This chapter is concerned with software reliability growth model (SRGM) to study the testing policy and operational reliability of the system in a distributed development environment. Distributed software is developed by a different team of software engineering who is also responsible to resolve the complex issues related to the quality and reliability of the software. In the present study, we assume that the software system consists of a finite number of reused components and newly developed components. For the reused components we do not consider software reliability growth phenomenon because of the effect of severity of the faults whereas, for the newly developed components we consider this phenomenon. A time dependent correction lag function is used for correcting the detected and isolated errors. The total expected delivery cost and software reliability are evaluated during the (i) testing phase and (ii) operational phase. In addition, optimal software release policies to decide when to stop testing of a software system and transfer it to customer are also proposed. Numerical results for the proposed model are calculated analytically as well as by using Adaptive Network-based Fuzzy Interference Systems (ANFIS) approach. The potential of real time software applications has been discussed which also highlight the new direction for the future research.

7.1 Introduction

Recent advancement in software technologies has effectively promoted the growth of computer-related applications in many fields of our daily life. These technologies are the basis of many changes in communication systems, manufacturing system, transportation and many other areas. Computer information products and services have become an indispensable and still rapidly growing element of global economy. Software development processes have become more distributed because of the strong global competition. The reliability of the computer software not only becomes more important, but faults in software design become more subtle. Commercial software developers attempt to make their software products more
popular than even before in the market by selling more and more pieces of their product. In the last few decades customers have become more demanding in terms of cost, schedule, reliability and quality. These all parameters can be resolved by optimum design of the software using well established testing and reliability methodologies.

Software reliability growth modeling is a tool which can be used to evaluate the changes in reliability performance and software development status. These models incorporate relationship between the testing time and the corresponding number of faults detected or removed. From time to time a number of SRGM for distributed environment in the literature has been proposed. Dai et al. (2005) did the modeling and analysis of correlated software failure of multiple types. Kapur et al. (2005a) analyzed the flexible software reliability growth models for distributed systems. In 2008a, Kapur et al. explored the software reliability growth model depending upon flexibility and learning process.

Recently, a few researchers have emphasized on the importance of time lag problem in SRGMs. In 2001, Schneidewind explored the fault correction process. After that many other researchers have contributed towards this field. Schneidewind (2002), Huang et al. (2004), Lo and Huang (2006), Gokhale et al. (2006), Huang and Lin (2006) studied an integrated failure detection and fault correction model with various debugging time lags for the software reliability models. Wu et al. (2007) and Xie et al. (2007) proposed the modeling and analysis of software fault-detection and fault-correction processes. Singh et al. (2007) considered the fault dependency concept with debugging time lag in software reliability growth modeling using a power function of testing time. Huang and Huang (2008) testified the feature of correction lag function by experimental results using finite and infinite queueing model. Shu et al. (2009) modeled the software fault detection and correction processes based on the correction lag.

The optimal software release time is that when customer gets the software at minimum cost and high level of reliability. In 1980, Okumoto and Goel proposed the optimum release time for software systems based on reliability and cost criteria. Dohi et al. (1997) described the optimal software release policies with debugging time lag. Yang and Xie (2000) considered the release time in the operational phase. Gokhale
et al., (2004) suggested the optimal release policies based on economic analysis considering cost, reliability, testing efficiency, testing effort. Cost models with different criteria were proposed by Pham and Zhang (2003). Thirumurugan and Williams (2007) proposed the analysis of testing and operational software reliability in SRGM based on NHPP.

Several SRGMs have been analyzed for the distributed software system but none has addressed the context of correction lag function in the distributed software system. Our study fills this gap. In this chapter, optimal release time has been determined for different reliability concepts (Testing phase and Operational phase) using correction lag phenomenon in software reliability growth models. The rest of the chapter is organized as follows. Section 7.2 proposes new software reliability growth model based on fault detection and correction processes. Section 7.3 is devoted for modeling the total fault detection, isolation and correction processes. Testing and operational reliability are discussed in section 7.4. In section 7.5, optimal release problem based on cost and reliability constraint is analyzed in detail. To validate the analytical results, sensitivity analysis is given in section 7.6. Finally, some conclusions, future research directions and challenging issues are discussed in section 7.7.

7.2 Distributed SRGM

We consider a complex software system which is distributed on the basis of severity of the faults. It is assumed that all faults presented in the software may not be of the same type and the reliability growth of all distributed systems may not have the same impact. Software system consists of two types of components as shown in fig. 7.1. Some of them are reused components whereas remaining are newly developed components. Both types of components take into account the time delay between the failure detection, isolation and correction processes. It is also assumed that error content function is exponentially distributed. Also the testing team gains efficiency with time known as learning phenomenon of the testing team. The error detection isolation rate is dependent on the failure severity which are different from each on other from the testing point of view in both cases of reused and newly developed components.
For modeling purpose, we made the following assumptions:

