Genetic Algorithm Based Optimal Testing Effort Allocation Problem for Modular Software

Anu G. Aggarwal[1], P. K. Kapur[2], Gurjeet Kaur[3] and Ravi Kumar[4]

Submitted in April 2010; Accepted in September 2011

Abstract - ***Software reliability growth models (SRGM) are used to assess modular software quantitatively and predict the reliability of each of the modules during module testing phase. In the last few decades various SRGM's have been proposed in literature. However, it is difficult to select the best model from a plethora of models available. To reduce this difficulty, unified modeling approaches have been proposed by many researchers. In this paper we present a generalized framework for software reliability growth modeling with respect to testing effort expenditure and incorporate the faults of different severity. We have used different standard probability distribution functions for representing failure observation and fault detection/ correction times. The faults in the software are labeled as simple, hard and complex faults. Developing reliable modular software is necessary. But, at the same time the testing effort available during the testing time is limited. Consequently, it is important for the project manager to allocate these limited resources among the modules optimally during the testing process. In this paper we have formulated an optimization problem in which the total number of faults removed from modular software is (which include simple, hard and complex faults) maximized subject to budgetary and reliability constraints. To solve the optimization problem we have used genetic algorithm. One numerical example has been discussed to illustrate the solution of the formulated optimal effort allocation problem.***

Index Terms - ***Non-homogenous Poisson process, software reliability growth model, Probability Distribution Functions, Fault Severity, Genetic Algorithm***

1. INTRODUCTION

Nowadays large and complex software systems are developed by integrating a number of small and independent modules. Modules can be visualized as independent softwares performing predefined tasks, mostly developed by separate teams of programmers and sometimes at different geographical locations. During the development of modular software, faults can crop in the modules due to human imperfection. These faults manifest themselves in terms of failures when the modules are tested independently during the module testing phase of software development life cycle. However, in today's computer invaded world these failures can lead to big losses in terms of money, time and life. Thus it is very important to evaluate software reliability of each module during modular testing phase.

To assess modular software quantitatively and predict the reliability of each of the modules during module testing, software reliability growth models (SRGM) are used. Numerous SRGM's, which relate the number of failures (fault identified) and the Execution time (CPU time/Calendar time) have been discussed in the literature [19,5,3]. All these SRGMs assume that the faults in the software are of the same type. However, this assumption is not truly representative of reality. The software includes different types of faults, and each fault requires different strategies and different amounts of testing effort for removal. Ohba [8] refined the Goel-Okumoto[1] model by assuming that the fault detection/removal rate increases with time and that there are two types of faults in the software. SRGM proposed by Bittanti et al. [22] and Kapur and Garg [13] has similar forms as that of Ohba [8] but they developed under different set of assumptions. These models can describe both exponential and S-shaped growth curves and therefore are termed as flexible models [22, 8, 13]. Kapur et al. [16] developed Flexible software reliability growth model with testing effort dependent learning process in which two types of software faults were taken. Further, they proposed an SRGM with three types of faults [19]. The first type of fault was modeled by an Exponential model of Goel and Okumoto [1]. The second type was modeled by Delayed S-shaped model of Yamada et al. [21]. The third type was modeled by a three-stage Erlang model proposed by Kapur et al. [19]. The total removal phenomenon was modeled by the superposition of the three SRGMs. Shatnawi and Kapur [11] later proposed a generalized model based on classification of the faults in the software system according to their removal complexity.

The above literature review reveals that in the last few decades several SRGM's have been proposed. This plethora of SRGM's makes the model selection a tedious task. To reduce this difficulty, unified modeling approaches have been proposed by many researchers. The work in this area started as early as in 1980s with Shantikumar [4] proposing a Generalized birth process model. Gokhale and Trivedi [23] used Testing coverage function to present a unified framework and showed how NHPP based models can be represented by probability

