A Genetic Algorithm Approach for Optimal Allocation of Software Testing Effort

Prashant Johri
School of Computing Science & Engineering
Galgotias University
Greater Noida, India

Md. Nasar
School of Computing Science & Engineering
Galgotias University
Greater Noida, India

Udayan Chanda
School of business
Galgotias University
Greater Noida, India

ABSTRACT
Allocation of limited testing efforts to a software development project is a complex task for software managers. The challenges become difficult when the nature of the development process is considered in the dynamic environment. Numerous software reliability growth models have been proposed in last one decade to minimize the whole testing effort expenditures, but generally under static assumption. The main purpose of this article is to distribute total testing resource optimally under dynamic condition. An elaborate optimization policy is proposed using genetic algorithm and numerical example is also demonstrated. Genetic Algorithms (GAs) works with a set of individuals, representing probable solutions of the task. The selection theory is applied by using a criterion, giving an evaluation for the individual with respect to the desired solution. This article also studies the optimal resource allocation problems for different conditions by investigative the activities of the model parameters.

General Terms
Evolutionary Algorithm

Keywords
Genetic Algorithm, testing effort allocation, Software reliability, SRGM

Nomenclature
a is the initial fault content in the software.

b is the fault detection rate.

f(t) is the number of fault removed at time 't'.

m(t) is the cumulative number of fault detected till time 't' due to the testing effort w1(t).

T the planning period.

C1(m(t),w2(t)) Cost per unit at time 't' for cumulative faults removed m(t) and debugging effort w2(t).

C_2 is the cost of testing per unit testing efforts.

W is the total resources utilized during the SDLC at any point of time 't'.

$W_1(t)$ current effort expenditure due to testing at time 't'

$W_2(t)$ current effort expenditure due to debugging the fault at time 't'

1. INTRODUCTION
Software development phase is a difficult but major component of any Software Development Life Cycle (SDLC). The challenge will become stiffer when the development process is considered in the dynamic environment. To reduce uncertainty in the process often, companies set up different project management tools to coordinate with the other components of the project. Software will be released to the users at the end of the testing phase of SDLC. With superior development and testing efforts, superior quality software can be guaranteed. But this will be time consuming and is undesirable in the prevalent competitive market conditions. Allocation of financial efforts to a software development project during the testing phase in the dynamic environment will be vital decision that a software manager has to make. During testing resources such as computer time and manpower is consumed. The fault detection and removal process will depend upon the nature and amount of resources used. Many software reliability growth models (SRGMs) is proposed in the last decade to discuss the minimization issue of the testing effort expenditures Chatterjee, Misra and Alam [4], Kapur and Garg [17]. Often these models are based on the assumptions that the testing effort consumption and testing time follows Rayleigh and exponential distribution. The time dependent activities of the testing effort has been studied by many authors Basili and Zelkowitz [1], Kapur and Garg [17], Huang, Kuo, and Chen [9] and Yamada, Hishitani, and Osaki [27].

Earlier studies explored that exponential curve can be used if the testing resources are regularly consumed with respect to the testing time and Rayleigh curve otherwise. Logistic and Weibull-type functions were also used to describe the testing effort. Another school of thought assumes that the resource consumption can be expressed as an explicit function of the number of faults removed and calendar time Musa, Iannino, and Okumoto [20]. More recently, Tamura and Yamada [25] have proposed an SRGM based on stochastic differential equations in order to consider the active position of the open source project assuming that the software failure intensity depends on the time, and the software fault exposure phenomena on the bug-tracking system keep an unbalanced state. As discussed, over the three decades many SRGMs have been proposed to minimize the total expenditures, but mostly under static theory. Here in this article we have tried to explore an optimal resource allocation plan of software during the testing phase under dynamic situation. This paper also studies the optimal resource allocation problems for diverse conditions by examining the activities of the model parameters and also proposes the related release policy.

