IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 6, Ver. III (Nov.-Dec. 2016), PP 109-116
www.iosrjournals.org

Four Problem Minimization in Software Testing Resource Allocation

K. SaiSindhuja, A. Yuva krishna
(CSE, PVPSIT/ JNTUK, India)
(CSE, PVPSIT/ JNTUK, India)

Abstract: *Cumulative software testing resource for software faults needs a software development planning model that were based on expected problems and this plan has been formulated for four software development planning problems. Previously optimization is done on effort, cost and fault minimization. Now cumulative estimation on development phase is absorbed with realistic assumptions for this search based software engineering. This helps in optimizing the testing resource allocation in realistic. Cumulative with linear programming are for solving the problem. Software testing Resource allocation and release time decisions are vital for the software systems. The objective behind such critical decisions may differ from firm to firm. The motive of the firm may be maximization of software reliability or maximization of number of faults to be removed from each module or it may be minimization of number of faults remaining in the software or minimization of testing resources. Taking into consideration these different aims, various authors have investigated the problem of resource allocation and release time problem. In this paper we investigate various software release policies and resource allocation problem, for example, policies based on the dual constraints of cost and reliability.*

Keywords: *development planning model, operational profile, software reliability model, software testing-resource allocation.*

I. Introduction

The test objective is to provide adequate coverage, requirements validation and improve the quality of code being tested such that future maintenance can be done easily and potential faults will be at a minimal. Testing will be performed using a vast number of tools with scheduled and unscheduled testing sessions. Each testing session will be administered after each deliverable is met. The following document will serve to provide more detail on this matter.

According to organization standards, we will begin the testing process by first prioritizing features. This will be done to decide which feature the testers will begin with. Based on the background of the game, prioritization of features will be decided by using the following criteria: Reference Count, Distance, Weighted Methods and Average Cyclomatic Complexity. The tester will run software analytic tools to calculate the metric value. Once this is complete, the total priority for the feature will be calculated by summing the weight of each criteria by the percentage of the recorded metric in relation with the other features. After prioritization of features is complete, the testers will begin with the highest prioritization feature.

Within the feature, the testers will begin by standardizing the code style according to organization standards. Following standardization, a combination of manual and automated testing will be performed to increase the coverage of the feature to at least 85% or above. If time permits, after achieving 85% coverage, the process of prioritization will be performed again with the remaining features to determine which feature to test next. After determining the feature to test, the same process will be followed as before (i.e., standardization, manual and automated testing). Additionally, regression testing will be done to ensure previous modifications are still valid. This procedure will be performed iteratively after each feature until all have been tested.

II. Literature Survey

The development of high superiority software satisfying cost, a schedule and resource requirement is a vital prerequisite for improved competitiveness of any organization. One major difficulty to master this challenge is the inevitability of defects in software products. The testing of software systems are focusing to strong conflicting forces. One of the most effective ways to do this is to apply software reliability engineering during testing and development phase. Software reliability engineering delivers the desired functionality for a product much more efficiently by quantitatively characterizing its expected use. The software reliability engineering tends to increase reliability while decreasing software development time and overall cost. Thus software reliability engineering balances customer needs for the major quantitatively measurable quality characteristics of reliability, availability, delivery time and life cycle cost more effectively. In the literature, several researchers have developed models for cost benefit analysis of the testing process, all based on the initial

cost model described by [6]. [7] developed and solved two resource allocation problems for modular software, minimizing the total number of remaining faults in the software modules when the amount of available testing resources is previously specified and vice versa. [8] Studied the allocation problem minimizing the mean number of remaining faults in the software modules with a reliability aspiration and budget constraint. [9] Determined the optimal amount of resources needed for software module testing using the hyper-geometric software reliability growth model. [10] Discussed resource allocation problem to maximize the total fault removal from software consisting of several independent components. For the resulting optimization problem, they demand marginal testing effort function (MTEF), where the testing resource consumption was represented in terms of fault removal. [11] Discussed dynamic programming approach to testing resource allocation Problem for modular software in this Two Optimization models are proposed for optimal allocation of testing resources among the modules of Software.

