DYNAMIC PROGRAMMING APPROACH TO TESTING RESOURCE ALLOCATION PROBLEM FOR MODULAR SOFTWARE

P.K. Kapur[1] P.C. Jha[1] A.K. Bardhan[1]

Abstract
Testing phase of a software begins with module testing. During this period modules are tested independently to remove maximum possible number of faults within a specified time limit or testing resource budget. This gives rise to some interesting optimization problems, which are discussed in this paper. Two Optimization models are proposed for optimal allocation of testing resources among the modules of a Software. In the first model, we maximize the total fault removal, subject to budgetary Constraint. In the second model, additional constraint representing aspiration level for fault removals for each module of the software is added. These models are solved using dynamic programming technique. The methods have been illustrated through numerical examples.

Key words: Software Reliability, Non Homogeneous Poisson Process, Resource Allocation, Dynamic Programming

1. Introduction

Growth in software engineering technology has led to production of software for highly complex situations occurring in industry, scientific research, defense and day to day life. Consequently, the dependence of mankind on computers and computer-based systems is increasing day by day. Any failure in these systems can cost heavily in terms of money and/or human lives. Though high reliability of hardware part of these systems can be guaranteed, the same cannot be said for software. Therefore a lot of importance is attached to the testing phase of the software development process, where around half the developmental resources are used [8]. 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.

It is always desirable to remove a substantial number of faults from the software. In fact the reliability of a software is directly proportional to the number of faults removed. Hence the problem of maximization of software reliability is identical to

[1] Department of Operational Research, Faculty of Mathematical Sciences, University of Delhi, Delhi 110007, INDIA

that of maximization of fault removal. At the same time testing resource are not unlimited, and they need to be judiciously used. In this paper we discuss and solve such a management problem of allocation of testing resources among modules, through a Software Reliability Growth Model (SRGM). A Software Reliability Growth Model (SRGM) is a relationship between the number of faults removed from a software and the execution time/CPU time/calendar time. Several attempts have been made to represent the actual testing environment through SRGMs [1,4,5,9]. These models have been used to predict the fault content, reliability and release time of a software. SRGMs have also been used to manage the testing phase. Again large software consists of modules. Often these modules are developed independently and each module may contain different number of faults and that of different severity. Therefore distinct SRGMs should be used to represent the testing process of each module, as testing for these modules are done independently. An SRGM with testing effort [9] has been chosen to represent the fault removal process for the two optimization problems discussed in this paper.

The first optimization model (P1) maximizes the total number of faults expected to be removed, when available testing resource is known. The management normally aspires for some reliability level that can be translated in terms of number of faults removed. In the second optimization model (P2) we add a constraint in (P1) in terms of minimum number of faults aspired to be removed from each module. Dynamic programming technique is used to solve these problems. This is the first time that this has been done in software engineering, according to our knowledge. Dynamic programming approach, which is easy to solve and understand provides global optima for these problems. The methodology discussed in the paper has been illustrated through numerical examples.

Notations

N	:	Number of modules in the Software (>1)
a_i	:	Expected number of faults in the i^{th} module (i=1,2,…,N)
b_i	:	Proportionality constant for the i^{th} module
$x_i(t)$	:	Current testing effort expenditure at testing time t and $X_i(t) = \int_0^t x_i(w)dw$ for i^{th} module
X_i, Z	:	The amount of testing resource to be allocated to the i^{th} module and total testing resource available.
$m_i(t)$	:	Number of faults removed in (0,t] the ith module, mean value function of NHPP, i = 1,…,N
T	:	Total testing time
X_i^*	:	Optimal value of X_i , i = 1,…,N
$f_n(Z)$	:	Optimal number of faults removed upto nth modules (i.e. corresponding to nth stage in a Dynamic Programming algorithm)

a_{io} : Aspiration level of ith 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. Mathematical Modelling