[1,2,3,4] *Department of Operational Research, University of Delhi, Delhi-110007*
E-Mail: [1]*anuagg17@gmail.com*, [2]*pkkapur1@gmail.com*, [3]*gurjeetkaur85@gmail.com* and [4]*sianaravi@gmail.com*

distribution functions of fault –detection times. Another unification methodology is based on a systematic study of Fault detection process (FDP) and Fault correction process (FCP) where FCPs are described by detection process with time delay. The idea of modeling FCP as a separate process following the FDP was first used by Schneidewind [10]. More general treatment of this concept is due to Xie et al [9] who suggested modeling of Fault detection process as a NHPP based SRGM followed by Fault correction process as a delayed detection process with random time lag. The unification scheme due to Kapur et al [17] is based on Cumulative Distribution Function for the detection/correction times and incorporates the concept of change point in Fault detection rate. These schemes have proved to be fruitful in obtaining several existing SRGM by following single methodology and thus present a perceptive investigation for the study of general models without making many assumptions. In this paper we made use of such unified scheme for presenting a generalized framework for software reliability growth modeling with respect to testing effort expenditure and incorporate the faults of different severity. We have used different standard probability distribution functions for representing failure observation and fault correction times Also , the total number of faults in the software are labeled as simple, hard and complex faults .It is assumed that the testing phase consists of three different processes, namely failure observation, fault isolation and fault removal. The time delay between the failure observation and subsequent removal is assumed to represent the severity of the fault.

Developing reliable modular software is necessary. But, at the same time the testing effort available during the testing time is limited. These testing efforts include resources like human power, CPU hours, and elapsed time, etc. Hence, to develop a good reliable software system, a project manager must determine in advance how to effectively allocate these resources among the various modules. Such optimization problems are called "Resource Allocation problems". Many authors have investigated the problem of resource allocation [2, 7]. Kapur et al [20, 15] studied various resource allocation problems maximizing the number of faults removed form each module under constraint on budget and management aspirations on reliability for exponential and S-shaped SRGMs [1,19,8].In this paper we have formulated an optimization problem in which the total number of faults removed from modular software is (which include simple, hard and complex faults) maximized subject to budgetary and reliability constraints.

To solve the effort allocation problem formulated in this research paper we use Genetic Algorithm(GA). GA stands up a powerful tool for solving search & optimization problems. The complex non linear formulation of the optimal effort allocation problem is the reason behind choosing genetic algorithm as the solving tool. GA always considers a population of solutions. There is no particular requirement on the problem before using GA's, as it can be applied to solve any kind of problem.

The paper is organized as follows. Section 2 gives the generalized framework for developing the software reliability growth model for faults of different severity. In section 3 parameter estimation and model validation of the proposed model is done through SPSS. The testing effort allocation problem is formulated in section 4. In section 5 genetic algorithm is presented for solving the discussed problem. Section 6 illustrates the optimization problem solution through a numerical example. Finally, conclusions are drawn and are given in section 7.

2.1 Notations

W(t) : Cumulative testing effort in the interval (0.t].

w(t) : Current testing-effort expenditure rate at testing time t.

$\frac{d}{dt}W(t) = w(t)$

$m_j(W_t)$: Expected number of faults removed of type j(j=simple, Hard, Complex Faults).

$m(W_t)$: Expected number of total faults removed.

b : Constant fault detection rate.

β : rate of consumption of testing-effort

$\lambda(W_t)$: Intensity function for Fault correction process (FCP) or Fault correction rate per unit time.

$G(W_t)$, $F(W_t)$, $H(W_t)$: Testing effort dependent Probability Distribution Function for Failure observation, Fault Detection and Fault Correction Times

$g(W_t)$, $f(W_t)$, $h(W_t)$: Testing effort dependent Probability Density Function for Failure observation, Fault Detection and Fault Correction Times

* : Convolution.

$\otimes$: Steiltjes convolution.

2.2 Basic Assumptions

The proposed model is based upon the following basic assumptions:

1. Failure occurrence, fault detection, or fault removal phenomenon follows NHPP.
2. Software is subject to failures during execution caused by faults remaining in the software.
3. The faults existing in the software are of three types: simple, hard and complex. They are distinguished by the amount of testing effort needed to remove them
4. Fault removal process is prefect and failure observation/fault isolation/ fault removal rate is constant.
5. Each time a failure occurs, an immediate effort takes place to decide the cause of the failure in order to remove it. The time delay between the failure observation and its subsequent fault removal is assumed to represent the severity of the faults. The more severe the fault, more the time delay.

