Optimal Software Release Time Policy using Goal Programming

Sanjeev Kumar
Research Scholar
Himalayan University

Sachin Gupta
Assistant Prof.
Vivekanand Institute of Professional Studies, New Delhi India

Dr. Deepak Tyagi
Prof.and Director
St. Anne Mary Education Society
New Delhi India

ABSTRACT
In this paper, the impact of software testing effort and the efficiency on the modeling of software reliability for optimizing the cost in case of release time policy has been discussed. Another important problem in the software development process is to determine when to stop testing and release the software. If testing is stopped too early, there may be too many defects in the software, resulting in too many failures during the operation and leading to significant losses. If too much time is spend on testing, there may be a high testing cost. Therefore, there must be a tradeoff between testing and releasing. The release time should be determined by the testing process, efforts and cost. The more defects have been detected and removed, the less time will be used for further testing. To eliminate this problem of releasing the software goal programming approach has been discussed**.**

Keywords
Software release time policy, goal programming, software testing efforts, software defect removal.

1. INTRODUCTION
With the wide spread use of computer, software is playing an important role in our life. In other words computers and computer-based systems have entered in every walk and talk of our lives. We have become heavily dependent on automated tools and intelligent systems for almost every activity. A mere delay in the operation of these systems can lead to huge financial losses; hence software reliability becomes a problem that cannot be ignored. Software reliability is consequently one of the most important features for the critical software system.

According to ANSI definition, software reliability is defined as the probability of failure free software operation for a specified period of time under a specified environment. In practice, it is very difficult for the project managers to measure software reliability and quality. In the seventies of last century, many models (namely called Software reliability growth models, SGRM) have been proposed to describe the software testing process. Among these models, GO (Goel and Okumoto)[4] model assumes that the defects detected up to time‘t’ follows a non-homogeneous Poisson process (NHPP), while Jelinski and Moranda[11] model assumes that the total numbers of defects at the start of testing is a known constant, and the failure rate at any time is proportional to the number of defects remaining at that moment.

It is the testing stage of the software development in which attempts are made to remove the most of the faults which are dormant in software. A successful test strategy begins by considering the requirement specification and continues by specifying test cases based on this requirement specification. Most of the earlier software reliability models assume the fault removal process to be perfect i.e. when an attempt is made to remove a faults are introduced. But this assumption is not realistic due to the complexity of the software system and incomplete understanding of the user’s requirements or the specifications by the team. The software testing team may not be able to fix the cause of the failure properly or they may not be introducing new faults during the removal. Therefore it is necessary to incorporate the effect of imperfect debugging into the software reliability growth modeling.

The problem of determining when we to stop testing are emerges. That is to say, if we stop testing too early, there may be too many defects in the software, which will result in too many failures during the operation, and lead to significant losses. On the other side, spending too much time in testing may result in a high testing cost and delay the introduction of the product into the market. Therefore, there is a tradeoff between software testing and releasing.

2. LITERATURE REVIEW
There are two commonly used approaches in the industry for software reliability modeling. The first approach utilizes software reliability growth models (SRGM) in later stages of software development cycle to estimate the software failure rate from the observed failures. As each failure occurrence initiates the removal of a fault, the number of failures that have been experienced by time t, denoted by M(t) , can be regarded as a reflected image of reliability growth. Each SRGM implicitly assumes a certain functional form of mean value function. M(t). Using the failure times or the times between failures collected during a testing project, the parameters of a SRGM like expected number of failures by the time t or failure intensity can be estimated. Software architecture greatly affects the reliability and availability of software systems. The second approach referred to as architecture based modeling, utilizes discrete state Markov modeling in the software reliability estimation (Taylor and Vander (2007))[30]. System failures can be avoided with better software architectures by raising exceptions when fault occurs and confining the failure to a particular module.