2.1 Resource Allocation Problem

Consider a software having N modules, which are being tested independently for removing faults lying dormant in them. The duration of module testing is often fixed when scheduling is done for the whole testing phase. Hence limited resources are available, that need to be allocated judiciously. If m_i faults are expected to be removed from the i^{th} module with effort X_i, the resulting testing resource allocation problem can be stated as follows [5,6].

$$\max \quad \sum_{i=1}^{N} m_i$$

subject to

$$\sum_{i=1}^{N} X_i = Z \,,\; X_i \geq 0 \,,\; i = 1,\ldots,N \qquad \ldots \qquad \ldots \qquad \text{(P1)}$$

Above optimization problem is the simplest one as it considers the resource constraint only. Later in this paper, we incorporate additional constraints to the basic model. For solving (P1) a functional relationship between fault removal and resource consumption is required, which is discussed in the following section.

2.2 SRGM For Modules

A Software Reliability Growth Model 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 [9]. In this paper we discuss the resource allocation problem using such a SRGM for the modules.

Model Assumptions

1. Software consist of a finite number of modules and testing for each module is done independently

2. A module is subject to failures at random time caused by faults remaining in the software.
3. On a failure, the fault causing that failure is immediately removed and no new faults are introduced.
4. Fault removal phenomenon is modelled by Non Homogeneous Poisson Process (NHPP).
5. The expected number of faults removed in $(t, t+\Delta t)$ to the current testing resource is proportional to the expected remaining number of faults.

Under assumption 5, following differential equation may easily be written for i^{th} module

$$\frac{\frac{d}{dt} m_i(t)}{x_i(t)} = b_i(a_i - m_i(t)), \; i = 1,\ldots,N \qquad \ldots \quad \ldots \quad (1)$$

Solving equation (1) with the initial condition that, at $t = 0$, $X_i(t) = 0$, $m_i(t) = 0$ we get

$$m_i(t) = a_i(1 - e^{-b_i X_i(t)}), \; i = 1,\ldots,N \qquad \ldots \quad \ldots \quad (2)$$

To describe the behaviour of testing effort, either Exponential or Rayleigh function has been used [5,9]. Both can be derived form the assumption that, " the testing effort rate is proportional to the testing resource available".

$$\frac{dX_i(t)}{dt} = c_i(t)[\alpha_i - X_i(t)], \; i = 1,\ldots,N \qquad \ldots \quad \ldots \quad (3)$$

where $c_i(t)$ is the time dependent rate at which testing resources are consumed, with respect to the remaining available resources. Solving equation (3) under the initial condition $X(0) = 0$ we get

$$X_i(t) = \alpha_i \left[1 - \exp \int_0^t c_i(k)dk \right], \; i = 1,\ldots, N \qquad \ldots \quad \ldots \quad (4)$$

When $c(t) = \beta$, a constant

$$X_i(t) = \alpha_i(1 - e^{-\beta_i t}), \; i = 1,\ldots,N \qquad \ldots \quad \ldots \quad (5)$$

If $c(t) = \beta.t$, (1) gives a Rayleigh type curve

$$X_i(t) = \alpha_i(1 - e^{-\beta_i \frac{t^2}{2}}), \; i = 1,\ldots,N \qquad \ldots \quad \ldots \quad (6)$$

In this paper we have chosen exponential function (5) to represent testing effort in the optimization problems.

2.3 Estimation Of Parameters

The testing effort data are given in the form of testing effort $x_k (x_1 < x_2 < ... < x_n)$ consumed in time $(0, t_i]$; $i = 1,2,..,n$. The testing effort model parameters α_i and β_i can be estimated by the method of least squares as follows.

Minimize $\sum_{i=1}^{n} \left[X_i - \hat{X} \right]$

subject to $\hat{X}_n = X_n$ (i.e. the estimated value of testing effort is equal to the actual value).