6. The fault isolation/removal rate with respect to testing effort intensity is proportional to the number of observed failures.

2.3 Modeling Testing Effort

The proposed SRGM in this paper takes into account the time dependent variation in testing effort. The testing effort (resources) that govern the pace of testing for almost all the software projects are Manpower and Computer time.

To describe the behavior of testing effort, Exponential, Rayleigh, or Weibull function has been used.

The testing-effort described by a Weibull-type distribution is given by:

$$W(t)=\alpha\cdot\left[1-\exp(-\int_0^t g(\tau)d\tau\right] \quad (1)$$

In equation (1), if g(t)=β.

Then, there is an exponential curve, and the cumulative testing-effort in (0,t] is $W(t)=\alpha\cdot\left[1-\exp(-\beta\cdot t)\right]$. (2)

Similarly in (1) if $g(t)=\beta\cdot t.$

Then, there is a Rayleigh curve and the cumulative testing-effort is given by: $W(t)=\alpha\cdot\left(1-\exp\left[-\frac{\beta}{2}\cdot t^2\right]\right)$. (3)

And if $g(t)=\gamma.\beta\cdot t.^{\gamma-1}$ in (1), then

$$W(t)=\alpha\cdot\left(1-\exp\left[-\beta\cdot t^{\gamma}\right]\right). \quad (4)$$

which is cumulative testing effort of Weibull curve.

2.4 Model Development

Let a_1, a_2 and a_3 be the simple, hard and complex faults respectively at the beginning of testing. Also 'a' is the total fault content i.e. $a= a_{1+} a_{2+} a_3$.

2.4.1 Modeling Simple Faults

Simple faults are the faults which can be removed instantly as soon as they are observed. The mean value function for the simple faults of the software reliability growth model with respect to testing effort expenditure can be written as [18]:

$$m_1(W_t)= a_1\ F(W_t) \quad (5)$$

where, $F(W_t)$ is testing effort dependent distribution function.

From Equation (5), the instantaneous failure intensity function $\lambda(W_t)$ is given by:

$$\lambda(W_t)=a_1F'(W_t) \quad (6)$$

Or we can write

$$\lambda(W_t)=\frac{dm/dt}{dW_t/dt}=[\,a_1-m(W_t)\,]\frac{F'(W_t)}{1-F(W_t)} \quad (7)$$

2.4.2 Modeling Hard Faults

The hard faults consume more testing time for the removal. This means that the testing team will have to spend more time to analyze the cause of the failure and therefore requires greater time to remove them. Hence the removal process for hard faults is modeled as a two-stage process and is given by[18]:

$$m_2(W_t)=a_2(F\otimes G)(W_t)\text{, and} \quad (8)$$

$$\lambda(W_t)=\frac{(f*g)(W_t)}{1-(F\otimes G)(W_t)}[\,a_2-m(W_t)\,] \quad (9)$$

2.4.3 Modeling Complex Faults

These faults require more testing time for removal after isolation as compared to hard fault removal. Hence they need to be modeled with greater time lag between failure observation and removal. Thus, the removal process for complex faults is modeled as a three-stage process:

$$m_3(W_t)=a_3(F\otimes G\otimes H)(W_t) \quad (10)$$

And the instantaneous failure intensity function $\lambda(W_t)$ is:

$$\lambda(W_t)=\frac{(f*g*h)(W_t)}{1-(F\otimes G\otimes H)(W_t)}[\,a_3-m(W_t)\,] \quad (11)$$

2.4.4 Modeling Total Faults

The total fault removal phenomenon is the superimposition of the simple, hard and complex faults, and is therefore given as:

$$m(W_t)=m_1(W_t)+m_2(W_t)+m_3(W_t) \quad (12)$$
$$=a_1F(W_t)+a_2(F\otimes G)(W_t)+a_3(F\otimes G\otimes H)(W_t)$$

