The optimal release policy for constructing the software reliability growth model (SRGM) that incorporates both imperfect debugging and change-point concepts has been investigated. In any realistic situation the change-point problem may be realized due to the change in many factors like testing strategies, environment, resource allocation, etc.. The cost of the software before release and the costs of removing errors before and after release are known as software cost factors. It is important to ensure when to stop testing or when to release the software so that the total system development cost can be minimized subject to reliability constraint. To evaluate the total expected cost, a warranty cost model using non-homogeneous process along with change-point phenomenon is discussed. Optimal release policies based on cost and reliability criteria are constructed for determining fairly accurate optimal software release time. The phenomenon and applicability of the proposed model are established via numerical results.

5.1 Introduction

Software reliability engineering is an exponentially growing field and day by day their importance and needs have been increasing. Software developers need to produce software that should be more reliable; this can be possible by expanding high cost for fixing failures and taking care of safety concerns and legal liabilities. It is very difficult for developer to produce a completely fault free software because many faults may be introduced during its development process. So the goal of software development organization is to evaluate realistic software reliability and provide useful guidance that product can meet market reliability expectation. Software reliability growth models (SRGMs) facilitate a mathematical relationship between the number of faults removed and the testing time. These models also provide the useful information that how we can improve the reliability of the system software. A large number of software reliability
growth models (SRGMs) have been proposed by many researchers, still new models are to be developed in this field so that they could fit a great number of reliability growth curves depending upon a variety of applications in different scenario.

Mostly software reliability growth models are based on the assumption that the debugging is perfect. But in real time system, during the testing phase debugging process are not always performed perfectly. The software is developed by human, so it is possible to introduce new errors into the software during development/testing phase. Sometimes, due to the complexity of fault, it could not be removed perfectly. This phenomenon is called imperfect debugging. Some studies of the effect of imperfect debugging on software cost model were made by Xie and Yang (2003) and Pham (2003), Williams (2007). Chen et al. (2001) and Shyur (2003) analyzed the software reliability growth model with imperfect debugging and change-point. Jain and Priya (2005) discussed various software reliability issues under operational and testing phase. Recently, discrete software reliability growth modeling was studied by Goswami et al. (2007) by incorporating the concept of errors of different severity and change-point. Chiu et al. (2008) studied the software reliability growth model from the perspective effects. Zio (2009) suggested the old problems and new challenges related to reliability engineering. Recently, Jin et al. (2010) considered reliability growth modeling for electronic systems considering latent failure modes.

A recent trend in the development of NHPP based SRGMs is the incorporation of some additional information including the change-point concept. In more realistic situations, expenditures on software testing based upon testing strategy, resource allocation, environment criterion etc., may be changed at some point of time. When any of these factors change during the software testing, the failure intensity function increases or decreases non-monotonically. In 1993, change-point problems in software and hardware reliability were discussed by Zhao. Recently, a few researchers (cf. Kwang, 2001; Zou, 2003; Huang, 2005b; Kapur et al., 2006; Lin and Huang, 2008) proposed that the software testing may be changed at some time point. They noticed that SRGMs with change-point achieve a great improvement in the accuracy of evaluation of software
reliability. The estimation of parameters for NHPP based SRGM with change-point was
done by Chang (2006). Zhao et al. (2006) examined the environmental effects on the
software reliability growth model with change-point. The notable works in this field was
done by Kapur et al. (2007, 2008b). They analyzed the effect of change-point on the
errors on different severity and used the testing time as power function.

It is very important for software developer to control a software development
process in terms of cost, reliability and optimal release time. The software is released in
the market for operational use when it is tested perfectly. But a major problem for
software developers is to decide at what time, the testing should be stopped and software
should be released so that the total cost can be minimized subject to reliability constraint.
The software release time problem of software is also of vital importance from the view
point of economics. This problem has been discussed from time to time by many
researchers under different constraints/policy. In literature, some research papers on
optimization problem have also appeared. Kimura et al. (1999) illustrated the economic
analysis of software release problem with warranty cost and reliability requirement.
Zheng (2002) suggested release policies by considering the reliability constraint for
software testing time.