- The software system consists of a finite number of reused and newly developed components.
- The fault detection, isolation, and correction phenomenon are governed by the non-homogeneous poisson process (NHPP).
- All faults are independent and detectable.
- The software system is subject to failure during execution caused by the faults remaining in the software system.
- When a failure occurs, it is not corrected immediately because of the correction lag in the debugging process.
- During the fault detection, isolation, and correction process, no new faults are introduced into the system.
- The software failure detection and isolation rates at any time are affected by the number of faults remaining in the software system.
- The time delays between the faults detection, isolation and its subsequent correction processes are assumed to represent the severing of faults, i.e., the more severe faults, the more delay in time.
These are the notations which have been used for formulating the model:

\[\begin{align*}
m(T) & : \text{Mean value function.} \\
m(T_{LC}) & : \text{Software life cycle length.} \\
b_i(t) & : \text{Error detection rate of type } i\text{th reused component.} \\
b_j(t) & : \text{Error detection/isolation rate of type } j\text{th newly developed component.} \\
b_k(t) & : \text{Error detection/isolation rate of type } k\text{th newly developed component.} \\
a_i(t) & : \text{Expected number of initial faults of } i\text{th reused components.} \\
a_j(t) & : \text{Expected number of initial faults of } j\text{th newly developed components.} \\
a_k(t) & : \text{Expected number of initial faults of } k\text{th newly developed components.} \\
m^i_d(t) & : \text{Expected number of faults of } i\text{th reused component detected in time interval } (0, t]. \\
m^j_d(t) & : \text{Expected number of faults of } j\text{th newly developed component detected in time interval } (0, t]. \\
m^k_d(t) & : \text{Expected number of faults of } k\text{th newly developed component detected in time interval } (0, t]. \\
m^i_l(t) & : \text{Expected number of isolated faults of } j\text{th newly developed component in time interval } (0, t]. \\
m^k_l(t) & : \text{Expected number of isolated faults of } k\text{th newly developed component in time interval } (0, t]. \\
\phi^i(t) & : \text{The number of remaining uncorrected faults at time } t \text{ for the } i\text{th reused components.} \\
\phi^j(t) & : \text{The number of remaining uncorrected faults at time } t \text{ for the } j\text{th newly developed components.} \\
\phi^k(t) & : \text{The number of remaining uncorrected faults at time } t \text{ for the } k\text{th newly developed components.} \\
m^i_c(t) & : \text{The expected number of faults corrected for the } i\text{th reused components.} \\
m^j_c(t) & : \text{The expected number of faults corrected for the } j\text{th newly developed components.} \\
m^k_c(t) & : \text{The expected number of faults corrected for the } k\text{th newly developed components.} \\
\alpha & : \text{Constant rate of error generation, where } 0 < \alpha < 1.
\end{align*}\]
Chapter-7: Testing and Operational Reliability …

\( \beta \) : Constant parameter in the logistic function.
\( s \) : Shape parameter of correction lag function.
\( \theta \) : Scale parameter of correction lag function.
\( P \) : Additional fraction of observed faults during testing.
\( EC(T) \) : Expected cost of the software development.
\( C_0 \) : Total expected cost per unit time of testing.
\( C_1 \) : Cost of removing a fault before release (during testing).
\( C_2 \) : Cost of removing a fault before release (during operational phase, \( C_2 > C_1 \)).

7.2.1 Correction Lag Function

Basically correction lag is the difference between the number of detected faults and the number of corrected faults. In this study, it is also assumed that the correction lag is the difference between the number of isolated faults and the number of corrected faults. One common assumption of conventional software reliability growth model is that detected faults are immediately corrected but in realistic situation the removal of the detected faults depends on many factors such as complexity of the faults, experience, skill of the personal, the size of debugging team, etc.. Detecting and isolating an error is one thing of software and correcting them is another. Generally when a fault is corrected after its detection, the number of corrected faults depend on the number of detected faults. Often there is a time delay between these two processes. Sometimes because of the complexity of the faults, all detected faults can not be corrected immediately so the correction lag cannot be ignored. We can use both the processes together. To model this concept of time lag, the gamma curve is used. The correction lag is assumed to be distributed according the severity of the faults and is given by

\[
\phi(t) = m_d(t) - m_c(t) = m_i(t) - m_c(t) = v t^{s-1} \exp\left( - \frac{t}{\theta} \right) \quad \cdots (7.1)
\]

where \( v = \frac{1}{\beta^s \alpha^s} \).
For reused components correction lag function is

\[ \phi^i(t) = \nu^i t^{i-1} \exp\left[-\left(\frac{t}{\theta^i}\right)^\nu\right] \] ... (7.2)

For newly developed component correction lag function is

\[
\begin{align*}
\phi^i(t) &= \nu^i t^{i-1} \exp\left[-\left(\frac{t}{\theta^i}\right)^\nu\right] \\
\phi^k(t) &= \nu^k t^{k-1} \exp\left[-\left(\frac{t}{\theta^k}\right)^\nu\right]
\end{align*}
\] ... (7.3)

where \( \phi^i(t) \) denotes the correction lag function for hard faults and \( \phi^k(t) \) denotes the correction lag function for complex faults.

### 7.2.2 Fault Correction Phenomenon of Reused Components \((1 \leq i \leq P)\)

It is assumed that there are \( P \) faults in reused components in the complete software system. Reused components are treated as simple and they are corrected in two stages. First these faults are detected and then corrected. The correction lag is considered as the time duration between the fault detection and correction phenomenon. Hence the fault correction of reused components is given by:

\[ \phi(t) = m^i_b(t) - m^i_c(t) = \nu^i t^{i-1} \exp\left[-\left(\frac{t}{\theta^i}\right)^\nu\right] \] ... (7.4)

