International Journal of New Media Studies (IJNMS), ISSN: 2394-4331
Volume 8 Issue 1, January-June, 2021, Impact Factor: 6.789

An Examination of Software Reliability Models in the Context of Software Product Release

Podduturu Sharanya[1], Dr. Deepak Sharma[2]

[1]Research Scholar in Computer Science and Engineering Dept, Monad University, Hapur, Pilkhuwa Distt, Uttar Pradesh, India
[2]Professor in Computer Science and Engineering Dept, Monad University, Hapur, Pilkhuwa Distt, Uttar Pradesh,. India.

ABSTRACT

The software manufacturer will have to pay for post-release repairs if a product with a large number of flaws is made available to users too soon. A product that is released too late runs the risk of missing a market window and incurring additional development costs. Software Reliability Growth Models (SRGMs) are used to estimate software release time and are capable of capturing the quantitative features of testing. From a cost-benefit perspective, SRGMs help developers determine whether to deliver software products at the best time by offering practical methods for reducing the anticipated overall cost of the software system. This paper presents the findings from a cost model study, which contributes to the discussion of when to stop testing software products. The research focuses on the relationship between the cost of development and the software product's delivery schedule, as well as the overall cost of the software, which includes risk charges like fines for late software delivery and repair expenses for defects found during the warranty period. We also look into different software release approaches, such as those that are based on the complementary constraints of dependability and cost.

Keywords: Non-Homogeneous Poisson Process (NHPP), Software Release Rules, Software Testing, Cost Models, and Software Reliability Growth Models.

INTRODUCTION

Today, science and technology demand high performance hardware and high quality software in order to achieve new breakthroughs in quality and productivity. It is the integrating potential of the software that has allowed designers to contemplate more ambitious systems, encompassing a broader and more multidisciplinary scope, with the growth in utilization of software components being largely responsible for the high overall complexity of many system designs. However, in stark contrast with the rapid advancement of hardware technology, proper development of software technology has failed miserably to keep pace in all measures, including quality, productivity, cost and performance.

When the requirements for and dependencies on computers increase, the possibility of a crisis from computer failures also increases. Hence, for optimizing software use, it becomes necessary to address issues such as the reliability of the software products. There are many probabilistic and statistical approaches to modelling software reliability. Using tools/techniques/methods, software developers can design several testing programs or automate testing tools to meet the client's technical requirements, schedule and budget. These techniques can make it easier to test and correct software, detect more bugs, save more time and reduce expenses significantly [14].

There has been much effort expended in quantifying the reliability of a software system through the development of models [42]. These models are collectively called Software Reliability Models (SRMs). The main goal of these models is to fit a theoretical distribution to time-between-failure data, to estimate the time-to-failure based on software test data, to estimate software system's reliability and to design a rule for determining the appropriate time to terminate testing and to release the software into the market place [6], [41], [51]. However, the success of SRMs depends largely on selecting the appropriate model that best satisfies the stakeholder's need.

While testing software, SRMs are useful in measuring reliability for the quality control and testing process control of software development. In particular, SRMs that describe software failure-occurrence or fault-detection phenomenon in the system phase are called Software Reliability Growth Models (SRGMs). In the testing and validation phase of the software product life-cycle, the common goal of these models is to support the trade-off between three dimensions, namely, quality, schedule and cost. Despite their shortcomings - excessive data requirements for even modest reliability claims, difficulty of taking relevant non-measurable factors (such as software complexity, architecture, quality of verification and validation activities, and test coverage) into account etc. - SRMs offer a way to quantify uncertainty that helps in assessing the reliability of software systems, and may well provide further evidence in minimizing development cost and predicting software release time [1], [13], [17], [24].

Although testing is an efficient way to detect and re-

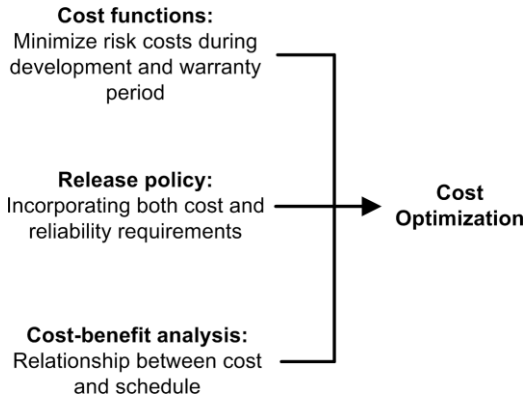


Figure 1. Identified Perspectives for Cost Optimization

solve faults to avoid failure of a software system, exhaustive testing is impractical. Therefore, software developers need to decide when to stop testing and release the software to the customers. From a cost-benefit viewpoint, SRGMs aid developers to decide the optimal release time of the software product by minimising the expected total software system cost. This paper helps answer the question of when to stop testing a software product by presenting the perspectives from a study of cost models. The study focuses on aspects of the relationship between development cost and schedule delivery of the software product and the total software cost including the risk costs, such as the penalty cost incurred due to late delivery of the software product and the cost of fixing a fault during the warranty period. We also investigate various software release policies, for example, release policies based on dual constraints of cost and reliability. The aim of this study is to help provide a better understanding of the usefulness of SRGMs in determining when to release a software product from the viewpoint of achieving an optimization between development cost and software quality.

I. COST OPTIMIZATION

Before releasing a software product, an important decision from an economic standpoint is whether to continue testing, stop testing or scrap the software [38]. Though prolonged testing is desirable from a reliability point of view, it adds substantial cost to the software development. On the other hand, if the cost of testing or the cost of delay in release is very high, the solution will tend to be not to test and to scrap the software due to high risk costs. In practice, cost optimization is a trade-off between three dimensions, namely, cost functions, release policy and cost-benefit analysis where the objective is to minimize total cost, i.e., development cost and risk cost, while maximizing benefits to the software manufacturer [42]. In Figure 1, we have identified these perspectives which contribute equally towards cost optimization of a software product. While appropriate cost functions aid project managers to minimize risk costs during development and in the warranty period, software release policies incorporating the dual constraints of cost and reliability help to obtain possible software release time values; and cost-benefit analysis helps to decide the optimum software release time.