In the first model, authors maximize the total fault removal, subject to cost Constraint. In another model, other constraint representing aspiration levels for fault removals for each module of the software is added. Authors solved this models using dynamic programming technique. A dynamic programming approach for finding the optimal solution has been proposed. The methods have been already illustrated through numerical examples. Further [12] studied various resource allocation problems maximizing the number of faults removed from each module under constraint on budget and management aspirations on reliability for exponential and S-shaped SRGMs [6,13,14]. They have discussed mathematical, dynamic and goal programming approaches to yield solutions of such class of optimization problems. [15] Also solved resource allocation problem maximizing the number of faults removed from each module under constraint on budget. [16] Investigated an optimal resource allocation problem in modular software systems during testing phase. The main goal is to minimize the cost of software development when the number of remaining faults is to minimize and a desired reliability objective is given. Authors analyzed the sensitivity of parameters of proposed software reliability growth models.

In addition, they also see the impact on the resource allocation problem if some parameters are either overestimated or underestimated. Authors evaluated the optimal resource allocation problems for various conditions by examining the behavior of the parameters with the most significant influence which have been stated by solved numerical example.[17] Considers two kinds of software testing resource allocation problems. The first problem is to minimize the total number of remaining faults, given a fixed quantity of testing-effort, and a reliability objective. The last problem is to optimize the amount of software testing effort given the total number of remaining faults in software, and a desired reliability objective.

Author has also proposed several strategies for module testing to help software project managers to solve these problems, and make the best decisions. Author provides several systematic solutions based on a nonhomogeneous Poisson process model (NHPP) model, allowing efficient allocation of a specified amount of testing resource expenditures for each and every software module under some specified constraints. Author also describes numerical examples on the optimal testing resource allocation problems and performed sensitivity analysis.[18] In this paper, ensemble models are developed to correctly estimate software reliability. Three linear ensembles and one non-linear ensemble are planned and tested. Based on the experiments conducted on the software reliability data obtained from research paper, it can observe that the non-linear ensemble outperformed all the other ensembles. Both theoretical and empirical works indicate that ensemble can be an effective and efficient way to improve accuracies. [19] Discussed an idea of genetic algorithm used in software reliability. The reliability of software systems is essentially a mathematical programming problem composed by the corresponding objective function and restriction constraints two parts. The problem is usually with a large number of local extreme points and often is no differentiable, discontinuous, multi-dimensional, highly nonlinear combinatorial optimization problem.

Therefore, to accurately solve the global optimal solution is generally difficult. Traditional solving algorithm can‘t obtain satisfied global optimal solutions. The genetic algorithm is applied once again in the reliability of software systems for the global optimization distribution of the reliability indicators. Comparing with the traditional software reliability allocation techniques, genetic algorithm has shown good results in the solving process of searching for optimal allocation schemes. At the same time, the reliability distribution of the software system is essentially a type of combinatorial optimization problems with constraint conditions, and relative to the traditional solving methods, genetic algorithm can display a better performance. [20] Formulated an optimization problem to allocate the resources among different modules under an assumption that change point is generated in each module by design due to change in testing strategy, running environment, defect density or testing skill maturity such that the total fault removal is maximized while incorporating the effect of both types of imperfect debugging. Also, a correct Understanding of modular system cannot be attained without giving a prime concern to the accurate effect of each module on the overall performance. The weightage of each and every module will be specified. This relative importance in this work is obtained using Analytical Hierarchy Process (ARP). Author has also considered the problem of determining minimum requirements for the testing

resources so that a desired proportion of faults are removed from each module. [21] Discussed to solving Optimal Testing Resource Allocation Problems OTRAPs with Multi-Objective Evolutionary Algorithms (MOEAs). Specifically, author formulated OTRAPs as two types of multi-objective problems. First, consider the reliability of the system and the testing cost as two objectives. Second, the total testing resources used is also taken into account as the third objective.

