SOFTWARE RELIABILITY ISSUES UNDER OPERATIONAL AND TESTING CONSTRAINTS

MADHU JAIN
Department of Mathematics, Institute of Basic Science
Dr. B.R. Ambedkar University, Khandari, Agra 282005, India

KRITI PRIYA
Institute of Technology and Science
Mohan Nagar, Ghaziabad, India

Received May 2003
Revised February 2004

Software reliability plays an important role in assuring the quality of a software. To ensure software reliability, the software is tested thoroughly during the testing phase. The time invested in the testing phase or the optimal software release time depends on the level of reliability to be achieved. There are two different concepts related to software reliability, viz., testing reliability and operational reliability. In this paper, we compare both types of software reliabilities to determine the optimal testing time of the software so as to minimize the total expected software maintenance cost. We consider a software has a number of clusters of modules, each having a different number of errors and a different failure rate. A hyperexponential model is employed for analyzing software reliability growth. Parameter estimation using the maximum likelihood estimation technique is also discussed. Numerical illustrations are taken to explore the effect of various parameters on reliability and maintenance cost. It is noticed that the operational reliability concept should be adopted for the software testing time problem.

Keywords: Software testing; operational reliability; hyperexponential; optimal policy; release time; warranty.

1. Introduction

With the advancement of technology, the intricacy of computers has also increased. The effective performance of a computer depends mainly on two components: hardware and software, of which the software component is important as it is more likely to fail than hardware. Therefore, software reliability plays a vital role in the functioning of a computer system. A software goes through many phases during its development, the most essential of which is the testing phase. The reliability of the software can be improved in the testing phase as the errors incurred during development are removed in this phase. The software reliability in the testing phase is known as the testing reliability of the software, which is the probability
of no failure during the testing phase. After delivery, the software is in its operational phase where it undergoes real-time operations. From the customer’s point of view, the reliability of the software in the operational phase is more important. This reliability is called the operational reliability of the software.

Determining the optimal testing time is a crucial task as it affects the reliability of the software as well as the cost of the software. The optimal software testing time should be such that it minimizes the cost of the software and ensures a desired level of reliability. A lot of research has also been done for analyzing the software reliability in different scenarios. Suresh and Babu (1997) studied the estimation and optimization of operational software reliability via a nonhomogeneous Poisson Process. Yang and Xie (2000) clarified that testing reliability and operational reliability in the context of software are two different approaches and showed that the testing reliability approach is misleading during decision-making of the optimal software release time. Rallis and Lansdowne (2001) estimated the reliability for a software system with sequential independent reviews. Kallelpalli and Tian (2001) analyzed the reliability for web applications by using Unified Markov Models. Popstoianova and Trivedi (2001) presented an architecture-based approach for quantitatively assessing component-based software systems.

Various problems related to the optimal release policies of software have been studied by Xia et al. (1993), Chatterjee et al. (1997), Kapur and Bhalla (1992), Yamada and Osaki (1986), and Kimura et al. (1999). Several software cost models and software reliability growth models (SRGMs) have been developed by various researchers. Kapur et al. (1993) developed an exponential SRGM with a bound on the number of failures. Zeephongsekul (1996) and Pham (1996) studied the reliability growth of a software model under imperfect debugging. Pham and Zhang (1999) developed a software cost model with warranty and risk costs. Pham and Zhang (2002) also predicted the operational software availability and discussed its applications to telecommunication systems. Zheng (2002) studied dynamic release policies for software systems with a reliability constraint.

It is realized in the real-time system that the reliability of a software depends on the reliability of individual modules. Each of the modules might have a different number of initial errors and a different failure rate; e.g., simple modules might have smaller number of errors as compared to complex ones. The hyperexponential model (Pham, 2000) is widely used for analyzing a software that has a number of modules, as it takes into consideration the failure pattern of each of the modules of the software.

In this study, we present optimal policies for software testing time so that at optimal cost a desired level of reliability could be achieved. The total expected maintenance cost of the software is computed by considering the initial testing cost, the testing cost per unit of time and the cost incurred during the warranty period of the software. The warranty period of the software is the duration in which the developer bears the maintenance cost of the software. The testing cost per unit of time is calculated by taking the discount rate into account (for determining the
present value of the cost). The SRGM is assumed to be hyperexponential, where each module of the software has a different number of initial errors and a different failure rate. The rest of the paper is organized as follows. Section 2 provides an insight to the SRGM along with the notations and assumptions being used in the formulation of the model. In Section 3, the parameter estimation methodology of the SRGM is described. In Section 4, the total expected maintenance cost of the software is computed to determine policies for optimal testing time. The reliability constraints are examined in Section 5. The optimal testing time is obtained by minimizing the total cost subject to the reliability constraints. Both types of reliabilities, i.e., testing reliability and operational reliability, are analyzed. In Section 6, numerical illustration is given by considering two modules in the software. Sensitivity analysis is carried out to explore the effect of system parameters on reliability and total maintenance cost. Finally, conclusions and the future scope of the work are outlined in Section 7.