In this paper we discuss an algorithm based on Genetic Algorithm (GA) to solve this type of problem. Optimal control theory is used to formulate the problem and to analyze the model properties. The proposed approach is helpful to solve the dynamic nature of the problem which is difficult to solve by ordinary optimization technique.

The rest of this paper is organized as follows. Section 2 gives the background of this paper. Section 3 describes the Model formulation and its solution. Section 4 takes experiments with deferential evolution. Section 5 concludes the paper.

2. Research background

Software reliability growth models (SRGM) provide measures to predict future failure behavior from known or assumed characteristics of the software. Numerous SRGMs have been proposed in software reliability literature under a set of assumptions and testing environment. The proposed SRGM in this paper takes into account the time dependent variation in the testing effort. The testing efforts that govern the pace of testing for almost all the software projects are [20].

(a) Manpower which contains

- ✓ Failure detection professionals.
- ✓ Failure rectification professionals.

(b) Computer time.

The vital role of manpower engaged in software testing is to run test cases and compare the test results with desired specifications. On a failure, the fault causing it is identified and then removed by failure rectification personnel. The computer facilities represent the computer time, which is necessary for failure detection and rectification. The influence of software testing effort has been included in several SRGMs Putnam [30]; Pillai and Nair [23]; Kapur, Garg, and Kumar [18]; Xie [26]; Ichimori, Yamada, and Nishiwaki [11]; Kapur and Bardhan [16]; Huang et al [9]; Kapur et al. [19], Myers [21] planned that software will be constructed and tested individually in a chronological step. Yamada et al. [27], Hou, Kuo, and Chang [7], Pham and Zhang [22] have recommended that system level software testing occurred only after the system was completely developed. Recently, Blackburn, Scudder, and Van Wassenhove [2], however, recommended that software development and system debugging and testing should be viewed as simultaneous activities. Kapur and Bardhan [16] investigated the association between the number of faults deleted with respect to time and/or testing effort. The authors intend that during the testing phase of a software development life cycle, errors are removed in two phases: initial a failure occurs and then the fault causing that failure is corrected; therefore the testing effort should be spent on two separate processes; failure detection and failure rectification. In their paper, the authors developed an SRGM incorporating time delay not only between the two phases but also through the segregation of resources between them and proposed two alternate methods for controlling the testing effort for achieving the preferred reliability or error detection level. Chiang and Mookerjee [5] discussed a development process in which system integration occurs when the number of errors in the system will achieve a certain threshold. Chang [3] proposed the sequential software release policy based on a state space model, considering a Gamma–Gamma-type invariant conditional distribution to define the state space model.

Jain and Priya [13], Zheng, S. [28] investigated software release policies to minimize development cost while satisfying a reliability objective in dynamic environment. Jain and Gupta [14] have proposed optimal released policy for module based software.

3. Model formulation and solution

We initiate our study by stating a common model with a very few assumptions. We are limiting our analysis to the case of a firm that controls its resources for testing and debugging under limited planning horizon. We are also consistent with the plan that the latent faults in the software system are discovered and removed during the testing period, and the number of faults remaining in the software system gradually decreases as the testing on progresses. Therefore, it is reasonable to presume the following differential equation:

$$\mathrm{f(t)} = \frac{d}{dt} m(t) = b w_1(t)[a - m(t)] \tag{1}$$

Now assume the software firm needs to minimize the entire expenditure over the limited planning horizon T. Then the objective function for the firm can be given by

$$\text{Min} \int_0^T [c_1(t) f(t) + c_2 w_1(t)] dt$$

Subject to

$$\mathrm{f(t)} = \frac{d}{dt} m(t) = f(b, w_1(t)) \tag{2}$$

$$\frac{m(t)}{a} \geq m_r \geq m(T) \geq m_a (= a m_r)$$

Where

$m(0)=0$ and $w_1(t)+w_2(t)=W$

$(w_1(t); w_2(t))\geq 0$ and $c_1(t) = c_1(m(t), w_2(t))$.