According to the assumptions, the SRGM with correction lag for reused \((i^{th})\) components, i.e., simple faults can be formulated as

\[ \frac{dm^i_b(t)}{dt} = b^i(t)\left[a^i(t) - m^i_c(t)\right] \] ... (7.5)

\[ m^i_c(t) = m^i_b(t) - \nu^i t^{i-1} \exp\left[-\left(\frac{t}{\theta^i}\right)^\nu\right] \] ... (7.6)

\[ b^i(t) = b^i, \quad a^i(t) = \alpha \exp(\beta t) \] ... (7.7)
Eq. (7.5) shows the fault detection process. Substituting the value of \( m^i_c(t) \) in eq. (7.5) and solving under the boundary condition \( m^i_d(0) = 0 \), we get

\[
m^i_d(t) = \frac{b'\alpha}{(\beta + b')} \left[ \exp(\beta t) - \exp(-b't) \right] + b'\nu' \sum_{n=0}^{\infty} \frac{\left( b' - \frac{1}{\theta'} \right)^n t^n \exp(-b't)}{n!(n+s)} \quad \ldots \text{(7.8)}
\]

From eqs (7.6) and (7.8), we get the expected number of faults corrected for \( i^{th} \) reused components as:

\[
m^i_c(t) = \frac{b'\alpha}{(\beta + b')} \left[ \exp(\beta t) - \exp(-b't) \right] + b'\nu' \sum_{n=0}^{\infty} \frac{\left( b' - \frac{1}{\theta'} \right)^n t^n \exp(-b't)}{n!(n+s)} - \nu' t^{s-1} \exp\left\{ \left( \frac{t}{\theta'} \right) \right\} 
\]

\[
\ldots \text{(7.9)}
\]

### 7.2.3 Modeling of Newly Developed Components

The two types of faults (hard and complex) are considered for the newly developed components. These faults are corrected into different stages according to their severity. The detection, isolation, correction, and correction lag function depend on the severity of the faults.

#### 7.2.3.1 Fault Correction of Hard Faults of Newly Developed Components

\((P + 1 \leq j \leq P + Q)\)

It is assumed that these faults are corrected in three stages. First these faults are detected then isolated and finally corrected. In this case, the testing team has to spend more time to analyze the behavior of the faults than the simple faults. The correction lag is used for both phenomenon detection and isolation. Hence the fault correction phenomenon for hard faults are as follows:

\[
\phi^i(t) = m^i_h(t) - m^i_c(t) = \nu' t^{s-1} \exp\left\{ \left( \frac{t}{\theta'} \right) \right\} 
\]

\[
\phi^i(t) = m^i_i(t) - m^i_c(t) = \nu' t^{s-1} \exp\left\{ \left( \frac{t}{\theta'} \right) \right\} 
\]

\[
\ldots \text{(7.10)}
\]
According to the assumptions, the SRGM with correction lag for newly developed 
(j\textsuperscript{th}) components, hard faults can be formulated as

\[
\frac{dm^j_i(t)}{dt} = b^i(t)[a^j_i(t) - m^j_i(t)] \quad \ldots (7.11)
\]

\[
\frac{dm^j_i(t)}{dt} = b^i(t)[m^j_i(t) - m^j_i(t)] \quad \ldots (7.12)
\]

\[
b^i(t) = b^i, \quad a^j_i(t) = \alpha \exp(\beta t) \quad \ldots (7.13)
\]

Eqs (7.11) and (7.12) show the fault detection and isolation processes, respectively.

Solving the above differential equations under the boundary condition 
\(m^j_i(0) = 0\) and 
\(m^j_i(0) = 0\), we get the expected number of detected faults for j\textsuperscript{th} newly developed components as

\[
m^j_i(t) = \frac{b^i\alpha}{(\beta + b^i)} \left[ \exp(\beta t) - \exp\left\{ - \left( b^i \right) t \right\} \right] + b^i \sum_{n=0}^{\infty} \frac{\left( b^j - \frac{1}{\theta^j} \right)^n t^{n+s} \exp\left\{ - b^i t \right\}}{n!(n+s)}
\]

\[
\ldots (7.14)
\]

Now, after putting the value of \(m^j_i(t)\) in eq. (7.12), the expected number of faults isolated for the j\textsuperscript{th} newly developed components are

\[
m^j_i(t) = \frac{(b^j)^\gamma \alpha}{(\beta + b^j)^\gamma} \left[ \exp(\beta t) - \exp\left\{ - \left( b^j \right) t \left( 1 + t(\beta + b^j) \right) \right\} \right] + \sum_{l=0}^{\infty} \sum_{n=0}^{\infty} \frac{(b^j)^\gamma \alpha}{n!(n+s+l)...(n+s) !} \sum_{t=0}^{\infty} \frac{\left( b^i - \frac{1}{\theta^i} \right)^n t^{n+s+l} \exp\left\{ - b^i t \right\}}{n!(n+s+l)...(n+s)}
\]

\[
\ldots (7.15)
\]

From eqs (7.10) and (7.15), we get the expected number of faults corrected for the j\textsuperscript{th} newly developed components as:
7.2.3.2 Faults Correction of Complex Faults of Newly Developed Components, 

\((P + Q + 1 \leq k \leq P + Q + R)\)

Complex faults are corrected in multi stages. First these faults are detected then isolated and finally corrected. These faults need more efforts for correcting and isolating as such require many stages. These faults need greater time lag between failure detection and isolation process because of its complexity. Hence the fault correction process for \(k^{th}\) newly developed components is given by:

\[
\phi^k(t) = m^k_D(t) - m^k_C(t) = \nu^k t^{s-1} \exp \left[ \left( \frac{t}{\theta^k} \right) \right]
\]

\[
\phi^k(t) = m^k_D(t) - m^k_C(t) = \nu^k t^{s-1} \exp \left[ \left( \frac{t}{\theta^k} \right) \right]
\]

where \(l = 1, 2, \ldots, n\)

According to the assumptions, the SRGM with correction lag for newly developed \((k^{th})\) components, i.e., complex faults can be formulated as

\[
\frac{dm^k_D(t)}{dt} = b^k(t)[a^k(t) - m^k_C(t)]
\]

\[
\frac{dm^k_C(t)}{dt} = b^k(t)[m^k_C(t) - m^k_C(t)]
\]

\[
b^k(t) = b^k, \ a^k(t) = \alpha \exp(\beta t)
\]
Chapter-7: Testing and Operational Reliability ....