One of the key issues of software engineering is to manage the process of testing
under a limited budget in a specified time interval. In this chapter, we investigate how to
integrate change-point concept into imperfect debugging software reliability growth
model (SRGM) with warranty cost. We suggest the optimal release policies under cost
and reliability constraints. The remaining part of this chapter is organized as follows;
Section 5.2 is devoted to the mathematical formulation of non-homogeneous software
reliability growth model (SRGM) with imperfect debugging and change-point concepts.
In section 5.3, we study the warranty cost model. To measure the parameters of the
proposed model, maximum likelihood function is given in section 5.4. We discuss
optimum release policies based on cost and reliability criteria in section 5.5. To examine
the behavior of parameters which have most significant influence on optimal results, the
sensitivity analysis is performed in section 5.6. Finally concluding remarks are given in section 5.7.

5.2 NHPP Model

We develop software reliability model which follows a non-homogeneous poisson process (NHPP) for the behavior of the mean of the cumulative number of detected faults. Let \( \{N(t), t \geq 0\} \) be a counting process that represents the cumulative number of faults detected up to testing time \( t \) with mean value function \( m(t) \). The SRGM based on the NHPP can be formulated as:

\[
\text{prob}[N(t) = n] = \frac{[m(t)]^n e^{-m(t)}}{n!}, \quad n = 0,1,2,3,\ldots
\]  

\[
\tag{5.1}
\]

The mean value function is assumed to be a non decreasing function in testing time with boundary conditions \( m(0) = 0 \) and \( m(\infty) = a \), where ‘\( a \)’ is the initial expected error content function.

5.2.1 SRGM with Imperfect Debugging and Change-Point

Most of the software reliability growth models focus on the software testing phase, where software defects are detected, isolated and removed and then software tends to grow. The point at which fault detection/introduction rate is changed, is called change-point. In this section, a SRGM is developed which incorporates both imperfect debugging and change-point concepts. It is assumed that during the software testing, fault detection rate and fault introduction rate change at some instant \( \tau \).

The fault detection rate function with change-point is defined as

\[
b(t) = \begin{cases} 
  b_1, & \text{for } 0 \leq t \leq \tau \\
  b_2, & \text{for } t > \tau 
\end{cases}
\]

\[
\tag{5.2}
\]
The fault introduction rate during testing with change-point is defined as

$$\beta(t) = \begin{cases} \beta_1, & \text{for } 0 \leq t \leq \tau \\ \beta_2, & \text{for } t > \tau \end{cases} \quad \text{...(5.3)}$$

The proposed model is based on the following assumptions:

- The fault removal phenomenon is modeled by non-homogeneous Poisson process (NHPP).
- The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
- The fault detection rate is constant over time.
- The time between $\text{(i-1)}^{\text{th}}$ and $i^{\text{th}}$ failures depends on the time to the $\text{(i-1)}^{\text{th}}$ failure.
- On the detection/removal of a software failure, it may be possible that the effort to remove the failure may not be perfect i.e. due to imperfect debugging a new fault may be introduced during removal of the error.
- The total number of faults at the beginning of the testing phase is finite i.e. the imperfect error debugging does not increase the initial error content.

These are the notations which have been used to formulate the model:

- $m(t)$ : Mean value function in the NHPP model.
- $\lambda(t)$ : Failure intensity function.
- $a$ : Initial number of errors in the software before starting of the testing phase.
- $b$ : Fault detection rate.
- $\beta$ : Fault introduction rate.
- $C_0$ : Initial testing cost.
- $C_t$ : Testing cost per unit time.
- $C_w$ : Maintenance cost per fault during the warranty period.
Chapter-5: Warranty Cost Analysis of Software Reliability…

\[ T \] : Release time of the software.

\[ T^* \] : Optimal release time of the software.

\[ T_w \] : Warranty period.

\[ \alpha \] : Discount rate of the cost.

EC(T) : Expected total maintenance cost of the software.

\[ C_w(T) \] : Maintenance cost during the warranty period.