Once the estimates of α_i and β_i are known, the parameters of the SRGMs (2) for the modules can be estimated through Maximum Likelihood Estimation method using the underlying Stochastic Process, which is described by a Non Homogeneous Poisson Process. During estimation, estimated values of α_i and β_i are kept fixed. If the fault removal data for a module is given in the form of cumulative number of faults removed y_j in time $(0,t_j]$. The likelihood function for that module is given as

$$L_i(a_i, b_i / (y_i, W_i) = \prod_{j=1}^{n} \frac{\left[m_i(t_j) - m_i(t_{j-1}) \right]^{y_j - y_{j-1}}}{(y_j - y_{j-1})!} e^{-(m_i(t_j) - m_i(t_{j-1}))}$$

3. Optimal Allocation Of Resources

From the estimates of parameters of SRGMs for modules, the total fault content in the software $\sum_{i=1}^{N} a_i$ is known. Modules testing aims at removing maximum number of them, within available resources. Hence (P1) can be restated as

Maximize $\sum_{i=1}^{N} m_i(X_i) = \sum_{i=1}^{N} a_i (1 - e^{-b_i X_i})$

Subject to $\sum_{i=1}^{N} X_i \le Z$, $X_i \ge 0$ $\quad i = 1, \ldots, N \quad \ldots \quad$ (P1A)

(P1A) can be solved using Dynamic Programming Approach. From Bellman's principle of optimality, we can write the following recursive equation [2].

$$f_1(Z) = \max_{X_1 = Z_1} \left\{ a_1 (1 - e^{-b_1 X_1}) \right\}$$

$$f_n(Z) = \max_{0 \le X_n \le Z} \left\{ a_n (1 - e^{-b_n X_n}) + f_{n-1}(Z - X_n) \right\}, \; n = 2,\ldots,N \; \ldots \qquad (7)$$

To index the modules, they can be arranged in descending order of their values of $a_i b_i$ *i.e.* $a_1 b_1 \ge a_2 b_2 \ge \ldots \ge a_N b_N$. Through this approach resources are allocated to the modules sequentially. But for some values of Z ($Z < Z_r$) one or more modules with higher index number i.e. having lower detectability may not get any allocation. We summarize this result in the following simple theorem.

Theorem - 1

If for any $n = 2,\ldots,N$; $1 \le e^{-\mu_{n-1} Z} \le \dfrac{\mu_{n-1} V_{n-1}}{a_n b_n}$, then values of $X_n, X_{n+1}, \ldots X_N$ are zero and problem reduces to (n-1) stage problem with

$$X_r = \frac{1}{b_r + \mu_{r-1}} \left[\mu_{r-1} Z - \log \left(\frac{\mu_{r-1} V_{r-1}}{a_r b_r} \right) \right], \; r = 1,\ldots,(n\text{-}1) \qquad \ldots \qquad (8)$$

where $\mu_i = \dfrac{1}{\sum_{j=1}^{i} \left(\frac{1}{b_j}\right)}$ and $\mu_i V_i = \prod_{j=1}^{i} (a_j b_j)^{(\mu_i / b_j)}$, $i = 1,\ldots,N$

Proof of the theorem is given in appendix.

As a result of the above allocation procedure, some modules may not be tested 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 percentages of the fault content in each 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 resource constraint and minimum desired level of faults to be removed from each of the modules in the software. The resulting testing resource allocation problem can be stated as follows:

$$\max \sum_{i=1}^{N} m_i = \sum_{i=1}^{N} a_i (1 - e^{-b_i X_i})$$

subject to

$$m_i = a_i (1 - e^{-b_i X_i}) \ge p_i a_i = a_{i_0} \; , \; i = 1, \ldots, N$$

$$\sum_{i=1}^{N} X_i \le Z\ ,\ \ X_i \ge 0\ ,\ i = 1, \ldots, N \qquad \ldots \qquad \text{(P2)}$$