A. Uncertainty in Releasing a Software Product

Software product development is characterized by unpredictability, and there are often large discrepancies between the initially planned and actual project objectives. The unpredictability of software product development is not new. Different criteria can be identified to formulate the cost optimization problem. Firstly, if the requirement is fault-free software or any other reliability goal, then the problem is to determine the minimal testing time to reach the reliability requirement. Secondly, if the total software cost (i.e., combined cost of developing and maintaining a software product) is to be considered, then the optimum release time is determined using appropriate cost functions, so that the total expected software cost can be minimized. Based on the Jelinski and Moranda (J-M) model [16], Koch and Kubat in their paper [28] introduced a balanced cost-benefit function considering not only the total costs of testing, but also the benefits derived from the application to determine the optimal release time. This cost-benefit function takes into account the planned delivery time, cost of correcting faults (in both testing and implementation phases of software product life-cycle), cost of goodwill due to faults after delivery, loss due to delayed delivery, benefits of using the testing team after the software release, and benefits associated with successful operation of the software per time unit. In such cases, the expected total software cost can be minimized using their cost function together with any SRM. Therefore, software release time is associated with the cost of software testing and the gain of an earlier release of the software. Gain is defined as the difference in cost incurred when all the faults are removed during the operational phase as against the cost when some faults are removed during the testing phase and others are removed during the operational phase [2]. In practice, a software manufacturer wants to determine the optimum testing time from a cost-benefit point of view.

If the software industry is unable to find easy-to-implement improvement strategies, the typical software manufacturer organization is likely to become increasingly less predictable in terms of cost and quality. In markets with increasing competition and smaller market windows, software manufacturer might experience increasing pressure to release software products prematurely, disregarding the total life-cycle effects [3]. In this case, uncertainties are:

Unknown software product behaviour: It is difficult, if not impossible, to guarantee that the software product meets the exact functional and non-functional requirements. This may lead to dissatisfied customers/end-users and to unforeseen, even potentially dangerous, situations. Apart from the fact that people's lives may be at risk,

such situations can have an enormous financial impact on the software manufacturer.

Unknown operational maintenance cost: The post-release or maintenance cost of the software may become unexpectedly high. If the exact status of the software with its documentation is unknown, a software manufacturer may be confronted with high maintenance costs for correcting failures. Future adaptive and perfective maintenance activities may be severely hampered.
The presence of these uncertainties may have a dra- matic impact on a software manufacturer's market posi- tion. Releasing a software product too late might severely undermine its market position, releasing a software product prematurely might lead to recalls and warranty, or even liability problems.

B. Cost Overruns

Estimation of accurate time-to-market (schedule) of a software product is a major topic of concern in the software product development industry [9]. Having computed a nominal value for a schedule, a software manufacturer may face the question of adjusting the schedule to either deliver the software product at an accelerated pace, or to improve efficiency. Any adjustment to the schedule will have a subsequent cost impact. As a schedule is further stretched, the manufacturer faces severe cost penalties. Since it is expensive to fix post-release failures, software manufacturers frequently decide to release the system as late as possible, i.e., at the deadline or after the deadline. If the software fails during the warranty period, additional costs are incurred by the manufacturer, which are known as the risk costs [35]. Risk costs also include the penalty cost which is incurred by the manufacturer for not delivering the software within the scheduled time [20], [45]. The penalty cost is usually proportional and exponential to the delivery time.

II. Software Cost Models

SRMs offer a way to quantify uncertainty that helps in assessing the reliability of software systems, and may well provide further evidence in minimizing development cost and predicting software release time. 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 Goel and Okumoto (G-O) [10]. Models are described, for example, by Yamada and Osaki [47]; Brettschneider [5], presenting a simplified decision-making model; Xie and Yang [43], incorporating the effect of imperfect debugging on software cost; Huang et al. [14], incorporating ways to improve test efficiency; Yamada et al. [44], incorporating life-cycle distribution and applying discount rate; Pham and Zhang [37], incorporating test coverage; Leung [29], incorporating a budget constraint; Kapur and Garg [20]; Ehrlich et al. [7]; Yang and Chao [50]; Boland and Singh [4]; Hou et al. [12]; Koch and Kubat [28], incorporating the penalty cost when the software is delivered after the scheduled delivery time; Pham and Zhang [35], [36], developing a generalized cost model which considered fault removal cost, warranty cost, and software risk cost due to software failures; Liu and Chang [32] also addressing the risk cost; and Kimura et al. [27] developing a software cost model considering software maintenance cost during the warranty period.

Using the G-O Non-Homogeneous Poisson Process (NHPP) reliability model, Okumoto and Goel [34] suggested a simple cost model which determines a point in time as an optimal software release time and cost of testing per time unit. The expected cost $C(T)$ of the software product released at a given time T is calculated by using the following cost function:

$$C(T) = C_1(T) + C_2(T) + C_3(T) \qquad (1)$$

The actual cost of a software project is given by $C(T)$, and is often called the software cost model. C_1 is the cost incurred by fault removal activities during testing, C_2 is the cost incurred by fault removal activities during the operational phase, C_3 is the general cost of software testing. Despite the fact that many software cost models have been proposed, for most of them, $C_1(T)$, $C_2(T)$, and $C_3(T)$ are common cost components that have been adopted. Using a formula given by Yang et al. [49] the software cost can be formulated as:

$$C(T) = C_0 + \sum_{i=1}^{6} C_i(T) \qquad (2)$$

where C_0 is the setup cost for software testing, C_4 is the risk cost due to software failures, C_5 is the cost to remove faults in the warranty period and C_6 is the penalty cost. In existing research, different formulations of cost components $C_i(T)$, $1 \leq i \leq 6$, have been proposed. Moreover, other cost components can be considered and added to the generalized cost model as well. It is known from [19] that the costs of quality can be categorized into prevention costs, appraisal costs, internal failure costs, and external failure costs. Later, Slaughter et al. [40] elaborated that while developing a software product, appraisal cost comprises cost of code inspections, testing, software measurement activities, etc.; prevention costs includes the costs of training man-power in design methodologies, quality improvement meetings, software design reviews, etc.; internal failure costs is a mixture of the costs of rework in programming, reinspection, retesting, etc.; external failure costs represent expenses incurred in field service and support, maintenance, liability damages, litigation expenses, etc. In the generalized cost model above, $C_1(T)$ and $C_3(T)$ can be viewed as a mixture of appraisal costs and internal failure costs; $C_2(T)$, $C_4(T)$, $C_5(T)$, and $C_6(T)$ can be viewed as external failure costs. Therefore, it can be noted that the cost defined in (2), $C(T)$, is only part of the total cost incurred in the development of the software product.

The formulation of $C_1(T)$ is generally considered to be proportional to the number of software faults removed during the testing phase.

$$C_1(T) = c_1 m(T) \tag{3}$$

where c_1 is the deterministic cost to remove each fault per unit time during testing and $m(T)$ is the expected number of saftware failures by time T.

Similarly, $C_2(T)$ is considered to be proportional to the number of software faults removed during the operational phase. Thus,

$$C_2(T) = c_2[m(T_{LC}) - m(T)] \tag{4}$$

where c_2 is the cost of fixing each fault during operation and $m(T_{LC})$ is the expected number of software failures in the life-cycle length of the software product. Since, c_1 is the deterministic cost to remove each fault per unit time durig testing and $c_2(T)$ is the cost of removing a fault in the operational phase; normally, $c_2 > c_1$.

$C_3(T)$ is assumed to be a power function of testing time T, i.e.,

$$C_3(T) = c_3 T^k \tag{5}$$

The parameter $k(0 < k \leq 1)$ reflects the fact that the increasing gradient is different in the beginning and at the end of testing. In the simplest case, $k = 1$. For SRMs considering test effort [13], [15], $C_3(T)$ is formulated as:

$$C_3(T) = c_r[W(T)]^k \tag{6}$$

The risk cost due to software failures, $C_4(T)$, is given as:

$$C_4(T) = c_4[1 - R(x|T)] \tag{7}$$

Pham and Zhang [36] developed a Net Gain in Reliability (NGIR) model and defined the expected NGIR, $E(T)$, of the software development process as the economical net gain in software reliability that exceeds the expected total cost of the software development.

$E(T)$ = Expected gain in reliability - (total development cost + risk cost)

Using (3, 4, 5 and 7) the NGIR is, therefore, given by Pham and Zhang [36] is:

$$E(T) = R(x|T)(C_g + c_4) - \left[C_3(T) + m(T)\left(c_1 + \frac{c_r}{2}m(T)\right) + c_4\right] \tag{8}$$

where, $m(T)\left(c_1 + \frac{c_w}{2}m(T)\right)$ is the expected total costs to remove all faults detected during the period $[0, T)$.

Pham and Zhang [35] and Sgarbossa and Pham [8] suggested the cost to remove a fault during the warranty period, $C_5(T)$, given by:

$$C(T) = C_0 + C_3(T) + c_1 m(T)\mu_y + C_5\mu_r[m(T + T_r) - m(T)] + C_R[1 - R(x|T)] \tag{9}$$

Finally, the penalty cost [12], [20], [28], $C_6(T)$, can be formulated as:

$$C_6(T) = I(T - T_d)C_p(T - T_d) \tag{10}$$

where, $I(.)$ is an indicator function, defined as, $I(t) \equiv 1$, if $t \geq 0$, else 0.

It is of great interest to determine an optimum software release time satisfying both cost and reliability requirements. Both the expected total software cost $C(T)$ and the software reliability $R(x|T)$ are assumed to be the evaluation criteria satisfying software cost and software reliability requirements simultaneously.

A. Cost Model for Imperfect and Explicit Debugging

Usually the costs of testing are based on software reliability models which assume that the fault is debugged as soon as it is detected, and the debugging process is perfect. According to Gokhale et al. [11], the time required to debug a fault, however, cannot be neglected; and hence at any given time, the number of faults debugged will be less than the number of faults detected. Thus, the cost of resolving a failure in practice consists of two parts: the cost of opening a modification request and diagnosing the fault that caused a failure; and the cost of removing a fault and verifying that the failure no longer occurs. The former depends on the fault detection process, and the latter depends on the debugging process. Gokhale et al. [11] denote c_7 (total cost of detecting a fault and resolving a failure during testing) as the cost associated with the former, and c_9 (cost of debugging a fault in the testing phase) with the latter. For a release time T, the economic model presented by Ehrlich et al. [7]:

$$E = C_B(T) + c_1 m(T) + c_2(a - m(T)) + c_8(\lambda(n, T)\eta I) \tag{11}$$

is modified by Gokhale et al. [11] to be:

$$E = C_B(T) + c_7 m_D(T) + c_9 m_R(T) + c_2(a - m_R(T)) + c_8(\lambda(n, t)\eta I) \tag{12}$$

where $m_D(t)$ and $m_R(t)$ denote the expected number of faults detected, and removed respectively, by time T. c_8 (cost to customer operations in the field) is considered as a function of the adjusted failure rate $\lambda(n, t)$ of the software.

