representing time between failures. The log-normal and gamma (with appropriate choice of parameters) are other functions with an increasing hazard rate which may also be appropriate for this phase.

Application of Reliability Theory to Software

When applied to software reliability, many of the basic concepts and definitions of reliability theory remain intact. Among these are the following:

- Definition of reliability $R(t)$ as the *probability* of successful program operation for at least t hours
- Probability density function $f(t)$ of time between software troubles, or, equivalently, the time rate of change of the probability of trouble
- Hazard rate $z(t)$ as the instantaneous trouble rate, or, equivalently, the time rate of change of the conditional probability of trouble (time rate of change of probability of trouble, given that no trouble has occurred prior to time t)

There are also major differences between hardware and software reliability. These are listed below:

- Stresses and wear do not accumulate in software from one operating period to another as in the case of certain equipment; however, program quality may be different at the start of each run, for the reason given below.
- In the case of hardware, it is usually assumed that between the burn-in and wearout stages an exponential distribution (which means a constant hazard rate) applies and that the probability of failure in a time interval t is independent of equipment age. However, for software, there may be a difference in the initial "state of quality" between operating periods due to the correction of errors in a previous run or the introduction of new errors as the result of correcting other errors. Thus it is appropriate to employ a reliability growth model which would provide a reliability prediction at several points in the cumulative operating time domain of a program.
- For equipment, age is used as the variable for reliability prediction when the equipment has reached the wearout stage. Since with software, the concern is with running a program repeatedly for short operating times, the time variable which is used for reliability purposes is the time between troubles. However, cumulative operating time is a variable of importance for predicting the timing and magnitude of shifts in the reliability function as a result of the continuing elimination of bugs or program modification.