A particular case of the proposed model is tabulated in Table 2.1

Faults	F(W_t)	G(W_t)	H(W_t)	m(W_t)
Simple	$W_t\sim$ exp(b_1)	-	-	$m_1(W_t)=a_1\left[1-e^{-bW_t}\right]$
Hard	$W_t\sim$ exp(b_2)	$W_t\sim$ exp(b_2)	-	$m_2(W_t)=a_2\left[1-\left((1+bW_t)e^{-bW_t}\right)\right]$
Complex	$I(W_t)$	$I(W_t)$	$W_t\sim N(\mu,\sigma^2)$	$m_3(W_t)=a_3[\Phi(W_t,\mu,\sigma^2)]$

MVF of Total Fault

$$m(W_t) = a_1\left[1-e^{-bW_t}\right] + a_2\left[1-\left((1+bW_t)e^{-bW_t}\right)\right] + a_3\left[\left(\Phi\left(W_t,\mu,\sigma^2\right)\right)\right]$$

Table 2.1: A Particular Case

2.5 Reliability Evaluation

Using the SRGM we can evaluate the reliability of the software during the progress of testing and predict the reliability at the release time. Reliability of software is defined as "given that the testing has continued up to time t, the probability that a software failure does not occur in time interval $(t, t+\Delta t)\,(\Delta t \geq 0)$". Hence the reliability of software is represented mathematically as

$$R(t) \equiv R(t+\Delta t \mid t) = \exp^{-(m(t+\Delta t)-m(t))} \quad (13)$$

Another measure of software reliability at time t is defined as "the ratio of the cumulative number of detected faults at time t to the expected number of initial fault content of the software" given by[4]:

$$R(t) = \frac{m(t)}{a} \quad (14)$$

To incorporate the effect of testing effort in the reliability estimation of each module Equation (14) can be modified as:

$$R(W_t) = \frac{m(W_t)}{a} \quad (15)$$

3. PARAMETER ESTIMATION AND MODEL VALIDATION

To measure the performance of the proposed model we have carried out the parameter estimation on the data set cited in M.Ohba [8](DS-I). The software was tested for 19 weeks during which 47.65 computer hours were used and 328 faults were removed. The estimation results for Exponential, Rayleigh, and Weibull function are given in table 3.1

Testing Effort Function	Parameter Estimation for DS-I			
	α	β	γ	R^2
Exponential function	19029.3	0.0001	-	0.992
Rayleigh function	49.2961	0.0137		0.974
Weibull function	782.603	0.0023	1.114	0.996

Table 3.1: Testing Effort Function Parameter Estimates

Weibull effort function is chosen to represent the testing effort as it provided the best fit on the testing effort data (based on the highest value of R^2.) Based upon these estimated parameters, parameters of proposed SRGM were estimated. The goodness of fit measures used are Mean Square Error (MSE) and Coefficient of multiple determination (R^2). The results are compared with SRGM proposed by Kapur et al. [19] with three types of fault. The results are tabulated in table 3.2 (Letting $b_1=b_2=b_3=b$)The goodness of fit curves for DS-I is given in Figure: 3.1

Paramter Estimates	Proposed Model	Kapur et al. Model [19]
a	353	378
b	0.05218	0.09722
μ	26.71107	-
σ	6.530279	-
$\mathbf{R^2}$	0.996	0.992
MSE	38.79684	75.31579