B. Cost Factor for Release of New Versions

To ensure ongoing software quality, new releases of given software are required. These releases provide the customer with improved and fault-free versions and the process of providing new versions continues throughout the software product life-cycle. A common situation in

practice is that the same software is released several times in different versions. Usually, these packages are not static and require changes to correct faults, improve performance, and add new and improved features [42].

In this light, Levin and Yadid [30] proposed a model for determining the release time of a new version of software by using the G-O model for the software failure process. The optimization is carried out by minimizing the expected total development cost. Four different cost factors associated with the release of new release are:

1) Based on the G-O model with parameters a and b, the expected number of faults detected during time [0,t), is given by $a[1 - \exp^{-bt}]$. Taking the sum of the fix cost ц associated with the next release and the average cost of correcting an fault c_1, given by Levin and Yadid [30] is:

$$ц + c_1 a[1 - \exp^{-bt}] \quad (13)$$

where fix cost includes the cost of documentation, distribution, installation, customer training etc., and the average cost of correcting a fault is assumed to be proportional to the number of detected faults.

2) Cost of improving the software during time (0,t] is thus given by:

$$c_5 v[\exp^{-zt} - 1 + zt]z \quad (14)$$

3) Next cost factor is called the cost of software obsolescence, obtained as:

$$\frac{c_6 x[\exp^{-yt} - 1 + yt]}{yt} \quad (15)$$

This factor represents the loss of market share since the longer it takes to release a new version, the more users turn to other competitors.

4) The optimum time to release a new version is then determined by minimizing the total cost per unit time, given by:

$$c(t) = \frac{ц + c_1 a[1 - \exp^{-bt}] + c_5 v[\exp^{-zt}}{\frac{-1 + zt]z + \frac{c_6 x[\exp^{-yt} - 1 + yt]}{yt}}{t}} \quad (16)$$

SOFTWARE RELEASE POLICIES

After the prescribed reliability goal is set and the focus is to achieve the target reliability, for any software cost model there is a need to determine the optimum release policy by minimizing the expected total cost subject to the reliability goal. A lot of software release policies discuss the best time to make a decision to stop testing software and release it to the customer. In the literature, the optimal software release problem has been discussed by researchers since the early 1980s [13], [20], [22–26], [46]. Software release policies which explain dual constraints of minimizing a total average software cost satisfying a reliability requirement have been studied by Koch and Kubat [28]; Okumoto and Goel [34]; Shanthikumar and Tufekci [39]; Yamada et al [45], [48]; Kimura et al. [27]; Kapur and Garg [21]; Huang [13]; Ahmad et al. [1], and Yang et al. [49]. Software release policies based on cost and reliability criteria and their variants such as controlling the test effort expenditures are discussed by [13], [24] and estimation of penalty cost by [20]. In addition, Kapur and Garg studied software release policies for Continuous time SRGMs [22], optimising two conflicting objectives, namely software cost subject to budget and reliability constraints. The policies for the Discrete time SRGMs are discussed by [26], [46]. These release policies are useful to control the total software testing cost in both testing and validation phases of the software product life-cycle.

A. Release Policy for Continuous Exponential SRGM

For an Exponential SRGM in Continuous time, Kapur et al. in [22], [23] defined the mean value function as $m(t) = a(1 - \exp^{-bt})$ and the failure intensity as $\lambda t \equiv m'(t) = ab\exp^{-bt}$. It may be observed that $\lambda(t)$ is a decreasing function in t with $\lambda(0) = ab$ and $\lambda(\infty) = 0$. Substituting cost components $c_1(T)$ and $c_2(T)$ from equations (3) and (4) in (1), the total cost function can be obtained as:

$$C(T) = c_1 m(T) + c_2(m(T_{LC}) - m(T)) + c_3 T \quad (17)$$

and the expected software reliability $R(x|T)$ given that the last failure occured in $T \geq 0(x \geq 0))$ is defined as:

$$R(x|T) = \exp^{-[m(T+x)-m(T)]} \quad (18)$$

While determining software release policies for SRGMs, three types of criteria are commonly considered:

Cost Criteria: The objective in this case is to find a release time T, such that the total expected software cost during the software product life-cycle is minimised. By differentiating total cost function $C(T)$ in equation (17) with respect to T, one obtains:

$$C'(T) = -(c - c)m'(T) + c_3 \quad (19)$$

where $C'(T) = 0$ if $m'(T) = \frac{c_3}{c_2 - c_1}$.

Reliability Criteria: The objective in this case is to find a release time T, satisfying $R(x|T) \geq R_0$, where $(0 < R_0 < 1)$ is the required level of reliability. From (18) $R(x|0) = \exp^{-m(x)}$ and $R(x|\infty) = 1$. By differentiating $R(x|T)$ with respect to T, one obtains:

$$R'(x|T) = \exp^{-[m(T+x)-m(T)]} (ab\exp^{-bT}(1 - \exp^{-bx})) \quad (20)$$

Cost and Reliability Criteria: The objective in this case is to either minimise cost subject to reliability not less

than a predefined reliability level or to reliability subject to cost not exceeding a predefined finite budget. The objective is, therefore, either minimise $C(T)$ subject to $R(x|T) \geq R_0$ or maximise $R(x|T)$ subject to $C(T) \leq C_B$, where C_B is the predefined budget level.

