Software reliability measures applied to system engineering

by JOHN D. MUSA
Bell Laboratories
Whippany, New Jersey

INTRODUCTION

Boehm, Brown, and Lipow have characterized the multidimensional nature of software quality in terms of a hierarchy of attributes. One of the high-level attributes is reliability, which they define qualitatively as the satisfactory performance of intended functions. This definition may be refined to the quantitative statement “probability of failure-free operation in a specified environment for a specified time.” A “failure” is an unacceptable departure of program operation from program requirements, where, as in the case of hardware, “unacceptable” must ultimately be defined by the user. The term “fault” will be used to indicate the program defect that causes the failure.

Several trends have recently combined to escalate the importance of quantitative software reliability measures:

1. The large and growing number of real-time and interactive systems has increased the operational and cost impacts of failures.
2. The increasing number, size, and complexity of computer networks and distributed processing systems have multiplied the risk and effects of failure.
3. The explosive growth of personal computing has created a demand for relatively foolproof software for unsophisticated users.

Measurement is seen to be important as soon as one recognizes that in software as in hardware there can be too much as well as too little reliability. Improvement of reliability, of course, costs money, and usually impacts development schedules and system performance (in the case of software, through increased memory, processing time, and peripherals requirements). The system engineer and the manager have to make design tradeoffs among the foregoing factors and it is best that this be done in quantitative terms. The need for a quantitative reliability measure continues throughout the development process, particularly during test, since reliability is a valuable indicator of system status. Finally, reliability or mean-time-to-failure (MTTF) is a useful metric for characterizing system operation and for controlling change during the maintenance phase. This paper will focus on the system engineering application, but it will also touch on monitoring the system test phase and controlling change during maintenance.

EXECUTION TIME THEORY OF SOFTWARE RELIABILITY

Software failures are caused by design or coding faults, while the hardware failures dealt with by hardware reliability theory are caused by physical deterioration. However, software and hardware reliabilities are mathematically very similar. Thus they may be manipulated in similar fashion and they may be combined to yield system reliability.

The basic concept of the execution time theory is that execution (processor or CPU) time is the best practical measure for characterizing the failure-inducing stress placed on software. Execution time and calendar time can be related because the relationship between the two is characteristically paced by one of the resources: failure identification (test team) personnel, failure correction (debugging) personnel, or computer time.

The execution time theory is based on assumptions that appear to be satisfactorily met by most executable programs and most development projects, assuming that testing is representative of the operational environment and that failures are observed.

A number of fundamental equations relating failures experienced, present MTTF, cumulative execution time, objective MTTF, failures to be experienced to reach the MTTF objective established for the project, and execution and calendar times required to meet the objective have been derived and summarized.

The equations mentioned above depend on four classes of parameters (described in detail in Reference 3, Section VI): program, planned, debug environment, and test environment. The two program parameters, the total failures expected during the maintained life of the software and the mean time to failure at the start of test, can be statistically reestimated (Reference 3, p. 436) as testing progresses.

A portable FORTRAN program has been developed to re-estimate the program parameters and compute status quantities significant to the manager. The program requires as input the execution time intervals between failures experienced, the MTTF objective, and the test environment.
parameter. The debug environment and planned parameters are required only if it is desired to predict the test completion date.

The program computes confidence intervals for the quantities it estimates. The 75 percent confidence interval has been found to be the most useful; it represents a good compromise between high confidence and a narrow range of estimation. A sample report generated for an actual project is shown in Figure 1. The lower and upper confidence bounds are sandwiched around the "most likely" (maximum likelihood) estimates. Note that "999999" indicates "no upper limit." For the project illustrated on 10/6/77, the most likely present MTTF is 66.7 hr and the 75 percent confidence interval for present MTTF is 63.3 hr or greater. The most likely date on which the 5000 hr MTTF objective will be reached is 11/22/77 and the 75 percent confidence interval for reaching the objective is sometime between now (10/6/77) and 12/23/77.

System Engineering

The use of the execution time theory in system engineering can be illustrated by looking at the tradeoffs that can be made between MTTF, cost, and schedules. These tradeoffs are made for the system test phase of the project. It is assumed that MTTF improvement is obtained by more extensive testing, which of course affects costs and schedules. Costs and schedules for other phases are assumed to be constant. This assumption is reasonable: reliability improvement techniques such as structured programming, design reviews, etc. are commonly implemented on a "yes-no" basis dependent on their cost effectiveness. One does not ordinarily trade off the degree to which structured programming is employed with MTTF.

The procedures and formulas used for computing system test duration and cost will be described, since these two computations are central to the system studies to which the execution time theory of software reliability can be applied. An example of a system study will then be presented to suggest some of the kinds of questions that can be answered.