Here, in the above optimal problem $0<m_r<1$, we have considered the preferred reliability level to be at least ma (where ma is unique). The planning period is [0, T] and the assumption $m(T)\geq m_a$ means that the firm intends to reaching at least the level ma at the end of the planning period. The planning problem is to find the allocation of resources that minimizes the total cost.

To resolve the above optimization problem, suppose $w_1^*(t)$ be an admissible control vector which reassigns (m0,t0) to a target (m(T),T), where final state m(T) is given but the final time T is not given t0 and m0 are the initial time and state and both are fixed, i.e. (t0,m(0)=(0,m0)). Suppose that m*(t) is corresponding to $w_1^*(t)$, then by Pontryagin Maximum principle, in order for $w_1^*(t)$ to be optimal, it is compulsory that there exists a non-zero, continuous vector function $\lambda^*(t)$ and a constant scalar $\lambda 0$ such that (Sethi and Thompson [24]):

$W_1^*(t)$ maximizes

$$[-c_1(t) + \lambda(t)] f(t) - c_2 w_1(t) \forall w_1(t) \geq 0 \tag{3}$$

We can interpret $\lambda(t)$ as the minor value of faults at time 't', this value must be negative because increasing the number of faults will increase the debugging expenditure. The physical explanation of the Hamiltonian H will be given as follows: $\lambda(t)$ stands for future cost incurred as one more fault is introduced in the system (at time t). Thus the Hamiltonian is the sum of testing cost c2w1(t), current cost c1(t)f(t) and future cost $\lambda(t)$f(t). In short, H represents the instantaneous total cost of the firm at time t.

The following necessary condition that hold for an optimal solution:

$$\frac{\partial H}{\partial w_1} = 0 \Rightarrow -(c_1 - \lambda) f w_1 - c_1 w_1 f - c_2 = 0$$

Other optimality condition is

$$\frac{\partial^2 H}{\partial w_1^2} \leq 0 \Rightarrow -(c_1 - \lambda) f_{w1w1} - c_{1w1w1} f - 2c_{1w1} f_{w1} \leq 0$$

Where $c_{1w1} = \frac{\partial c_1}{\partial w_1}$ and $c_{1w1w1} = \frac{\partial^2 c_1}{\partial w_1^2}$

From above optimality conditions, we will get the following results:

$$\Rightarrow w_1^*(t) = \frac{(\lambda(t) - c_1(t))b(a - m(t)) - c_2}{c_{1w1(t)}(t)b(a - m(t))} \quad (4)$$

And,

$$w_2^*(t) = w - w_1^*(t) \quad (5)$$

The above optimization problem is solved by a powerful computerized heuristic search and optimization method, ie. genetic algorithm (GA) Goldberg [29], David [10] and Lin [12]. Genetic Algorithms are direct, parallel, stochastic method for global search and optimization, which imitates the evolution of the living beings, described by Charles Darwin. GA is part of the group of Evolutionary Algorithms (EA)[15].

The genetic algorithms use the three main principle of the natural evolution: reproduction, natural selection and diversity of the species, maintained by the differences of each generation with the previous.

Genetic Algorithms works with a set of individuals, representing possible optimal solutions of the task. The selection principle is applied by using a principle, giving an evaluation for the individual with respect to the preferred solution. The superlative suited individuals form the next generation.

PROCEDURE OF GA

Step I: Start

Step II: Generate population of chromosomes randomly

Step III: calculate the fitness of every chromosome in the population

Step IV: Generate a new population by repeating following steps until the new population is complete

[Selection] Select two parent chromosomes from the set of population according to their fitness

[Crossover] With a crossover probability, crossover the parents to form new offspring (children). If crossover is not performed, offspring will be the exact copy of parents.

[Mutation] With a mutation probability, mutate offspring at each locus (position in chromosome)

[Accepting] set new offspring in the new population

[Replace] Use new generated population for further sum of the algorithm.

[Test] If end condition is satisfied, stop and return the best solution in the current population

[Loop] Go to step III for fitness evaluation

Chromosome representation Genetic Algorithm starts with the initial population of solutions represented as chromosomes. A chromosome comprises genes where each gene represents a specific feature of the solution.