(P2) can be solved using Dynamic Programming Approach either by reducing the dimensionality of the problem through Lagrange multiplier or converting to (P1) by substitution. We first consider the dimensionality reduction in Dynamic Programming Approach [2] as follows.

$$\max_{X} \min_{\alpha} \phi(X, \alpha) = \sum_{i=1}^{N} \left[a_i (1 - e^{-b_i X_i}) + \alpha_i \left\{ a_i (1 - e^{-b_i X_i}) - a_{i_0} \right\} \right]$$

$$\text{subject to } \sum_{i=1}^{N} X_i \le Z \qquad X_i, \alpha_i \ge 0 \ \ i = 1, \ldots, N \qquad \text{(P3)}$$

Where α_i (i = 1, … , N) is Lagrange multiplier for i^{th} constraint corresponding to the i^{th} module. The above problem can be solved by Dynamic Programming approach in which Kuhn-Tuckker optimality conditions are obtained at each stage [2]. At any stage α_i (i = 1,…,N) can be zero or non-zero depending upon ineffectiveness or effectiveness of constraint respectively. Hence each stage has two possibilities and corresponding to each possibility of preceding stage present stage has two possibilities. So at any stage i, total number of cases is 2^{i-1}. Infact, above problem reduces to that of finding an optimal path by searching for an optimal solution at each stage in which only one option could be chosen. This procedure does not provide a closed form solution. Hence without further elaboration of the above method, the substitution method is adopted for converting the problem (P2) to the problem (P1) as follows:

$m_i(X_i) \ge a_{i_0}$ implies $a_{\text{i}}(1 - e^{-b_{\text{i}} X_{\text{i}}}) \ge a_{i_0}$

Hence, $X_i \ge -\frac{1}{b_{\text{i}}} \log\left[1 - \frac{a_{\text{i}_0}}{a_{\text{i}}} \right] = Z_i$ (say), i = 1, … , N

Therefore (P2) can be restated as,

Maximize $\sum_{i=1}^{N} m_i = \sum_{i=1}^{N} a_i (1 - e^{-b_i X_i})$

subject to $X_i \ge Z_i \quad i = 1, \ldots, N$

$$\sum_{i=1}^{N} X_i \le Z\ ,\ X_i \ge 0\ ,\ i = 1, \ldots, N \qquad \text{(P4)}$$

Let $Y_i = X_i - Z_i$ (i = 1, … , N), then (P4) can be written as the problem (P1) given below

$$\max \sum_{i=1}^{N} m_i = \max \sum_{i=1}^{N} \overline{a}_i (1 - e^{-b_i Y_i})$$

subject to

$$\sum_{i=1}^{N} Y_i \le Z - \sum_{i=1}^{N} Z_i = \overline{Z} \text{ (say)}$$

$$Y_i \ge 0, \; i = 1, \ldots, N$$

$$\overline{a_i} = a_i - a_{i_0}, \; i = 1, \ldots, N \qquad \text{(P5)}$$

The Problem (P5) is similar to the problem (P1) and hence using theorem-1 the problem (P5) can also be solved.

If for any i = 2, … , N $1 \le e^{-\mu_i \overline{Z}} \le \frac{\mu_{i-1} V_{i-1}}{\overline{a_i} b_i}$,then $Y_i, Y_{i+1}, \ldots, Y_N$ are zeroes, then problem (P5) reduces to a $(i-1)$ stage problem and its solution is given as

$$Y_n = \frac{1}{b_n + \mu_{n+1}} \left[\mu_{n-1} Z - \log \left(\frac{\overline{\mu}_{n-1} V_{n-1}}{\overline{a_n} b_n} \right) \right], \; n = 1, \ldots, (i\text{-}1) \qquad \ldots \qquad (9)$$

$$f(\overline{Z}) = \sum_{n=1}^{i-1} \overline{a}_i - V_n e^{-\mu_n \overline{Z}} \qquad \ldots \qquad \ldots \qquad (10)$$

Through equation (9) optimal allocation of resources to the modules can be calculated. In the following section we numerically illustrate these results.