Eqs (7.18) and (7.19) show the fault detection and multi stage isolation processes, respectively. After solving these equations under the boundary conditions

\[ m^k_b(0) = m^k_{b1}(0) = m^k_{b2}(0) = 0 = \ldots \ldots m^k_l(0) = 0, \]

The expected number of detected faults for the \( k \)th newly developed components is obtained as

\[
m^k_b(t) = \frac{b^k \alpha}{(\beta + b^k)} \left[ \exp(\beta t) - \exp\left\{ (b^k t) \right\} \right] + b^k \sum_{n=0}^\infty \frac{\left( b^k - \frac{1}{\theta^k} \right)^n t^{n+s} \exp\left( - b^k t \right)}{n!(n+s)}
\]

\[
\ldots (7.20)
\]

Now, expected number of faults isolated for the \( k \)th newly developed components is

\[
m^k_i(t) = \sum_{l=1}^m \left[ \frac{(b^k)^{l+1}}{(\beta + b^k)^{l+1}} \left\{ \exp(\beta t) - \exp\left\{ (b^k t) \right\} \right\} \left\{ 1 + \sum_{C=1}^l \frac{(t(\beta + b^k))^C}{C!} \right\} \right]

+ \sum_{d=0}^l \sum_{n=0}^\infty \left( b^k \right)^{d+1} \nu^k \frac{\left( b^k - \frac{1}{\theta^k} \right)^n t^{n+s+d} \exp\left( - b^k t \right)}{n! \prod_{i=0}^d (n+s+i)!}
\]

\[
\ldots (7.21)
\]

From eqs (7.17) and (7.21), we get the expected number of faults corrected for the \( k \)th newly developed components as

\[
m^k_c(t) = \sum_{l=1}^m \left[ \frac{(b^k)^{l+1}}{(\beta + b^k)^{l+1}} \left\{ \exp(\beta t) - \exp\left\{ (b^k t) \right\} \right\} \left\{ 1 + \sum_{C=1}^l \frac{(t(\beta + b^k))^C}{C!} \right\} \right]

+ \sum_{d=0}^l \sum_{n=0}^\infty \left( b^k \right)^{d+1} \nu^k \frac{\left( b^k - \frac{1}{\theta^k} \right)^n t^{n+s+d} \exp\left( - b^k t \right)}{n! \prod_{i=0}^d (n+s+i)!} - \nu^k t^{t-1} \exp\left[ \left( t/\theta^k \right) \right]
\]

\[
\ldots (7.22)
\]
7.3 Total Fault Phenomenon

We consider that the software system contains P (simple faults) reused component and Q (hard faults) and R (complex faults) newly developed components. Our interest in the present investigation is to evaluate the total expected number of faults detected, isolated and corrected from each distributed component. Now we calculate the total detection, isolation and correction phenomenon for the entire software system.

7.3.1 Total fault detection phenomenon

The mean value function for the fault detected in all components i.e. $i^{th}$ reused and $(j^{th}$ and $k^{th}$) newly developed components for the superposed NHPP is:

$$m_D(t) = \sum_{i=1}^{P} m_i^D(t) + \sum_{j=P+1}^{P+Q} m_j^D(t) + \sum_{k=P+Q+1}^{P+Q+R} m_k^D(t) \quad \ldots (7.23)$$

$$m_D(t) = \sum_{i=1}^{P} \frac{b^i\alpha}{(\beta+b^i)} \left[ \exp(\beta t) - \exp( - (b^i t)) \right] + b^i\nu \sum_{n=0}^{\infty} \frac{\left( b^i - \frac{1}{\theta^i} \right)^n}{n!(n+s)} t^{n+s} \exp(-b^i t)$$

$$+ \sum_{j=P+1}^{P+Q} \frac{b^j\alpha}{(\beta+b^j)} \left[ \exp(\beta t) - \exp( - (b^j t)) \right] + b^j\nu \sum_{n=0}^{\infty} \frac{\left( b^j - \frac{1}{\theta^j} \right)^n}{n!(n+s)} t^{n+s} \exp(-b^j t) \quad \ldots (7.24)$$

$$+ \sum_{k=P+Q+1}^{P+Q+R} \frac{b^k\alpha}{(\beta+b^k)} \left[ \exp(\beta t) - \exp( - (b^k t)) \right] + b^k\nu \sum_{n=0}^{\infty} \frac{\left( b^k - \frac{1}{\theta^k} \right)^n}{n!(n+s)} t^{n+s} \exp(-b^k t)$$