2. The SRGM

A software system consists of various modules, which are different from each other from the testing point of view, e.g., new and complex modules may have different failure rates as compared to the reused and simple ones. We employ a hyperexponential growth model that is based on the assumption that each module in the software has a different initial number of errors and a different failure rate.

The following notations are used for the mathematical formulation purpose:

- \( n \) Number of modules in the software
- \( a_i \) Number of initial faults in module \( i \)
- \( b_i \) Failure rate of each fault in module \( i \)
- \( m(t) \) Expected number of failures detected at time \( t \)
- \( EC(T) \) Total expected software maintenance cost
- \( T \) Software release time
- \( T^* \) Optimum software release time
- \( T_w \) Warranty period
- \( R_{op} \) Operational reliability
- \( R_{tc} \) Testing reliability
- \( c_0 \) Initial testing cost
- \( c_t \) Testing cost per unit time
- \( c_w \) Maintenance cost per fault during the warranty period
- \( C_w(T) \) Maintenance cost during the warranty period
- \( \alpha \) Discount rate of the cost

The mean value function for hyperexponential distribution is given by

\[
m(t) = \sum_{i=1}^{n} a_i [1 - e^{-b_i t}].
\] (2.1)
The failure intensity function can be obtained as
\[ \lambda(t) = \frac{dm(t)}{dt} = \sum_{i=1}^{n} a_i b_i e^{-b_i t}. \] (2.2)

3. Parameter Estimation

Parameter estimation is of primary importance in software reliability prediction. We suggest the parameter estimation for the hyperexponential model by using the maximum likelihood estimation (MLE) technique. The MLE method for estimating the parameters for a software containing \(n\) modules is given in Appendix A.

For a software of two modules, i.e., \(n = 2\), the mean value function reduces to
\[ m(t) = a_1 [1 - e^{-b_1 t}] + a_2 [1 - e^{-b_2 t}]. \]

Assuming that the data are given for the cumulative number of detected errors \(y_j\) in a given time interval \((0, t_j)\) where \(j = 1, 2, \ldots, k\) and \(0 < t_1 < t_2 < \cdots < t_k\), the log likelihood function (LLF) for the mean value function \(m(t)\), can be written as
\[ \text{LLF} = \sum_{j=1}^{k} (y_j - y_{j-1}) \log \left[ a_1 (1 - e^{-b_1 t_j}) + a_2 (1 - e^{-b_2 t_j}) - a_1 (1 - e^{-b_1 t_{j-1}}) - a_2 (1 - e^{-b_2 t_{j-1}}) - a_1 (1 - e^{-b_1 t_k}) + a_2 (1 - e^{-b_2 t_k}) \right]. \] (3.1)

There are four unknown parameters, i.e., \(a_1, a_2, b_1,\) and \(b_2\) to be estimated using the MLE. The following system of equations maximize the log likelihood function:
\[ \sum_{j=1}^{k} \frac{(e^{-b_1 t_{j-1}} - e^{-b_1 t_j})(y_j - y_{j-1})}{a_1(e^{-b_1 t_{j-1}} - e^{-b_1 t_j}) + a_2(e^{-b_2 t_{j-1}} - e^{-b_2 t_j})} - (1 - e^{-b_1 t_k}) = 0, \] (3.2)
\[ \sum_{j=1}^{k} \frac{(e^{-b_2 t_{j-1}} - e^{-b_2 t_j})(y_j - y_{j-1})}{a_1(e^{-b_1 t_{j-1}} - e^{-b_1 t_j}) + a_2(e^{-b_2 t_{j-1}} - e^{-b_2 t_j})} - (1 - e^{-b_2 t_k}) = 0, \] (3.3)
\[ \sum_{j=1}^{k} \frac{(t_j e^{-b_1 t_{j-1}} - t_{j-1} e^{-b_1 t_j})(y_j - y_{j-1})}{a_1(e^{-b_1 t_{j-1}} - e^{-b_1 t_j}) + a_2(e^{-b_2 t_{j-1}} - e^{-b_2 t_j})} - t_k e^{-b_1 t_k} = 0, \] (3.4)
\[ \sum_{j=1}^{k} \frac{(t_j e^{-b_2 t_{j-1}} - t_{j-1} e^{-b_2 t_j})(y_j - y_{j-1})}{a_1(e^{-b_1 t_{j-1}} - e^{-b_1 t_j}) + a_2(e^{-b_2 t_{j-1}} - e^{-b_2 t_j})} - t_k e^{-b_2 t_k} = 0. \] (3.5)