The benefits of MOEAs over state-of-the-art single objective approaches to OTRAPs will be shown through empirical studies. This study has revealed that a well-known MOEA, namely No dominated Sorting Genetic Algorithm II (NSGA-II), performs well on the first problem formulation, but fails on the second one. Author did experimental study and compared MOEAs with Single-Objective Approaches and Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) With NSGA-II on Hence, a Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) is proposed. [22] Formulated an optimization problem in which the total number of faults removed from modular software which includes simple, hard and complex faults.

During the software testing many of the resources like time, effort and budget are consumed. The main work of the manager is to allocate the resources in a constrained manner such that the effort can be optimally allocated and overall budget is minimized. In this paper author proposed an imperfect debugging SRGM during testing and resource allocation is done based on optimizing the effort and reliability. Author used two testing resource allocation schemes one by minimizing the number of remaining faults and allocating the resources to attain the maximum reliability. An experimental result also shows the proposed model well fitted for software testing.

III. Methodology

There are many papers that have addressed the problems of optimal resource allocation. In this paper, we first consider two kinds of software testing-resource allocation problems, and then propose several strategies for Module testing. Namely, we provide systematic methods for the software project managers to allocate a specific amount of test expenditures for each module under some constraints, such as *1) minimizing the number of remaining faults with a reliability objective*, or *2) minimizing the amount of testing-effort with a reliability objective*. Here we employ a srgm with generalized *logistic* testing-effort function to describe the time-dependency behaviors of detected software faults, and the testing-resource expenditures spent during module testing.

Let t_0(> 0) be the release time of software (time length of software testing). Suppose that the software consists of n components. For each software component i(= 1, 2, . . . , n),the expected cumulative number of software faults by testing time t is given by the following exponential form [23]:

$$H_i(t) = a_i(1 - \exp[-r_i W_i(t)]), \qquad (1)$$

Where a_i is the expected number of initial fault contents before testing and r_i is the fault-detection rate per unit testing effort for the component i. The function $W_i(t)$ is called the testing-effort function and is measured by the person hour or the person day consuming during the time interval (0, *t*]. If the mean value function of an NHPP is given by $H_i(t)$, then the resulting stochastic model is quite similar but somewhat different from the so-called testing-effort dependent software reliability model [23], because the expected cumulative number of a whole software system is represented by

$$\begin{aligned} H(t) &= \sum_{i=1}^{n} H_i(t) \\ &\neq \left(\sum_{i=1}^{n} a_i\right)\left(1 - \exp\left[-\sum_{i=1}^{n} r_i\, W_i(t)\right]\right). \end{aligned} \qquad (2)$$

This slightly different assumption focuses on the additive component structure in the architecture-based modeling. The expected residual number of software faults for component i at time t is given by

$$a_i - H_i(t) = a_i \exp[-r_i W_i(t)] \neq a_i \exp\left[-\sum_{i=1}^{n} r_i\, W_i(t)\right].$$

Define the cost structure for software testing:

c_1: fixing cost of a software fault detected in testing phase

c_2: fixing cost of a software fault detected in operational phase

c_3: testing cost per software testing effort,

Where, without any loss of generality, $c_2 > c_1$, i.e., the debugging cost in operational phase is greater than that in testing phase. Then, the expected total software cost for component i is given by

$$C_i(t_0) = c_1 H_i(t_0) + c_2\{H_i(T) - H_i(t_0)\} + c_3 W_i(t_0), \qquad (3)$$