Also failure intensity function for the expected number of total faults detected is obtained as

$$\lambda_D(t) = \frac{d}{dt} m_D(t) \quad \ldots (7.25)$$
\( \lambda_D(t) = \sum_{i=1}^{P} \left[ b^i \alpha' \beta \exp(\beta t) + b^i \exp(-b^i t) \right] + b' \sum_{n=0}^{\infty} \frac{\left( b^i - \frac{1}{\theta^j} \right)^n}{n!} \exp(-b't) \left[ \frac{(n+s)}{t} - b^i \right] \)

\[ + \sum_{j=P+1}^{P+Q} \frac{b^j \alpha' \beta \exp(\beta t) + b^j \exp(-b^j t)}{(\beta + b^j)} b^j \sum_{n=0}^{\infty} \frac{\left( b^j - \frac{1}{\theta^j} \right)^n}{n! (n+s)} \exp(-b^j t) \left[ \frac{(n+s)}{t} - b^j \right] \]

\[ + \sum_{k=P+Q+1}^{P+Q+R} \frac{b^k \alpha' \beta \exp(\beta t) + b^k \exp(-b^k t)}{(\beta + b^k)} b^k \sum_{n=0}^{\infty} \frac{\left( b^k - \frac{1}{\theta^k} \right)^n}{n! (n+s)} \exp(-b^k t) \left[ \frac{(n+s)}{t} - b^k \right] \]

\[ \cdots \quad (7.26) \]

### 7.3.2 Total fault isolation phenomenon

Faults are isolated in newly developed (j and k\(^{th}\)) components not in the reused (i\(^{th}\)) components. So the mean value function of the superposed NHPP is given as:

\[ m_1(t) = \sum_{j=P+1}^{P+Q} m_1^{(j)}(t) + \sum_{k=P+Q+1}^{P+Q+R} m_1^{(k)}(t) \quad \cdots \quad (7.27) \]

\[ m_1(t) = \sum_{j=P+1}^{P+Q} \frac{\left( b^j \right)^2 \alpha'}{(\beta + b^j)^2} \left( \exp(\beta t) - \exp\left( (b^j t)(1 + t(\beta + b^j)) \right) \right) \]

\[ + \sum_{j=0}^{P+Q} \sum_{n=0}^{\infty} \frac{\left( b^j \right)^{j+1} \left( b^j - \frac{1}{\theta^j} \right)^n}{n!} \prod_{i=0}^{j} \left( n+s+i \right)! \exp(-b't) \]

\[ + \sum_{k=P+Q+1}^{P+Q+R} \sum_{i=1}^{m} \frac{\left( b^k \right)^{i+1}}{(\beta + b^k)^{i+1}} \left( \exp(\beta t) - \exp\left( -b^k t \right) \left( 1 + \sum_{C=i}^{\infty} \frac{t(\beta + b^k)^C}{C!} \right) \right) \]

\[ + \sum_{k=0}^{\infty} \sum_{n=0}^{\infty} \left( b^k \right)^{k+1} \frac{\left( b^k - \frac{1}{\theta^k} \right)^n}{n!} \prod_{i=0}^{k} \left( n+s+i \right)! \exp(-b^k t) \quad \cdots \quad (7.28) \]
The failure intensity function for total isolated faults is obtained as

\[
\lambda_i(t) = \frac{d}{dt} m_i(t) \quad \text{... (7.29)}
\]

\[
= \sum_{j=P+1}^{P+Q} \left( \frac{b_j}{b_j + b_j} \right) \beta \exp(\beta t) - \exp\left\{ - \left( \frac{b_j}{b_j + b_j} \right) \right\} \left( \beta + b_j \right) - b_j \left( 1 + t \left( \beta + b_j \right) \right) \\
+ \sum_{l=0}^{1} \sum_{n=0}^{\infty} \left( \frac{b_j}{b_j + b_j} \right)^{n+l} \left( \frac{1}{\theta} \right) t^{n+l} \exp\left( - \frac{b_j}{\theta} t \right) \left[ \frac{(n+s+l)}{t} - b_j \right] \\
+ \sum_{k=P+Q+1}^{P+Q+R} \sum_{l=1}^{\infty} \left( \frac{b_k}{b_k + b_k} \right)^{l+k} \beta \exp(\beta t) - \exp\left\{ - \left( \frac{b_k}{b_k + b_k} \right) \right\} \left( \beta + b_k \right) - b_k \left( 1 + t \left( \beta + b_k \right) \right) \\
+ \sum_{d=0}^{1} \sum_{n=0}^{\infty} \left( \frac{b_d}{b_d + b_d} \right)^{d+n} \left( \frac{1}{\theta} \right) t^{n+d} \exp\left( - \frac{b_d}{\theta} t \right) \left[ \frac{(n+s+d)}{t} - b_d \right] 
\]

\[
\quad \text{... (7.30)}
\]

### 7.3.3 Total fault correction phenomenon

In this case, the faults are corrected in all the components (reused and newly developed) of software according to their severity. Thus the mean value function is given by

\[
m_c(t) = \sum_{i=1}^{P} m_c^i(t) + \sum_{j=P+1}^{P+Q} m_c^j(t) + \sum_{k=P+Q+1}^{P+Q+R} m_c^k(t) \quad \text{... (7.31)}
\]
\[
\lambda_c(t) = \frac{d}{dt} m_c(t)
\]  
\[\cdots (7.33)\]
\[
\sum_{i=1}^{p} b_i\alpha \left[ b \exp(\beta t) + b_i \exp(-b_i t) \right] + b_v \sum_{n=0}^{\infty} \frac{\left( b_i - \frac{1}{\theta^i} \right)^n}{n!(n+s)} \exp(-b_i t) t^{n+s} \left[ \frac{(n+s)}{t} - b_i^j \right]
\]

\[
v^i \exp\left(-\frac{t}{\theta^i}\right) t^{s-1} \left[ \left( s-1 \right) - \frac{1}{\theta^i} \right] + \sum_{l=0}^{p+Q} \sum_{l=0}^{m} \frac{\left( b_i^j \right)^{l+1} \alpha}{\theta^j \left( \beta + b_i^j \right)} \left[ b \exp(\beta t) - \exp(-b_i t) \right] \left[ (\beta + b_i^j) \right]^{-n} t^{n+s+l} \left[ \left( n+s+l \right) - b_i^j \right]
\]