The mean value function \( m(t) \) can be obtained as [cf. Shyur, 2003]:

\[
m(t) = \begin{cases} 
\frac{a}{1-b_1} [1 - \exp(-(1-b_1)b_1 t)], & 0 \leq t \leq \tau \\
\frac{a}{1-b_2} [1 - \exp(-(1-b_1)b_1 \tau -(1-b_2)b_2 (t-\tau))] + \frac{m(\tau)(b_1-b_2)}{1-b_2}, & t > \tau 
\end{cases}
\]

The corresponding failure intensity function is

\[
\lambda(t) = \frac{dm(t)}{dt} = \begin{cases} 
ab_1[\exp(-(1-b_1)b_1 t)], & 0 \leq t \leq \tau \\
ab_2[\exp(-(1-b_1)b_1 \tau -(1-b_2)b_2 (t-\tau))], & t > \tau 
\end{cases}
\]

\textbf{5.3 Warranty Cost Model}

Software testing is divided into the testing phase and operational/warranty phase. It is evident that the cost of remaining an error in operational phase is more expensive than the testing phase because during the debugging process some new type of faults may be introduced into the software in the operational phase.

When the cost of the software development is estimated, the software developers have to consider the cost of after-sales support. This is known as the warranty cost. The computation of this warranty cost is dependent on the release time of the software.
Now, we discuss the optimal testing time by considering the development and warranty cost with the discount rate. The discount maintenance cost is considered to take care of the present value of the money. First of all we construct cost model for the software by assuming that there are three types of costs i.e., (i) an initial testing cost, (ii) testing cost per unit time and (iii) the maintenance cost during the warranty period.

Hence the total expected software maintenance cost is given by

\[
EC(T) = C_0 + C_i \int_0^T \exp(-\alpha T) dt + C_w(T) \quad \ldots(5.6)
\]

There are two cases for maintenance cost during warranty period:

**Case I**: In this case, it is assumed that during the warranty period, the software reliability growth does not occur. Here we assume that only minor errors that will not affect the reliability of the software, can be corrected. Then maintenance cost during warranty is obtained as

\[
C_w(T) = \begin{cases} 
C_w \int_T^{T+T_w} \lambda(T) \exp(-\alpha t) dt, & 0 < t \leq \tau \\
C_w \int_T^{T+T_w} \lambda(T) \exp(-\alpha t) dt, & t > \tau
\end{cases}
\]

\[
= \begin{cases} 
C_w \frac{ab_1}{\alpha} \exp(-(1-\beta_1)b_1T) \exp(-\alpha T)[1 - \exp(-\alpha T_w)], & 0 \leq t \leq \tau \\
C_w \frac{ab_2}{\alpha} \exp(-(1-\beta_1)b_1\tau - (1-\beta_2)b_2(T-\tau)) \exp(-\alpha T)[1 - \exp(-\alpha T_w)], & t > \tau
\end{cases}
\]

Now, using eqs (5.6)-(5.7), we get

(i) \[
EC(T) = C_0 + C_i \int_0^T \exp(-\alpha t) dt + C_w \frac{ab_1}{\alpha} \exp(-(1-\beta_1)b_1T) \\
\exp(-\alpha T)[1 - \exp(-\alpha T_w)], \quad \text{for} \ 0 \leq t \leq \tau
\]

\ldots(5.8)
(ii) \[ EC(T) = C_0 + C_i \int_0^T \ exp(-\alpha t)\,dt + C_w \frac{ab_2}{\alpha} \exp(- (1-\beta_1)b_1 \tau - (1-\beta_2)b_2 (T-\tau)) \]
\[ \exp(-\alpha T)[1-\exp(-\alpha T_w)], \quad \text{for } t > 0 \]...

Differentiating equation (5.8) with respect to ‘T’ and equating to zero, where \((0 \leq t \leq \tau)\), we get
\[ T = T_1 = \frac{1}{b_1(1-\beta_1)} \ln \frac{ab_1}{\alpha} \frac{C_w}{C_i} \left[ \alpha + b_1(1-\beta_1) \right] [1-\exp(-\alpha T_w)] \]...

And differentiating equation (5.9) with respect to ‘T’ and equating to zero, where \((t > \tau)\), we have
\[ T = T_2 = \frac{1}{b_2(1-\beta_2)} \ln \frac{ab_2}{\alpha} \frac{C_w}{C_i} \exp(-\tau [(1-\beta_1)b_1 - (1-\beta_2)b_2] [\alpha + b_2(1-\beta_2)] [1-\exp(-\alpha T_w)] \]

Since \( \left[ \frac{d^2 EC(T)}{dT^2} \right]_{T=T_1} > 0 \), and \( \left[ \frac{d^2 EC(T)}{dT^2} \right]_{T=T_2} > 0 \),

\( EC(T) \) has a minimum value at \( T_1 = T^* \), \( 0 < t \leq \tau \); and \( T_2 = T^* \), \( t > \tau \).

