Optimal Testing Time Allocation of Modular Software For Exponential SRGM Incorporating Imperfect Debugging

P. C. Jha* Ritu Arora** P. K. Kapur*
* Department of Operational Research, University Of Delhi, Delhi.
**Maharaja Agrasen Institute of Technology, GGSIP University. Delhi
Email: arora_ritu21@yahoo.co.in

ABSTRACT
The development of high quality software satisfying cost, schedule and resource requirements is an essential prerequisite for improved competitiveness of any organization. Testing is an important stage of SDLC as it provides the measure of software reliability and assists to judge the performance, safety, fault tolerance or security of the software. As most software systems are modular, it is of great importance for the management to allocate the limited testing time among the software modules in an optimal way. During testing phase the modules are tested independently to remove maximum possible number of faults within specified testing time. While removing faults from each module, there is a possibility that either a fault is imperfectly debugged or some new faults may be generated. Firstly the faults are not perfectly debugged and secondly some new faults may be generated while removing the existing faults. Here, we discuss two types of optimization problems to allocate optimal allocation of testing time among the modules of software using dynamic resource allocation strategy. In the first optimization model, we maximize the mean number of detected faults, subject to the time constraint. In the second model, additional constraint representing aspiration level for fault removal for each module of the software is added. These methods have been illustrated through numerical example.

KEYWORDS
Software Reliability, Non-homogenous Poisson process, Testing Resource Allocation, Modular Software System, Dynamic Programming

1.0 INTRODUCTION
The size and complexity of computer systems have grown rapidly for the last several decades. Software cost as a percentage of total computer system costs continue to increase, while associated hardware costs continue to decrease. Any failure in these systems can cost heavily in terms of money and/or human lives. Consequently, the dependence of mankind on computers and computer based systems is increasing day by day. Due to design and other human errors, no complex software system can be made completely free of faults. In order to reduce the number of delivered faults in software, it is estimated that many companies spend between 50-80% of their software development effort on testing. Thus reducing the effort on testing is a key issue in attaining high productivity in software development. The latent faults in the software results in system failure in operational phase, and it is essential to remove as many faults as possible through testing. There is an urgent need for software designers to improve and assure software reliability, which is a main aspect of software quality in general. Therefore a lot of importance is attached to the testing phase of the software development process; where around half the developmental resources are used [9]. Essentially testing is a process of executing a program with the explicit intention of finding faults and it is this phase, which is amendable to mathematical modeling. The imperfect debugging model was discussed [10] to introduce the effect of error generation into reliability modeling. Later on several SRGM was developed in the literature on imperfect debugging [6]. In fact in practical situations, while trying to fix the faults, we may not be able to fix all faults perfectly and sometime, new faults may be introduced while removing the faults. The first phenomenon is known as imperfect debugging while the second as faults generation[3]. A software reliability growth model relates the amount of testing resources/time to the reliability growth. A SRGM explains the time dependent behavior of fault removal. As modules are tested independently distinct SRGMs would represent their reliability growth. The influence of testing effort can also be included in the SRGMs [1,4,5,12].

Complex software systems are usually composed of a number of different modules. During the module testing phase, all the testing activities of different modules are competing for the limited testing time. Therefore, the software project manager should monitor the testing process closely and effectively allocate the resources/time in order to reduce the testing cost and to meet the given reliability requirements. It is always desirable to remove a substantial number of faults from the software. In fact the reliability of software is directly proportional to the number of faults removed. Several Software Reliability Growth Models (SRGM) have been proposed in the literature to estimate the software reliability measures such as the remaining number of faults, failure rate and reliability growth during the testing phase. These models are applied to the software testing data collected during the testing phase and then are often used to predict the software failures in the operational phase. SRGM describes the failure occurrence and/or failure removal phenomenon of the testing process and consequently enhancement in reliability with respect to time (CPU time, calendar time or execution time or test cases as a unit of time) and/or testing effort [5].
In this paper, a general formulation is presented and it allows the testing process of different modules to be modeled by different models. We propose a dynamic time allocation for software module testing [8]. The first optimization model (P1) maximizes the total number of faults expected to be removed given available testing time is known. The management normally aspires for some reliability level that can be translated in terms of number of faults removed. The second optimization model (P2), we add a constraint in (P1) in terms of minimum number of faults aspired to remove.