B. Release Policy for Discrete Exponential SRGM

The mean value function (number of faults detected in n test run) for a Discrete Exponential SRGM is given by Kapur et al. [23], [25] as:

$$m(n) = a[1-(1-b)^n], a > 0, 0 < b < 1 \quad (21)$$

and the discrete failure intensity is given as:

$$\lambda(N) = m(N+1) - m(N) = ab(1-b)^N \quad (22)$$

given that $\lambda(N)$ decreases as N increases, where $\lambda(0) = ab$ and $\lambda(\infty) = 0$. The cost during the software product life-cycle N_{LC}, when the software is released after N test runs is:

$$C(N) = c_1 m(N) + c_2(m(N_{LC}) - m(N)) + c_3 N \quad (23)$$

Comparing the cost when the software is released after $(N+1)$ and N test runs yields:

$$C(N+1) - C(N) = -(c_2 - c_1)\lambda(N) + c_3 \quad (24)$$

Discrete software reliability $R(x|N)$ is defined as the probability that software failure does not occur in $(N, N+x]$ test runs, given that the last failure occurred in N test runs, given by Kapur et al. [23], [25] is:

$$R(x|N) = \exp^{-m(N+x)-m(N)} \quad (25)$$

where x is the number of test cases.

Combining (24) and (25), the cost and reliability criteria are discussed by Kapur et al. in [25].

C. Release Policy under Penalty Cost

If the software manufacturer fails to release the software product at the scheduled delivery time, additional costs are incurred by the manufacturer termed as penalty cost. To determine penalty cost, Yamada et al. [45] assumed T_s, i.e., scheduled delivery time of the software, is a random variable with cumulative distribution function (CDF) $G(t)$ and finite probability density function (PDF) as $g(t)$. Using $C_P(t)$ as the penalty cost incurred in time $(0, t]$ due to delay in software release, they obtained the expected penalty cost in $(T_s, T]$ as:

$$\int_0^T C_P(T-t)dG(t) \quad (26)$$

Thus, the total expected software cost during the software product life-cycle obtained by Yamada et al. [45] is:

$$C(T) = C(T|T_s) = c_1 m(T) + c_2(m(T_{LC} - m(T)) + c_3 T + \int_0^T C_P(T-t)dG(t) \quad (27)$$

Later, Kapur and Garg [20] added that the expected penalty cost is an increasing function in t. Differentiating $C(T)$ in (24) with respect to T, they the authors obtained the expected penalty cost as:

$$C'(T) = -[(c_2 - c_1)m'(T) - \frac{d}{dT}\int_0^T C_P(T-t)dG(t)] + c_3 \quad (28)$$

Using (28), Kapur et al. [20], [23] derived the release policy under penalty cost based on minimizing $C(T)$ subject to $R(x|T) \geq R_0$ where $T \geq T_s$. Two were cases considered: (i) when T_s is deterministic; and (ii) when T_s has an arbitrary distribution.

D. Release Policy with Test Effort

For software release policies, the testing cost is directly proportional to the testing time T. Therefore, if T becomes infinitely large, so does the testing cost. In reality, no software developer will spend infinite resources on testing the software. Test effort curves are typically used to measure testing resources, such as CPU time, man power etc. Assuming test effort to be Exponential, Kapur & Garg [23] discussed the release policy for an Exponential SRGM with the added assumption that testing resources are described by an Exponential curve. For an Exponential type test effort curve, $w(t) = \alpha\beta\exp^{-\beta t}$ describes instantaneous testing resources. The test effort expenditure in time t is generally given as:

$$W(t) = \int_0^t w(x)dx = \alpha(1-\exp^{-\beta t}) \quad (29)$$

Generally, the total test effort expenditures does not exceed α even if the software is tested for an indefinite time. Based on minimizing cost subject to reliability not less than a predefined reliability objective R_0, Kapur & Garg [23] formulated the software release policy with test effort as:

$$C(T) = c_1 m(T) + c_2(m(T_{LC} - m(T)) + c_r W(T) \quad (30)$$

The software reliability $R(x|T)$ is given by:

$$R(x|T) = \exp^{[\exp^{-bm(T)} - \exp^{-bm(T+x)}]} \quad (31)$$

When the test effort curve is a Weibull curve instead of Exponential, release policies for cost, reliability and combined cost and reliability are discussed by Lin and Huang [31]. In this paper, the authors assumed that in certain cases, the policies of testing resources allocation could

be changed. Based on this assumption, they presented concepts of multiple change-points into Weibull-type test effort functions. The testing resource allocation problem has also been studied by Jha et al. [18] by minimizing the total software testing cost of a modular software system, given a reliability constraint and an upper bound on the amount of available testing resources.

E. **Bicriterion Release Policy**

For Exponential Continuous time models, the Bicriterion release policy is discussed by Kapur and Garg [23]. This policy optimizes two objectives simultaneously, namely total expected software cost not exceeding a specified budget and software reliability not less than a given reliability level. Such a release policy gives enough flexibility in finding the optimum release time for the software, based on relative importance associated with both cost and reliability. The Bicriterion software release policy aims at minimizing cost and maximizing reliability simultaneously such that the total expected cost during the software product life-cycle does not exceed a specified budget and conditional reliability is not less than a pre-specified reliability objective.

Mathematically, Kapur and Garg [23] state,

$$\begin{aligned} &\text{maximize } \log R(x|T), \\ &\text{minimize } \bar{C}(T) \\ &\text{subject to} \\ &\bar{C}(T) \leq 1 \\ &R(x|T) \geq R_0 \\ &T \geq 0, 0 < R_0 < 1 \end{aligned}$$

where $\bar{C}(T) = \frac{C(T)}{C_B}$. This is reduced to a single objective optimization problem by introducing:

$$\lambda = \begin{matrix} \lambda_1 \\ \lambda_2 \end{matrix} \in R^2,$$

$$\text{where } \lambda_1 \geq 0, \lambda_2 \geq 0, \quad \sum_{i=1}^{2} \lambda_i = 1$$

Here R^2 is the coefficient of multiple determinations and $\lambda_i (i = 1, 2)$ is the priority for the i^{th} component. Using λ_1 and λ_2, a degree of flexibility is introduced over the other release policies where optimization is based either on cost or reliability functions and thus the previously stated formula is further reformulated as

$$\begin{aligned} \text{rcl maximize } F(T) = \lambda_1 \log R(x|T) - \lambda_2 \bar{C}(T) \\ \text{subject to} \\ \bar{C}(T) \leq 1 \\ R(x|T) \leq R_0 \\ T \geq 0, 0 < R_0 < 1 \end{aligned}$$

If $\lambda_1 = 0$ and $\lambda_2 = 1$ and C_B is sufficiently large, this formulation reduces to the classic cost optimisation problem discussed by Okumoto and Goel [34]. Whereas, if $\lambda_1 = 1$ and $\lambda_2 = 0$, this formulation reduces to the reliability optimisation problem to suit high reliability projects, such as nuclear reactors, space exploration etc., since the reliable operation of these projects is critically dependent on the reliable operation of their software components.

