A Markovian model for reliability and other performance measures of software systems

by AMRIT L. GOEL and KAZU OKUMOTO
Syracuse University
Syracuse, New York

INTRODUCTION

Several studies have been undertaken in recent years to investigate the software failure phenomenon, with the objective of developing analytical models for quantitative assessment of software performance. Most of these studies assume that the times between software failures follow an exponential distribution with a failure rate that depends on the number of errors in the system (see, for example, Jelinski and Moranda,5 Littlewood and Verrall6 and Shooman11). A key assumption made in most of these studies is that the errors are removed with certainty when detected. However, as pointed out in Miyamoto7 and Thayer et al.,12 errors are not always corrected when detected. The existing models do not provide a solution for such situations.

In this paper we develop a Markovian model, which we call an Imperfect Debugging Model (IDM), for studying software failures when errors are not removed/corrected with certainty, i.e., for the case of imperfect debugging. Also, expressions are derived for the following probabilistic performance measures:

- Time to a completely debugged system.
- Time to a specified number of remaining errors.
- Number of remaining errors at time t.
- Number of errors detected by time t.
- Reliability function.

Actual failure data from a large Department of Defense (DoD) software project are analyzed and the results compared with the observed values.

MODEL DEVELOPMENT

The following assumptions are made for developing the model:

1. An error causing a software failure, when detected, is corrected with probability p, while with probability q (p + q = 1) we fail to completely remove it. Thus, q is the probability of imperfect debugging.
2. The time, T_i, to a software failure, when i errors remain in the system, follows an exponential distribution with parameter iλ. The parameter λ represents the mean error occurrence rate per unit time.
3. The time to remove an error will be neglected in this model.
4. No new errors are introduced during the debugging process.
5. At most, one error is removed at correction time.

Let X(t) denote the number of errors remaining in a software system at time t. We will use this random variable to describe the behavior of the error removal process as a function of time. Further, let N be the number of errors at the beginning of the debugging phase, i.e., X(0) = N.

Suppose that there are j errors in the package at some time. Then from Assumption 1, we note that after the occurrence of the next failure

\[ X(t) = \begin{cases} i-1 & \text{with probability } p, \\ i & \text{with probability } q. \end{cases} \]  

In other words, if we were to observe the X(t) process at times of software failures, then its behavior is governed by Equation 1. The transition probabilities \( P_{ij} \) from state i to state j are given by

\[ P_{ij} = \begin{cases} p & j=i-1 \\ q & j=i \\ 1 & i=j=0 \\ 0 & \text{otherwise} \end{cases} \]  

for \( i, j = 0, 1, 2, \ldots, N. \)

A diagrammatic representation of transitions between states corresponding to Equation 2 is given in Figure 1.

Assumption 2 implies that the Probability Density Function (PDF) and the Cumulative Distribution Function (CDF) of the random variable T_i are, respectively given by

\[ f_i(t) = iλ e^{-iλt} \]  

and

\[ F_i(t) = 1 - e^{-iλt}. \]

We note that even though the stochastic process X(t)
makes transitions from state to state in accordance with Equation 2, the times spent in various states are random and are given by Equation 3. Hence, the process \( \{X(t), t \geq 0\} \) forms a semi-Markov process (see, for example, Ross\(^9\)). A typical realization of this process is shown in Figure 2. It should be pointed out that in our formulation the process \( X(t) \) undergoes both real and virtual transitions. This means that after an attempt to remove an error the state of \( X(t) \) may change or may remain unchanged. In Figure 2, real transitions occur at states \( N\), \( N-2 \) and \( i \) while a virtual transition occurs at state \( N-1 \).

Let \( Q_{ij}(t) \) denote the one-step transition probability that, after making a transition into state \( i \), the process \( X(t) \) next makes a transition into state \( j \) by time \( t \). In other words, if a software package has \( i \) remaining errors at time zero, then \( Q_{ij}(t) \) represents the probability that the next failure, resulting in \( j \) remaining errors, will be by time \( t \). Then, we can show that

\[
Q_{ij}(t) = P_{0j} F_i(t)
\]

for \( i, j = 0, 1, 2, \ldots, N \). Substituting (2) in (5) yields

\[
Q_{ij}(t) =\begin{cases} 
  p F_i(t) & j = i - 1 \\
  q F_i(t) & j = i \\
  1 & j = 0 \\
  0 & \text{otherwise}
\end{cases}
\]

For known parameters \( N, p \) and \( \lambda \), the probabilities \( Q_{ij}(t) \) are obtained from Equation 6. This equation represents the basic model that will be used in the following sections for obtaining the various performance measures for software systems.