Over long periods of calendar or test time, there will

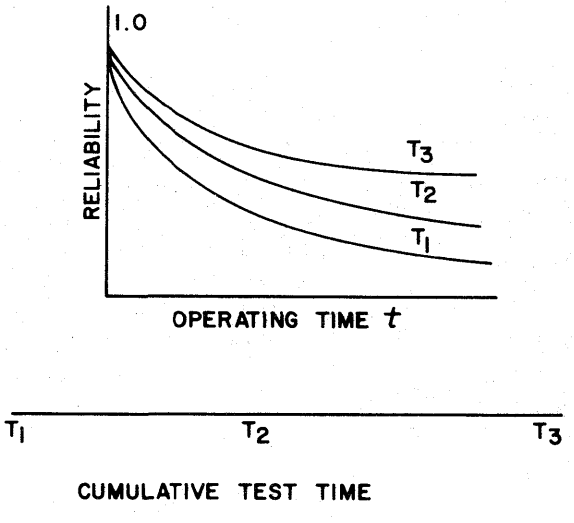

Figure 3—Reliability growth

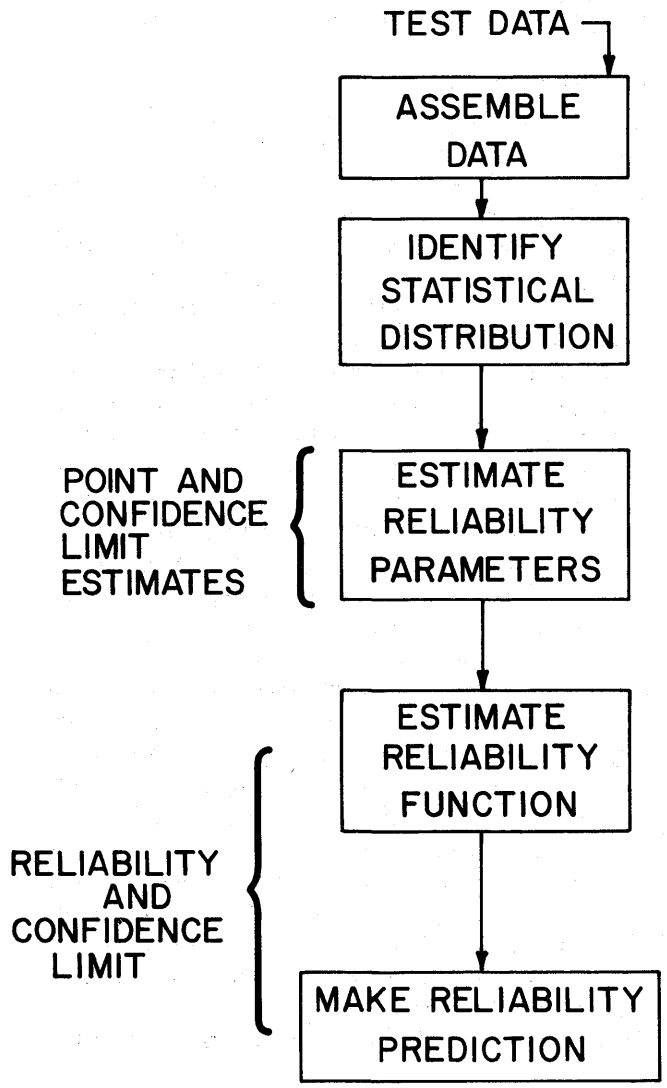

Figure 4—Steps in reliability prediction

be shifts in the error occurrence process such that different hazard rate and probability density functions are applicable to different periods of time; or, the same hazard and probability density functions may apply but the parameter values of these functions have changed. This shift is depicted in Figure 3, where the reliability function, which is a decreasing function of *operating time* is shown shifted upward at various points in *cumulative test time*, reflecting long-term reductions in the trouble rate and an increase in the time between troubles.

Approach

The steps which are involved in one approach to software reliability prediction are shown in Figure 4.

Step 1. Assemble Data

Data must first be assembled in the form of a time between troubles distribution as was indicated in Figure 2. At this point, troubles are also classified by type and severity.

Step 2. Identify Statistical Distribution

In order to identify the type of reliability function which may be appropriate, both the empirical relative frequency function of time between troubles (see example in Figure 5) and the empirical hazard function are examined. The shapes of these functions provide qualitative clues as to the type of reliability function which may be appropriate. For example:

- A monotonically decreasing $f(t)$ and a constant $z(t)$ suggest an exponential function.
- An $f(t)$ which has a maximum at other than $t = 0$ and a $z(t)$ which monotonically increases suggests:
  - —Normal function or
  - —Gamma function or
  - —Weibull function with $\beta > 1$.
- A monotonically decreasing $f(t)$ and a monotonically decreasing $z(t)$ suggest a Weibull function with $\beta < 1$.

After some idea is obtained concerning the possible distributions which may apply, point estimates of the parameters of these distributions are obtained from the sample data. This step is necessary in order to perform goodness of fit tests and to provide parameter