4. Numerical Example

It is assumed that parameters a_i and b_i for the i[th] module (i=1,.....N) are already estimated using the software failure data. Consider a software having 10 modules whose parameter estimates are as given in Table-1. Suppose the total resource available for testing is 97000. First the problem (P1) is solved and from the recursion equation (7) optimal allocation of resources (X_i^*) for the modules are computed. These are listed in Table-1 along with the corresponding expected number of fault removed, percentages of faults removed and faults remaining for each module. The total number of faults that can be removed through this allocation is 152 (i.e. 60.6% of the fault content is removed from the Software). It is observed

that in some modules (module-9,10) the remaining faults after allocation is high. This can lead to frequent failure during 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…10). Since faults in each module are integral values, nearest integer larger then 50% of the fault content in each module is taken as lower limit that has to be removed. The new allocation of resource along with expected number of fault removed, percentages of faults removed and faults remaining for each module after solving the problem (P2) through the problem (P5) is summarized in Table-2. The total number of faults that can be removed through this allocation is 146.8(i.e. 58.4% of the fault content is removed from the Software). In addition to the above if it is desired that a certain percentage of the total faults are to be removed then additional testing resources would be required. It is interesting to study this tradeoff and Table-3 summarizes results, where the required percentage of faults removed is 60%. To achieve this, 3000 units of additional testing effort is required. The total number of faults that can be removed through this allocation is 150.8(i.e. 60.09% of the fault content is removed from the Software). Analysis given in Tables-1, 2 and 3 help in providing the developer an insight into the resource allocation and the corresponding fault removal phenomenon and the objective can be set accordingly.

Table - 1

Module	a_i	b_i	X_i^*	m_i*	% of faults removed	% of faults remaining
1	63	5.33E-05	25435	46.7689	74.24	25.76
2	13	0.000252	5280.7	9.56979	73.61	26.39
3	6	0.000526	2459.5	4.3553	72.59	27.41
4	51	5.17E-05	21549	34.2571	67.17	32.83
5	15	0.000171	6354.5	9.93004	66.2	33.8
6	39	5.72E-05	16554	23.8778	61.23	38.77
7	21	9.94E-05	8857.2	12.2916	58.53	41.47
8	9	0.000174	3412.3	4.03476	44.83	55.17
9	23	5.06E-05	5845.6	5.88626	25.59	74.41
10	11	8.78E-05	1251.9	1.14528	10.41	89.59
Total	251		97000	152.117	60.6	39.4

Table-2

Module	a_i	a_{io}	Z_i*	Y_i*	$m_i(Y_i)$	m_i*	% of faults removed	% of faults remaining
1	63	32	13300	7495.5	10.2	42	67	33
2	13	7	3064.6	1235.6	1.61	8	66.21	33.79
3	6	3	1317.3	672.1	0.89	4	64.89	35.11
4	51	26	13793	2969.7	3.56	30	57.96	42.04
5	15	8	4464.8	440.4	0.51	8	56.71	43.29
6	39	20	12565	0	0	20	51.28	48.72
7	21	11	7465.7	0	0	11	52.38	47.62
8	9	5	4652.5	0	0	5	55.56	44.44
9	23	12	14586	0	0	12	52.17	47.83
10	11	6	8978.1	0	0	6	54.55	45.45
Total	251	130	84187	12813.3	16.8	146	58.48	41.52