Initial population For a given total testing effort W, GA generates the initial population randomly. It will initialize the values within the limits of each variable randomly.

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 gives a best solution. In the effort allocation problem, the fitness function is the objective of testing effort optimization problem.

However to solve the problem dynamically the initial value of $\lambda(0)$ and $f(0)$ must be given first then from objective function we will find the optimal value of effort and fault detected at time t.

Table 1: Parameters of the GA for testing effort allocation

Parameter	**Value**
Population Size	30
Number of Generation	70
Selection Mode	Tournament
Crossover Probability	0.8
Mutation Probability	0.2

Based on above parameters problem is solved using MATLAB version 7.4.0 [8], [6]

4. Numerical analysis

Assume that a=100, b=0.2,
$\lambda(0)$=40, f(0)=2,
c_0=1000, c_2=5000,

The parameter values of 'a' and 'b' can be estimated for any given dataset outside the normative model by considering Equation (1). Though these parameters are mutually correlated with $w_1(t)$ and $w_2(t)$, due to the interdependence between the two, one effort can be expressed in the form of other. In this analysis, the objective is to check the significance of allocation of the testing effort (w_1), we use differential evolution to allocate testing effort.

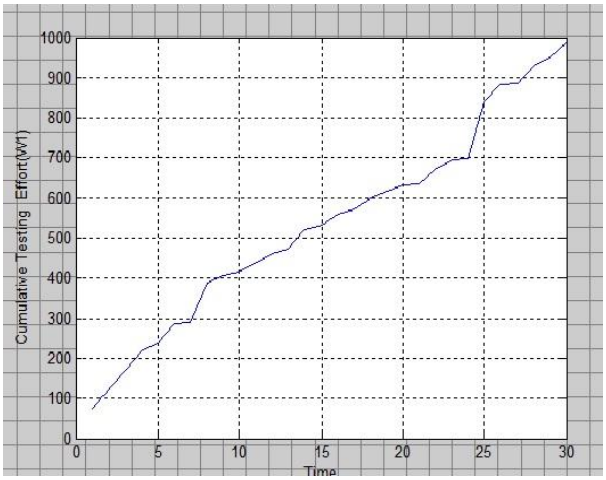

Figure 1. Allocation of testing effort (w_1) versus time.

Below table shows allocation of effort with respect to time.

Time	Cumulative Testing Effort(W_1)
1	72.929
2	124.41
3	169.8
4	221.27
5	239.1
6	287.13
7	290.17
8	388.08
9	406.81
10	415.05
11	437.57
12	459.73
13	472.87
14	521.33
15	532.69
16	559.25
17	572.17
18	600.52
19	615.92
20	632.66
21	635.83
22	670.29
23	696.49
24	698.52
25	838.43
26	884.4
27	887.97
28	930.18
29	950.45
30	988.67

In this study we tried to explore the conditions under which a software product can be transferred from its development phase to operational use. The literary meaning of the term 'to release' is to specify that the software has met a defined quality level during testing and is ready for mass distribution.

An ideal software release policy addresses the following concern of a release manager:

- Avoid excessive time to market due to over testing.
- It will manage all product related activities, from concept to retirement
- It will help to optimize product performance and plan upcoming investments
- Analyze, manage and improve the reliability of software products.
- Balance customer needs for competitive price, timely delivery and a reliable product.
- Determine whether the software is good enough to release to customer, minimizing the risks of releasing software with serious problem.