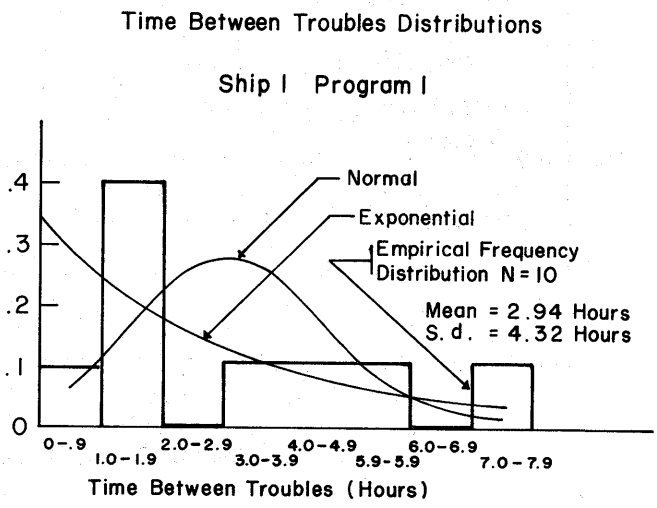

Figure 5—Probability density functions

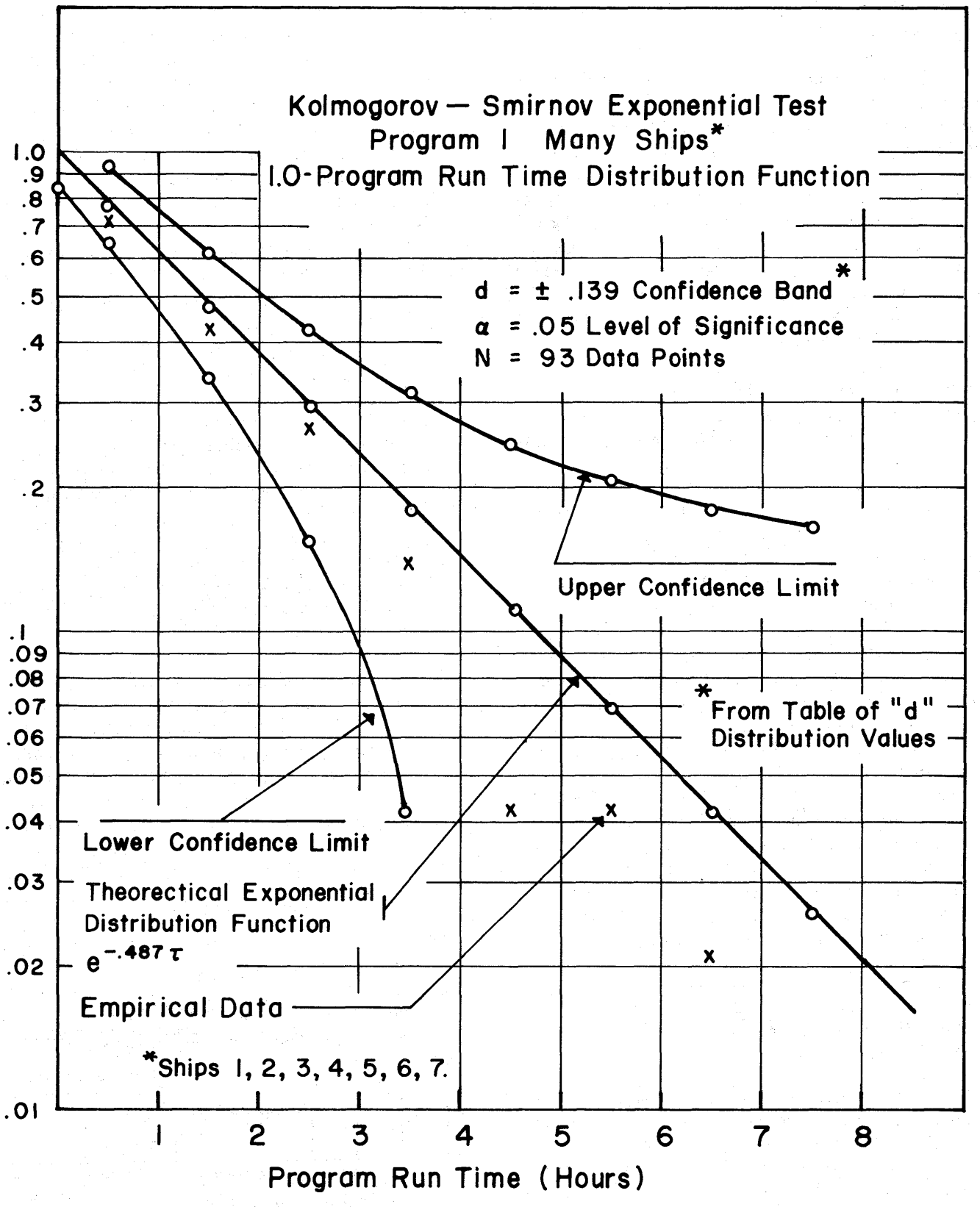

Figure 6—Goodness of fit test

estimates for the reliability function. In order to make a goodness of fit test, it is necessary to provide an estimate of the theoretical function to be used in the test. This is obtained by making point estimates of the applicable parameters. In the case of the one parameter exponential distribution, the point estimate would simply involve computing the mean time between troubles = total cumulative test time/number of troubles, which is the maximum likelihood estimator of the parameter T in the exponential probability density function $f(t) = 1/Te^{-t/T}$.