**System Test Duration**

Estimates of system test duration (excluding the test planning effort) may be made before testing begins as follows. The total inherent faults $N_o$ is estimated from data on faults per source instruction at the start of system test. The total expected failures $M_o$ may be equated to $N_o$ unless the ratio of faults corrected to failures detected departs appreciably from 1 (in this case, a correction must be made—see Reference 3, pp. 447-8). Data taken by the author and by Akiyama and Endres give a range of 3.36 to 7.98 faults per thousand source instructions for assembly language programs at the start of system test; the weighted (by number of instructions) mean is 5.43 faults per thousand instructions. It is likely that these numbers are also applicable to higher-order languages, although the author does not currently have data on this.

The initial MTTF $T_o$ is estimated from

$$T_o = \frac{1}{KN_o}$$

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

- Code is not "straight line" but has many loops and branches, except in very trivial cases, and
- The machine state varies, and hence the fault associated with an instruction may or may not be exposed at one particular execution of the instruction.

At present, $K$ must be determined from a similar program. It may be possible in the future to relate $K$ to program structure in some way. On six projects for which data is available, $K$ ranges from $1.54 \times 10^{-7}$ to $2.99 \times 10^{-7}$.

The calendar time interval $t$ consists of the sum of one to three periods. In each period, a different resource (indicated by the value of the index $k$): $C$—computer time, $F$—failure correction personnel, $I$—failure identification personnel) is limiting or produces the maximum ratio of calendar time to execution time for that period. Thus the duration of each period is computed separately based on its limiting resource, and then the durations are summed. We have

$$t = \sum \frac{\Delta x_k}{P_k \cdot \rho_k}$$

where $\Delta x_k$ is the limiting resource requirement for the period (e.g., 100 person days), $P_k$ is the limiting resource quantity available (e.g., 5 persons), and $\rho_k$ is the limiting resource utilization factor. The resource utilization factor reflects the possibility [particularly for failure correction personnel (see Reference 3, pp. 432-3)] that all of an available resource cannot be usefully employed.
Each resource requirement during its limiting period is given by

\[ \Delta \chi_k = \Delta \tau_k + \mu_k \Delta m_k, \]

where \( \Delta \tau_k \) is the resource expenditure per unit execution time, \( \mu_k \) is the resource expenditure per error, \( \Delta m_k \) is the execution time interval for the period, and \( \Delta m_k \) is the number of failures experienced in the period.

The number of failures experienced in the period is given by

\[ \Delta m_k = M_k T_0 \left[ \frac{1}{T_k} - \frac{1}{T_{k+1}} \right]. \]

The execution time interval may be determined from

\[ \Delta \tau_k = \frac{M_k T_0}{C} \ln \frac{T_{k+1}}{T_k}, \]

where \( C \) is the testing compression factor and \( T_{k+1} \) and \( T_k \) represent the MTTFs at the boundaries of the limiting resource period.

The boundaries of the different resource-limited periods \( T_{k+1} \) and \( T_k \) are the present and objective MTTFs and the transition points

\[ T_{kk'} = \frac{C (P_k \mu_k \theta_k - P_{k'} \mu_{k'} \theta_{k'})}{P_k \rho_k \theta_k - P_{k'} \rho_{k'} \theta_{k'}}, \]

that lie within that range, where \( (k, k') \) have the values \( (C, F), (F, I), \) and \( (I, C) \). One must determine which resource-limited periods actually occur from an examination of the boundaries and a determination of the maximum calendar time/execution time ratio for each period. The maximum calendar time/execution time ratio for each period is given by

\[ \max \left[ \frac{\theta_k T + C \mu_k}{P_k \rho_k \theta_k} \right]. \]

where \( T \) is any MTTF in the range \( [T_k, T_{k+1}] \).

System test cost

Estimates of system test cost are made as follows. Determine the number of failures

\[ m = M_k \left( 1 - \frac{T_0}{T_F} \right) \]

that must be experienced and the associated execution time

\[ \tau = \frac{M_k T_0}{C} \ln \left( \frac{T_F}{T_{k+1}} \right). \]

To increase the MTTF from \( T_0 \) to the MTTF objective \( T_F \), each of the three total resource expenditures \( \chi_k \) is given by

\[ \chi_k = \theta_k T + \mu_k m, \]

where \( j \) has the values \( C, F \), and \( I \). The \( \chi_k \) are multiplied by cost rates and the results totaled to yield overall cost.