**Case II**: Here we assume that during the warranty period, the software reliability growth occurs, even after the testing phase. In this case only major errors that will improve the reliability of the software can be corrected. Then \( C_w(T) \) is given by

\[ C_w(T) = \begin{cases} 
C_w \int_0^{T^*} \lambda(t) \exp(-\alpha t)\,dt, & \text{for } 0 \leq t \leq \tau \\
C_w \int_0^{T^*} \lambda(t) \exp(-\alpha t)\,dt, & \text{for } t > \tau 
\end{cases} \]
\begin{align*}
C_w & \left[ \frac{ab_1}{(1 - \beta_1) b_1 + \alpha} \exp(- (1 - \beta_1) b_1 + \alpha) \right] T \left( 1 - \exp(-(1 - \beta_1) b_1 + \alpha) \right), \quad \text{for} \ 0 \leq t \leq \tau \\
& \left[ \frac{ab_2}{(1 - \beta_2) b_2 + \alpha} \exp(- (1 - \beta_2) b_2 + \alpha) \right] T \left( 1 - \exp(-(1 - \beta_2) b_2 + \alpha) \right), \quad \text{for} \ t > \tau 
\end{align*}

Substituting above values from eq. (5.12) in eq. (5.6), we get

(i) \( EC(T) = C_0 + C_1 \int_0^T \exp(- \alpha t) dt + C_w \left[ \frac{ab_1}{(1 - \beta_1) b_1 + \alpha} \exp(-(1 - \beta_1) b_1 + \alpha) \right] T \left( 1 - \exp(-(1 - \beta_1) b_1 + \alpha) \right), \quad \text{for} \ 0 \leq t \leq \tau \) 

\( \ldots (5.13) \)

(ii) \( EC(T) = C_0 + C_1 \int_0^T \exp(- \alpha t) dt + C_w \left[ \frac{ab_2}{(1 - \beta_2) b_2 + \alpha} \exp(-(1 - \beta_2) b_2 + \alpha) \right] T \left( 1 - \exp(-(1 - \beta_2) b_2 + \alpha) \right), \quad \text{for} \ t > \tau \) 

\( \ldots (5.14) \)

Differentiating equation (5.12) with respect to ‘\( T \)’ and equating to zero, where (0 \( \leq t \leq \tau \)), we get

\( T = T_3 = \frac{1}{b_1 (1 - \beta_1)} \ln \frac{C_w a b_1}{C_1} \left[ 1 - \exp(-(1 - \beta_1) b_1 + \alpha) T_w \right] \) 

\( \ldots (5.15) \)

Again, differentiating equation (5.13) with respect to ‘\( T \)’ and equating to zero, where (\( t > \tau \)), we find

\( T = T_4 = \frac{1}{b_2 (1 - \beta_2)} \ln \frac{C_w a b_2 \exp(z)}{C_1} \left[ 1 - \exp(-(1 - \beta_2) b_2 + \alpha) T_w \right] \) 

\( \ldots (5.16) \)

where \( z = -(1 - \beta_1) b_1 \tau + (1 - \beta_2) b_2 \tau \).

Again \( \left[ \frac{d^2 EC(T)}{dT^2} \right]_{T = T_3} > 0 \), and \( \left[ \frac{d^2 EC(T)}{dT^2} \right]_{T = T_4} > 0 \).
Thus EC(T) has a minimum value at $T_3 = T^*$ for $0 \leq t \leq \tau$, and $T_4 = T^*$ for $t > \tau$.