Notations

\( N \): Number of modules in the software system (>1)

\( m_i(t) \): Number of faults removed in \((0, t]\) the \(i^{th}\) module. Mean value function of NHPP, \(i = 1, 2, \ldots, N\)

\( a_i \): Expected number of faults in the \(i^{th}\) module \((i = 1, 2, N)\)

\( b_i \): Proportionality constant for the \(i^{th}\) module

\( T_i \): The amount of testing time to be allocated to the \(i^{th}\) module and

\( Z \): Total testing time available

\( Z^* \): Optimal value of \(X_i\), \(i = 1, 2, \ldots, N\)

\( f_n(Z) \): Optimal number of faults removed up to \(n^{th}\) modules (i.e. Corresponding to \(n^{th}\) stage in a Dynamic Programming Algorithm)

\( a_{io} \): Aspiration level of \(i^{th}\) module (i.e. number of faults desired to be removed from \(i^{th}\) module)

\( p_i \): The minimum proportion of total faults to be removed from \(i^{th}\) module

2. DYNAMIC RESOURCE ALLOCATION

2.1 Assumption and Nomenclature

1. A software system is composed of \(N\) modules, which are tested independently during the module-testing phase [11].
2. A module is subject to failures at random times caused by the faults remaining in software.
3. Fault removal phenomenon is modeled by Non-Homogeneous Poisson Process (NHPP).
4. A detected fault is immediately removed and no new fault is introduced.
5. The expected number of faults removed in \((t, t + \Delta t)\) to the current testing time is proportion of the expected remaining number of faults.

Under assumption 5, following differential equation may easily be written for module

\[
\frac{d}{dt}m_i(t) = b_i[a_i - m_i(t)], \quad i = 1, 2, \ldots, N
\]  

(1)

Solving equation (1) with the initial condition that, at \(t = 0\), \(T_i(t) = 0, m_i(t) = 0\)

\[
m_i(t) = a_i \left(1 - e^{-b_i T_i(t)}\right), \quad i = 1, 2, \ldots, N
\]  

(2)

2.2 Optimal Allocation of Testing / Time

Consider the software having \(N\) modules, which are being tested independently for removing faults lying dormant in them. From the estimates of parameters of SRGMs for modules, the total fault content in the software \(\sum_{i=1}^{N} a_i\) is known. The limited time are available, that need to be allocated judiciously. Modules testing aims at removing maximum number of them within available time. If \(m_i\) faults are expected to be removed from the \(i^{th}\) module with testing time \(T_i\), the resulting allocation problem can be stated as follows.

Maximize \(\sum_{i=1}^{N} m_i(T_i) = \sum_{i=1}^{N} a_i \left(1 - e^{-b_i T_i}\right)\)

Subject to \(\sum_{i=1}^{N} T_i \leq Z, \quad T_i \geq 0, \quad i = 1, 2, \ldots, N\)  

(P1)

(P1) can be solved using Dynamic Programming Approach. From Bellman’s principle of optimality, we can write the following recursive equation [2]

\[
f_n(Z) = \max_{r \leq Z_n} \left\{a_i \left(1 - e^{-b_i r}\right)\right\}
\]

\[
f_n(Z) = \max_{0 \leq T_i \leq Z} \left\{a_i \left(1 - e^{-b_i T_i}\right) + f_{n-1}(Z - T_i)\right\},
\]  

(3)

To index the modules, they can be arranged in descending order of their values of \(a_i b_i\) i.e. \(a_1 b_1 \geq a_2 b_2 \geq \ldots \geq a_N b_N\). Through this approach time is allocated to the modules sequentially. But for some values of \(Z (Z < Z_n)\) one or more modules with higher index number i.e. having lower detectability may not get any
Optimal Testing Time Allocation of Modular Software For Exponential SRGM Incorporating Imperfect Debugging

We summarize this result in the following simple theorem.

**Theorem-1**

If for any \( n = 1, 2, \ldots, N \), \( 1 \leq e^{\mu_n Z} \leq \frac{\mu_{n-1} V_{n-1}}{a_n b_n} \), then values of \( T_n, T_{n+1}, \ldots, T_N \) are zero and problem reduces to \((n-1)\) stage problem with

\[
T_r = \frac{1}{b_r + \mu_{r-1}} \left[ \mu_r Z - \log \left( \frac{\mu_{r-1} V_{r-1}}{a_r b_r} \right) \right], \quad r=1, \ldots, (n-1) \tag{4}
\]

Where \( \mu_i = \frac{1}{\sum_{j=1}^{i} \frac{1}{b_j}} \) and

\[
\mu_i V_i = \prod_{j=1}^{i} \left( a_j b_j \right)^{\left( \mu_j / b_j \right)}, \quad i = 1, 2, \ldots, N
\]

Proof of the theorem is mentioned in [7].