Table 1: The summary of the literature review

S.No.	Authors and study published in the year	Objective and the Discussion
1.	Hudson (1967) [9]	The software development was considered as birth and death process. The fault is considered as birth and fault correction as death. In his theory he proposed that the rate of detection of faults is proportional to the number of faults remaining and the rate of fault detection increases with time.
2.	Jelinki and Moranda (1972)[11]	Developed software reliability growth model which is based on the assumption that at the beginning of the testing , there are *a u* faults in the software program with *a u* being an unknown but fixed number.
3.	Musa et.al. (1987).[22]	The model prepared by Jelinki and Moranda is of binomial type as classified by Musa et.al. The Jelinki and Moranda model has poor predictive capabilities in many cases where as the model proposed by Musa is rich in clarity of modeling and has a better conceptual insight and predictive validity. In this model the execution time is viewed in two ways like (a) operating time of the software product and (b) Cumulative execution time that occurs during the test phase of development and the post delivery maintenance.
4.	Scheindewind (1972) [26]	Presented a software reliability model from an empirical viewpoint assuming fault detections per time interval as a non homogeneous Poisson process with an the exponential mean value function. He applied least square method or maximum likelihood estimation to the determination of the parameters of the process and also suggested that the time lag between failure detection and correction be determined from actual data and used to correct the time scale in forecast.
5.	Goel and Okumoto (1979)[4]	Proposed a model assuming expected number of initial software faults as N as compared to the fixed but unknown actual number of the initial software faults u_0 in the Jelinki and Moranda.
6.	Piwowarski et al.(1983) [25]	Proposed a model for block coverage in dependence of the number of test cases executed during functional testing, which can easily be extended to become a model of the number of failures experienced in terms of time.
7.	Yamada et al.(1986) [33]	Extended Goel-Okumoto model stating that the ratio between the expected number of software failure occurring in (t, t+Δt) with $\Delta t \rightarrow 0$ is proportional to the expected number of undetected faults.
8.	Littlewood et al. (1986)[18]	The Bayesian extension to the Jelinski and Moranda model was proposed by Littlewood in 1986 to improve the parameter estimation of the model and hence its predictive capability.
9.	Tohma et al. (1989)[31]	He gave a structural approach to the estimation of the number of residual software faults based on hyper- geometric distribution.
10.	Tang and Iyer (1992)[29]	They presented analysis and modeling of correlated failures in multicomputer systems.
11.	Lee et al. (1993)[17]	Presents measurement-based evaluation of operating system fault tolerance.
12.	Lyu(1996)[19]	He developed a model based on assumptions that the number of failures experienced by time t follows a Poisson distribution. The

		number of software failures that occurs in (t, t+Δt) with $\Delta t \rightarrow 0$ is proportional to the expected number of undetected faults. Whenever a failure occurs, faults that caused it is removed instantaneously and without introducing any new fault into the software. Since each fault is perfectly repaired after it has caused a failure, the number of inherent faults in the software at the beginning of the testing is equal to the number of failures that will have occurred after an infinite amount of testing.
13.	Gokhale et al. (1996)[5]	Introduced the enhanced non-homogeneous Poisson process (ENHPP) model as a unifying frame work for finite failure NHPP models with the assumptions that the N faults inherent in the software at the beginning of testing are uniformly distributed over the potential fault sites and at time t. Whenever a failure occurs, the fault that caused it is removed instantaneously and without introducing any new fault into the software.
14.	Krishnamurthy and Mathur (1997)[16]	They gave an approach for the estimation of reliability of a software system using reliability of its components.
15	Gokhale et al. (1998)[6]	Presented an analytical approach to architecture-based software reliability prediction.
16.	Smidts and Sova (1999)[27]	Considered an architecture-oriented modeling approach for software reliability estimation based on decomposition of requirements into software functions and attributes.
17.	Huang et al (2001)[8]	Proposed the incorporation of testing effort into the modeling process, which can be measured by the human power, the number of test cases, or the execution-time information.
18.	Chen et al. (2001)[3]	Included testing coverage into time-basis adjustment for more accurate software reliability measurement.
19	Cai and Lyu (2004)[2]	Carried out an empirical study on reliability and fault correlation models for diverse software systems.
20.	Mohanta (2005)[21]	Proposed a fuzzy Markov model for determination of fuzzy state probabilities of generating units.
21.	Teng et al. (2006)[29]	Studied reliability modeling of hardware and software interactions.
22.	Guillermo and Manic, (2006)[7]	Have carried out fuzzy perform ability analysis of disk arrays.
23.	Lyu (2007)[20]	Proposed a recent trend in software architecture is that as information engineering is becoming the central focus for today's businesses, service-oriented systems and the associated software engineering will be the standards for business development.
24.	Suri (2009),[28]	Has developed a simulator for risk assessment of software project based on performance measurement.
25.	Yadav and khan (2009) and Khaled (2009)[32]	Have given a critical review of software reliability models. In the analysis of certain software systems, few parameters such as software failure and repair rates cannot be exactly estimated.

