CHAPTER 2

A DISCRETE SOFTWARE RELIABILITY GROWTH MODEL WITH LEADING AND DEPENDENT ERRORS

20 INTRODUCTION

Software reliability modelling is very important due to the fact that it is not possible to produce fault free software. The fault in the software occur due to human imperfection. These faults manifest themselves in terms of failure when the software is run. Testing phase in the software development process aims at detecting and removing these faults/errors and making the software more reliable. Thus it is very important to evaluate software reliability during testing phase, based on software error data analysis. Modes concerned with the relationship between cumulative number of error detected through software testing and time span of testing are called software reliability growth models (SRGMs). Based on non-homogeneous Poisson Process (NHPP), several SRGHs [1,4-6,8-9] have been developed to predict remaining errors in the software and to evaluate measures such as mean time between failures, software reliability etc. Moreover, each software system is developed
for a different objective and so it is not possible to develop an SRGM which can analyze failure data for all software systems. Most of the SRGMs developed use calendar time or CPU time as the unit of software error detection/removal period. However, at times the number of test runs can be a more appropriate unit of software fault detection/removal period. Such as SRGM is called a discrete SRGM and relates the number of faults detected/removed to the number of test runs during the testing phase. A test run can be a single computer test run or a series of computer test runs executed in an hour, day, week or even month. Very few discrete SRGMs, have been developed in the literature. Mostly, the error detection/removal phenomenon has been described by the exponential or s-shaped SRGMs. The s-shaped error removal phenomenon can be attributed to error depending [6] or to time lag between the failure due to an error and its subsequent removal [8]. Bittanti [1] attributes s-shapedness to increased error removal during the later part of the testing phase. None of these models, however, describes the interface between independent leading errors and errors whose removal is dependent on these leading errors.
In this chapter, we propose a new discrete SRGM assuming, the software contains two types of errors, leading and dependent. A leading error is defined as one that is immediate removed on its causing a failure. A dependent error is termed as one whose removal is delayed until the corresponding leading error is removed. The removal of a leading error helps in isolating the cause of failure of its corresponding dependent error. Applicability of the model has been shown by applying it to several software error data cited in [2].

Besides, modelling a software error detection process, it is also of utmost importance to know when to stop testing and release the software for use. Several criteria have been suggested in this regard [5, 7]. In this chapter, we also discuss a release policy for the proposed discrete SRGM by minimizing cost subject to discrete failure intensity not exceeding a specified value. We first estimate the parameters of the proposed SRGM by the method of maximum likelihood using software error data cited in [2]. Using the estimated values, we discuss the optimal release policy based on cost and intensity criteria. Results are illustrated by numerical examples.
ASSUMPTIONS

1. Software is subject to failures at random test runs caused by errors remaining in the software.
2. The error removal phenomenon is modelled by NHPP.
3. When a failure occurs, an intermediate effort is made to detect the error causing the failure and remove the error.
4. The errors in the software are divided into two categories: Leading (Independent) Faults Dependent Faults
5. The number of errors in the software is finite and is the sum of leading and dependent errors.
6. The expected discrete failure intensity for leading errors is proportional to the current remaining leading errors.
7. The expected discrete failure intensity for dependent errors is proportional to the current remaining dependent errors and the ratio of leading errors removed to the total number of errors.
8. The error removal process does not introduce any new errors in the software.
9. Software life cycle is assumed to be more than the optimal number of test cases/runs before releasing the software.
10. Software is never released without testing.

11. Corresponding to the error detection phenomenon at the manufacturer/user end, there exists an equivalent error detection phenomenon at the user/manufacturer end.

22 NOTATIONS

$\varepsilon$ : expected initial error content in the software, $\varepsilon \geq 0$.

$a_{1}a_{2}$ : expected initial leading (dependent) error content in the software, $a_{1} \geq 0$, $a_{2} \geq 0$, $a = a_{1}a_{2}$.

$b < c^{d}$ : proportionality constant for leading (dependent) errors, $0 < b \leq 1$, $0 \leq c \leq 1$.

$p$ : proportion of leading errors in the software, $0 < p \leq 1$, $a_{1} = p \cdot a$

$N$ : Test run lag between removal of leading and dependent errors, $N \geq 0$.

$m_{1}(n), m_{2}(n))$ : mean value function for leading (dependent) errors, $m_{1}(n) = 0$ for $n < 0$, $m_{1}(\infty) = a_{1}$, $m_{2}(n) = 0$ for $n < N$, $m_{2}(\infty) = a_{2}$.