### 5.4 Warranty Cost Model with Reliability Constraint

The software reliability of the NHPP model is defined as the probability that a software failure will not occur during the testing time interval $(T, T+x]$. The software reliability function is formulated as:

$$R(x/T) = \exp[-\{m(T + x) - m(T)\}]$$

$$= \begin{cases} \exp[-\exp(-(1-\beta_i)b_iT)m_i(x)], & \text{for } 0 \leq t \leq \tau \\ \exp[-\exp(-(1-\beta_1)b_1\tau - (1-\beta_2)b_2(T-\tau))m_2(x)], & \text{for } t > \tau \end{cases} \quad \ldots(5.17)$$

where

$$m_i(x) = \frac{a}{1-\beta_i}[1-\exp(-(1-\beta_i)b_i x)], \quad i = 1, 2.$$  

Let $T_{R1}$ be the optimal release time for the case (i) for testing time $T_1 (0 \leq t \leq \tau)$ and $T_2 (t > \tau)$ satisfying the relation $R(x/T_1) = R_0$, $R(x/T_2) = R_0$, respectively. Similarly $T_{R2}$ be the optimal release time for the case (ii) for testing time $T_3 (0 \leq t \leq \tau)$ and $T_4 (t > \tau)$ satisfying the relation $R(x/T_3) = R_0$ and $R(x/T_4) = R_0$ respectively. Now, we get

$$T_{R1} = \frac{1}{b_1(1-\beta_1)} \left\{ \ln(m_1(x)) - \ln\left(\frac{1}{R_0}\right) \right\} \quad \ldots(5.18)$$

and

$$T_{R2} = \frac{1}{b_2(1-\beta_2)} \left\{ \ln(m_2(x)) + [(1-\beta_2)b_2 - (1-\beta_1)b_1]T - \ln\left(\frac{1}{R_0}\right) \right\} \quad \ldots(5.19)$$
Let $R_0 \ (0 < R_0 \leq 1)$ be the desired level of reliability and $T = T^*$ (optimum release time) which minimizes the expected cost of the software with reliability objective $R_0$. Thus, the optimization problem can be formulated as

$$\text{Minimize } EC(T),$$

subject to $R(x/T) \geq R_0$ \quad \ldots (5.20)

### 5.5 Optimal Software Release Policies

The software release time problem is basically concerned with the cost of the software. The goal of the software developers is to determine an optimum software release time satisfying both cost and reliability requirements. The optimal software release policies can be established by minimizing the total expected software cost under the specified constraints so that the achieved software reliability is not less than a pre-specified level of reliability. Such cost-reliability optimization problem can be solved to predict the optimal software release policies.

The optimal software release policies for the NHPP SRGM based on cost and reliability criteria are as follows:

**Optimal Testing Time with Cost**

(A) **Optimal Testing Policy 1 (OTP 1) for case I:**

- $P_{A.1}: \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) < \lambda(T_2)$,

  then $T^* = T_1$.

- $P_{A.2}: \lambda(0) < \lambda(T_1) \leq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_2)$,

  then $T^* = T_2$.

- $P_{A.3}: \lambda(0) < \lambda(T_1) \leq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) < \lambda(T_2)$,

  then $T^* = 0$. 

109
Chapter-5: Warranty Cost Analysis of Software Reliability…

\[ P_{A.4} : \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_2), \]

then \( T^* = \max(T_1, T_2). \)

\[ (B) \quad \text{Optimal Testing Policy 2 (OTP 2) for case (ii):} \]

\[ P_{B.1} : \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) < \lambda(T_4), \]

then \( T^* = T_3. \)

\[ P_{B.2} : \lambda(0) < \lambda(T_3) \leq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_4), \]

then \( T^* = T_4. \)

\[ P_{B.3} : \lambda(0) < \lambda(T_3) \leq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) < \lambda(T_4), \]

then \( T^* = 0. \)

\[ P_{B.4} : \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_4), \]

then \( T^* = \max(T_3, T_4). \)

\[ \text{Optimal Testing Time with Cost and Reliability Constraints} \]

\[ (C) \quad \text{Optimal Testing Policy 3 (OTP 3) for case I:} \]

\[ P_{C.1} : \text{If} \quad \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) < \lambda(T_2) \quad \text{and} \quad \text{R(x/0)} > \text{R}_0, \]

then \( T^* = T_1. \)

\[ P_{C.2} : \text{If} \quad \lambda(0) < \lambda(T_1) \leq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_2) \quad \text{and} \quad \text{R(x/0)} > \text{R}_0, \]

then \( T^* = T_2. \)

\[ P_{C.3} : \text{If} \quad \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \quad \text{and} \quad \lambda(\tau) > \lambda(T_2) \quad \text{and} \quad \text{R(x/0)} > \text{R}_0, \]

then \( T^* = \max\{T_1, T_2\}. \)
Chapter-5: Warranty Cost Analysis of Software Reliability…

\textbf{P.C.4:} If \( \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \) and \( \lambda(\tau) < \lambda(T_2) \) and \( R(x/0) > R_0 \),
then \( T^* = 0 \).