**EXPRESSIONS FOR PERFORMANCE MEASURES**

**Distribution of time to a completely debugged software system**

Suppose \( i \) is the number of errors remaining in a software system at some time during the debugging process and let \( G_{i,0}(t) \) represent the probability that the time required to go from \( i \) to 0 errors is less than or equal to \( t \). In other words, \( G_{i,0}(t) \) represents the CDF of the time required to get a completely debugged system when the current number of remaining errors is \( i \).

Recall that at time zero, \( X(0) = N \) and at the time of the first failure

\[
X(t) =\begin{cases} 
  N-1 & \text{with probability } p \\
  N & \text{with probability } q
\end{cases}
\]

as shown in Figure 1. Suppose that the debugging at the first error occurrence is perfect. Then the probability of going from \( N \) to \( N-1 \) errors in time \( [u, u+du] \) is given by \( dQ_{N,N-1}(u) \). If the system clears an error at time \( u \), then the process \( X(t) \) restarts with \( (N-1) \) remaining errors at time \( u \) and the probability of going from \( N-1 \) to 0 in time \( t-u \) is \( G_{N-1,0}(t-u) \). Hence the probability of going from \( N \) to 0 in time \( t \) is

\[
\int_0^t G_{N-1,0}(t-u) dQ_{N,N-1}(u) = Q_{N,N-1} G_{N-1,0}(t),
\]

where * denotes convolution.

Similarly, if the debugging at the first error occurrence is imperfect, the probability of going from \( N \) to 0 in time \( t \) is

\[
\int_0^t G_{N,0}(t-u) dQ_{N,N}(u) = Q_{N,N} G_{N,0}(t).
\]

Since the events depicted in Equations 8 and 9 are mutually exclusive, we get the renewal equation

\[
G_{N,0}(t) = Q_{N,N} G_{N,0}(t) + Q_{N,N} G_{N,0}(t).
\]
In general, we get the renewal equation
\[ G_{i,0}(t) = Q_{i-1,i} \cdot G_{i-1,0}(t) + Q_{i,i} \cdot G_{i,0}(t) \] (11)
for \( i=1, 2, \ldots, N \)
where \( G_{0,0}(t) = 1 \).

Using the Laplace-Stieltjes (L-S) transform technique (see Ross, for example) to solve the renewal Equation 11, we obtain the probability that the software system will be completely debugged by time \( t \) as
\[ G_{N,0}(t) = \sum_{j=1}^{N} C_{N,j}(1-e^{-\omega j t}). \] (12)
where
\[ C_{N,j} = \binom{N}{j} (-1)^{j-1}. \]

Distribution of time to a specified number of remaining errors

In many instances a completely debugged software is not cost-effective and we may be willing to tolerate a certain number of remaining errors, say \( n_0 \), which will ensure some desired reliability. The distribution of time to \( n_0 \) is then of interest.

Using an approach similar to the one above, we get the renewal equation
\[ G_{i,n_0}(t) = Q_{i-1,i} \cdot G_{i-1,n_0}(t) + Q_{i,i} \cdot G_{i,n_0}(t), \] (13)
for \( i=n_0 + 1, \ldots, N \)
where \( G_{n_0,n_0}(t) = 1 \). Then the probability that the software system will have \( n_0 \) errors by time \( t \), starting with \( N \) errors at time 0, is obtained by
\[ G_{N,n_0}(t) = \sum_{j=1}^{N-n_0} B_{N,j,n_0}(1-e^{-\omega j t}), \] (14)
where
\[ B_{N,j,n_0} = \frac{N!}{n_0! j! (N-n_0-j)!} (-1)^{j-1} \frac{j}{n_0+j}. \]

Distribution of number of remaining errors

Let \( P_{N,n_0}(t) \) represent the probability that there are \( n_0 \) errors remaining in a software package at time \( t \), given that there are \( N \) errors at the beginning of debugging, i.e.,
\[ P_{N,n_0}(t) = P\{X(t)=n_0 \mid X(0)=N\} \]
which is the so-called state occupancy probability. Conditioning on the next failure and following an approach similar to the above, we get the following renewal equations:
\[ P_{n_0,n_0}(t) = e^{-\omega n_0 t} + Q_{n_0,n_0} \cdot P_{n_0,n_0}(t), \] (15)
\[ P_{n_0,n}(t) = Q_{n_0,n} \cdot G_{n,n}(t), \] (16)
for \( n_0 \leq N \)
where \( n_0 < N \).