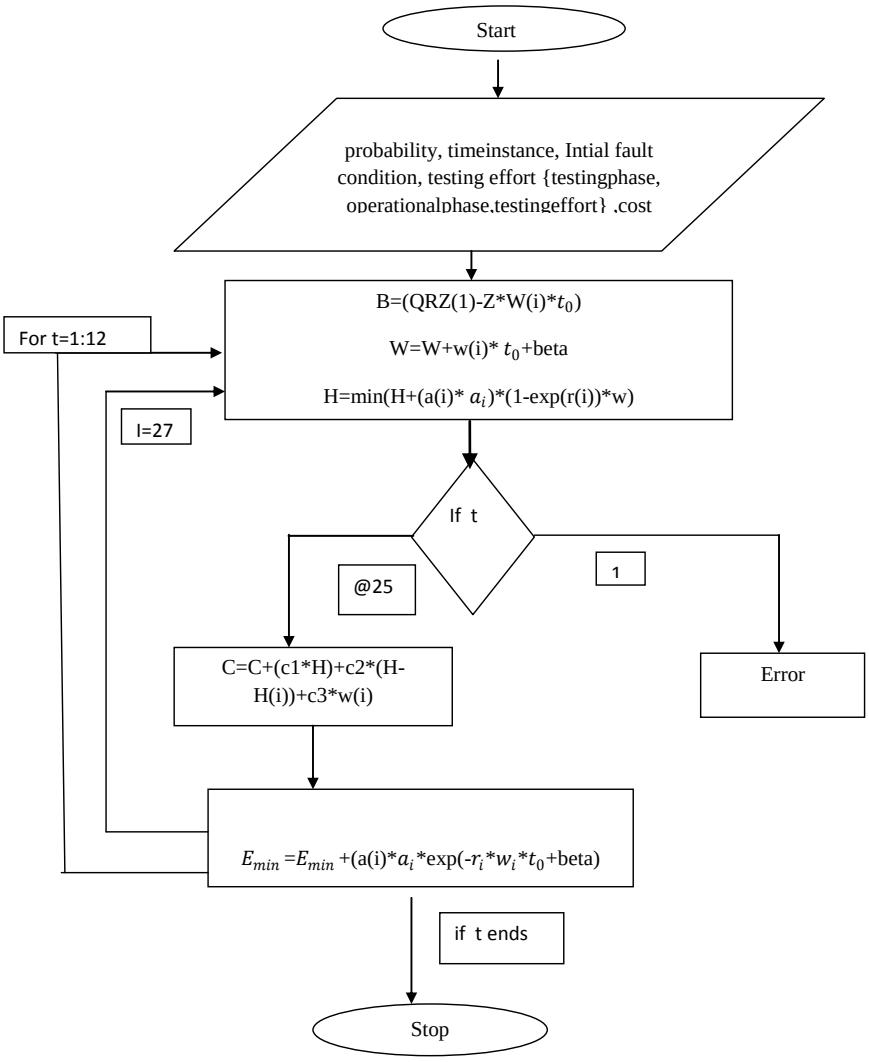

Figure1:flowchart

Where T is the software lifetime in the operational phase. So, the expected total software cost for a whole software system is given by $C(t_0) = \sum_{i=1}^{n} C_i(t_0)$ for a given release schedule t_0 and possesses an additive property for respective component development costs. The software testing-effort function $W_i(t_0)$ is a function of the testing timet_0. To simplify the discussion, we employ the simple linear model:

$$W_i(t_0) = w_i(t_0) + \beta_i \,, \qquad (4)$$

Where w_i is the testing effort per unit testing time and β_i is a fixed effort for component i. It is common to see that the cumulative number of software failures caused by software faults is estimated by the operational profile in the design phase. Let ϕ_i be the frequency to use the component i in the operational phase with $\sum_{i=1}^{n} \varphi = 1$. Similar to Helander et al. [6], define

$$\varphi_i = \sum_{i=1}^{n} \mu_{ij} p_j \,, \qquad (5)$$

Where μ_{ij} is the ratio of the software operation j(= 1, 2, . . . , m) processed by component i, p_j is the probability that the software operation j is executed, and

$$\sum_{j=1}^{m} p_j = 1 \,, \sum_{i=1}^{n} \mu_{ij} = 1 \qquad (6)$$

From the above preliminary, the net operational time for each component i is given by $\varphi_i(T-t_0)$. Recently fujiietal [5] proposed an accelerated life testing model to describe the operational prole by using the environmental factor. Then the mean value function for component i is described by

$$H_i(t) = \begin{cases} H_t(t_0) & (0 \le t < t_0) \\ H_0(t_0) & (t_0 \le t < T) \end{cases} \qquad (7)$$