However, in the Bicriterion software release policy, λ_1 and λ_2 can be fixed or variable according to the priority attached to the reliability and cost functions. Using $\bar{c}_i = \frac{c_i}{C_B}$, $i = 1, 2, 3$ the objective function $F(T)$ is formulated as:

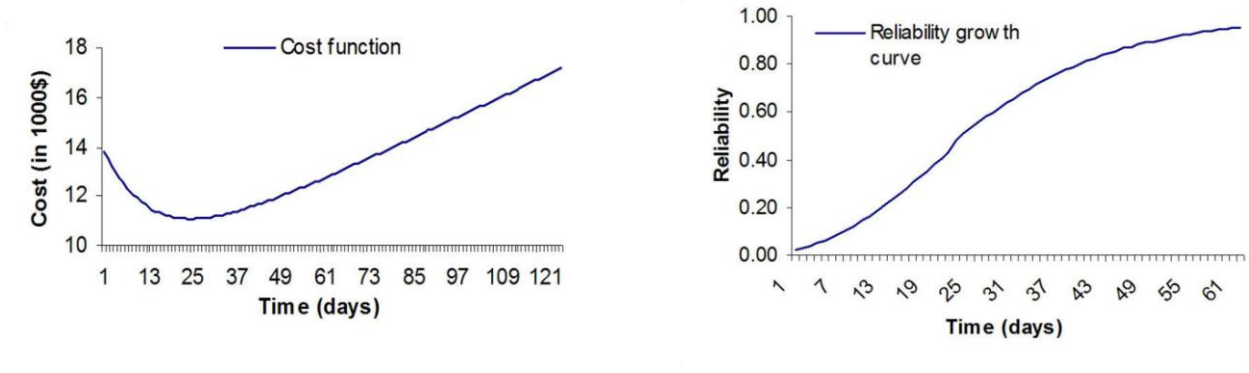
$$\begin{aligned} F(T) = m(T)[\lambda_1 + \lambda_2(\bar{c}_2 - \bar{c}_1)] - \\ \lambda_1 m(T + x) - \bar{c}_2 \lambda_2 m(T_{LC}) - \bar{c}_3 \lambda_2 T \end{aligned} \tag{32}$$

The different values of λ_1 and λ_2 give rise to different values of optimal software release time (T^*) and hence, different $R(x|T^*)$ and $C(T^*)$. Giving more weight to reliability (i.e., higher λ_1) helps obtain an optimal solution with higher value of $R(x|T^*)$. If the emphasis is on maximising reliability only (i.e., $\lambda_1 = 1, \lambda_2 = 0$), then the highest possible reliability value can be achieved by exhausting the total budget.

F. Data Analysis

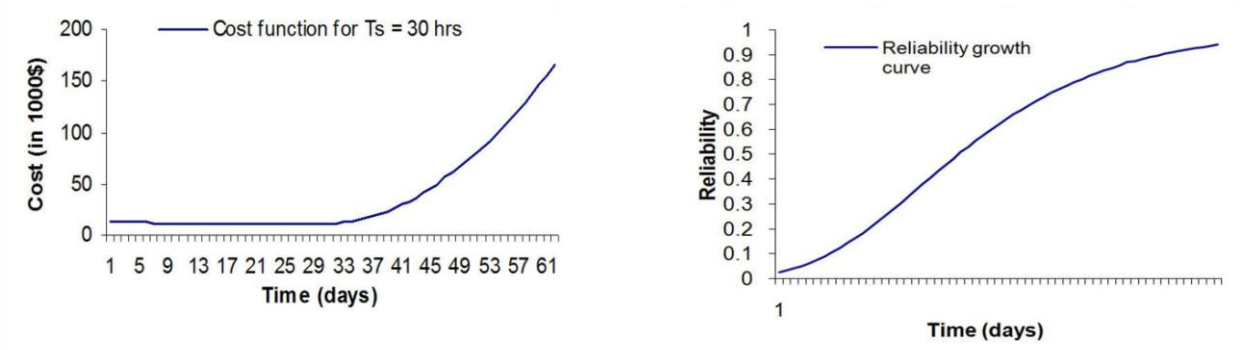
To obtain the cost and reliability values for the software release policies discussed in this section, we consider the data provided by Musa et al. [33]. The G-O model was used for this study, which is one of the earliest NHPP-based SRGMs developed and has been widely used in the literature. The data is for software tested for 125 CPU hrs over 11 days with a total of 32 faults being detected. Using this data set, cost parameters were assumed as $c_1 = 150$, $c_2 = 250$, $c_3 = 70$, the desired reliability level as 0.87, budget cost $C_B = 20000$ and penalty cost = 150. We have assumed these values as an example, since it is expected that software developers have reliable estimates of various model and cost parameters from past experiences. In table I, we have summarized the cost parameters and the desired reliability level. From the failure data recorded in this study, the constant parameters a and b for both Continuous and Discrete Exponential G-O model were obtained as $a = 58.07821$ and $b = 0.0703236$. Under these parameters, the optimal release time obtained for the Discrete Exponential model is 47.55 days at the time point when reliability is 0.87 and cost is $ 11870.43. The plots of (a) cost function and (b) reliability growth curve for the Discrete time G-O exponential model are shown in Figs. 2a and 2b, respectively.