\[
- b_i^j \left[ \left( n+s+l \right) - b_i^j \right] + \sum_{l=0}^{i} \sum_{n=0}^{\infty} \frac{\left( b_i^j \right)^{l+1} v^j \left( b_i^j - \frac{1}{\theta^i} \right)^n}{n! \prod_{i=0}^{l} (n+s+i)} t^{n+s+l} \exp(-b_i^j t) \left[ \left( n+s+l \right) - b_i^j \right]
\]

\[
\left\{ \sum_{c=1}^{l} \left( \beta + b_i^j \right)^c - b_i^j \left\{ \sum_{c=1}^{l} \left( t \left( \beta + b_i^j \right)^c \right) \right\} \right\} + \sum_{d=0}^{i} \sum_{n=0}^{\infty} \frac{\left( b_i^j \right)^{l+1} v_k \left( b_i^j - \frac{1}{\theta^i} \right)^n}{n! \prod_{i=0}^{d} (n+s+i)} t^{n+s+d} \exp(-b_i^j t) \left[ \left( n+s+d \right) - b_i^j \right] - v^i \exp\left(-\frac{t}{\theta^i}\right) t^{s-1} \left[ \left( s-1 \right) - \frac{1}{\theta^i} \right]
\]

\[
\ldots (7.34)
\]

### 7.3.4 Special Cases

If there are single component in the software i.e. \( i = 1 \), we use \( a(t) = a, b(t) = b \).

Then our model converts to Shu et al. (2009)’s model. In this case, we obtain

\[
m_d(t) = a[1 - \exp(-bt)] + b_v \exp(-bt) \sum_{n=0}^{\infty} \frac{\left( b_i^j - \frac{1}{\theta^i} \right)^n}{n!(n+s)} \exp(-b_i^j t) \quad \ldots (7.35)
\]

\[
m_c(t) = a[1 - \exp(-bt)] + b_v \exp(-bt) \sum_{n=0}^{\infty} \frac{\left( b_i^j - \frac{1}{\theta^i} \right)^n}{n!(n+s)} \exp(-b_i^j t) - v_i \exp\left(-\frac{t}{\theta^i}\right) \quad \ldots (7.36)
\]

If \( v = 0 \), i.e. correction lag does not exist in the software testing and faults are removed as soon as they are detected, i.e., \( m_d(t) = m_c(t) \), then our model matches with the classical fundamental SRGM, (Goel-Okumoto Model). In this case, we have
7.4. Testing and Operational Reliability

Software reliability is defined as the probability of failure-free software operations for a specified period in a specified environment. The software reliability can be expressed in terms of conditional probability as

\[ R_{\text{test}}(x/t) = \Pr(x_k > x \mid t_k = t) \] \hspace{1cm} \text{\ldots}(7.38)

This represents the reliability during the next failure interval of \(x\) units given the failure history during \(t\) units.

7.4.1 Testing Reliability

During the testing stage, the software developer improves the software. The testing process follows the non-homogeneous poisson process (NHPP). For any \(t \geq 0\) and \(x > 0\), we get the reliability function as

\[ R_{\text{test}}(x/t) = \exp[-\{m_c(t+x) - m_c(t)]\] \hspace{1cm} \text{\ldots}(7.39)

Here \( R_{\text{test}}(x/t) \) measures the software reliability during the testing phase and \((t + x)\) should not extend to the operational phase.

7.4.2 Operational Reliability

If the time interval \(x_k\) is in the operational phase and the software has been tested for \(t\) units and then released to the customers, then the time to next failure will follow exponential distribution with parameter \(\lambda(t)\), where \(\lambda(t)\) is the failure intensity function of the NHPP calculated at time \(t\).

Thus the operational reliability function is given by

\[ R_{\text{op}}(x/t) = \exp[-\lambda(t)x] \] \hspace{1cm} \text{\ldots}(7.40)
Chapter-7: Testing and Operational Reliability ....

7.5 Optimal Software Release Policies

In this section, we study the reliability concepts to obtain the optimal software release time. It is important for software developer to determine when to stop the testing of the software and release it in the market for operational use. If more time is spent on testing, more errors can be removed and the software leads to be more reliable but with this the expected cost of the software will also increase. On the other hand, if the testing time is too short, the expected cost of the software reduces but the software may be unreliable because of the latent faults which have not been removed. Optimal release policies help a software vendor to calculate the software release cost by taking into account the following.

- Software development cost.
- Testing reliability.
- Operational reliability.

7.5.1 Software release time based on cost criteria

Finding software faults is not only tedious job, but is also very expensive. In recent years, the costs of developing software and the correction lag during fault detection and removal phenomenon are the major expenses in a system. As software projects become larger, the rate of software defects increases geometrically. The total expected cost for software development is given by

\[
EC(T) = C_0(T) + C_1(1 + P)m_c(T) + C_2\left[m_d(T_{L_E}) + m_i(T_{L_E}) - (1 + P)m_c(T)\right] \quad \ldots (7.41)
\]

The optimal release policies (ORP) based on the cost criteria is as follows:

\[
T^* = T_1 \quad \text{when} \quad \lambda(0) > \lambda(T_1)
\]

\[
T^* = 0 \quad \text{when} \quad \lambda(0) \leq \lambda(T_1)
\]
7.5.2 Optimal release time based on cost-reliability criteria

It is important for a software developer to know when to stop testing and release the software. The optimization problems with reliability constraint can be formulated as

\[(OP) \quad \text{Minimize} \quad EC(T) \]