On solving the above equations simultaneously, we can get the values of the parameters \(a_1, a_2, b_1,\) and \(b_2\).

4. Optimal Testing Time for Cost Minimization

In this section, we describe some optimal policies to determine the testing time so as to minimize the total expected cost of the software. The cost model is developed by assuming that after the delivery of the software there is a warranty period in which
the maintenance cost of the software is paid by the developer. We also include a discount rate in the testing cost and the maintenance cost to determine the present value of the total cost of the software.

The following assumptions are made to obtain total expected software maintenance cost:

- Initial cost \( c_0 \) is required for testing the software.
- Testing cost varies with time and is affected by the present value (cost) of the software.
- The warranty period \( T_w \) is of constant length and the cost incurred during the warranty period depends on whether the software reliability growth occurs after the testing phase or not.

Thus, EC\( (T) \) can be given by

\[
EC(T) = c_0 + c_t \int_0^T e^{-\alpha t} dt + C_w(T) . \quad (4.1)
\]

The following two cases arise for determining the warranty cost \( C_w(T) \).

**4.1. Case 1: The software reliability growth does not occur after the testing phase**

Here, the warranty cost is given by

\[
C_w(T) = c_w \int_T^{T + T_w} \lambda(T)e^{-\alpha t} dt . \quad (4.2)
\]

Therefore, Eq. (4.1) becomes

\[
EC(T) = c_0 + c_t \int_0^T e^{-\alpha t} dt + c_w \int_T^{T + T_w} \lambda(T)e^{-\alpha t} dt . \quad (4.3)
\]

Differentiating Eq. (4.3) w.r.t. \( T \) and equating it to 0, we get

\[
\sum_{i=1}^n a_i b_i (\alpha - b_i) e^{-b_i T} - \frac{\alpha c_t}{c_w (1 - e^{-\alpha T})} = 0 . \quad (4.4)
\]

Since \( b_i \ll 1 \) \( (i = 1, 2, \ldots, n) \), Eq. (4.4) is reduced to

\[
\sum_{i=1}^n A_i (1 - b_i T) = K , \quad (4.5)
\]

where \( A_i = a_i b_i (\alpha - b_i) \) \( (i = 1, 2, \ldots, n) \) and

\[
K = \frac{\alpha c_t}{c_w (1 - e^{-\alpha T})} ,
\]

which gives the optimal release time as

\[
T^* = T_1 = \frac{\sum_{i=1}^n A_i - K}{\sum_{i=1}^n A_i b_i} . \quad (4.6)
\]

Since \( \left| \frac{d^2 EC(T)}{dT^2} \right|_{T=T_1} > 0 \), EC\( (T) \) has a minimum value at \( T_1 = T^* \). Now,

\[
\lambda(T_1) = \sum_{i=1}^n a_i b_i e^{-b_i T_1} .
\]
Thus, the optimal release policy 1 (ORP 1) in this case is stated as follows:

P1: \( T^* = T_1 \) when \( \lambda(0) > \lambda(T_1) \)

P2: \( T^* = 0 \) when \( \lambda(0) \leq \lambda(T_1) \)

The validity of the above policies is discussed in Appendix B.