In the present scenario, it is difficult to analyze software reliability and availability due to such uncertain parameters. The models generally assume that once a fault is discovered it is removed immediately that is, software have instantaneous repair time. The reality is that applications executing in the field can take significant amount of time may be days or weeks to get a fault removed. The second problem, which is generally faced, is the quality of the failure data. For example repeat failures generally occur due to the fact that faults are not removed instantaneously. Another problem is that operational profile testing is generally ignored i.e. it is assumed that the software is going to be tested in the same manner that it is used in the field, which is not true in practice. Thus, there is a need of further improvements in the existing models so that

reliability and availability can be computed more efficiently and accurately.

3. OPTIMAL SOFTWARE RELEASE POLICY

Software reliability growth models can captures the quantitative aspects of the software testing process, and can be used to provide a reasonable software release time. During the software testing phase, developers can use the SRGM to determine when to stop testing. If the reliability goal is achieved, the software product is ready for release.

To shift the software from the testing phase to the operational phase, theoretical determination of the release time of software is very important problem in terms of cost. In recent years, the problem of optimal software release time has been analyzed and discussed by many authors.
The release time problem is to decide upon the optimal testing termination time T. In the present problem we consider optimization of testing under budgetary constraint. The mathematical formulation of the problem is given as,

$$Maximize: R(x \mid T)$$

$$Subject\ to: C(T) \leq Cb,$$
$$T \geq 0$$

Where
$R(x|T) = expected\ reliability\ of\ software\ at\ time\ T,$

$C(T) =$
$cost\ incurred\ during\ software\ testing\ upto\ time\ T,$

$Cb = \ total\ budget\ assigned\ for\ software\ testing.$

The above problem can be extended in terms of getting desired reliability with the minimum cost.

Maximize R(x|T)
Minimize C(T)

Subject to
C(T) <= Cb
R(x|T)>= R0
T>=0, R0>=0
We proposed a goal programming model to deal with the problem of error free software with optimal time of release.
There exist various variables (factors) that affect the software development. The present model is base on the following factors.

Number of functional point: The number of functional point can be considered as phases in software development. As the number of functional point increases, the complexity of the software deployment also increases. Let there are four phases in terms of functional point Requirement analysis, Design, Integration and Testing.

Budget assign for the development of software: Budget for different phases is different. Any change at any stage results in delay to procure the budget and in some cases sourcing of finance also matters.

Number of people participates: Number of people participate at different phases according to the requirement analysis also effects the release time policy as the wrong deployment of manpower results in delay of project.

Project duration: Estimation of time and completion of project within the expected time must be taken under account and the risk associated with the delay in project is also estimated.

Testing efforts: The efforts applied to make software error free also affect its release time policy. For simplicity of the problem we are assuming that the testing efforts for different phases of software development are different.

4. GOAL PROGRAMMING MODEL

Although few of above mentioned variable is taken under account, but the given problem is extended to the any number of variable on which the release time policy depends.

B_1, B_2, B_3 and B_4 are the budget per day assign to the Requirement analysis, Design, Integration and Testing respectively. 'B' be the total assign budget.

M_1, M_2, M_3 and M_4 is the number of man power per day assign to the Requirement analysis, Design, Integration and Testing respectively. 'M' be the total manpower assign.

E_1, E_2, E_3 and E_4 is the efforts assign per day to the Requirement analysis, Design, Integration and testing respectively. 'E' be the total efforts required.

R, D, I, and T denotes the decision variables, in terms of time taken in respective phases of requirement analysis, Design, Integration and testing respectively and T_t be the total time taken in testing.

Minimize Z = R + D + I + T

Subject to:
B_1 R + B_2 D + B_3 I + B_4 T ≤ B
M_1 R + M_2 D + M_3 I + M_4 T ≤ M
E_1 R + E_2 D + E_3 I + E_4 T ≤ E

R, D, I, T ≥ 0

The above mentioned Linear programming model can be converted to goal programming model

Minimize Z = $P_1\, d_1^+ + P_2\, d_2^+ + P_3\, d_3^- + P_4\, d_4^+$

Subject to:
B_1 R + B_2 D + B_3 I + B_4 T + $d_1^- - d_1^+$ = B
M_1 R + M_2 D + M_3 I + M_4 T + $d_2^- - d_2^+$ = M
E_1 R + E_2 D + E_3 I + E_4 T + $d_3^- - d_3^+$ = E
R + D + I + T + $d_4^- - d_4^+$ = T_t
R, D, I, T, $d_1^-, d_1^+, d_2^-, d_2^+, d_3^-, d_3^+, d_4^-, d_4^+ \geq 0$.