Table-3

Module	a_i	a_{io}	Z_i*	Y_i*	$m_i(Yi)$	X_i*	m_i*	% of faults removed	% of faults remaining
1	63	32	13300	8624.7	11.74	21924.6	44	69.43	30.57
2	13	7	3064.6	1474.2	1.93	4538.82	9	68.7	31.3
3	6	3	1317.3	786.52	1.048	2103.79	4	67.47	32.53
4	51	26	13793	4134.5	5.13	17927.4	31	61.05	38.95
5	15	8	4464.8	793.16	0.984	5257.95	9	59.9	40.1
6	39	20	12565	0	0	12565.5	20	51.3	48.7
7	21	11	7465.7	0	0	7465.66	11	52.38	47.62
8	9	5	4652.5	0	0	4652.5	5	55.55	44.45
9	23	12	14586	0	0	14585.7	12	52.18	47.82
10	11	6	8978.1	0	0	8978.11	6	54.55	45.45
Total	251	130	84187	15813	20.8	100000	151	60.09	39.91

5. Conclusion

In this paper we have discussed a couple of optimization problems occurring during module testing phase of software development life cycle. 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 judicious allocation of resources. It is observed that after certain duration of testing, fault removal becomes difficult in the sense that greater effort will be required to remove each additional fault. As the reliability of software is of utmost importance scientific decision making is required while deciding the resource budget. The tradeoff as shown in section-4 can be useful in this regard.

Alternatively if the developer is not too keen on an optimal solution but is satisfied with an efficient solution, Goal Programming approach may be desirable in that case. We are further looking into this aspect.

Appendix:

Proof of the theorem-1

We have following recursion equations given in (7):

$$f_1(Z) = \max_{X_1 = Z} \left\{ a_1 (1 - e^{-b_1 X_1}) \right\}$$

$$f_n(Z) = \max_{0 \le X_n \le Z} \left\{ a_n (1 - e^{-b_n X_n}) + f_{n-1}(Z - X_n) \right\}, n = 2, \ldots, N$$

The above problem can be solved through forward recursion in N stages as follows.

Stage-1: Let $n=1$ then we have

$$f_1(Z) = \max_{X_1 = Z} \left\{ a_1 (1 - e^{-b_1 X_1}) \right\} = a_1 (1 - e^{-b_1 Z})$$

Stage-2: Let $n=2$ then we have

$$f_2(Z) = \max_{0 \le X2 \le Z} \left\{ a_2 (1 - e^{-b_2 X_2}) + f_1(Z - X_2) \right\}$$

Substituting $f_1(Z - X_2)$ in above we have

$$f_2(Z) = \max_{0 \le X2 \le Z} \left\{ a_2 (1 - e^{-b_2 X_2}) + a_1 (1 - e^{-b_1 (Z - X_2)}) \right\}$$

Now let, $F_2(X_2) = \left\{a_2(1-e^{-b_2X_2}) + a_1(1-e^{-b_1(Z-X_2)})\right\}$ then

$f_2(Z) = \max_{0 \le X2 \le Z} \{F_2(X_2)\}$

The maxima can be found through calculus.

$$\frac{dF_2(X_2)}{dX_2} = a_2b_2e^{-b_2X_2} - a_1b_1e^{-b_1(Z-X_2)}$$

The sufficiency condition can be checked through the second derivative condition:

$$\frac{d^2F_2(X_2)}{dX_2^2} = -a_2b_2^2e^{-b_2X_2} - a_1b_1^2e^{-b_1(Z-X_2)} \le 0$$

The following three situations can occur.

(i) $\frac{dF_2(X_2)}{dX_2} < 0$ (ii) $\frac{dF_2(X_2)}{dX_2} = 0$ (iii) $\frac{dF_2(X_2)}{dX_2} > 0$

If $\frac{dF_2(X_2)}{dX_2} < 0$, then X_2 =0.

At X_2 =0 $\frac{dF_2(X_2)}{dX_2} = a_2b_2 - a_1b_1e^{-b_1Z} < 0$

i.e. $1 \le e^{b_1Z} < \frac{a_1b_1}{a_2b_2}$

Which implies $a_1b_1 > a_2b_2$,in other words the detectability in module -1 is higher than module –2.