The foregoing approach implicitly assumes that idle time for all resources during the project can be profitably employed in other activities and should not be charged as a cost. If this is not true for any resource, the cost for that resource should be determined by multiplying \( \tau \) from (2) by the total number of personnel (or dedicated computers) and the resource rate.

Sensitivity of results to parameter accuracy

The reader may be concerned about the accuracy with which parameters on a particular project can be estimated. If this is a problem, one should note that inaccuracies usually affect absolute rather than relative values. Many and perhaps most system engineering decisions are concerned with relative values of alternatives. In any case, calculations can be performed with different values of a parameter to determine the sensitivity of a decision to a parameter inaccuracy. As experience is gained and more data is available, it should be possible to determine parameters more accurately and hence improve the absolute accuracy with which costs, schedules, etc. can be estimated.

Example

Consider a cost optimization problem to illustrate the application of the foregoing concepts. An online system is being planned to process orders received by a business, generate bills, break down the work involved into tasks and write work orders on those tasks, order materials, etc. It is desired to establish the mean-time-to-failure (MTTF) objective for the system that will minimize total system cost over an estimated lifetime of two years. Faults are not to be corrected in the field for this system; they will be fixed at the next release. Assume for simplicity that the hardware components of the system are much more reliable than the software and hence may be neglected in this analysis. Also, for simplicity, assume that the entire system test period is failure-correction-personnel limited. The system is expected to operate 250 days/yr, 8 hr/day. The average total cost impact of a failure (in terms of reduced efficiency, extra supervisory time and other work required to "straighten out the mess," etc.) is $10,000. The software consists of 100,000 source (400,000 object) instructions. Programmer loaded salary is $30/hr and computer (CPU) time is $1000/hr. The system test team has eight members and there are 40 program designers available for debugging. The utilization factor for failure correction personnel is 0.138 (computed from Reference 3, Equation (10), using a probability of 0.9 and a queue length of 3). Average instruction execution rate is one million instructions per second. On similar projects, a value
of fault exposure ratio \( K = 2.4 \times 10^{-7} \) has been experienced and the average fault rate has been 6.25 faults per thousand instructions. Data taken in similar environments indicates that six person hours are required for failure correction per failure and that this effort is independent of amount of execution time. Similarly, four person hours of system test team effort and one hour of chargeable computer time are required per hour of execution time and two person hours of system test team effort and one-half hour of computer time are required per failure. Assume a testing compression factor \( C \) of one.

We compute a value of \( M_0 = N_0 = 625 \), using the program size and average fault rate. The linear execution frequency, determined by dividing object instruction execution rate by number of object instructions is 2.5 sec\(^{-1}\) or 9000 hr\(^{-1}\). Hence from (1) we obtain \( T_F = 0.741 \) hr. Using (8) and (9) we find

\[
m = 625 - \frac{463}{T_F}
\]  
and

\[
\tau = 463 \ln \left( \frac{T_F}{0.741} \right)
\]  

Substituting the resource expenditure rates and (11) and (12) in (10) for each resource we obtain

\[
\chi_c = 313 - \frac{232}{T_F} + 63 \ln \left( \frac{T_F}{0.741} \right)
\]

\[
\chi_r = 3750 - \frac{2778}{T_F}, \quad \text{and}
\]

\[
\chi_f = 1250 - \frac{926}{T_F} + 1852 \ln \left( \frac{T_F}{0.741} \right)
\]

The cost of system test will be

\[
$1000 \chi_c + $30(\chi_r + \chi_f) = $463,000 - \frac{343,000}{T_F} + $519,000 \ln \left( \frac{T_F}{0.741} \right).
\]  

The number of failures during operation will be the total operating lifetime divided by MTTF, or \( \frac{4000}{T_F} \). Hence the cost of failures will be \( \frac{40,000,000}{T_F} \). The expression for the sum of these costs,

\[
$463,000 + \frac{39,700,000}{T_F} + $519,000 \ln \left( \frac{T_F}{0.741} \right)
\]

is of the form

\[
a + \frac{b}{T_F} + c \ln \frac{T_F}{T_0}
\]

A simple minimization using calculus yields

\[
T_F|_{\text{MIN}} = \frac{b}{c}
\]

Thus we obtain a value of 76.5 hr for the MTTF objective that minimizes system life cycle costs. The cost of system test and operational failures for this value is $3,389,000.

To determine the duration of system test, note that since there is only one limiting resource period, (2) becomes

\[
t = \frac{\chi_r}{\chi_r S_r}
\]

where \( \chi_r \) is the resource expenditure for the entire system test period and is given by (10). Hence, using (14) in (20), along with the number of failure correction personnel and their utilization factor, we obtain

\[
t = 679 - \frac{503}{T_F}.
\]

At the minimum cost point, the system test period will require 672 hr or 84 eight-hr days.