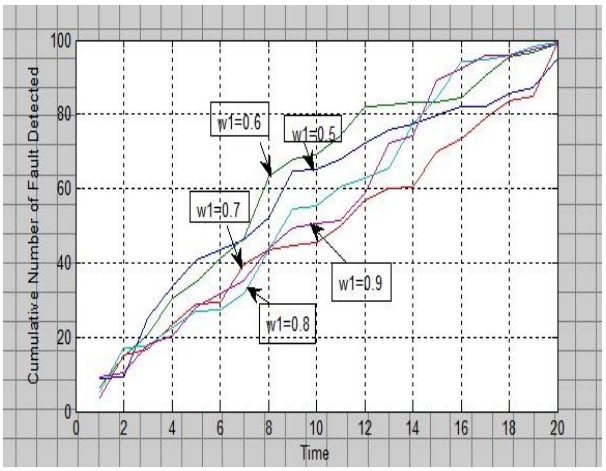

Figure 2 Cumulative number of faults removed versus time.

Table 1: Parameters of the GA for cumulative fault removing

Parameter	**Value**
Population Size	20
Number of Generation	80
Selection Mode	Tournament
Crossover Probability	0.8
Mutation Probability	0.2

In this analysis the objective is to check when to introduce software in market and also to check the significance of cumulative fault removal till time t, hence its value has been optimize using genetic algorithm, GA create initial population for 'a' (total fault content in software). In the analysis it has been seen that as the value of testing effort gradually increased keeping the other parameters constant. Above figure shows the cumulative number of fault removed versus time. Below figure shows the future cost of fault removal.

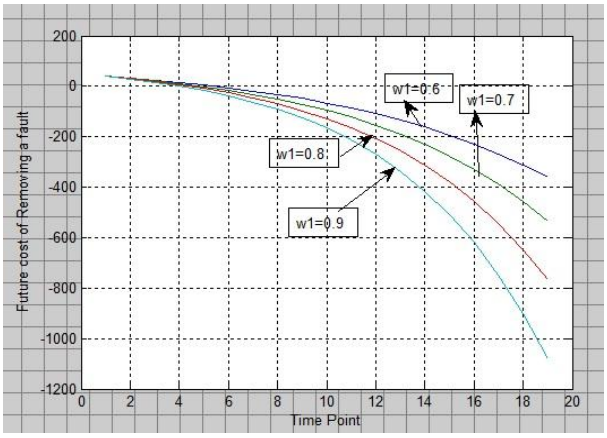

Figure 3. Shadow cost versus time

5. Conclusion

In this paper we propose an alternate foundation for optimal allocation of testing resources using genetic algorithm and also optimize cumulative fault removing with respect to time. We recommend that software testing and debugging should be viewed as simultaneous behavior. During the investigation, we have examined to allocate effort in dynamic environment using GA. This means that the developers and testers can devote their time and resources to complete their testing tasks based on well controlled expenditures and also we demonstrated when to release software in market after specified reliability level.

6. REFERENCES

[1] Basili, V.R. and Zelkowitz, M.V. (1979), 'Analyzing Medium Scale Software Development', in Proceedings of the 3rd International Conference on Software Engineering, pp. 116–123.

[2] Blackburn, J.D., Scudder, G.D., and Van Wassenhove, L.N. (2000), 'Concurrent Software Development', Communications of the ACM, 43, 200-214

[3] Chang, Y.-C. (2004), 'A Sequential Software Release Policy', Annals of the Institute of Statistical Mathematics, 56, 193–204.

[4] Chatterjee, S., Misra, R.B., and Alam, S.S. (1997), 'Joint Effect of Test Effort and Learning Factor on Software Reliability and Optimal Release Policy', International Journal of Systems Science, 28, 391–396.

[5] Chiang, I.R., and Mookerjee, V.S. (2004), 'A Fault Threshold Policy to Manage Software Development Projects', Information Systems Research, 15, 3–19.

[6] Painton, L., Campbell, J., 1995. Genetic algorithms in optimization of system reliability. IEEE Transaction on Reliability. 44 (2), 172–178.

[7] Hou, R.H., Kuo, S.Y., and Chang, Y.P. (1997), 'Optimal Release Times for Software Systems with Scheduled Delivery Time Based on the HGDM', IEEE Transactions of Computer, 46, 216–221.

[8] Global Optimization Toolbox User's Guide R2012 a The MathWorks, Inc.