As a result of the above allocation procedure, some modules may not test at all. This situation is not advisable. Again management often aspires to achieve certain minimum reliability level for the software and that for each module of the software i.e. a certain percentage of the fault content in module of the software is desired to be removed. Hence (P1) needs to be suitably modified to maximize removal of faults in the software under time constraint and minimum desired level of faults to be removed from each of modules in the software. The resulting testing time allocation can be stated as follows:

\[
\text{Max } \sum_{i=1}^{N} m_i = \sum_{i=1}^{N} a_i \left( 1 - e^{-b_i T_i} \right)
\]

Subject to

\[
m_i = a_i \left( 1 - e^{-b_i T_i} \right) \geq p_i a_i = a_{i0}, \quad i = 1, 2, \ldots, N \tag{P2}
\]

\[
\sum_{i=1}^{N} T_i \leq Z, \quad T_i \geq 0, \quad i = 1, 2, \ldots, N
\]

(P2) can be solved using Dynamic Programming Approach. The substitution method is adopted for converting the problem (P2) to the problem (P1) as follows:

\[
m_i (T_i) \geq a_{i0} \Rightarrow a_i \left( 1 - e^{-b_i T_i} \right) \geq a_{i0}
\]

Hence \( T_i \geq \frac{-1}{b_i} \log \left[ 1 - \frac{a_{i0}}{a_i} \right] = Z_i \) (say), \( i = 1, 2, \ldots, N \)

Therefore (P2) can be restated as

\[
\text{Max } \sum_{i=1}^{N} m_i = \sum_{i=1}^{N} a_i \left( 1 - e^{-b_i T_i} \right)
\]

Subject to \( T_i \geq Z_i, \quad i = 1, 2, \ldots, N \tag{P3} \)

\[
\sum_{i=1}^{N} T_i \leq Z, \quad T_i \geq 0, \quad i = 1, \ldots, N
\]

Let \( Y_i = T_i - Z_i (i = 1, 2, \ldots, N) \), then (P3) can be written as the problem (P1) given below

\[
M \text{ax } \sum_{i=1}^{N} m_i = \text{Max } \sum_{i=1}^{N} a_i \left( 1 - e^{-b_i Y_i} \right)
\]

Subject to

\[
\sum_{i=1}^{N} Y_i \leq Z - \sum_{i=1}^{N} Z_i = \bar{Z} \quad \text{(say)} \tag{P4}
\]

\( Y_i \geq 0, \quad i = 1, 2, \ldots, N \)

\( a_i = a_i - a_{i0}, \quad i = 1, 2, \ldots, N \)

The problem (P4) is similar to the problem (P1) and hence using theorem-1 the problem (P4) can also be solved.

If for any \( n = 1, 2, \ldots, N \), \( 1 \leq e^{\mu_n Z} \leq \frac{\mu_{n-1} V_{n-1}}{a_n b_n} \), then

\[
Y_i, Y_{i+1}, \ldots, Y_N \quad \text{are zeroes and problem reduces to } (i-1) \text{ stage problem with}
\]

\[
y_n = \frac{1}{b_n + \mu_{n+1}} \left[ \mu_{n+1} Z - \log \left( \frac{\mu_{n+1} V_{n+1}}{a_n b_n} \right) \right], \quad n=1, \ldots, (i-1) \tag{5}
\]

Where \( \mu_i = \frac{1}{\sum_{j=1}^{i} b_j} \) and

\[
\mu_i V_i = \prod_{j=1}^{i} \left( a_j b_j \right)^{\left( \mu_j / b_j \right)}, \quad i = 1, 2, \ldots, N
\]

\[
f \left( \bar{Z} \right) = \sum_{i=1}^{N} a_i \left( 1 - e^{-\mu_i \bar{Z}} \right)
\]

Optimal time for each of the modules can be obtained using equation (5). Numerical illustration of these results is given in the following section.

3. NUMERICAL EXAMPLE

It is assumed that parameters \( a_i \) and \( b_i \) for the \( i^{th} \) module \( (i = 1, 2, \ldots, N) \) are already estimated using the software failure data. Consider a software having 10 modules whose parameter estimates are as given in Table-I. Suppose the total C.P.U time available for testing is 85000 seconds. First the problem (P1) is solved & from recursion equation (3) optimal allocation of time \( T_i^* \) for the modules are computed.
The total number of faults that can be removed through first allocation is 297 (i.e. 67.2 % of the fault content is removed from the software). It is observed that in some modules (module-8, 10), the remaining faults after the allocation is high. This can lead to frequent failures in the operational phase. Obviously, this will not satisfy the developer and he may desire that at least 50% of fault content from each of the modules of the software is removed (i.e. \( p_i = 0.5 \) for each \( i = 1, 2, \ldots, 10 \)). Since faults in each module are integral values, nearest integer larger than 50% of the fault content in each module is taken as lower limit that has to be removed. The new allocation of time along with expected number of faults removed, percentage of faults removed and faults remaining for each module after solving the problem (P4) is summarized in Table-III. The total number of faults that can be removed through this allocation is 338. (i.e. 76.4 % of the fault content is removed from the software). Analysis given in Table- II and III help in providing the developer an insight into the time allocation and the corresponding fault removal phenomenon and the objective can be set accordingly.