In the case of a multiple parameter distribution, the process is more involved. For the Weibull distribution, the following steps are required to obtain parameter point estimates:

- A logarithmic transformation of the hazard function is performed in order to obtain a linear function from which initial parameter values can be obtained.
- The initial values are used in the maximum likelihood estimating equations in order to obtain parameter point estimates.
- The probability density, reliability and hazard functions are computed using the estimated parameter values.

At this point, a goodness of fitness test can be performed between the theoretical probability density and the empirical relative frequency function or between the theoretical and empirical reliability functions. The Kolmogorov-Smirnov (K-S) or Chi Square tests can be employed for this purpose. An example of using the K-S test is shown graphically in Figure 6. This curve shows a test with respect to an exponential reliability function. Shown are the upper and lower confidence limits, the theoretical function and the empirical data points. Since the empirical points fall within the confidence band, it is concluded that the exponential is not an unreasonable function to employ in this case.

Step 3. Estimate Reliability Parameters Confidence Limits

The point estimate of a reliability parameter provides the best single estimate of the true population parameter value. However, since this estimate will, in general, differ from the population parameter value due to sampling and observational errors, it is appropriate to provide an interval estimate within which the population parameter will be presumed to lie. Since only the lower confidence limit of the reliability function is of interest, one-sided confidence limits of the parameters are computed. In Figure 7 is shown an example of the results of the foregoing procedure, wherein, for an exponential distribution, the point estimate of mean time between troubles (MTBT) is 2.94 hours (hazard rate of .34 troubles per hour) and the lower confidence limit estimate of MTBT is 2.27 hours (hazard rate of .44 troubles per hour). The lower confidence limit of MTBT for an exponential distribution is computed from the expression $T_i = 2n\bar{t}/\chi^2_{2n,1-\alpha}$ where T: is the lower confidence limit of MTBT, n is number of troubles, $\bar{t}$ is the MTBT (estimated from sample data), χ^2 is a Chi-Square value and α is the level of significance.

Step 4. Extimate Reliability Function

With point and confidence limit estimates of parameters available, the corresponding reliability functions can be estimated. The point and lower limit parameter estimates provide the estimated reliability functions $R = e^{-.34t}$ and $R = e^{-.44t}$, respectively, in Figure 7. In this example, the predicted reliabilities pertain to the occurrence of all categories of software trouble, i.e., the probability of no software troubles of any type occurring within the operating time of t hours.

Step 5. Make Reliability Prediction

With estimates of the reliability function available, the reliability for various operating time intervals can be predicted. The predicted reliability is then compared with the specified reliability. In Figure 7, the predicted reliability is less than the specified reliability (reliability objective) throughout the operating time range of the program. In this situation, testing must be continued until a point estimate of MTBT of 5.88 hours (.017 troubles per hour hazard rate) and a lower confidence limit estimate of MTBT of 4.55 hours (.022 troubles per hour hazard rate) are obtained. This result would shift the lower confidence limit of the predicted reliability function above the reliability objective.

Estimating test requirements

For the purpose of estimating test requirements in terms of test time and allowable number of troubles, curves such as those shown in Figure 8 are useful. This set of curves, applicable only to the exponential reliability function, would be used to obtain pairs of (test