4.2. Case 2: The software reliability growth occurs after the testing phase

Now, the warranty cost is given by

\[
C_w(T) = c_w \int_T^{T+T_w} \lambda(t)e^{-\alpha t} \, dt. \tag{4.7}
\]

Substituting this value of \( C_w(T) \) in Eq. (4.1), we get the total expected software maintenance cost as

\[
EC(T) = c_0 + c_t \int_0^T e^{-\alpha t} \, dt + c_w \int_T^{T+T_w} \lambda(t)e^{-\alpha t} \, dt. \tag{4.8}
\]

To obtain the minimum value of \( T \), we differentiate Eq. (4.8) w.r.t. \( T \) and equate it to 0, which gives

\[
\sum_{i=1}^n a_i b_i \frac{1 - e^{-(\alpha + b_i)T_w} e^{-b_i T}}{\alpha + b_i} - c_t = 0. \tag{4.9}
\]

Since \( b_i \ll 1 \) \( (i = 1, 2, \ldots, n) \), Eq. (4.9) is reduced to

\[
\sum_{i=1}^n B_i (1 - b_i T) = c_t, \tag{4.10}
\]

where

\[
B_i = \frac{a_i b_i}{\alpha + b_i} \frac{1 - e^{-(\alpha + b_i)T_w}}{e^{-b_i T}} \quad (i = 1, 2, \ldots, n),
\]

which gives the optimal release time as

\[
T^* = T_2 = \frac{\sum_{i=1}^n B_i - c_t}{\sum_{i=1}^n B_i b_i}. \tag{4.11}
\]

Since \( |d^2 EC(T)/dT^2|_{T=T_2} > 0 \), \( EC(T) \) has a minimum value at \( T_2 = T^* \). Now, \( \lambda(T_2) = \sum_{i=1}^n a_i b_i e^{-b_i T_2} \).

Thus, the optimal release policy 2 (ORP 2) in this case is stated as follows:

P1: \( T^* = T_2 \) when \( \lambda(0) > \lambda(T_2) \)

P2: \( T^* = 0 \) when \( \lambda(0) \leq \lambda(T_2) \)
5. Optimal Testing Time with Reliability Constraints

Here, we consider the optimal software release problem under reliability constraints. The testing reliability can be computed by

\[ R_{te}(x \mid T) = \exp\left[ -\{m(x + T) - m(T)\} \right]. \] (5.1)

The operational reliability is given by

\[ R_{op}(x \mid T) = \exp\left[ -\lambda(T)x \right]. \] (5.2)

Let the minimum reliability to be achieved be \( R_0 \). The optimal release problem can be formulated as follows:

(OPT) \hspace{1cm} \text{Minimize } EC(T) \\
Subject to \hspace{0.5cm} R(x \mid T) \geq R_0. \hspace{1cm} (5.3)

The above problem can be restated into two problems depending on the testing and operational reliability.

For the testing reliability, the optimization problem can be stated as

(OPT) \hspace{1cm} \text{Minimize } EC(T) \\
Subject to \hspace{0.5cm} \exp\left[ -\{m(x + T) - m(T)\} \right] \geq R_0. \hspace{1cm} (5.4)

And for the operational reliability, the problem is given by

(OPO) \hspace{1cm} \text{Minimize } EC(T) \\
Subject to \hspace{0.5cm} \exp\left[ -\lambda(T)x \right] \geq R_0. \hspace{1cm} (5.5)

Let \( T_{Re} \) denote the optimal release time satisfying Eq. (5.4) and \( T_{Rp} \) denote the optimal release time satisfying Eq. (5.5).

5.1. Optimal release policies (ORP)

In this section, we focus on the optimal release time for the above-formulated constrained optimization problems (OPT) and (OPO). In Section 4.1, we mentioned that for the unconstrained cost minimization problem, \( T^* = T_1 \) when \( \lambda(0) > \lambda(T_1) \). Hence, in this case, if the reliability constraint is satisfied, i.e., \( R_{te}(x \mid T) \geq R_0 \), we choose \( T^* \) to be \( T_1 \). But if the reliability constraint is not satisfied, i.e., \( R_{te}(x \mid T) < R_0 \), we take \( T^* \) to be the maximum of \( T_1 \) and \( T_{Re} \) as the optimal testing time is the duration in which the required reliability is achieved and the expected total cost of the software is minimized.

Again, when \( \lambda(0) \leq \lambda(T_1) \), we take \( T^* = 0 \) if \( R_{te}(x \mid T) \geq R_0 \). However, if the reliability constraint is not satisfied, we take \( T^* \) to be the maximum of \( T_1 \) and \( T_{Re} \), which gives \( T^* = T_{Re} \).
Now we provide the optimal release policies for the problems (OP_T) and (OP_O) as below:

**ORP for testing reliability optimization problem (OP_T):**
Following are the optimum release policies ORP 3 and ORP 4 for both the cases as described in Section 3:

**Case 1: Optimal release policy 3 (ORP 3)**

\[ P_{te.31}: \text{If } \lambda(0) > \lambda(T_1) \text{ and } R_{te}(x|0) < R_0, \text{ then } T^* = \max\{T_1, T_{R_{te}}\}. \]
\[ P_{te.32}: \text{If } \lambda(0) > \lambda(T_1) \text{ and } R_{te}(x|0) \geq R_0, \text{ then } T^* = T_1. \]
\[ P_{te.33}: \text{If } \lambda(0) \leq \lambda(T_1) \text{ and } R_{te}(x|0) < R_0, \text{ then } T^* = T_{R_{te}}. \]
\[ P_{te.34}: \text{If } \lambda(0) \leq \lambda(T_1) \text{ and } R_{te}(x|0) \geq R_0, \text{ then } T^* = 0. \]

**Case 2: Optimal release policy 4 (ORP 4)**

\[ P_{te.41}: \text{If } \lambda(0) > \lambda(T_2) \text{ and } R_{te}(x|0) < R_0, \text{ then } T^* = \max\{T_2, T_{R_{te}}\}. \]
\[ P_{te.42}: \text{If } \lambda(0) > \lambda(T_2) \text{ and } R_{te}(x|0) \geq R_0, \text{ then } T^* = T_2. \]
\[ P_{te.43}: \text{If } \lambda(0) \leq \lambda(T_2) \text{ and } R_{te}(x|0) < R_0, \text{ then } T^* = T_{R_{te}}. \]
\[ P_{te.44}: \text{If } \lambda(0) \leq \lambda(T_2) \text{ and } R_{te}(x|0) \geq R_0, \text{ then } T^* = 0. \]

**ORP for operational reliability optimization problem (OP_O):**
The optimum release policies ORP 3 and ORP 4 for the operational reliability optimization problem (OP_O) are given below:

**Case 1: Optimal release policy 3 (ORP 3)**

\[ P_{op.31}: \text{If } \lambda(0) > \lambda(T_1) \text{ and } R_{op}(x|0) < R_0, \text{ then } T^* = \max\{T_1, T_{R_{op}}\}. \]
\[ P_{op.32}: \text{If } \lambda(0) > \lambda(T_1) \text{ and } R_{op}(x|0) \geq R_0, \text{ then } T^* = T_1. \]
\[ P_{op.33}: \text{If } \lambda(0) \leq \lambda(T_1) \text{ and } R_{op}(x|0) < R_0, \text{ then } T^* = T_{R_{op}}. \]
\[ P_{op.34}: \text{If } \lambda(0) \leq \lambda(T_1) \text{ and } R_{op}(x|0) \geq R_0, \text{ then } T^* = 0. \]

**Case 2: Optimal release policy 4 (ORP 4)**

\[ P_{op.41}: \text{If } \lambda(0) > \lambda(T_2) \text{ and } R_{op}(x|0) < R_0, \text{ then } T^* = \max\{T_2, T_{R_{op}}\}. \]
\[ P_{op.42}: \text{If } \lambda(0) > \lambda(T_2) \text{ and } R_{op}(x|0) \geq R_0, \text{ then } T^* = T_2. \]
\[ P_{op.43}: \text{If } \lambda(0) \leq \lambda(T_2) \text{ and } R_{op}(x|0) < R_0, \text{ then } T^* = T_{R_{op}}. \]
\[ P_{op.44}: \text{If } \lambda(0) \leq \lambda(T_2) \text{ and } R_{op}(x|0) \geq R_0, \text{ then } T^* = 0. \]

6. **Numerical Illustrations**

In this section, we provide numerical illustrations to examine the analytical results and suggested policies. The fault data used in the illustrations have been selected on the basis of the information available in the literature (Musa, 1987).

Figures 1–3 demonstrate the effect of various parameters on the total maintenance cost EC(T). In Figure 1, the effect of discount rate α on EC(T) is shown by varying the testing time T. It can be noted that EC(T) decreases with an increase in α. This is due to the fact that the cost incurred in the maintenance of the software is low when the present value of the money is less. Figure 2 depicts the effect of the maintenance cost per fault during the warranty period, i.e., c_w on EC(T). It is
Fig. 1. EC(T) vs. T for $a_1 = 30$, $a_2 = 50$, $b_1 = 0.005$, $b_2 = 0.007$, $c_w = 150$, $c_0 = 100$, $c_t = 5$, $T_w = 2000$.

Fig. 2. EC(T) vs. T for $a_1 = 30$, $a_2 = 50$, $b_1 = 0.005$, $b_2 = 0.007$, $\alpha = 0.001$, $c_0 = 100$, $c_t = 5$, $T_w = 2000$.