Where $H_t(t_0)$ is given in Eq.(1), and

$$H_0(t) = H_t(t_0 + \varphi_i(t - t_0)) \qquad (8)$$

For an arbitrary i. If $\varphi_i \ge 1$, then the operational circumstance is accelerated and the failure occurrence time in the operational phase is shorter than the fault-detection time in the system test, so the fault detection rate in the operational circumstance is greater than that in the system test. Otherwise, i.e., if $\varphi_i < 1$, the system testing circumstance is equal to or severer than the operational circumstance on debugging. After a few algebras, we have

$$H_i(T) - H_i(t_0) = a_i\,(1 - \exp[-r_i(w_i\{t_0 + \varphi_i\,(T - t_0)\} + \beta_i)]) - a_i(1 - \exp[-r_i\,w_i + \beta_i]), \qquad (9)$$

so, the expected total software cost is represented by

$$C(t_0) = c_1 \sum_{i=1}^{n} a_i(1 - \exp[-r_i(w_i t_0 + \beta_i)]) + c_2 \sum_{i=1}^{n} a_i\{\exp[-r_i(w_i t_0 + \beta_i)] - \exp[-r_i\,(w_i\{t_0 + \varphi_i\,(T - t_0)\} + \beta_i)]\} + c_3\,(w_i\,t_0) + \beta_i). \qquad (10)$$

For a given t0, we consider a testing-effort allocation problem. Regarding the expected total software cost in Eq.(10) as a function of the testing effort per unit testing time, w_i, i.e. replacing by $C(t_0) = C(w_i)$, i = 1, 2, . . . , n, with $\sum_{i=1}^{n} \varphi_i = 1$, we formulate three planning problems for optimization. Let $\overline{Q}$, $\overline{Z}$ and $\overline{R}$ be the upper levels of the cumulative testing efforts, expected total software cost, and the residual number of software faults, respectively. Then, we formulate the following three optimization problems with constraints:

Problem 1 (Effort minimization):

$$\min_{w_i; i = 1,2,3 \ldots, n} \quad \sum_{i=1}^{n} (w_i\,t_0 + \beta_i) \qquad (11)$$

$$s.t. \quad \sum_{i=1}^{n} C(w_i) \le \overline{Z} \qquad (12)$$

$$\sum_{i=1}^{n} a_i\ \exp[-r_i\{w_i t_0 + \beta_i\}] \le \overline{R} \qquad (13)$$

$$w_i \ge 0 \text{ for } i = 1, \ldots . n.$$

Problem 2 (Cost minimization):

$$\min_{w_i; i = 1,2,3 \ldots, n} \quad \sum_{i=1}^{n} C(w_i) \qquad (14)$$

$$s.t. \quad \sum_{i=1}^{n} (w_i\,t_0 + \beta_i) \le \overline{Q} \qquad (15)$$

$$\sum_{i=1}^{n} a_i\ \exp[-r_i\{w_i t_0 + \beta_i\}] \le \overline{R} \qquad (16)$$

$$w_i \geq 0 \text{ for } i = 1,\ldots.n.$$

Problem 3 (Fault minimization):
min

$$w_i; i = 1,2,3 \ldots., n \qquad \sum_{i=1}^{n} a_i \exp[-r_i\{w_i t_0 + \beta_i\}] \qquad (17)$$

$$s.t. \qquad \sum_{i=1}^{n} (w_i t_0 + \beta_i) \leq \overline{Q} \qquad (18)$$

$$\sum_{i=1}^{n} C(w_i) \leq \overline{Z} \qquad (19)$$

$w_i \geq 0$ for i =1,.….n.

Problem 4 (sensitive analysis):
The TE reaches its maximum value at time

$$t_{max} = \frac{\ln\frac{A}{K}}{aK} \qquad (20)$$

The conditional reliability function after the last failure occurs at time t is obtained by [1], [2]

$$R_{cond}(t) \equiv R_{cond}(t + \Delta t|t) = \exp\left[-\left(m(t + \Delta t) - m(t)\right)\right]. \qquad (21)$$

Taking the logarithm on both sides of the above equation, we obtain

$$\ln R_{cond}(t) = -\left(m(t + \Delta t) - m(t)\right). \qquad (22)$$

Here we will define another measure of reliability, i.e., the ratio of the cumulative number of detected faults at time t to the expected number of initial faults.

$$R(t) \equiv \frac{m(t)}{a}.$$

Note that R(t)is an increasing function in t. Using R(t), we can obtain the required testing time needed to reach the reliability objectiveR_0, or decide whether the reliability objective can be satisfied at a specified time. If we know that the value of R(t) has achieved an acceptable level, then we can determine the right time to release this software.