$m(n)$ : $m_{1}(n) + m_{2}(n)$.

$\lambda(n)$ : failure intensity for $m(n)$ ($\lambda(0) = 0$).

$C_{1}, C_{2}$ : Cost of fixing a leading error before (after) release of the software, $C_{2}, C_{1} \geq 0$. 

66
\( C_4 \): Cost of fixing a dependent error before release of the software, \( C_4 > C_3, 0 \).

\( C_5 \): Cost of a test run.

\( C(r) \): Total expected software cost incurred during software life cycle, when the software is released after \( n \) test runs.

\( n \): Software life cycle in terms of number of runs.

\( \lambda_0 \): Desired failure intensity.

\( n^* \): Optimal number of test runs executed before releasing the software.

23 MODEL ANALYSIS

The number of leading errors removed on the \((n+1)^{th}\) test run as per assumption (6) may be expressed as

\[
m_1(n+1) = m_1(n) = b(a_1 - m_1(n)) \quad \text{...... (2.1)}
\]

Solving (2.1), under the initial condition \( m_1(0) = 0 \), we have

\[
m_1(n) = a_1(1 - (1-b)^n) \quad \text{...... (2.2)}
\]

Equation (2.2) models the leading error removal phenomenon. The failure intensity for leading errors is given by

\[
a_1 b(1-b)^n
\]

It may be noted that failure intensity for leading errors
The number of independent errors removed on the \((n+1)^{th}\) test run, according to assumption (7) is expressed as

\[
m_2(n+1) - m_2(n) = C' a_2 m_2(n) \frac{m(n+1-N)}{a} \quad \ldots \ldots \text{(2.3)}
\]

Solving (2.3), under initial condition \(m_2(0) = 0\), we get

\[
m_2(n) = a_2 \left[ 1 - I \left( \frac{Ca}{a} \left( 1-(1-b)^x \right) \right) \right] \quad \ldots \ldots \text{(2.4)}
\]

The failure intensity for dependent errors is given as

\[
a_2 \left[ 1 - \frac{Ca}{a} \left( 1-(1-b)^x \right) \right] \frac{Ca}{a} \left( 1-(1-b)^{n-N+1} \right)
\]

It may be noted that failure intensity for dependent errors increases for all \(n \ (> N)\) satisfying

\[
\frac{Ca}{a} \left[ 1-(1-b)^n \right] \left( 1-(1-b)^{n-N+1} \right) \quad b(1-b)^{n-N}
\]

and then decreases.

Now,

\[
m'(n) = m_1(n) + m_2(n)
\]

\[= a - a_1 (1-b)^n - a_2 \left[ 1 - \frac{Ca}{a} \left( 1-(1-b)^x \right) \right]
\]

for simplicity, we assume test run lag to be negligible, i.e. \(N=0\), so

\[
m(n) = a - a_1 (1-b)^n - a_2 \left[ 1 - \frac{Ca}{a} \left( 1-(1-b)^x \right) \right] \quad \ldots \ldots \text{(2.5)}
\]

\[= a \left[ 1 - p(1-b)^n - (1-p) I \left[ 1 - p \left( 1-(1-b)^x \right) \right] \right] \quad \ldots \ldots \text{(2.6)}
\]
equation (2.5) or (2.6) represents the expected number of errors removed in test runs.

The failure intensity for \( m(n) \) is

\[
\lambda(n+1) = m(n+1) - m(n)
\]

\[
= a \cdot b(1-b)^n + \frac{Ca^2}{a} \left(1-(1-b)^{n+1}\right)
\]

\[
\sum_{x=0}^{n} \left(1 - \frac{Ca^4}{a} \left(1-(1-b)^x\right)\right) \quad \ldots \ldots (2.7)
\]

It may be noted that either \( \lambda(n) \) decreases for all \( n \geq 1 \) or increases for \( n \leq n_x \) and decreases for \( n > n_x \) where \( n_x \) satisfies

\[
\lambda(n_x-1) \cdot \lambda(n_x) \lambda(n_x+1)
\]

i.e., \( \lambda(n) \) is maximum for \( n = n_x \)

24 PARAMETER ESTIMATION

The proposed mean value function \( m(n) \) has four unknown parameters. To estimate the parameters, we use the method of maximum likelihood.