Fig. 3. EC(T) vs. $a_1$ for $a_2 = 50$, $b_1 = 0.005$, $b_2 = 0.007$, $\alpha = 0.001$, $c_w = 150$, $c_0 = 100$, $c_t = 5$, $T_w = 2000$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
clear that as $c_w$ increases, $EC(T)$ also increases. Besides, $EC(T)$ first decreases with the testing time $T$ and then it increases. This signifies that the total maintenance cost can be decreased drastically if the duration of the testing phase is small; but if testing is carried out for a longer period, then the maintenance cost will increase gradually after a certain period. The variation of $EC(T)$ with the initial number of errors in module 1, i.e., $a_1$ is depicted in Figure 3 for $n = 1$ and $n = 2$. It can be seen that the software consisting of only one module requires lower maintenance cost as compared to that comprising two modules.

In Figures 4–7, we compare the testing and operational reliability of the software. Figures 4 and 5 illustrate the testing and operational reliabilities for $n = 1$ and $n = 2$ by varying $a_1$ and $a_2$, respectively. Figures 6 and 7 show the reliability by varying the failure detection rate of modules 1 and 2, respectively. Both operational reliability and testing reliability of the software decrease with $a_i$ ($i = 1, 2$) and increase with $b_i$ ($i = 1, 2$). This implies that the software is less reliable if there are more number of errors present in it initially. On the other hand, if the failure detection rate is increased, the software could be made more reliable.

![Graph showing software reliability](image)

Fig. 4. Software reliability by varying $a_1$ for $T = 900$, $a_2 = 50$, $b_1 = 0.005$, $b_2 = 0.007$, $x = 100$, $R_0 = 0.8$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Fig. 5. Software reliability by varying $a_2$ for $T = 900$, $a_1 = 30$, $b_1 = 0.005$, $b_2 = 0.007$, $x = 100$, $R_0 = 0.8$.

Fig. 6. Software reliability by varying $b_1$ for $T = 900$, $a_1 = 30$, $a_2 = 50$, $b_2 = 0.007$, $x = 100$, $R_0 = 0.8$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Fig. 7. Software reliability by varying $b_2$ for $T = 900, a_1 = 30, a_2 = 50, b_1 = 0.005, x = 100, R_0 = 0.8$.

Now, we examine the suggested optimal policies for the testing time ($T$). Table 1 demonstrates the optimal release policies ORP 1 and ORP 2 for $n = 1$ and $n = 2$. Evidently, the optimal testing time, $T^*$, decreases with the testing cost per unit time ($c_t$). This means that the testing of the software can be completed in a shorter duration if more money is invested in the testing phase, which is obvious. Also, $T^*$ shows an increasing trend with the warranty period, $T_w$. Figure 8 exhibits the optimal release policies 3 and 4 (ORP 3 and ORP 4) graphically for OP$_T$ and OP$_O$ by taking $n = 2$. Note that the solution to the problem OP$_T$ is $T^* = T_{R_w} = 858.5$ and $R_{op}$ corresponding to this solution is 0.74 which does not satisfy the reliability requirement of 0.8. The solution of the problem OP$_O$ is $T^* = T_{R_{op}} = 906.2$, which clearly satisfies the desired reliability. Though the testing time obtained by applying the testing reliability constraint is optimal with respect to the problem OP$_T$, it does not match with that of OP$_O$, which indicates that the testing reliability can be achieved in lesser testing time as compared to the operational reliability. This implies that the optimal release problem should be formulated according to the operational reliability constraint since the testing reliability constraint can lead to an incorrect value of the testing time. The figure also shows the difference between the testing times of the unconstrained and constrained optimization problems. The optimal testing time $T^* = 552.07$ for the unconstrained optimization problem and
Table 1. ORP 1 and ORP 2 for \( a_1 = 30, a_2 = 50, b_1 = 0.005, b_2 = 0.007,\)
\( \alpha = 0.001, c_w = 150, c_0 = 100, c_t = 5, T_w = 2000.\)