\textbf{P.C.5:} If \( \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \) and \( \lambda(\tau) < \lambda(T_2) \) and \( R(x/0) < R_0 \),
then \( T^* = \max\{T_1, T_{R1}\} \).

\textbf{P.C.6:} If \( \lambda(0) < \lambda(T_1) \leq \lambda(\tau) \) and \( \lambda(\tau) > \lambda(T_2) \) and \( R(x/0) > R_0 \),
then \( T^* = \max\{T_2, T_{R2}\} \).

\textbf{P.C.7:} If \( \lambda(0) > \lambda(T_1) \geq \lambda(\tau) \) and \( \lambda(\tau) > \lambda(T_2) \) and \( R(x/0) < R_0 \),
then \( T^* = \max\{T_1, T_2, T_{R1}\} \).

\textbf{(D)} \textbf{Optimal Testing Policy 4 (OTP 4) for case II:}

\textbf{P.D.1:} If \( \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \) and \( \lambda(\tau) < \lambda(T_4) \) and \( R(x/0) > R_0 \),
then \( T^* = T_3 \).

\textbf{P.D.2:} If \( \lambda(0) < \lambda(T_3) \leq \lambda(\tau) \) and \( \lambda(\tau) > \lambda(T_4) \) and \( R(x/0) > R_0 \),
then \( T^* = T_4 \).

\textbf{P.D.3:} If \( \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \) and \( \lambda(\tau) > \lambda(T_4) \) and \( R(x/0) > R_0 \),
then \( T^* = \max\{T_3, T_4\} \).

\textbf{P.D.4:} If \( \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \) and \( \lambda(\tau) < \lambda(T_4) \) and \( R(x/0) > R_0 \),
then \( T^* = 0 \).

\textbf{P.D.5:} If \( \lambda(0) > \lambda(T_3) \geq \lambda(\tau) \) and \( \lambda(\tau) < \lambda(T_4) \) and \( R(x/0) < R_0 \),
then \( T^* = \max\{T_3, T_{R1}\} \).
\textbf{P_D.6:} If $\lambda(0) < \lambda(T_3) \leq \lambda(\tau)$ and $\lambda(\tau) > \lambda(T_4)$ and $R(x/0) > R_0$,
then $T^* = \max\{T_4, T_{R2}\}$.

\textbf{P_D.7:} If $\lambda(0) > \lambda(T_3) \geq \lambda(\tau)$ and $\lambda(\tau) > \lambda(T_4)$ and $R(x/0) < R_0$,
then $T^* = \max\{T_3, T_4, T_{R1}\}$.

\section*{5.6 Sensitivity Analysis}

To verify the proposed software reliability growth model that incorporates both imperfect debugging and change-point concept, we perform the sensitivity analysis by computing the expected maintenance cost and software reliability. The program has been coded using 'MATLAB' to check the validity of the analytical results. The effect of various parameters on the expected cost and reliability are explored by varying the parameters namely $a$ (initial number of errors), $b_1$ and $b_2$ (error detection rates) and $\alpha$ (discount rate of cost) for both cases. Some other default parameters are fixed as $C_0 = 100$, $C_t = 5$, $C_w = 150$, $T_w = 5$, $\beta = 0.04$, $x = 0.9$.

There are two cases for describing the expected maintenance cost $EC(T)$ and reliability $R(T)$. The graphical representation of the expected maintenance cost $EC(T)$ has been done in figs (5.1)-(5.4). The reliability $R(T)$ has also been displayed in figs (5.5)-(5.7).