Table 3.2: Parameter Estimates for DS-I

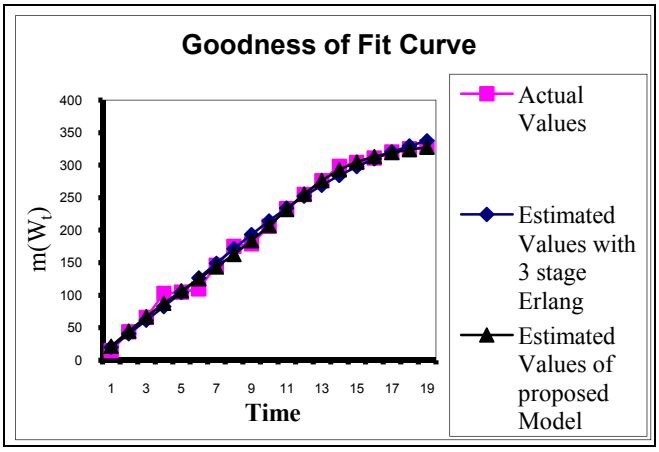

Figure3.1: Goodness of Fit Curve for DS-I

4. TESTING RESOURCE ALLOCATION PROBLEM

4.1 Notations:

j : 1,2,3; Simple faults-1;Hard Faults-2, Complex Faults-3

i : Module, 1,2..N

N : Total number of modules

$m_i(W_t)$: Mean value function for i^{th} module

b_{ji} : Constant fault detection rate for j^{th} fault type in i^{th} module

a_{ji} : Constant, representing the number of j^{th} fault type lying dormant in i^{th} module at the beginning of testing,

c_{ji} : Cost of removing j^{th} fault from i^{th} module

W_i : Testing effort for i^{th} module

R_i : Reliability of each module

B : Total cost of removing different types of faults

W : Total testing effort expenditure

4.2 Mathematical Formulation

Consider software with 'N' modules where each module is different in size, complexity, the functions they perform etc. In each module there are three types of faults; simple, hard and complex. The software has to be released in the market at a predefined software release time with limited availability of testing resources expenditure. Further the cost of removing the fault from each module is dependent on its severity.

Therefore, the problem of maximizing the faults of each of N independent modules such that reliability of each module is at least R_0 is formulated as:

Maximize

$$m(W_i) = \sum_{i=1}^{N} m_i(W_i)$$
$$= \sum_{i=1}^{N} \left(a_{1i} \left(1 - e^{-b_{1i} W_i} \right) \right) + \sum_{i=1}^{N} \left(a_{2i} \left(1 - (1 + b_{2i} W_i) e^{-b_i W_i} \right) \right)$$
$$+ \sum_{i=1}^{N} \left(a_{3i} \left[\left(\Phi_i \left(W_i, \mu_i, \sigma_i^2 \right) \right) \right] \right)$$

Subject to:

$$\sum_{i=1}^{N} \left(C_{1i} m_{1i}(W_i) + C_{2i} m_{2i}(W_i) + C_{3i} m_{3i}(W_i) \right) \le B \qquad i = 1, 2 \ldots N$$

$$\sum_{i=1}^{N} W_i \le W \qquad i = 1, 2 \ldots N$$

$$R_i \ge R_0 \qquad i = 1, 2 \ldots N \qquad \text{(P1)}$$

$$W_i \ge 0 \qquad i = 1, 2 \ldots N$$

5. GENETIC ALGORITHM FOR TESTING RESOURCE ALLOCATION

The above optimization problem is solved by a powerful computerized heuristic search and optimization method, viz. genetic algorithm (GA) that is based on the mechanics of natural selection and natural genetics. In each iteration (called generation), three basic genetic operations i.e., selection /reproduction, crossover and mutation are executed.

For implementing the GA in solving the allocation problem, the following basic elements are to be considered.

5.1 Chromosome Representation

Genetic Algorithm starts with the initial population of solutions represented as chromosomes. A chromosome comprises genes where each gene represents a specific attribute of the solution. Here the solution of the testing-effort allocation problem in modular software system includes the effort resources consumed by individual modules. Therefore, a chromosome is a set of modular testing effort consumed as part of the total testing effort availability.

5.2 Initial Population

For a given total testing time W, GA generates the initial population randomly. It initialize to random values within the limits of each variable.

5.3 Fitness Of A Chromosome

The fitness is a measure of the quality of the solution it represents in terms of various optimization parameters of the solution. A fit chromosome suggests a better solution. In the effort allocation problem, the fitness function is the objective of testing effort optimization problem along with the penalties of the constraints that are not met.

5.4 Selection

Selection is the process of choosing two parents from the population for crossover. The higher the fitness function, the more chance an individual has to be selected.

The selection pressure drives the GA to improve the population fitness over the successive generations. Selection has to be balanced with variation form crossover and mutation. Too strong selection means sub optimal highly fit individuals, will take over the population, reducing the diversity needed for change and progress; too weak selection will result in too slow evolution. We use "***Tournament selection"*** here.