<table>
<thead>
<tr>
<th>(T_w)</th>
<th>(c_t)</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n = 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>710.8360</td>
<td>491.1135</td>
<td>388.9484</td>
<td>321.6540</td>
<td>271.3911</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>716.2291</td>
<td>496.5066</td>
<td>394.3415</td>
<td>327.0471</td>
<td>276.7842</td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td>720.9867</td>
<td>501.2643</td>
<td>399.0991</td>
<td>331.8047</td>
<td>281.5418</td>
<td></td>
</tr>
<tr>
<td>1900</td>
<td>728.9302</td>
<td>509.2077</td>
<td>407.0426</td>
<td>339.7482</td>
<td>289.1853</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>732.2498</td>
<td>512.5273</td>
<td>410.3622</td>
<td>343.0678</td>
<td>292.8049</td>
<td></td>
</tr>
<tr>
<td>(n = 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>810.2664</td>
<td>622.1191</td>
<td>536.6120</td>
<td>480.9470</td>
<td>439.6693</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>811.8879</td>
<td>624.1284</td>
<td>538.7900</td>
<td>483.2389</td>
<td>442.0706</td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td>816.0235</td>
<td>628.1315</td>
<td>542.7383</td>
<td>487.1456</td>
<td>445.9503</td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>819.6777</td>
<td>631.6749</td>
<td>546.2323</td>
<td>490.6083</td>
<td>449.3903</td>
<td></td>
</tr>
<tr>
<td>1900</td>
<td>822.9003</td>
<td>634.8011</td>
<td>549.3155</td>
<td>493.6641</td>
<td>452.1263</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>825.7852</td>
<td>637.5969</td>
<td>552.0715</td>
<td>496.3949</td>
<td>455.1388</td>
<td></td>
</tr>
</tbody>
</table>

Policy 2

| \(n = 1\) |       |     |     |     |     |     |
| 1500  | 402.9559 | 183.2335 | 81.0863 | 13.7739 | 0.0000 |
| 1600  | 402.9671 | 183.2446 | 81.0794 | 13.7853 | 0.0000 |
| 1700  | 402.9732 | 183.2507 | 81.0855 | 13.7911 | 0.0000 |
| 1800  | 402.9765 | 183.2541 | 81.0889 | 13.7949 | 0.0000 |
| 1900  | 402.9784 | 183.2559 | 81.0907 | 13.7963 | 0.0000 |
| 2000  | 402.9794 | 183.2569 | 81.0917 | 13.7973 | 0.0000 |

| \(n = 2\) |       |     |     |     |     |     |
| 1500  | 522.7557 | 339.4290 | 256.0320 | 201.6954 | 161.4045 |
| 1600  | 522.7607 | 339.4327 | 256.0355 | 201.6987 | 161.4076 |
| 1700  | 522.7637 | 339.4352 | 256.0377 | 201.7008 | 161.4096 |
| 1800  | 522.7653 | 339.4365 | 256.0389 | 201.7019 | 161.4107 |
| 1900  | 522.7662 | 339.4372 | 251.7025 | 201.7025 | 161.4113 |
| 2000  | 522.7666 | 339.4376 | 256.0400 | 201.7028 | 161.4116 |

the corresponding total maintenance cost is \( EC(T) = 2582.1. \) After imposing the operational reliability constraint, \( T^* \) becomes 906.2 and the corresponding cost becomes 3196.6. It is clear that for ensuring the reliability of the software more maintenance cost is required. On the whole, we infer the following from the numerical illustrations:

- The total maintenance cost of the software increases if the duration of the testing phase is increased to a larger extent.
- A software is less reliable if there are more number of errors present in it before testing.
- If the failure detection rate of the software is increased then the software can be made more reliable.
The operational reliability constraint should be adopted in spite of the testing reliability constraint as the latter could lead to an inaccurate value of the optimal testing time.

7. Conclusion

We have developed an hyperexponential SRGM that may be employed to compute the total expected software maintenance cost of a module-based software by considering the behavior of each of the modules. The method provided for estimating the parameters of the model can be used for any other SRGM. The optimal software release time is obtained by minimizing the total maintenance cost of the software such that a desired reliability could be achieved. Two different reliability concepts,
operational reliability and testing reliability, are studied. It can be concluded that
the operational reliability constraint should be imposed for computing the optimal
testing time rather than the testing reliability constraint. The operational reliability
leads to more accurate results from the customer’s point of view. The suggested
policy may be helpful in determining the optimal time to release a software subject
to reliability requirement while minimizing the total maintenance cost.