Figs 5.1(i)-5.1(iii) exhibit the effects of $a$, $b_1$ and $\alpha$ on $EC(T)$ for the cases 1 whereas figs 5.2(i)-5.2(iii) depict the effects of same parameters for case 2 before the change-point. Figs 5.1(i) and 5.1(ii) reveal that $EC(T)$ first decreases (approximately up to $t = 40$) and then after it increases for the testing time and finally it attains constant behavior. We notice the same trend in figs 5.2(i) and 5.2(ii) which have been drawn for case 2 before change-point. Figs 5.1(ii) and 5.2(ii) also show the same pattern for $b_1$ with respect to time for case 1 and case 2, respectively. We also notice that $EC(T)$ increases with respect to $a$ and $b_1$ for both cases. In figs 5.1(iii) and 5.2(iii), $EC(T)$ first decreases
gradually up to $t = 60$ and after that it increases sharply. It is also seen that after $t = 60$, $EC(T)$ decreases with respect to $\alpha$.

In figs 5.3(i)-5.3(iv) and figs 5.4(i)-5.4(iv), the effect of variations of parameters $a$, $b_1$, $b_2$ and $\alpha$ on $EC(T)$ for both cases 1 and 2 respectively, after change-point are plotted. We observe the similar trend for both cases as noted in figs 5.1(i)-5.1(iii) i.e. $EC(T)$ first decreases and after some time it grows rapidly with respect to testing time. We also see that, $EC(T)$ increases for high number of initial faults $a$ and high value of error detection rate $b_2$. On the other hand, for higher values of $b_1$ and $\alpha$, the expected cost $EC(T)$ decreases.

Figs (5.5)-(5.7) illustrate the pattern of the reliability of the software by varying the parameters $a$, $b_1$, $b_2$ and $\tau$ for both cases i.e. before change-point and after change-point. Figs 5.5(i) and 5.6(i) show the trend of reliability by varying the parameter $a$ before change-point and after change-point, respectively. We notice that after the change-point, the reliability increases as compared to before change-point. We also see the same pattern of reliability in figs 5.5(ii) and 5.6(ii). It can be easily seen that as initial number of faults (a) increases, the reliability increases. We notice the same effect of parameters $b_2$ and $\tau$ on $EC(T)$ in figs 5.7(i) and 5.7(ii) and in figs 5.5(ii) and 5.6(ii). Also, we found that $EC(T)$ increases as error detection rate increases. In figs 5.7(i) and 5.7(ii), reliability is plotted by varying the parameters $b_2$ and $\tau$, respectively, for after the change-point case. In these graphs, the reliability is increasing sharply as compared to before change-point case.

From the above results, it is over all concluded that there is a steep decreasing trend in the expected maintenance cost $EC(T)$ for the lower values of $T$; and after some time, the $EC(T)$ shows the linear increment for the further increased values of $T$. It is demonstrated that the expected maintenance cost $EC(T)$ is greatly affected by the testing time $T$. The reason behind the high value of $E\{C(T)\}$ in the beginning may be the initial testing cost, which is taken significantly high.
5.7 Concluding Remarks

The software reliability growth model developed in this investigation includes the imperfect debugging phenomenon as well as change-point concepts. The software cost subject to reliability constraint discussed may provide an insight to achieve maximum reliability within a given budget. It is noticed that if the cost could be less than the given budget and the customers are satisfied, then it is a great profit for an organization. The proposed model and findings have been validated by taking numerical illustration, which demonstrates the applications of investigation done for different types of software and large-scale real time embedded systems.
Fig. 5.1: Expected maintenance cost Vs T for case 1 before change-point for different values of (i) a (ii) b1 (iii) \( \alpha \).

Fig. 5.2: Expected maintenance cost Vs T for case 2 before change-point for different values of (i) a (ii) b1 (iii) \( \alpha \).
Fig. 5.3: Expected maintenance cost Vs T for case 1 after change-point for different values of (i) a, (ii) b₁, (iii) b₂, and (iv) α.
Fig. 5.4: Expected maintenance cost Vs T for case 2 after change-point for different values of (i) $a$ (ii) $b_1$ (iii) $b_2$ and (iv) $\alpha$. 
Fig. 5.5: Software reliability Vs $T$ before change point for different values of (i) $\alpha$ (ii) $b_1$.

Fig. 5.6: Software reliability Vs $T$ after change point for different values of (i) $\alpha$ (ii) $b_1$.

Fig. 5.7: Software reliability Vs $T$ after change-point for different values of (i) $b_2$ (ii) $\tau$. 

118