5.5 Crossover

Crossover is the process of taking two parent solutions and producing two similar chromosomes by swapping sets of genes, hoping that at least one child will have genes that improve its fitness. In the testing resource allocation problem, crossover diversifies the population by swapping modules with distinct time consuming, particularly when the population size is small.

5.6 Mutation

Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly disturbing genetic information.

The important parameter in the mutation technique is the mutation probability. The mutation probability decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. In our problem of testing resource allocation, we have used a mutation probability of 10%.

With the basic modules of genetic algorithm described above, the procedure for solving the optimal effort allocation problem is as follows [6]:

Step 1: Start

Step 2: Generate random population of chromosomes

Step 3: Evaluate the fitness of each chromosome in the population

Step 4: Create a new population by repeating following steps until the new population is complete:

[Selection] Select two parent chromosomes from a population according to their fitness
[Crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover is performed, offspring is the exact copy of parents.
[Mutation] With a mutation probability, mutate offspring at each locus (position in chromosome)
[Accepting] Place new offspring in the new population
[Replace] Use new generated population for further sum of the algorithm.
[Test] If the end condition is satisfied, stop and return the best solution in the current population
[Loop] Go to step 3 for fitness evaluation

6. NUMERICAL EXAMPLE

The Effort Allocation Problem described in section 4 is illustrated numerically in this section. Consider a software system consisting of three modules, whose parameters have already been estimated using software failure data. These parameter estimates for each module is shown in Table 6.1. The total testing resources available is assumed to be 5000 units. Total cost for removing the different types of faults is 10000 units. Also, it is desired that the reliability of each module is at least 0.9.

odule	a_1	a_2	a_3	b	c_1	c_2	c_3	μ	σ
1	313	107	81	0.00368	5	10	15	16.292	5.586
2	332	97	76	0.00234	5	10	15	14.987	4.123
3	298	64	32	0.0018	5	10	15	12.456	7.654

Table 6.1: Parameter Estimates for effort allocation problem

Based on the above information, the problem (P1) is solved using genetic algorithm. The parameters used in GA evaluation are given in table 6.2.

Parameter	Value
Population Size	106
Number of Generations	26
Selection Method	Tournament
Crossover Probability	0.9
Mutation Probability	0.1

Table 6.2: Parameter of the GA

The optimal testing time allocation to each type of fault in module and hence total fault removed from each module and their corresponding cost of removing is shown in table 6.3.

Module	W	m_1	m_2	m_3	m	Reliability	Cost of removing faults
1	1192.22	309	100	81	490	0.978	3758.87
2	1602.294	324	86	76	486	0.962	3622.524
3	2202.934	292	58	32	382	0.969	2521.486
Total	4997.448				1358		9902.88

Table 6.3: The optimal testing effort expenditure with the corresponding cost of each module

7. CONCLUSION

In this paper we have discussed the problem for modular software at the unit testing stage. We have made use of unified scheme for presenting a generalized framework for Software reliability growth modeling with respect to testing effort expenditure and incorporated the faults of different severity. The faults in each module are of three types-simple, hard and complex. Further we have optimally allocated the testing effort to each type of fault and the modules and have found out the different types of faults removed in the modules with a fixed budget and a prerequisite level of reliability. Genetic Algorithm is developed to solve the problem of resource allocation. Numerical example is discussed to illustrate the solving of the discussed optimization problem through GA.

FUTURE SCOPE

The present study is done under the assumption of independence of the failures of different modules. In future, dependence of the failures from different modules as well as the architecture styles and connectors reliability can also be studied.