Appendix A

Assuming that the data are given for the cumulative number of detected errors \( y_j \)
in a given time interval \((0, t_j)\) where \( j = 1, 2, \ldots, k \) and \( 0 < t_1 < t_2 < \cdots < t_k \),
then the log likelihood function (LLF) for the mean value function \( m(t) \) can be
written as

\[
\text{LLF} = \sum_{j=1}^{k} \left( y_j - y_{j-1} \right) \log[m(t_j) - m(t_{j-1})] - m(t_k). \tag{A.1}
\]

The following system of equations give the maximum of the above function:

\[
\sum_{j=1}^{k} \frac{\partial}{\partial p} m(t_j) - m(t_{j-1}) \frac{\partial}{\partial p} m(t_{j-1}) \frac{\partial}{\partial p} m(t_k), \tag{A.2}
\]

where \( p \) is the unknown parameter and \( m(t) = \sum_{i=1}^{n} a_i[1 - e^{-b_it}] \).

For a hyperexponential distribution, the mean value is of the form

\[
m(t) = \sum_{i=1}^{n} a_i[1 - e^{-b_i t}] .
\]

For a software containing \( n \) modules, the parameters \( a_i \) and \( b_i \) for \( i = 1, 2, \ldots, n \)
can be obtained by solving the following equations simultaneously:

\[
\sum_{j=1}^{k} \frac{(e^{-b_i t_{j-1}} - e^{-b_i t_j})(y_j - y_{j-1})}{\sum_{i=1}^{n} a_i(e^{-b_i t_j} - e^{-b_i t_j})} - (1 - e^{-b_i t_k}) = 0 \tag{A.3}
\]

and

\[
\sum_{j=1}^{k} \frac{(t_i e^{-b_i t_{j-1}} - t_{i-1} e^{-b_i t_j})(y_j - y_{j-1})}{\sum_{i=1}^{n} a_i(e^{-b_i t_j} - e^{-b_i t_j})} - t_k e^{-b_i t_k} = 0. \tag{A.4}
\]

Further details of estimating the parameters by the above-mentioned method can
be found in Pham (2000).

Appendix B

Equation (2.2) implies that

\[
\lambda(T) = \sum_{i=1}^{n} a_i b_i e^{-b_i T}. \tag{B.1}
\]
We consider two cases:

(i) When \( \lambda(0) > \lambda(T_1) \), then Eq. (B.1) gives

\[
\sum_{i=1}^{n} a_i b_i > \sum_{i=1}^{n} a_i b_i e^{-b_i T_1}
\]

\[\Rightarrow a_i b_i > a_i b_i e(-b_i T_1) \quad i = 1, 2, \ldots, n\]

\[\Rightarrow 1 > e(-b_i T_1) \quad i = 1, 2, \ldots, n\]

\[\Rightarrow T_1 > 0 \text{ so that we can choose } T_1 \text{ as optimal testing time, i.e., } T^* = T_1.\]

(ii) When \( \lambda(0) \leq \lambda(T_1) \), \( T_1 \leq 0 \). Since optimal testing time cannot be taken as negative, in this case \( T^* = 0 \).

References


**Madhu Jain**, born in 1959, received her MSc, MPhil, PhD, and DSc degrees in Mathematics from the University of Agra. She was a Gold medallist at the MPhil level. She has more than 125 research publications in refereed international/national journals and more than a dozen books to her credit. She was awarded the Young Scientist award, an SERC visiting fellowship of the Department of Science & Technology (DST), Govt. of India, and a Career award of University Grants Commission (UGC), India. She has successfully completed six sponsored major research projects of DST, UGC, and Council of Scientific & Industrial Research (CSIR). Her current research interest includes wireless communications, performance modelling, soft computing, bioinformatics, and operations research. She has visited more than 25 reputed Universities/Institutes in the United States, Canada, the United Kingdom, Germany, France, Holland, and Belgium. She has participated and presented her research works in more than 25 international and 50 national conferences/seminars. Twelve candidates have received their PhD degrees under her supervision. After having taught in DAV College, Dehradun, and IIT. Delhi, as visiting faculty, she is teaching and conducting research at the Institute of Basic Science, Dr B.R. Ambedkar University, Khandari, Agra.

**Kriti Priya**, born in 1977, received her MSc. degree (2000) in Industrial Mathematics and Computer Applications from Dr B.R. Ambedkar University, Agra. She worked with HCL Infosystems, Noida, as a software trainee and with Datum Technologies (as a software engineer) for two years. She had three years of teaching experience in Operations Research and Mathematics. At present she is working as a lecturer of mathematics in Institute of Technology and Science, Mohan Nagar, Ghaziabad, India. Her research interests include software reliability, computer communication systems, and multimedia.