For software release policy under penalty cost, the optimal release time is 47.55 days at the time point when reliability is 0.87 and cost is $ 58070.81. The plots of (a) cost function and (b) reliability growth curve for the



(a) **(b)**

Figure 2. Plots of (a) cost function and (b) reliability growth curve for the Discrete Exponential software release policy



(a) **(b)**

Figure 3. Plots of (a) cost function and (b) reliability growth curve for the Penalty cost software release policy

TABLE I.
SUMMARY OF SELECTED DATA SET [33]

Notation	Value
c_1($)	150
c_2($)	250
c_3($)	70
x(days)	1
$T_{L\&}$ (days)	52
R_0 (Desired reliability level)	0.87
C_B (Budget)	20, 000
Penalty cost($)	150

policy under penalty cost are shown in Figs. 3a and 3b, respectively.

From the failure data, the constant parameters a and b for the Continuous G-O model with exponential test effort function were obtained as a = 61.09838 and b = 0.00634789, and the parameters α, ν of test effort function were obtained as α = 2171.339 and ν = 0.004861476. Note that the value of x for the test effort-based software release policy is 56 CPU hr. Under these parameters, the optimal release time obtained for the test effort model is 52 days when the reliability level is 0.87 and cost is $ 42694.57. The plots of (a) cost function and (b) reliability growth curve for the test effort-based software release policy are shown in Figs. 4a and 4b, respectively.

TABLE II.
DATA VALUES FOR BICRITERION SOFTWARE RELEASE POLICIES

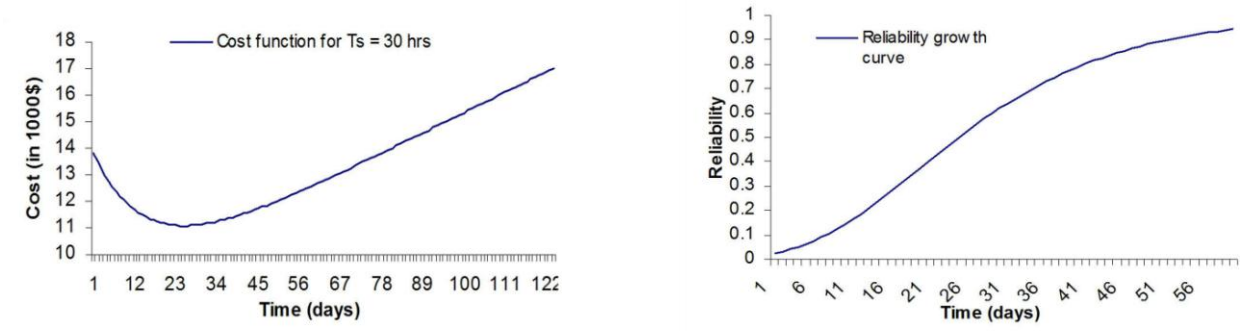
λ_1	λ_2	T^***(days)**	$R(x\|T^*)$	$C(T^*)$
0.6	0.4	68.62418	0.968868	13187.19
0.5	0.5	63.18735	0.954702	12828.30
0.4	0.6	57.9011	0.934981	12489.00

For the Bicriterion software release policy, different values for λ_1 and λ_2 give rise to a different optimal release time T^* and hence, different $R(x|T^*)$ and $C(T^*)$. These values are given in table II. The introduction of λ_1 and λ_2 gives more flexibility to the software project manager in setting objectives and thus one may have a trade-off between cost and reliability depending upon the importance of each. The plots of (a) cost function and (b) reliability growth curve for the Bicriterion software release policy are shown in Figs. 5a and 5b, respectively.

From the above data analysis of the policies discussed earlier, it is clear that the optimal software release time is very close. In practice, the type of policy to be adopted depends mainly on the cost model chosen to estimate the expected cost. For example, if we consider the penalty cost in the cost model then the cost incurred would be more after the scheduled time of software delivery compared to the cost incurred if a cost model is chosen

(a) **(b)**

Figure 4. Plots of (a) cost function and (b) reliability growth curve for the Test Effort software release policy



(a) **(b)**

Figure 5. Plots of (a) cost function and (b) reliability growth curve for the Bicriterion software release policy

without penalty cost. Therefore, the type of the software release policy to be chosen depends on:

- the objectives of the release policy;
- system constraints;
- cost model; and
- the SRGM chosen to describe the failure process.

In general, no particular policy can be considered as best in general. It solely depends on the objectives set by the software project manager/developers, the system constraints and the testing profile to be attained at the release time. Hence, one must first define these and formulate the policy accordingly to obtain the optimum software release time.

CONCLUSIONS

Most of the existing research on determining the op- timal time to release software gives insufficient consid- eration to cost optimization, and the formulations of the problem are generally based on the treatment (such as minimization) of the expected cost, either during the testing or in the warranty period. Since considering cost control during both development and maintenance phases is more meaningful in achieving the overall goal of minimization of total expected costs than simply consid- ering cost in relation to only one of these phases, these formulations are flawed. If these formulations are used, then the solution obtained may give management a false impression that the cost of the software product is at a low level (i.e., has been minimized). In fact, what has been minimized or guaranteed to be below a certain level is the expected cost either during the testing or in the warranty period, not the total expected cost; thus, there exists a certain level of risk that the cost of the software may be unexpectedly high and the project may run over budget.