Subject to \( R(x/T) \geq R_0 \) \hspace{1cm} \ldots (7.42)

The above optimization problem can be reformulated as follows:

(i) The optimization problem for the testing reliability can be given as:

\[(ORP_{te}) \quad \text{Minimize} \quad EC(T) \]

subject to \( R_{te}\left(\frac{x}{T}\right) = \exp[-(m(x+T)-m(T))] \geq R_0 \) \hspace{1cm} \ldots (7.43)

(ii) The optimization problem for the operational reliability can be stated as

\[(ORP_{op}) \quad \text{Minimize} \quad EC(T) \]

Subject to \( R_{op}\left(\frac{x}{T}\right) = \exp[-\lambda_c(T)x] = R_0 \) \hspace{1cm} \ldots (7.44)

7.5.3 Software release policies based on reliability criteria

These are some notations which have been used for describing the release policies.

\( T_C \) : Software release time which minimizes \( EC(T) \), and \( T_C \geq 0 \).

\( T_{te}^R \) : Software release time which minimizes value of \( EC(T) \) and satisfies eq. (7.43), \( T_{te}^R \geq 0 \).

\( T_{op}^R \) : Software release time which minimizes value of \( EC(T) \) and satisfies eq. (7.44), \( T_{op}^R \geq 0 \).

\( T_{pe}^* \) : Optimal solution to \( P_{te} \).
Theorem 1: When $\lambda(t)$ is strictly decreasing for $t \geq 0$ then

Case 1: If $R_{op}(x/T) \geq R_0 \geq 0$ then $T_{pu}^* = T_{pu}^o = T_C$

Case 2: If $R_{te}(x/T) \geq R_0 \geq R_{\infty}(x/T)$ then $T_{pu}^* = T_C$ and $T_{pu}^o = \max(T_C, T_{op}^R)$

(a) If $T_C \geq T_{op}^R$ then $T_{pu}^* = T_{pu}^o = T_C$

(b) If $T_C < T_{op}^R$ then $T_{pu}^* = T_C < T_{pu}^o = T_{op}^R$

Case 3: If $R_{op}(x/T) < R_0$ then $T_{pu}^* = \max(T_C, T_{te}^R)$ and $T_{pu}^o = \max(T_C, T_{op}^R)$

(a) If $T_C \geq T_{op}^R$ then $T_{pu}^* = T_{pu}^o = T_C$

(b) If $T_{te}^R < T_C < T_{op}^R$ then $T_{pu}^* = T_C < T_{pu}^o = T_{op}^R$

(c) If $T_C < T_{te}^R$ then $T_{pu}^* = T_{te}^R < T_{pu}^o = T_{op}^R$

Theorem 2: When $\lambda(t)$ is first-increasing, then decreasing and $T_0$ is the inflection point of $m(t)$, then

Case 1: If $T_0 \leq T$ then $R_{op}(x/T) < R_{te}(x/T)$

Case 2: If $T_0 > T$ then let $T_1$ be the solution to $\lambda(T_1) = \lambda(T)$, $T_1 > T$. Now we have

(a) If $T_1 \geq T + x$ then $R_{op}(x/T) > R_{te}(x/T)$

(b) If $T_1 < T + x$ then

(i) $R_{op}(x/T) > R_{te}(x/T)$, if $M < 0$

(ii) $R_{op}(x/T) = R_{te}(x/T)$, if $M = 0$
(iii) \( R_{op} \left( \frac{x}{T} \right) < R_{te} \left( \frac{x}{T} \right) \) if \( M > 0 \)

where \( M = \lambda x - m(T + x) + m(T) \).

**Theorem 3:** When \( \lambda(t) \) is first-increasing, then decreasing and \( T_0 \leq T \), then

**Case 1:** If \( R_{op} \left( \frac{x}{T_0} \right) \geq R_0 \) then \( T_{p_u}^* = T_{p_q}^* = \max(T_c, T_0) \)

**Case 2:** If \( R_{te} \left( \frac{x}{T_0} \right) \geq R_0 > R_{op} \left( \frac{x}{T_0} \right) \) then

\( T_{p_u}^* = \max(T_c, T_0), T_{p_q}^* = \max(T_c, T_{op}^R) \) and

(a) If \( T_c \geq T_{op}^R \) then \( T_{p_u}^* = T_{p_q}^* = T_c \)

(b) If \( T_0 < T_c < T_{op}^R \) then \( T_{p_u}^* = T_c < T_{p_q}^* = T_{op}^R \)

(c) If \( T_c < T_0 \) then \( T_{p_u}^* = T_0 < T_{p_q}^* = T_{op}^R \)

**Case 3:** If \( R_{op} \left( \frac{x}{T_0} \right) < R_0 \) then \( T_{p_u}^* = \max(T_c, T_{te}^R), T_{p_q}^* = \max(T_c, T_{op}^R) \) and

(a) If \( T_c \geq T_{op}^R \) then \( T_{p_u}^* = T_{p_q}^* = T_c \)

(b) If \( T_{te}^R < T_c < T_{op}^R \) then \( T_{p_u}^* = T_c < T_{p_q}^* = T_{op}^R \)

(c) If \( T_c < T_{te}^R \) then \( T_{p_u}^* = T_{te}^R < T_{p_q}^* = T_{op}^R \)