Algorithm:
Step1: set l=0
Step2: Calculate

$$X_i = \frac{l}{r_i}[\ln(\frac{v_i a_i r_i}{Z} \exp[-r_i C_i] \sum_{i=1}^{N-l} \frac{l}{r_i})] \; i = 1,2,3 \ldots. N - l.$$

Step3: Rearrange the index i such that $X_1^* \geq X_2^* \geq \cdots \geq X_{N-l}^*$.

Step4: if $X_{N-l}^* \geq 0$ then stop.

Else update X_{N-l}^* = 0; l=l+1.

END If.
Step5: Go to step 2.

The optimal solution has the form

$$X_i^* = \frac{l}{r_i}[\ln(\frac{v_i a_i r_i}{Z} \exp[-r_i C_i] \sum_{i=1}^{N-l} \frac{l}{r_i})] \; i = 1,2,3 \ldots. N - l.$$

Algorithm always converges in at worst,N-1 steps.

IV. results

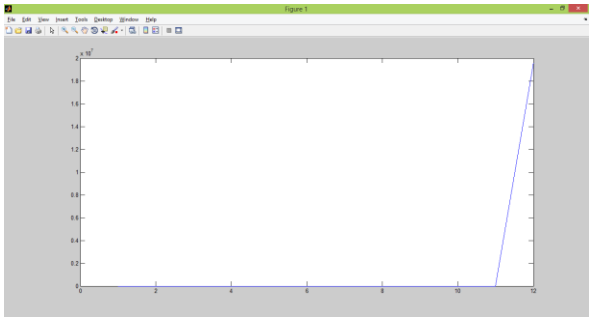

Fig2: Depends on effort for problem1

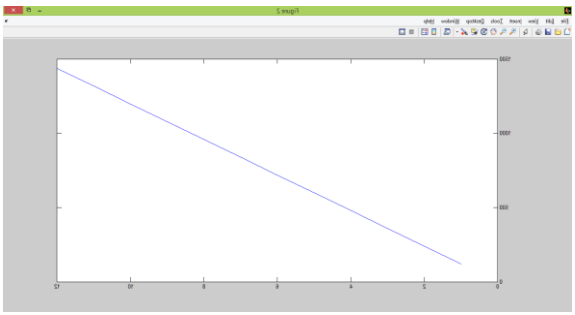

Fig 3: Depends on faults for problem1

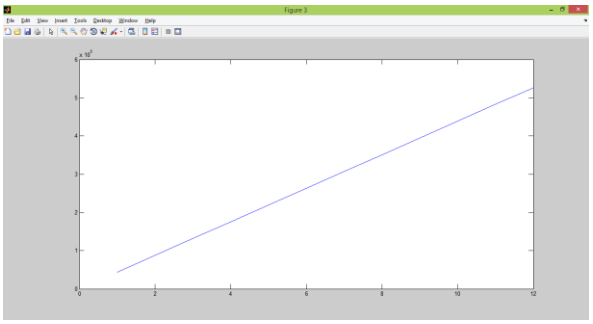

Fig4: Depends on cost for problem1

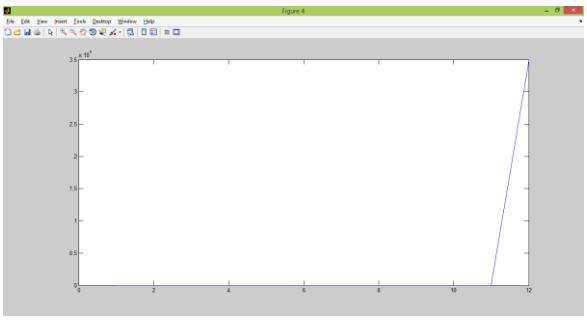

Fig5: Depends on sensitive of cost for problem1

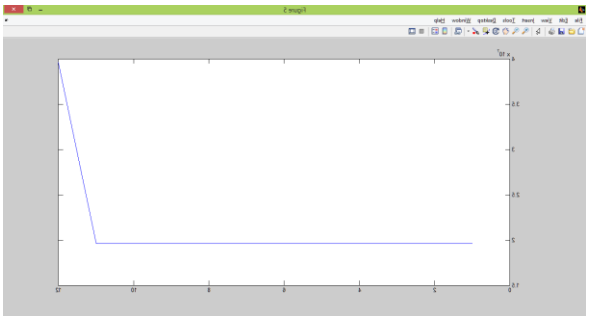

Fig6: Depends on effort for problem2