[9] Huang, C.-Y., Kuo, S.-Y. and Chen, J.Y. (1997), 'Analysis of a Software Reliability Growth Model with Logistic Testing Effort Function', in Proceeding of 8th International Symposium on Software Reliability Engineering, pp. 378–388.

[10] David, L. Handbook of Genetic Algorithms. New York : Van Nostrand Reinhold. 1991

[11] Ichimori, T., Yamada, S. and Nishiwaki, M. (1993), 'Optimal Allocation Policies for Testing-resource Based on a Software Reliability Growth Model', in Proceedings of the Australia–Japan Workshop on Stochastic Models in Engineering, Technology and Management, pp. 182–189.

[12] Lin, C., Shen, S., Yeh, Y., & Ding, J. (2001). Dynamic Optimal Control Policy In Advertising Price and Quality. International Journal of Systems Science, 32, 2. Business Source Premier Database

[13] Jain, M. and Priya, K. (2002). Optimal policies for software testing time. Journal of Computer Society of India, 32, 25-30.

[14] Jain, M. and Gupta, R. (2011). Optimal Release Policy of Module-Based Software. Quality Technology and Quantitative Management Vol. 8, No. 2, pp. 147-165 Lin, C., Shen, S., Yeh, Y., & Ding, J. (2001).

[15] Kosters W.A., Kok J.N. and Floreen P., Fourier Analysis of Genetic Algorithms, Theoretical Computer Science, Elsevier, 229, 199, pp. 143-175.

[16] Kapur, P.K., and Bardhan, A.K. (2002), 'Testing Effort Control Through Software Reliability Growth Modelling', International Journal of Modelling and Simulation, 22, 90–96.

[17] Kapur, P.K., and Garg, R.B. (1990), 'Cost Reliability Optimum Release Policy for a Software System with Testing Effort', OPSEARCH, 27, 109–116.

[18] Kapur, P.K., Garg, R.B., and Kumar, S. (1999), Contributions to Hardware and Software Reliability, Singapore: World Scientific.

[19] Kapur, P.K., Gupta, A., Shatnawi, O.R., and Yadavalli, V.S.S. (2006), 'Testing Effort Control Using Flexible Software Reliability Growth Model with Change Point', International Journal of Performability Engineering, 2, 245–262.

[20] Musa, J.D., Iannino, A., and Okumoto, K. (1987), Software Reliability: Measurement, Prediction, Applications, New York: Mc Graw Hill.

[21] Myers, G.J. (1976), Software Reliability: Principles and Practices, New York: John Wiley & Sons.

[22] Pham, H., and Zhang., X. (1999), 'A Software Cost Model with Warranty and Risk Costs', IEEE Transactions on Computer, 48, 71–75.

[23] Pillai, K., and Nair, V.S.S. (1997), 'A Model for Software Development Effort and Cost Estimation', IEEE Transations on Software Engineering, 23, 485–497.

[24] Sethi, S.P., and Thompson, G.L. (2005), Optimal Control Theory – Applications to Management Science and Economics (2nd ed.), New York: Springer.

[25] Tamura, Y., and Yamada, S. (2009), 'Optimisation Analysis for Reliability Assessment Based on Stochastic Differential Equation Modelling for Open Source Software', International Journal of Systems Science, 40, 429–438.

[26] Xie, M. (1991), Software Reliability Modeling, Singapore: World Scientific.

[27] Yamada, S., Hishitani, J., and Osaki, S. (1993), 'Software Reliability Growth Model with Weibull Testing Effort: A Model and Application', IEEE Transactions on Reliability, R-42, 100–105.

[28] Zheng, S. (2002). Dynamic release policies for software systems with a reliability constraint. IIE Transactions, 34, 253-262.

[29] Goldberg, D. E., 1989, Genetic Algorithms: in Search Optimization and Machines Learning (New York: Addison-Wesley).

[30] Putnam, L. (1978), 'A General Empirical Solution to the Macro Software Sizing and Estimating Problem', IEEE Transactions on Software Engineering, SE-4, 345–361.