Sensitivity analyses can be conducted for each of the quantities involved in the calculation. For example, let \( K \) be \( 1.6 \times 10^{-7} \) rather than \( 2.4 \times 10^{-7} \). Repeating the calculations outlined above yields a value of 50.8 hr for the MTTF objective that minimizes system life cycle costs.

Another useful technique is to vary parameters that are under the control of the manager to determine the effects on schedules and costs. For example, suppose that the length of the system test period is unsatisfactory. Let us examine the effect of staffing with 60 rather than 40 programmers for debugging. This will reduce the failure correction personnel utilization factor to 0.121 (from Reference 3, Equation (10), using a probability of 0.9 and a queue length of 3). Equation (21) now becomes

\[
t = 517 - \frac{383}{T_F}
\]

The costs will remain unchanged, since the total resource expenditure required to reach a given MTTF remains the same. Thus minimum cost will still occur at the same value of \( T_F \). However, the time required for system test is reduced to 512 hr (64 days).

The reader might suggest increasing the debugging staff still further to improve schedules. In actuality, possible improvement is restricted. We have oversimplified the example for explanatory purposes by dealing with only one limiting resource period. The other periods would come into play and limit further reduction in system test duration.

The reliability of the software for one day (eight hours of operation), assuming the MTTF objective of 76.5 hr is attained, is given by

\[
R = \exp \left( - \frac{T'}{T} \right) = 0.90.
\]

where \( T' \) is the period of operation and \( T \) is the MTTF. This figure can be combined with reliabilities of hardware components to give overall system reliability (Reference 3, pp. 439-442).
MONITORING TEST PROGRESS AND REENGINEERING THE SYSTEM

Recall that status estimates like those shown in Figure 1 can be obtained throughout the system test period. One can therefore continually track present MTTF and its confidence bounds. The number of failures, execution time, and calendar time required to reach the MTTF objective are also computed. By use of (10) and cost rates, remaining system test cost can be computed. The estimates are usually better than those made before test, and they generally improve in quality as testing proceeds.

The effects of changing the MTTF objective or various resources can be investigated if schedule or costs are unsatisfactory. Consequently the manager can not only determine the present status of his system test effort in terms of a parameter (MTTF) directly related to operational requirements, but he can explore alternatives for accomplishing his objective or the effects of altering it. Thus the techniques we have discussed can be decision-making aids. Many of the decisions represent a system re-engineering.

A plot of MTTF history for an actual project is given in Figure 2. The dot-dashed center curve is the maximum likelihood estimate and the solid outer curves delineate the bounds of the 75 percent confidence interval. Note the general upward progress with some downward swings. Although there may be some statistical variation, the downward swings have usually been found to be correlated with design changes or the introduction of new code. The present MTTF is very sensitive to remaining errors when only a few remain; hence its upper confidence limit can be noisy.

SOFTWARE MAINTENANCE

Failures continue to occur (and usually, be corrected) for virtually all software systems of any size during the operational phase. They may occur at increasing intervals, but in many cases, the MTTF of a system exhibits a general stability about some value over the long term, although there are many swings about this value. This behavior is generally due to the periodic installation of design changes (with a resultant drop in MTTF) followed by periods of error removal, during which the MTTF improves. It is particularly true of operating systems and other software provided in computation centers; this is illustrated in Figure 3.

If the MTTF can be tracked and plotted as illustrated and if service objectives can be set for the system, a quantitatively-based mechanism for change control can be established. When MTTF falls below the service objective, the system is frozen until improvement occurs. The manager may use the amount of margin above the service objective as a guide to the size of the change he will permit at any given time.
CONCLUSIONS

The theory outlined in this paper has proved to be a good framework for understanding, measuring, and predicting the reliability of computer programs. It constitutes an approach that is compatible with hardware reliability theory as to combination of components and thus permits reliability analysis of hardware-software systems. The theory has been applied to several software development projects of different kinds and several operational systems. Substantial experience has been gained in its use. It can be used in system engineering, test monitoring, and change control of operational software. As more data is collected from various projects, it should be possible to improve the estimates of some of the parameters, due to added insight into how they vary with program environment factors. This would result in further improvement in the estimation of status quantities, particularly completion date. Experience in application of the theory should lead to its further refinement and broadening, resulting in greater accuracy and wider utility.

ACKNOWLEDGMENT

The author is indebted to R. E. Archer, Jr., D. T. Chai, J. P. Collins, and T. R. Peters for their helpful comments and suggestions.

REFERENCES