Suppose, data is available for \( k \) observed pairs \( (n_i, y_i) \) \( (i=1,2,...,k) \), where \( y_i \) is the cumulative number of faults removed by \( n_i \) test runs \((0 \leq y_1 \leq y_2 \quad \ldots \quad y_k)\) \((0 \leq n_1 \leq n_2 \quad \ldots \quad n_k)\). The likelihood function for the unknown parameters with \( m(n) \) in (2.6) is

69
\[ L(a, b, c, p \mid (n_i, y_i)) = \epsilon^{-p \left[-m(n_i) \sum_{i=1}^{k} \frac{m(n_i) - m(n_{i-1})}{(y_i - y_{i-1})^{Y_i - Y_{i-1}}} \right]} \]

with \( n_0 = y_0 = 0 \). Taking Log of (8), we get

\[ \ln \left[ L(a, b, c, p \mid (n_i, y_i)) \right] = -m(n_k) + \sum_{i=1}^{k} (y_i - y_{i-1}) \ln \left[ m(n_i) - m(n_{i-1}) \right] \] \[ - \sum_{i=1}^{k} (y_i - y_{i-1}) ! \] \[ \ldots \ldots (2.9) \]

From (2.9), maximum likelihood estimates of \( a, b, c \) and \( p \) are obtained using DNCONF subroutine of IMSL MATLIB Library, under the following constraints

\[ 0 < a < \infty \]
\[ 0 < b < 1 \]
\[ 0 \leq c < 1 \]
\[ 0 \leq p \leq 1 \]

We have applied the proposed model for the following four real discrete software failure data sets cited in [2]

DS1 - The failure data is for a command, control and communication system software tested for twelve months. During this period 2657 errors were removed.
DS2 - The software failure data is for a command and control system software. The software was tested for fifteen weeks and 1138 errors were removed.

DS3 - The data is for a communication and control system software tested for fifteen weeks during which period 1483 errors were removed.

DS4 - This data set is also for a command and control system software tested for fifteen weeks and 2702 errors were removed.

The following table gives the maximum likelihood estimates of the proposed model parameters $a, b, c$ and $p$ for the four data sets described above:

<table>
<thead>
<tr>
<th>DATA SETS</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>3115</td>
<td>0.1642</td>
<td>0.2104</td>
<td>0.7929</td>
</tr>
<tr>
<td>DS2</td>
<td>1385</td>
<td>0.1339</td>
<td>0.0743</td>
<td>0.8555</td>
</tr>
<tr>
<td>DS3</td>
<td>2562</td>
<td>0.4407</td>
<td>0.2533</td>
<td>0.1855</td>
</tr>
<tr>
<td>DS4</td>
<td>3942</td>
<td>0.0742</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

From the estimates of $b, c$ and $p$ obtained for the four failure data sets, it is observed that DS1 and DS2 have a high percentage of leading errors, while DS3 has a low percentage of leading errors and DS4 does not have any dependent errors and all the errors are leading. The model
for DS4 data set reduces to a discrete exponential model. From the estimates obtained for the parameters, it is evident that the software either may not contain any dependent errors or it may contain a varying proportion of dependent errors. The above mentioned data sets simply justify the existence of such a discrete a software error removal phenomenon. Figures 1 to 4 show the graphs of actual and estimated failures for the above data sets.

25 OPTIMAL SOFTWARE RELEASE PROBLEM

It is very important for the software manager to know when to stop testing and release the software for use. Several researches have studied this problem based on different criteria for release [4-5 and reference cited there in]. In this chapter, we obtain the optimal number of test runs expected before release such that the total expected software cost during the life cycle of the software is minimized subject to discrete failure intensity not exceeding a prespecified value.

Mathematically, we may say

Minimize $C(n) = \sum_{i=1}^{4} c_i \left( m_i(n) - m_1(n) \right) + c_5 n \quad (2.10)$
subject to

\[ \lambda(n) \leq \lambda_0 \] \quad \ldots \ldots \quad (2.11)

From (2.10),

\[ C(n+1) - C(n) = -\left( C_2 - C_1 \right) \left( m_1(n+1) - m_1(n) \right) - \]
\[ \left( C_3 - C_4 \right) \left( m_2(n+1) - m_2(n) \right) + C_5 \]

for simplicity, assuming \( C_3 = C_1 \) and \( C_4 = C_2 \), we have

\[ C(n+1) - C(n) = -\left( C_2 - C_1 \right) \lambda(n+1) + C_5 \] \quad \ldots \ldots \quad (2.12)