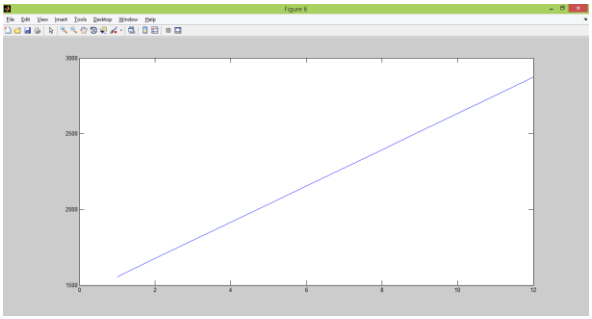

Fig7: Depends on faults for problem2

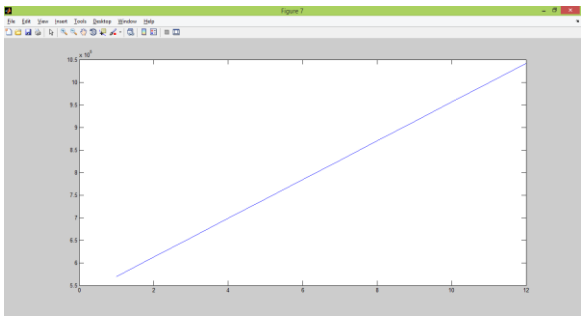

Fig8: Depends on cost for problem2

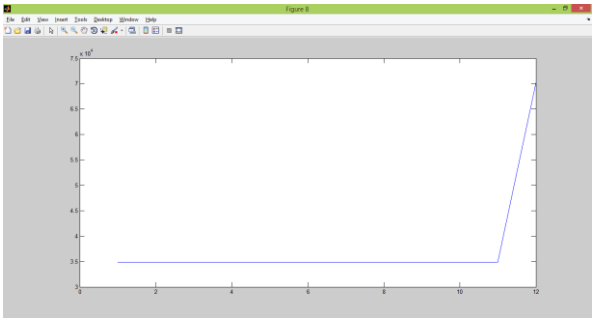

Fig9: Depends on sensitive of cost for problem2

V. Conclusion

In this paper we developed four development planning models which constraints, based on expected total cost, cumulative software testing resources, expected software faults and sensitive of cost. These models have been based on actual software development project and the optimal testing resource allocation policies. Since both software reliability and software cost depend on the operational profile. In the future, in our approach resources allocated based on point of time. Therefore, there is a problem of dynamic allocation of testing resource. To avoid dynamic allocation problem globalization technique based of software computing can be used. The globalization software computing techniques are genetic algorithm.