REFERENCES

[1]. A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, Vol. 28, No. 3, pp 206–211, 1979.
[2]. H. Ohetera and S Yamada., "Optimal allocation and control problems for software testing resources", IEEE Transactions on Reliability, Vol. 39, No. 2, p. 171-176, 1990.
[3]. H. Pham, System Software Reliability, Reliability Engineering Series, Springer, 2006.
[4]. J. G. Shanthikumar, , "A General Software Reliability Model For Performance Prediction, Microelectronics Reliability", Vol. 21, pp. 671–682, 1981.
[5]. J.D. Musa, A. Iannino and K. Okumoto, Software Reliability: Measurement, Prediction, Applications, McGraw Hill, 1987.
[6]. K. Sastry ,'Single and Multiobjective Genetic Algorithm Toolbox for Matlab in C++' (IlliGAL Report No. 2007017) Urbana, IL: University of Illinois at Urbana-Champaign, 2007

[7]. Lo. Huang and Lyu Kuo, "Optimal allocation of testing resources considering cost, reliability, and testing – effort", Proceedings of the 10th IEEE Pacific International Symposium on dependable Computing, 2004

[8]. M. Ohba, "Software reliability analysis models," IBM Journal of Research and Development, vol. 28, no. 4, pp. 428–443, 1984.

[9]. M. Xie, and B. Yang, "Optimal Testing time Allocation for Modular Systems", International Journal of Quality and Reliability Management, Vol. 18, No. 4, 854-863,2001.

[10]. N.F. Schneidewind, , Analysis Of Error Processes In Computer Software, Sigplan Notices, Vol. 10, pp. 337–346, 1975.

[11]. O. Shatnawi, P.K. Kapur, "A Generalized Software Fault Classification Model", WSEAS Transactions on Computers, Vol. 2, No. 9, pp. 1375-1384, 2008

[12]. P. K Kapur., S.Younes, and S. Agarwala, "Generalised Erlang model with n types of faults," *ASOR Bulletin*, Vol. 14, No.1, pp. 5–11, 1995.

[13]. P. K. Kapur and. R. B. Garg, "Software reliability growth model for an error-removal phenomenon," *Software Engineering Journal*, Vol. 7, No. 4, pp. 291–294, 1992.

[14]. P. K., Kapur. V. B. Singh, and B Yang., "Software reliability growth model for determining fault types," in Proceedings of the 3rd International Conference on Reliability *and Safety Engineering (INCRESE '07)*,pp. 334–349, 2007.

[15]. P.K Kapur., P.C. Jha, A.K. Bardhan, , "Dynamic programming approach to testing resource allocation problem for modular software", in Ratio Mathematica, Journal of Applied Mathematics, Vol. 14, pp. 27-40, 2003.

[16]. P.K. Kapur, D. N .Goswami, A. Bardhan, O. Singh, "Flexible software reliability growth model with testing effort dependent learning process", Applied Mathematical Modelling, Vol. 32, No. 7, pp. 1298–1307,2008

[17]. P.K. Kapur, J. Kumar and R. Kumar, "A Unified Modeling Framework Incorporating Change Point For Measuring Reliability Growth During Software Testing". To appear in OPSEARCH.

[18]. P.K. Kapur, O. Shtnawi, A Aggarwal., R. Kumar, "Unified Framework for Developing Testing Effort Dependent Software Relaibility Growth Models", WSEAS Transactions on Systems, Vol. 8 No. 4,pp 521-531,2009

[19]. P.K. Kapur, R. B. Garg and S. Kumar, Contributions to Hardware and Software Reliability, Singapore, World Scientific Publishing Co. Ltd., 1999.

[20]. P.K.Kapur, P.C. Jha, A.K. Bardhan, "Optimal allocation of testing resource for a modular software", Asia Pacific Journal of Operational Research, Vol. 2, No. 3, pp. 333-354, 2004.

[21]. S. Yamada, M. Ohba, and S. Osaki, "S-shaped software reliability growth models and their applications," *IEEE Transactions on Reliability*, vol. 33, No. 4, pp. 289–292, 1984.

[22]. S. Bittanti, P.Bolzern, E.Pedrotti, and R. Scattolini, "A flexible modeling approach for software reliability growth," in *Software Reliability Modelling and Identification*, G. Goos and J. Harmanis, Eds., Springer, Berlin, Germany, pp 101–140, 1998.

[23]. S.S Gokhale., T. Philip, P.N. Marinos and K.S. Trivedi, "Unification of Finite Failure Non-Homogeneous Poisson Process Models through Test Coverage", In Proc. Intl. Symposium on Software Reliability Engineering (ISSRE 96), pp 289-299, October 1996.