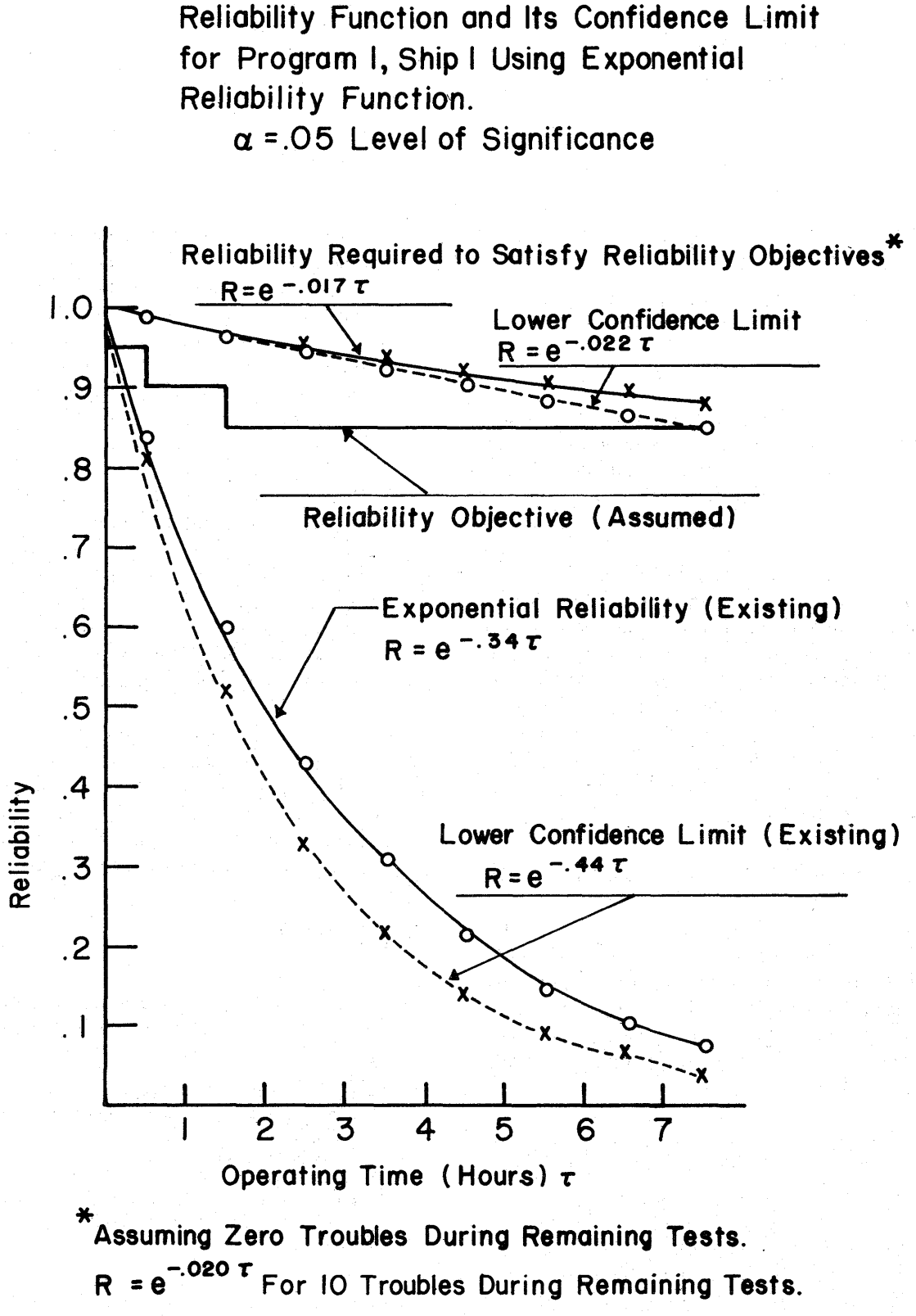

Figure 7—Reliability prediction

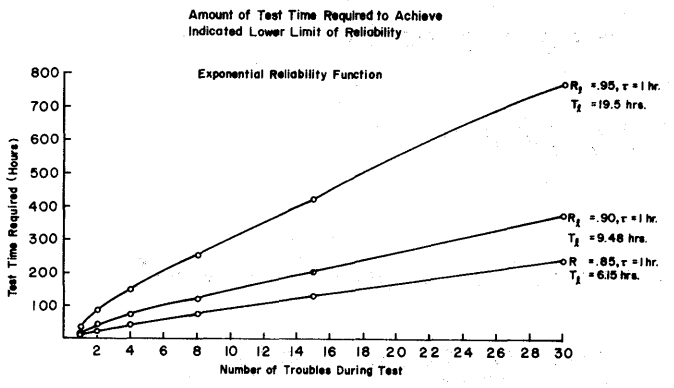

Figure 8—Test requirements

time, number of troubles) values. The satisfaction during testing of one pair of values is equivalent to satisfying the specified lower limit of reliability R_l for t hours of operation. For example, if a program reliability specification calls for a lower reliability confidence limit of .95 after 1 hour of *operating time*, this requirement would be satisfied by a *cumulative test time* of 100 hours and no more than 2 troubles; a cumulative test time of 200 hours and no more than 6 troubles; a cumulative test time of 300 hours and no more than 10 troubles, etc. The required test time is estimated from the relationship $T = [t\chi^2_{2n,1-\alpha}/2Ln(1/R_l)]$, where T is required test time, t is required operating time, χ^2 is a Chi Square value, n is number of troubles, R_l is the required lower limit of reliability and α is level of significance.