To study the behavior of the cost function, we consider the following cases

(i) When \( \lambda(n) \) is decreasing for all \( n \)

if \( C_5 > (C_2 - C_1) \lambda(1) \), minimum cost is achieved for \( n=0 \),

else

if \( C_5 = (C_2 - C_1) \lambda(1) \), minimum cost is achieved for \( n=0 \) and \( 1 \), else

if \( C_5 < (C_2 - C_1) \lambda(1) \), and there exists \( n_1 (>1) \)

satisfying

\[ (C_2 - C_1) \lambda(n_1 - 1) > C_5 > (C_2 - C_1) \lambda(n_1) \], cost is minimum

when \( n = n_1 - 1 \), else

if \( C_5 < (C_2 - C_1) \lambda(1) \) and there exists \( n_1 (>1) \)

such that

\[ (C_2 - C_1) \lambda(n_1 - 1) > C_5 = (C_2 - C_1) \lambda(n_1) \]

\( C(n) \) is minimum for both \( n=n_1 \) and \( n=n_1 - 1 \)
When $\lambda(n)$ increases for $n \leq n_v \ (1)$ and decreases for $n \geq n_v$, such that $\lambda(n - 1) < \lambda(n) \forall x, \lambda(n + 1)$

1. If $c_5 \geq (C - C_1) \lambda(n)$, minimum cost is achieved for $n = 0$.

2. If $c_5 < (C - C_1) \lambda(n)$, $C(n)$ is minimum for both $n = n_1$ and $n = n_1 - 1$, else

   a. If $c_5 < (C - C_1) \lambda(n_1)$, $C(n)$ is minimum for both $n = n_1$ and $n = n_1 - 1$.

   b. If $c_5 > (C - C_1) \lambda(n_1)$, $C(n)$ increases for $n \leq n_1 - 1$ and is minimum for $n = n_1 - 1$, else if $c_5 < (C - C_1) \lambda(n_2)$ and $c_5 > (C - C_1) \lambda(n_1)$, $C(n)$ increases for $n \leq n_2 - 1$ and is minimum for both $n = n_1$ and $n = n_1 - 1$.

Now, for a specific intensity requirement $\lambda_o (> 0)$, if $\lambda(n)$ is decreasing for all $n$ and $\lambda(1) > \lambda_o$, these exist $n > 1$ such that $\lambda(n - 1) > \lambda_o \geq \lambda(n)$, else
If $\lambda(n)$ is increasing for $n \leq n_x$ and decreasing for $n > n_x$, and $\lambda(n_x), \lambda_0$, these may exist $n_3$ and $n_4$

$(0, n_3, n_x, n_4)$ such that

$$\lambda(n_3) \leq \lambda_0, \lambda(n_3 + 1) \geq \lambda_0 \quad \ldots \quad (2.13)$$

$$\lambda(n_4) \leq \lambda_0, \lambda(n_4 - 1) > \lambda_0 \quad \ldots \quad (2.14)$$

Combining the cost and intensity requirements, we may state the following theorem for optimal release policy (assuming unique $n$ exists for minimum cost) theorem. Assume $C_2 > C_1 > 0, \lambda_0 > 0, C_5 > 0$.

(a) $\lambda(n)$ is decreasing for all $n \geq 1$

(i) $\lambda(1) \leq \lambda_0$

if $C_5 \geq (C_2 - C_1) \lambda(1)$, $n^* = 1$

if $C_5 < (C_2 - C_1) \lambda(1)$ and there exists $n_1$ such that

$(C_2 - C_1) \lambda(n_x) > n_1 \geq (C_2 - C_1) \lambda(1)$, $n^* = n_1 - 1$

(ii) $\lambda(1) > \lambda_0$ and there exists $n > 1$ satisfying

$\lambda(n_x - 1) \geq \lambda(n_i)$

if $C_5 \geq (C_2 - C_1) \lambda(1)$, $n^* = n_i$

if $C_5 < (C_2 - C_1) \lambda(1)$ and there exists $n_1$ such that

$(C_2 - C_1) \lambda(n_x - 1) > n_1 \geq (C_2 - C_1) \lambda(1)$,

$$n^* = \max(n_x, n_1 - 1)$$

(b) $\lambda(n)$ increases for $n \leq n_x$ and decreases for $n > n_x$

(i) $C_5 \geq (C_2 - C_1) \lambda(n_x)$

if $\lambda(n_x) \leq \lambda_0$, $n^* = 1$
1f $\lambda(n_x) \leq \lambda_0$ and there exists $n_3$ satisfying (2.13),
\[ n^* = 1 \]

1f $\lambda(n_x) \leq \lambda_0$ and there exists $n_4$ satisfying (2.14),
\[ n^* = n_4 \]