5. MODEL ILLUSTRATION

The proposed goal programming model is tested using an illustration. The impact of changing the priority in objective function can also be done. Although the model can be solved by many software's but due to easiness we are using solver of MS Excel to illustrate model.

Minimize Z = $P_1\, d_1^+ + P_2\, d_2^+ + P_3\, d_3^- + P_4\, d_4^+$
Subject to:
8,000 R +5,000 D + 2,000 I + 7,000 T + $d_1^- - d_1^+$ = 25,000
10 R + 3 D + 4 I + 8 T + $d_2^- - d_2^+$ = 25
2,000 R + 750 D + 480 I + 1,280 T + $d_3^- - d_3^+$ = 4,700
R + D + I + T + $d_4^- - d_4^+$ = 11

R, D, I, T, $d_1^-, d_1^+, d_2^-, d_2^+, d_3^-, d_3^+, d_4^-, d_4^+ \geq 0$

6. CONCLUSION

The present research deals with the optimal software release time policy problem. Extensive literature review has been done and various methods that are dealing with the problem is identified. The goal programming method to deal with release time is proposed as it is multi criteria decision making problem. Various parameters like requirement analysis time, design, implementation and testing time is taken under account using the constraints like budget, men hours, testing efforts and duration of project. At last the model is illustrated using numerical problem.

Table2: Illustration of the goal programming problem

software release time model														
variables	R	D	I	T	u1	v1	u2	v2	u3	v3	u4	v4		
	Requirem	design	integratio	testing	underachi	overachie	underachi	overachie	underachi	overachiv	underachi	overachie	left hand	right hand
objective coefficient						0.5						0.5		
constraint	8000	5000	2000	7000	1	-1							91000	91000
	10	3	4	8			1	-1					84	84
	2000	750	480	1280					1	-1			15820	15820
	1	1	1	1							1	-1	11	11
variables	R	D	I	T	u1	v1	u2	v2	u3	v3	u4	v4		
solution v	5.653846	9.153846	0	0	0	0	0	0	0	2353.077	0	3.807692		
objective	1.903846													

7. REFERENCES

[1] Aggarwal K.K, Singh Yogesh, "Software Engineering". (2001) New Age International Publishers.

[2] Cai, X. and Lyu, M.R. (2004), An Empirical Study on Reliability and Fault Correlation Models for Diverse Software Systems, in Proceedings 15th International Symposium on Software Reliability Engineering, Saint-Malo, France, pp.125-136.

[3] Chen, M.H., Lyu, M.R., and Wong, E. (2001), Effect of Code Coverage on Software Reliability Measurement, IEEE Transactions on Reliability Vol. 50, No.2, pp. 165-170.

[4] Goel. A.L, Okumoto K. (1979)"Time dependent error detection rate model for software reliability and other performance measures" IEEE Transactions on Reliability; R-28(3): 206-211.

[5] Gokhale, S. S., Philip, T., Marinos, P. N. and Trivedi, K. S. (1996), Unification of Finite Failure Non-Homogeneous Poisson Process Models through Test Coverage, Technical Report, Center for Advanced Computing and Communication, Department of Electrical and Computer Engineering, Duke University.

[6] Gokhale, S., Wong W.E., Trivedi, K.S. and Horgan, J.R. (1998), An Analytical Approach to Architecture-Based Software Reliability Prediction, IEEE International, Computer Performance and Dependability Symposium, Durham, NC, pp. 13-19.

[7] Guillermo, N. and Manic, M. (2006), Fuzzy performability analysis of disk arrays, International Symposium on Industrial Electronics, pp. 9-13.

[8] Huang, C.Y., Kuo, S.Y. and Lyu, M.R. (2001), A Framework for Modeling Software Reliability Considering Various Testing Efforts and Fault Detection Rates, IEEE Transactions on Reliability, Vol. 50,No. 3, pp. 310-321

[9] Hudson, G.R. (1967), Program Errors as Birth and Death Process, Tech. report SP-3011, System Development Corp.