PRELIMINARY RESULTS AND CONCLUSIONS

A Naval Electronics Laboratory Center (NELC) sponsored study* was performed, employing the concepts and techniques described in this report, on Naval Tactical Data System (NTDS) data. The data utilized involved 19 programs, 12 ships and 325 software trouble reports. The major preliminary results and conclusions follow:

1. On the basis of Analysis of Variance tests, it was found that NTDS programs are heterogeneous with respect to reliability characteristics. There was greater variation of reliability between programs than within programs. This result suggests that program and programmer characteristics (source of between program variation) is more important in determining program reliability than is the stage of program checkout or cumulative test time utilized (source of within program variation). This result indicates a potential for obtaining a better understanding of the determinants of software reliability by statistically correlating program and programmer characteristics with measures of program reliability.
2. Goodness of fit tests indicated much variation among programs in the type of reliability function which would be applicable for predicting reliability. This result and the Analysis of Variance results suggest that program reliability should be predicted on an individual program basis and that it is not appropriate to merge sets of trouble report data from different programs in order to increase sample size for reliability prediction purposes.
3. Based on its application to NTDS data, the approach for reliability prediction and quality control which has been described appears feasible. However, the methodology must be validated against other test and operational data. Several interactive programs, written in the BASIC language, which utilize this approach, have been programmed at NELC*.

Another model by Jelenski and Moranda* has been developed and validated against NTDS and NASA data. Other approaches, such as reliability growth models, multiple correlation and regression studies and utilization of data smoothing techniques will be undertaken as part of a continuing research program.

BIBLIOGRAPHY

1 R M BALZER
EXDAMS—extendable debugging and monitoring system
AFIPS Conference Proceedings Vol 34 Spring 1969
pp 567–580
2 W J CODY
Performance testing of function subroutines
AFIPS Conference Proceedings Vol 34 Spring 1969
pp 759–763
3 J C DICKSON et al
Quantitative analysis of software reliability
Proceedings—Annual Reliability and Maintainability Symposium San Francisco California 25–27 January 1972
pp 148–157

* N. F. Schneidewind, "A Methodology for Software Reliability Prediction and Quality Control," *Naval Postgraduate School*, Report No. NPS55SS72032B, March 1972.

* Programmed by Mr. Craig Becker of the Naval Electronics Laboratory Center.

* Jelenski, Z. and Moranda, P. B., "Software Reliability Research," McDonnell Douglas Astronautics Company Paper WD1808, Navember 1971.

4 BERNARD ELSPOS et al
Software reliability
Computer January–February 1971 pp 21–27
5 ARNOLD F GOODMAN
The interface of computer science and statistics
Naval Research Logistics Quarterly Vol 18 No 2 1971 pp 215–229
6 K U HANFORD
Automatic generation of test cases
IBM Systems Journal Vol 9 No 4 1970 pp 242–256
7 Z JELINSKI P B MORANDA
Software reliability research
McDonnell Douglas Astronautics Company Paper WD 1808 November 1971
8 JAMES C KING
Proving programs to be correct
IEEE Transactions on Computers Vol C-20 No 11 November 1971 pp 1331–1336
9 HENRY C LUCAS
Performance evaluation and monitoring
Computing Surveys Vol 3 No 3 September 1971 pp 79–91
10 R B MULOCK
Software reliability engineering
Proceedings—Annual Reliability and Maintainability Symposium San Francisco California 25–27 January 1972 pp 586–593
11 R J RUBEY R F HARTWICK
Quantitative measurement of program quality
Proceedings—1968 ACM National Conference pp 672–677
12 N F SCHNEIDEWIND
A methodology for software reliability prediction and quality control
Naval Postgraduate School Report No NPS55SS72032B March 1972