### Table – II

<table>
<thead>
<tr>
<th>Software Module, ( i )</th>
<th>Mean number of remaining faults in module ( i, a_i )</th>
<th>( b_i (10^{-4}) )</th>
<th>( T_i^* )</th>
<th>( m_i^* )</th>
<th>% of faults removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89</td>
<td>5.0923</td>
<td>23.2993</td>
<td>93.197474</td>
<td>338.297474</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>3.69411</td>
<td>16.80523</td>
<td>62.241591</td>
<td>338.241591</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>2.5336</td>
<td>13.83276</td>
<td>99.571183</td>
<td>338.571183</td>
</tr>
<tr>
<td>4</td>
<td>39</td>
<td>0.8819</td>
<td>8.3420</td>
<td>59.5862</td>
<td>338.5862</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>0.7274</td>
<td>6.4794</td>
<td>22.679</td>
<td>338.679</td>
</tr>
<tr>
<td>6</td>
<td>39</td>
<td>1.3509</td>
<td>10.1534</td>
<td>337.1534</td>
<td>338.1534</td>
</tr>
<tr>
<td>7</td>
<td>59</td>
<td>0.6824</td>
<td>8.1534</td>
<td>20.153</td>
<td>338.153</td>
</tr>
<tr>
<td>8</td>
<td>68</td>
<td>4.14062</td>
<td>3.019436</td>
<td>18.019436</td>
<td>338.019436</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>8.16064</td>
<td>5.0923</td>
<td>20.0923</td>
<td>338.0923</td>
</tr>
<tr>
<td>10</td>
<td>37</td>
<td>0.6824</td>
<td>8.16064</td>
<td>20.16064</td>
<td>338.16064</td>
</tr>
</tbody>
</table>

### Table – III

<table>
<thead>
<tr>
<th>Software Module, ( i )</th>
<th>( a_i )</th>
<th>( b_i )</th>
<th>( T_i^* )</th>
<th>( m_i^* )</th>
<th>% of faults removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89</td>
<td>0.000418</td>
<td>5093.4895</td>
<td>88.99789</td>
<td>99.99758646</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>0.000509</td>
<td>3267.5248</td>
<td>23.2993</td>
<td>93.197474</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>0.000396</td>
<td>4009.0184</td>
<td>16.8052</td>
<td>62.241591</td>
</tr>
<tr>
<td>4</td>
<td>39</td>
<td>0.000253</td>
<td>6137.1441</td>
<td>38.8327</td>
<td>99.571183</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>0.000229</td>
<td>6682.7088</td>
<td>34.4798</td>
<td>76.646711</td>
</tr>
<tr>
<td>6</td>
<td>39</td>
<td>0.000172</td>
<td>8052.9383</td>
<td>36.7379</td>
<td>94.19985924</td>
</tr>
<tr>
<td>7</td>
<td>39</td>
<td>0.000088</td>
<td>14476.534</td>
<td>31.9387</td>
<td>54.1334274</td>
</tr>
<tr>
<td>8</td>
<td>68</td>
<td>0.000073</td>
<td>17182.796</td>
<td>14.9937</td>
<td>22.04963438</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>0.000153</td>
<td>5812.6261</td>
<td>8.275927</td>
<td>59.11376777</td>
</tr>
<tr>
<td>10</td>
<td>37</td>
<td>0.000068</td>
<td>14106.22</td>
<td>3.019436</td>
<td>18.019436</td>
</tr>
<tr>
<td>Total</td>
<td>442</td>
<td>85000</td>
<td>297.3999</td>
<td>67.28503926</td>
<td></td>
</tr>
</tbody>
</table>

#### 4. CONCLUSION

In this paper, we have discussed an optimization problem occurring during module testing phase of software development life cycle under imperfect debugging. A dynamic programming approach for finding the optimal solution has been proposed. Using simple recursion equations it is shown how fault removal for each module and that of the software can be maximized by allocation of time. Numerical examples show that by adopting the optimal testing time allocation among software modules, the software system reliability can be much improved.

#### 5. REFERENCES


Continued on Page No. 533
Continued from Page No. 516