[10] Huo R.H, Chen I.Y, Cheng.Y.P, Kuo. S.Y,(1996) "Optimal Release Policies for Hyper- Geomertic Distribution Software reliability growth model, " IEEE Tran.Rel., vol 45, no.4 pp 646-651 Dec. 1996.

[11] Jelinski Z, Moranda PB. "Software reliability research" In: Freiberger W, (Ed.) Statistical Computer Performance Evaluation 1972; New York: Academic Press: 465-497.

[12] Kapur P.K., AggarwalS., Garg R.B.(1984.), "Bi-criterian Release Policy for Exponential Software Reliability Growth Models ", Operational Research, vol 28,pp 165-180,

[13] Kapur PK, Garg RB, Kumar S.(1999) "Contributions to hardware and software reliability"

[14] Kapur PK, Garg RB. "A software reliability growth model for an error removal phenomenon" Software Engineering Journal 1992; 7: 291-294.

[15] Khaled, M.S. (2009), What is Hampering the Performance of Software Reliability Models, A Literature review, Proceedings of the International Multi Conference of Engineers and Computer Scientists , Hong Kong, pp. 231-238.

[16] Krishnamurthy, S. and Mathur, A.P. (1997), On the estimation of reliability of a software system using reliability of its components, in Proceedings of the 8th IEEE International

[17] Lee, I., Tang, D., Iyer R.K. and Hsueh M.C. (1993), Measurement-Based Evaluation of Operating System Fault Tolerance, IEEE Transactions on Reliability, pp.238-249.

[18] Littlewood, B., Abdel-Ghaley, A.A and Chan, P.Y. (1986), Evaluation of Competing Software .

[19] Lyu, M. R. (1996), Handbook of Software Reliability Engineering, IEEE Computer Society measurement, International Journal of Computer Science and Network Security, Vol.9 No.6, pp. 23-30.

[20] Lyu,M.R. (2007), Software Reliability Engineering, A Roadmap , in proceedings of international conference on Future of Software Engineering, Washington, pp.153-170

[21] Mohanta, D.K. (2005), Fuzzy Markov model for determination of fuzzy state probabilities of generating units including the effect of maintenance. IEEE Trans. On Power System, Vol.20, no.4, pp. 2117-2124.

[22] Musa, J. D., Iannino, A. and Okumoto, K. (1987) Software Reliability - Measurement, Prediction, Application, McGraw Hill, New York.

[23] Ohba M. "Software reliability analysis models" IBM Journal of Research and Development 1984; 28: 428-443.

[24] Pham H. "System Software Reliability"(2005); Springer Series in Reliability Engineering. Reliability and its Interdisciplinary Nature.

[25] Piwowarski, P., Ohba, M. and Caruso, J. (1983), Coverage Measurement Experience During Function Test, in Proceedings of Fifteenth International IEEE Conference on Software Engg., pp.387-301.

[26] Schneidewind, N.F.(1972), An Approach to Software Reliability Prediction and Quality Control, Fall Joint Computer Conference, AFIP Press, Montvale, NJ. Vol. 41, pp. 837- 847.Singapore: World Scientific Publishing Co. Ltd.

[27] Smidts, C. and Sova, D. (1999), An Architectural Model for Software Reliability Quantification: Sources of Data, Reliability Engineering and System Safety Vol.64, pp.279–290.

[28] Suri, P.K. (2009), Simulator for Risk assessment of software project based on performance Symposium on Software Reliability Engineering, Mexico, pp.146-145.

[29] Tang, D. and Iyer, R.K. (1992), Analysis and Modeling of Correlated Failures in Multicomputer systems, IEEE Trans. Computers, Vol. 41, No. 5, pp. 567-577.

[30] Taylor, R. and Vander, Hoek A. (2007), Software Design and Architecture: The Once and Future Focus of Software Engineering, International conference on Future of Software Engineering , IEEE-CS Press, pp. 226-243.

[31] Tohma, Y., Tokunaga, K., Nagase, S. and Murata, Y. (1989), Structural approach to the estimation of the number of residual software faults base on the hyper-geometric distribution, IEEE Transactions on Software Engineering, Volume SJ2-15, no. 3, pp. 345-355.

[32] Yadav, A. and Khan R.A.(2009), Critical review on software reliability models, International Journal of recent trends in Engineering, Vol 2, No. 3, pp. 114-116.

[33] Yamada H, Ohtera H, and Narihisa H. "Software reliability growth models with testing effort" IEEE Trans. On Reliability 1986; R-35(1): 19-23.