The distribution of the number of remaining errors at time \( t \) is
\[ P_{n_0,n}(t) = G_{N,n_0}(t) - G_{N,n_0-1}(t), \] (17)
where
\[ G_{N,0}(t) = 1, \]
\[ G_{N-1,0}(t) = 0. \]

Finally, the expected number of remaining errors in the software at time \( t \) is
\[ E[X(t) \mid X(0)=N] = \sum_{n_0=0}^{N} n_0 P_{N,n_0}(t) = Ne^{-\omega t}. \] (18)

From Equation 18, we note that the number of errors remaining in the software system is expected to decrease exponentially over the debugging time.

Expected number of errors detected by time \( t \)

We introduce a new random variable \( N(t) \) which denotes the total number of errors detected by time \( t \). The process \( \{N(t), t \geq 0\} \) is called a counting process. We are interested in obtaining the expression for the expected number of errors detected, \( M_N(t) \), during the debugging period, \( t \), when the initial number of errors is \( N \), i.e.
\[ M_N(t) = E[N(t) \mid X(0)=N] \]
which is called a Markov renewal function. By conditioning on the next software failure, we obtain the renewal equations
\[ M_{N,j}(t) = F_j(t) + p M_{j-1,n} \cdot F_j(t) + q M_{j,n} \cdot F_j(t), \] (19)
where \( M_N(t) = 0 \). From (19) we obtain
\[ M_N(t) = \frac{N}{p} (1-e^{-\omega t}). \] (20)

By taking the derivative of (20) we can get the error detection rate at time \( t \),
\[ M_N(t) = N\omega e^{-\omega t} \] (21)
which is exponentially decreasing over time. This implies that more errors are detected during the early stages of debugging. Note that if we let \( t \to \infty \) we have
\[ M_N(\infty) = \frac{N}{p} \]
which is the expected number of software errors detected by the end of debugging.

Let us now consider the case when the detected errors are separated as new errors and errors which were not corrected due to imperfect debugging. Let \( N_1(t) \) be a random variable which denotes the total number of imperfect debugging errors by time \( t \). Then we can show that
\[ D_N(t) = q M_N(t). \] (22)
From the collection of the Computer History Museum (www.computerhistory.org)

772 National Computer Conference, 1979

where

\[ D_n(t) = \mathbb{E}[N_n(t) | X(0) = N]. \]

Note that \( D_n(x) = q \frac{N}{p} \). Equation 22 implies that 100q percent of the software failures during debugging will be due to imperfect debugging.

Software system reliability

So far we have studied the stochastic behavior of the number of errors in the software system during the debugging period. Now we investigate the distribution of the time between software failures and study the problem of reliability growth. From the second section recall that the random variable \( T_i \) denotes the time to next failure when the number of remaining errors is \( i \) and \( F_i(t) \) is the CDF of \( T_i \). Let \( X_k \) denote the time between the \((k-1)\)st and \( k\)th software failures and \( \Phi_k(x) \) be the CDF of \( X_k \). Note that \( X_k \) does depend on the number of remaining errors at the \((k-1)\)st failure but this number is not explicitly known. Further, let \( \eta_k \) be a random variable which denotes the number of remaining errors between the \((k-1)\)st and \( k\)th software failures. Then, from the second section we have

\[ \eta_1 = N \]

\[ \Phi_1(x) = F_N(x), \]

and

\[ \Phi_k(x) = pF_{N-1}(x) + qF_N(x). \]

In general, we have

\[ \Phi_k(x) = p(X_k \leq x) = \sum_{i=N-(k-1)}^{\infty} p(X_k \leq x | \eta_k = i) p(\eta_k = i) \]

or

\[ \Phi_k(x) = \sum_{i=0}^{k-1} p(X_k \leq x | \eta_k = N-k+j+1) p(\eta_k = N-k+j+1) \]

or

\[ \Phi_k(x) = \sum_{i=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j F_{N-(k-1)}(x). \] (23)

This is called a mixture of exponential distributions with binomial mixing portions. It can be shown that \( \Phi_k(x) \) is a Decreasing Failure Rate (DFR) distribution. The reliability function at the \( k\)th stage, i.e., between \((k-1)\)st and \( k\)th failure, is given by

\[ R_k(x) = p(X_k > x) \]

\[ = 1 - \Phi_k(x) \]

\[ = \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j \hat{F}_{N-(k-1)}(x) \] (24)

where in general

\[ \hat{F}_k(x) = 1 - F_k(x) = e^{-\lambda x}. \]