References

[1]. R. S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, 6th Edition, 2005.
[2]. M. R. Lyu, Handbook of Software Reliability Engineering, McGraw Hill, 1996.
[3]. J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction and Application, McGraw Hill, 1987.
[4]. M. Xie, Software Reliability Modeling, World Scientific Publishing Company, 1991.
[5]. AL.Goel, and K. Okumoto, "Time dependent error detection rate model for software reliability and other performance measures,‖ IEEE Transaction on Reliability. vol. 28 no. 3, pp. 206-211,1979.
[6]. C. Y. Huang, M. R. Lyu, and S. Y. Kuo, ―A Unified Scheme of Some Non-Homogenous Poisson Process Models for Software Reliability Estimation,‖ IEEE Transaction on Software Engineering, Vol. 29, No. 3, pp. 261-269, March 2003.
[7]. H. Ohetera, and S.Yamada, "Optimal allocation and control problems for software testing resources," IEEE Transaction on Reliability, vol. 39 no. 2, pp. 171-176, 1990.
[8]. S. Yamada, T. Ichimori, and M.Nishiwaki, "Optimal allocation policies for testing-resource based on a software reliability growth model," Mathematical and Computer Modelling, vol. 22, pp. 295-301, 1995.
[9]. R. Huo, S. Kuo, Y.Chang, "Needed resources for software module test, using the hyper-geometric software reliability growth model, " IEEE Transaction on Reliability., vol. 45 no. 4, pp. 541-549, 1996.
[10]. P.K. Kapur, A.K. Bardhan, and V.S.S. Yadavalli, "On allocation of resources during testing phase of a modular software," Int. Journal Syst. Sci, vol.38, pp. 493-499, 2007.

[11]. P.K.Kapur, P.C. Jha, and AK.Bardhan, "Dynamic programming approach to testing resource allocation problem for modular software,‖ Ratio Mathematica, Journal of Applied Mathematics, vol. 14, pp. 27-40, 2003.
[12]. P.K.Kapur, P.C.Jha, AK.Bardhan, "Optimal allocation of testing resource for a modular software, " Asia Pacific Journal of Operational Research, vol. 21 no. 3,pp. 333-354,2004.
[13]. M.Obha, "Software reliability analysis models, "IBM Journal of Research and Development, vol. 28, pp.428-443, 1984.
[14]. P.K. Kapur, R.B. Garg, and S. Kumar, Contributions to Hardware and Software Reliability, World Scientific: Singapore 1999.
[15]. M. Khan, N. Ahmad, and L.Rafi, "Optimal Testing Resource Allocation for Modular Software Based on a Software Reliability Growth Model: A Dynamic Programming Approach, " Proceedings of the International Conference on Computer Science and Software Engineering, 2008.
[16]. C. Y. Huang, J. H. Lo, S. Y. Kuo and M. R, ―Optimal Allocation of Testing-Resource Considering Cost, Reliability, and Testing-Effort,‖ Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on 3-5 March 2004, pp. 103-112.
[17]. Chin-Yu Huang, and Michael R. Lyu, Optimal Testing Resource Allocation, and Sensitivity Analysis in Software Development, ―IEEE Transactions on Reliability, VOL. 54, NO. 4, DECEMBER 2005.
[18]. [18] N. Raj Kiran and V. Ravi, ―Software reliability prediction by soft computing techniques,‖ The Journal of Systems and Software 81 (2008) 576–583.
[19]. Xue Guoxing, " Research of Software Reliability Based on Genetic Algorithm, " IEEE International Conference on Signal Processing Systems 2009.
[20]. Kapur, P K, Anu G Aggarwal and Gurjeet Kaur Optimal Testing Resource Allocation for Modular Software Considering Cost, Testing Effort and Reliability using Genetic Algorithm. International Journal of Reliability, Quality and Safety Engineering. 16(6): 495-508. 2010.
[21]. Zai Wang,, Ke Tangand Xin Yao, " Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems, " IEEE Transaction on Reliability, VOL. 59, NO. 3, SEPTEMBER 2010.
[22]. Anu G. Aggarwal, P. K. Kapur, Gurjeet Kaur and Ravi Kumar, "Genetic Algorithm Based Optimal Testing Effort Allocation Problem for Modular Software, "BVICAM‘s International Journal of Information Technology, Proceedings of the 4th National Conference; INDIACom-2010.
[23]. P.K. Kapur, Hoang Pham, Udayan Chanda and Vijay Kumar (2012): Optimal allocation of testing effort during testing and debugging phases: a control theoretic approach, International Journal of Systems Science, DOI:10.1080/00207721.2012.669861.