Similarly $\frac{dF_2(X_2)}{dX_2} > 0$ implies $X_2 = Z$

and we have $\frac{a_2b_2}{a_1b_1} > e^{b_2Z} \ge 1$

Hence $a_2b_2 > a_1b_1$, the testing resources would be allocated to module -2 first as the detectability is higher there.

Finally if $\frac{dF_2(X_2)}{dX_2} = 0$

$X_2^* = \frac{1}{b_1+b_2}\left\{b_1Z - \log\frac{a_1b_1}{a_2b_2}\right\}$, i.e. $X_2^* = \frac{1}{\mu_1+b_2}\left\{\mu_1Z - \log\frac{\mu_1V_1}{a_2b_2}\right\}$,

and $$f_2(Z) = \sum_{i=1}^{2} a_i - e^{-\frac{a_1 b_1}{a_2 b_2} Z} \left\{ a_2 \left(\frac{a_1 b_1}{a_2 b_2}\right)^{\frac{b_2}{b_1 + b_2}} + a_1 \left(\frac{a_2 b_2}{a_1 b_1}\right)^{\frac{b_1}{b_1 + b_2}} \right\}$$

i.e. $$f_2(Z) = \sum_{i=1}^{2} a_i - e^{-\mu_2 Z} V_2$$

Where $$\mu_1 = \frac{1}{\frac{1}{b_1}} = b_1,\ V_1 = a_1,\ \mu_2 = \frac{1}{\frac{1}{b_1} + \frac{1}{b_2}} = \frac{b_1 b_2}{b_1 + b_2}$$

$$V_2 = \left\{ a_2 \left(\frac{V_1 \mu_1}{a_2 b_2}\right)^{\frac{\mu_2}{\mu_1}} + V_1 \left(\frac{a_2 b_2}{V_1 \mu_1}\right)^{\frac{\mu_2}{b_2}} \right\}$$

Now proceeding by induction it can be shown for nth stage,

$$X_n^* = \frac{1}{\mu_{n-1} + b_n} \left\{ \mu_{n-1} Z - \log \frac{\mu_{n-1} V_{n-1}}{a_n b_n} \right\}$$

and $$f_n(Z) = \sum_{i=1}^{n} a_i - e^{-\mu_n Z} V_n \text{ for } n=1\ldots N$$

The proof is complete.

References

1. Goel A.L., Software Reliability Models: Assumptions, limitations and applicability, IEEE Trans. On software engineering, SE-11, pp. 1411-1423, 1985.
2. Hadley, G., Nonlinear and Dynamic Programming, Addision-Wesley, Reading Mass, 1964.
3. Ichimori, T, Yamada, S. And Nishiwaki M., Optimal allocation policies for testing-resource based on a Software Reliability Growth Model, Proceedings of the Australia –Japan workshop on stochastic models in engineering, technology and management, pp. 182-189, 1993.
4. Kapur P.K. and Garg R.B.; Cost reliability optimum release policies for a software system with testing effort, OPSEARCH, vol. 27, no. 2, pp. 109-118, 1990.
5. Kapur P.K., Garg R.B. and Kumar, S.; Contributions to Hardware and Software Reliability, World Scientific, Singapore, 1999.

6. Kapur, P.K. and Bardhan, A.K., Modelling, allocation and control of resources: an interdisciplinary approach in software reliability and marketing, Operations Research, Eds. M. Agarwal and K. Sen, Narosa Publishing House, New Delhi 2001.
7. Kubat P. and Koch H.S., Managing test procedures to achieve reliable software, IEEE Trans. On Reliability, Vol. R-32, pp. 299-303, 1983.
8. Musa J.D., Iannino A. And Okumoto K, Software Reliability-Measurement, Prediction and Application, Mc Graw Hill, 1987.
9. Yamada S. And Ohtera H. And Narihisa H., Software Reliability Growth Model with testing effort, IEEE Trans. On Reliability, vol. R-35, pp. 19-23, 1986.