### 7.6 Sensitivity Analysis

In this section, we perform computational experiment for exploring the detection, isolation, and correction process of SRGM. The numerical results obtained from analytically approach are compared with the neuro-fuzzy results by building Adaptive Network Based Fuzzy Inference System (ANFIS) in MATLAB 6.5. Neuro Fuzzy (NF) is characterized by their membership function by defining the tolerance limit for achievements. ANFIS is built by using the fuzzy toolbox of the MATLAB
Gaussian function is used for describing the membership function. For all approximations, ANFIS are trained for 10 epochs. For illustration purpose, we choose default parameters as:

<table>
<thead>
<tr>
<th>Input Variables</th>
<th>No. of membership function</th>
<th>Linguistic Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>3</td>
<td>Low Medium High</td>
</tr>
<tr>
<td>α</td>
<td>3</td>
<td>Low Medium High</td>
</tr>
<tr>
<td>(\nu_1)</td>
<td>3</td>
<td>Low Medium High</td>
</tr>
<tr>
<td>(b_1)</td>
<td>3</td>
<td>Low Medium High</td>
</tr>
<tr>
<td>(b_2)</td>
<td>3</td>
<td>Low Medium High</td>
</tr>
</tbody>
</table>

Table 7.1: Linguistic values of the membership functions for various input parameters

\[ \alpha = 0.5, \beta = 0.3, b_1 = 0.2, b_2 = 0.3, b_3 = 0.5, \theta_1 = 3, \theta_2 = 4, \theta_3 = 5, \nu_1 = 10, \nu_2 = 19, \nu_3 = 17, s = 1, p = 0.01, T_e = 200. \]

Fig. 7.2(i) shows the difference between cumulative number of errors detected and corrected whereas fig. 7.2(ii) reveals the difference between isolated and corrected faults. In figs 7.2(i-ii), the failure intensity function is plotted. The total expected cost and reliability of testing and operation phase are exhibited in figs 7.4 and 7.5, respectively. In all figures, the analytical (ANFIS) results are shown by the continuous (broken) lines.

From figs 7.2(i) and 7.2(ii), we examine the combined effect of detection and correction and isolation and correction, respectively on cumulative number of errors. For both cases, \(m_0(t)\), \(m_i(t)\) and \(m_c(t)\) initially increase slightly but after some time
increase sharply. The corresponding fuzzy membership function is drawn in fig. 7.2(iii)

In figs 7.3(i-iv), the graphs for EC(T) are plotted with respect to time t for different parameters $\alpha, \theta, b_1,$ and $b_2$, respectively. We note in fig. 7.3(i) that EC(T) first decreases till $T=11$, and after that it increases slightly. It is also seen that as $\alpha$ increases, EC(T) increases. We see the same patterns of EC(T) in both figs 7.3(ii) and 7.3(iii). Fig 7.3(iv) reveals the same pattern as seen in fig 7.3(i). The fuzzy membership functions for figs 7.3(i)-7.3(iv) is shown in fig. 7.3(v).

Figs 7.4 (i-ii)-7.7 (i-ii) exhibit the effects of testing and operational reliability for different parameters. The shapes of the corresponding membership functions for figs 7.4(i)-7.7(i) are shown in figs 7.4(ii)-7.7(ii). From these figs, it is observed that the reliability decreases as $\alpha$ and $\upsilon_1$ increase as shown in figs 7.5(i) and 7.6(i). Also reliability increases with the increase in $b_1$ and $b_2$ for both phases (testing and operational). In these figs, it is noticed that the values of testing reliability are higher than that of operational reliability for fixed values of other parameters.

In fig. 7.8, we compare the testing and operational reliability of the software. We notice that the testing reliability does not match accurately with operational reliability, i.e. the testing reliability can be achieved in lesser time as compared to the operational reliability. The optimal testing time $T^*=79.18$ for the operational reliability and corresponding expected cost EC(T) is 6923.83. After some time, $T^*$ becomes 0.94 and the corresponding cost becomes 10202.88.

Overall we can conclude that

- By increasing the failure detection rate, the software can be made more reliable.
- After testing phase, if we are able to remove most of the errors, then our software becomes more reliable.
- It is well understood that more cost is required for ensuring the reliability of the software. ANFIS provides an easy and fast solution which seems to be quite closer to analytical results.
7.7 Concluding Remarks

The decision making regarding software release time is one of the key issues for the software developer. In this chapter, SRGM is developed to demonstrate the optimal release time assuming that the testing is completed in three phases namely detection, isolation and correction according to their severity. A very realistic assumption of correction lag has been incorporated to formulate the SRGM. The impact of these two different concepts of reliability based on testing and operational phases on the optimal release time have examined. The suggested policies may be helpful in determining the optimal release time of the complex software subject to reliability requirement based on the total expected cost criteria.
Fig. 7.2: Comparison of different cumulative number of errors Vs time

Fig. 7.2 (iii): Membership functions for input parameters Vs time for mean value function.
Fig. 7.3: EC(t) Vs time for different parameters (i) $\alpha$ (ii) $\theta$ (iii) $b_1$ (iv) $b_2$.

Fig. 7.3 (v): Membership functions for input parameters for EC(t).
Chapter-7: Testing and Operational Reliability ….

Fig. 7.4 (i): Reliability by varying $\alpha$

Fig. 7.4 (ii): Membership functions for input parameter $\alpha$

Fig. 7.5 (i): Reliability by varying $\mu_1$

Fig. 7.5 (ii): Membership functions for input parameter $\mu_1$
Fig. 7.6 (i): Reliability by varying $b_1$

Fig. 7.6 (ii): Membership functions for input parameter $b_1$

Fig. 7.7 (i): Reliability by varying $b_2$

Fig. 7.7 (ii): Membership functions for input parameter $b_2$. 
Fig. 7.8: Optimal Release Time of the Software