(11) $C_{5} \cdot (C - C) \lambda(n_x), C_{5} \leq (C - C) \lambda(1)$ and there exists $n_1 \cdot n_x$ such that
\[ (C - C) \lambda(n_1 - 1) \leq C_{5} \cdot (C - C) \lambda(n_1) \]

1f $\lambda(n_x) \leq \lambda_0$, $n^* = n_1 - 1$

1f $\lambda(n_x) \leq \lambda_0$ and there exist $n_3$ and $n_4$ satisfying (2.13) and (2.14) respectively, then

1f $n_1 \geq n_4 + 1$, $n^* = n_1 - 1$, else

1f $C(n_3) < C(n_4)$, $n^* = n_3$

1f $C(n_3) > C(n_4)$, $n^* = n_4$

1f $C(n_3) = C(n_4)$, $n^* = n_3$ or $n_4$

1f $\lambda(n_x) \leq \lambda_0$ and there exists $n_4$ satisfying (2.14) then

1f $n_1 \geq n_4 + 1$, $n^* = n_1 - 1$ else $n^* = n_4$

(111) $C_{5} \cdot (C - C) \lambda(n_x), C_{5} \leq (C - C) \lambda(1)$ and there exists $n_2 \cdot n_x$ and $n_1 \cdot n_x$ satisfying
\[ (C - C) \lambda(n_2 - 1) \leq C_{5} \cdot (C - C) \lambda(n_2) \text{ and} \]
\[ (C - C) \lambda(n_1 - 1) \geq C_{5} \cdot (C - C) \lambda(n_1) \]

1f $\lambda(n_x) \leq \lambda_0$

1f $C(1) \cdot C(n_1 - 1)$, $n^* = n_1 - 1$

76
if \( C(1) = C(n_1 - 1) \), \( n^* = 1 \) or \( n_1 - 1 \)

if \( C(1) \neq C(n_1 - 1) \), \( n^* = 1 \)

if \( \lambda(x) \geq \lambda_x \) and there exist \( n_3 \) and \( n_4 \) satisfying (2.13) and (2.14) then

if \( C(1) \neq C(n_1 - 1) \)

if \( n_1 \geq n_4 + 1 \), \( n^* = n_1 - 1 \)

if \( n_3 \leq n_2 - 1 \) or \( (n_3 \geq n_2 - 1 \) and \( C(n_3) \geq C(1) \))

if \( C(1) \neq C(n_4) \), \( n^* = 1 \)

if \( C(1) = C(n_4) \), \( n^* = 1 \) or \( n_4 \)

if \( C(1) \neq C(n_4) \), \( n^* = n_4 \)

if \( C(n_3) \neq C(n_4) \), \( n^* = n_3 \)

if \( C(n_3) \neq C(n_4) \), \( n^* = n_4 \)

if \( C(n_3) = C(n_4) \), \( n^* = n_3 \) or \( n_4 \)

if \( C(1) = C(n_1 - 1) \)

if \( n_1 \geq n_4 + 1 \), \( n^* = n_1 - 1 \) or \( 1 \), else \( n^* = 1 \)

if \( C(1) \neq C(n_1 - 1) \), \( n^* = 1 \)

if \( \lambda(x) \neq \lambda_x \) and there exists \( n_4 \) satisfying (2.14) then

if \( n_1 \geq n_4 + 1 \), \( n^* = n_1 - 1 \) else \( n^* = n_4 \)

Other cases can be similarly discussed
26 NUMERICAL EXAMPLES

Using $a=1385$, $b=0.1229$, $c=0.0745$, $p=0.8555$, we discuss the optimal release policy for the software system described by DS2. We assume $C_1=5$, $C_2=10$, $C_3=40$, $n_l=150$ and $n_0=.999$. Using these values, we have $n_1=27$ ($C(n)$ is minimum for $n=26$) and $n_l=49$. Using (ii) of (a) in the theorem, we get $n^* = \max(n_1, n_l-1) = 49$. The intensity for $n=49$ is $.995$ and cost is $8948.4$. If only cost was to be minimized, $n^*$ would have been 26, cost as $8378.8$ but intensity would have been quite high (8.0). Figures 5 and 6 show the graphs of cost and intensity functions respectively.

27 CONCLUSION

In this chapter, we have proposed a new discrete SRGM. At times discrete SRGMs are more suitable to describe software error detection/removal phenomenon than continuous SRGMs. In the proposed model the assumption of error independence has been relaxed.

Moreover, the proposed model can cater for various types of software growth modelling from pure exponential to highly s-shaped. Thus the proposed model can be applied to different testing environments.