From a practical point of view the reliability obtained in (24) is not easy to work with. For computational purposes we use the following approximation.

\[ R_k(x) = e^{-\lambda x - \lambda k x}, \quad k = 1, 2, \ldots \] (25)

For details of this approximation, see Goel and Okumoto. 2

APPLICATION TO A LARGE SOFTWARE PROJECT

In this section we analyse the error data from a large software project and compute various performance measures using the results of the two preceding sections. The error data is taken from Fries, 1 and represents software errors from a large DoD systems development project. The project consisted of approximately 320,000 assembly language instructions. It included the operational software and the simulation software necessary to develop and test the former. Software Problem Reports (SPRs) were written in the time period from the beginning of configuration management (approximately start of integration testing) to delivery of the software. A total of 2036 SPRs were encountered during this period and they were categorized into 20 major groups. Data are also included about the source of errors, the type of correction made, and the time to find and fix the error.

For purposes of this analysis, we consider 1612 errors reported during the last 12 months of the software testing phase. For estimating the parameters \( N, p \) and \( \lambda \) we use the method given in Goel and Okumoto. 2 The estimated values of these parameters are

\[ \hat{N} = 2108, \quad \hat{p} = 0.936 \quad \text{and} \quad \hat{\lambda} = 0.1127. \]

Substituting \( \hat{N}, \hat{p} \) and \( \hat{\lambda} \) for \( N, p \) and \( \lambda \), respectively, in Equation 18, the estimated value of the expected number of remaining errors over the testing period \( t \) is obtained as

\[ \hat{E}[X(t) | X(0) = \hat{N}] = 2108e^{-0.936 \cdot 0.1127 t}. \]

A plot of this quantity is shown in Figure 3. Also shown in this figure are the observed values by month. The fitted curves for \( M_k(t) \) and \( D_k(t) \) from Equations 20 and 22 are shown in Figure 4 along with the actual values for these quantities. From Figures 3 and 4 we see that the model seems to describe the behavior of the software error phenomenon very well.

The reliability function for the system is obtained from Equation 25 as

\[ R_k(x) = e^{-\lambda x - \lambda k x}. \]

Plots of reliability for \( k = 1613(50)1863 \), i.e., for the cases when the number of errors removed is 1612(50)1862, are shown in Figure 5. From these plots the extent of reliability growth with \( k \) can be easily evaluated. For example, \( R_{1613}(0.1) = 0.24 \) while \( R_{1863}(0.1) = 0.74 \), i.e., the value of \( R(0.1) \) increases by 200 percent when \( k \) goes from 1613 to 1863.

The plots in Figure 5 can also be used to determine the expected time required to achieve a desired reliability. Suppose our objective is to have a reliability of 0.3 at 0.2 weeks.
We want to know the number of errors that must be removed to achieve this reliability. In other words we want to know the value of \( k \) that yields \( R_k(0.2) = 0.3 \). From Figure 5, we see that \( R_{1813}(0.2) = 0.3 \) so that \( k = 1813 \) errors to achieve desired reliability for the software system. Then the number of remaining errors will be

\[
\hat{n}_0 = 2108 - .936(1813 - 1)
\]

or

\[
\hat{n}_0 = 412.
\]

The expected time required to remove \((N - n_0)\) errors is

\[
\mu_{N,n_0} = \frac{1}{p\lambda n_0 + 1} \sum_{j=n_0+1}^{N} \frac{1}{j} = \frac{1}{p\lambda} \ln \frac{N-1}{n_0+1}.
\]

For our case, we get

\[
\hat{\mu}_{2108,412} = \frac{1}{(0.936)(0.1127)} \ln \frac{2108+1}{412+1} = 15.46 \text{ months}.
\]

In other words, to achieve the desired reliability, we will need to continue testing for 15.46 - 12 = 3.46 additional months.

**CONCLUSION**

We have developed an imperfect debugging model (IDM) for software systems and derived expressions for various performance measures in terms of the first passage time distribution of the underlying semi-Markov process. The failure data from a large software project were analyzed using the model developed in this paper. A comparison of the fitted and the observed values indicates that the model provides a good description of the underlying failure phenomenon. Also, reliability curves were used to determine the debugging time required to achieve a desired level of software system performance.

**REFERENCES**