In this paper, we have studied the cost optimization problem and its impact on optimum software release time in detail. It is clearly shown that cost-benefit analysis, cost functions and software release policies are the desired criteria for scheduled delivery of a software product and to minimize cost overruns during the warranty period. The main contribution of the research presented is to demon- strate the important fact that, in the optimal software release problem, the uncertainty involved in computing total software cost should not be neglected. Based on this standpoint, we have discussed the existing cost functions which are important in studying the total cost of a software product during its life-cycle and are incorporated in obtaining optimum release policies. Further, we have surveyed the software release policies based on the dual constraints of cost and reliability for both Continuous and Discrete time SRGMs. In addition, we have also discussed the Bicriterion policy and the release policies for penalty cost incurred due to missed schedules and to control the test effort expenditures during testing. These software release policies are important to minimize risk and estimate total development cost.

We have also presented data analysis to provide a comparative summary among the cost models surveyed and their usage in terms of determining the best possible release policy for different scenarios. The benefits of using such an analysis can prove invaluable for a project manager, since both cost and reliability curves are very readable and easily interpretable, especially for non-experts, facilitating the strategy that has to be followed for better management of a new software project. Due to the fact that only a few outliners have been considered, a decision maker benefits greatly from comparing the performance results of different models.

It has been repeatedly stressed that software manufacturers are confronted with serious problems when trying to report the pre-release level of product reliability obtained and the expected post-release maintenance cost, based on the level of reliability and the maintainability of the resulting product. The applicability of the existing theory is limited, and the exploratory case studies confirm this to be a problem area. This hampers the determination of the zone of cost effectiveness, especially for larger and more complex software products. This problem area has been known for decades, but no solution has been proposed that has found wide acceptance. The traditional development methods are not able to cope with this, possibly implying that the release trade-off question will become more difficult in the near future due to increasing uncertainty. It might be worthwhile, although ambitious, to pursue research in the area of totally new development approaches, eliminating, or at least reducing, this uncertainty level and moving the decision-making process from complete uncertainty to informed uncertainty.

Nomenclature

c_1	Deterministic cost to remove each fault per unit time during testing phase
c_2	Cost of fixing each fault during operation
c_3	Software test cost per unit time
c_4	Risk cost per software failure
c_5	Average cost of responding to a request for improvement
c_6	Opportunity loss of a software user
c_r	The cost per unit test effort expenditure
C_p	The function for penalty cost
C_g	Coefficient of gain in reliability if the software works successfully
C_R	The loss due to software failure
C_B	The total budget allocated for the software during the software product life-cycle
T	Software release time (same as testing time)
$T_{L\&}$	The life-cycle length of the software product
T_d	Scheduled release time of the software
T_r	Warranty period of the software
T_s	Scheduled delivery time of the software
a, b	Parameters of G-O Exponential model
v, y, z	Model parameters reflecting the dynamic of change in requirements
ц	Fix costs (documentation, distribution, installation, customer training etc.)
a, β	Parameters of Exponential test effort curve
$\lambda(n, t)$	Failure rate of the software after accounting for debugging activities
η	Expected execution time of the software release per field site
I	Number of field sites
$m(T)$	Expected number of software failures by time T
$m_D(t)$	Expected number of faults detected by time with explicit debugging
$m_R(t)$	Expected number of faults removed by time with explicit debugging
μ_y	The expected time to remove a fault during testing period
μ_r	The expected time to remove a fault during warranty period w
$W(T)$	The total test effort spent in $(0, T]$
$w(t)$	Instantaneous testing resource
R_0	Reliability objective
$R(x\|T)$	Reliability function of software by time T for a mission time x
E	Economic consequences involved in stopping test at time T

REFERENCES

[1] N. Ahmad, M. U. Bokhari, S. M. K. Quadri, and M. G. M. Khan. The Exponentiated Weibull Software Reliability Growth Model With Various Testing-efforts and Optimal Release Policy. International Journal of Quality and Reliability Management, 25(2):211–235, 2008.

[2] D. S. Bai and W. Y. Yun. Optimum Number of Errors Corrected before Releasing a Software System. IEEE Transactions on Reliability, 37(1):41–44, 1988.

[3] E. W. Berghout and M. Nijland. Full Life-cycle Manage- ment and the IT Management Paradox. In D. Remeny and
A. Brown, editors, Make or Break Issues in IT Manage- ment, pages 77–107. Butterworth-Heinemann, 2001.

[4] P. J. Boland and H. Singh. Determining the Optimal Release Time for Software in the Geometric Poisson Reliability Model. International Journal of Reliability, Quality and Safety Engineering, 9(3):201–213, 2002.

[5] R. Brettschneider. Is Your Software Ready for Release?
IEEE Software, pages 100–108, 1989.

[6] S. R. Dalal and C. L. Mallows. When Should One Stop Testing Software? Journal of American Statistical Association, 83(403):872–879, 1988.

[7] W. Ehrlich, B. Prasanna, J. Stampfel, and J. Wu. Deter- mining the Cost of A Stop-Test Decision. IEEE Software, 10(2):33–42, 1993.

[8] F. Sgarbossa and H. Pham. A Cost Analysis of Systems Subject to Random Field Environments and Reliability. IEEE Trasactions on Systems, MAN, and Cybernetics - Part C: Applications and Review, 40(4):429-437, 2010.

[9] N. E. Fenton and S. L. Pfleeger. Software Metrics: A Rig- orous & Practical Approach. PWS Publishing Company, 1997.

[10] A. L. Goel and K. Okumoto. Time-Dependent Error Detection Rate Model for Software Reliability and other Performance Measures. IEEE Transactions on Reliability, R-28(3):206–211, 1979.

[11] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi. Incorporating fault debugging activities into software reliability models: a simulation approach. IEEE Transactions on Reliability, 55(2):281–292, 2006.

[12] R. Hou, S. Kuo, and S. Chang. Optimal Release Times for Software Systems with Scheduled Delivery Time Based on the HGDM. IEEE Transactions on Computers, 46(2):216– 221, 1997.

[13] C. Huang. Cost-Reliability-Optimal Release Policy for Software Reliability Models Incorporating Improvements in Testing Efficiency. The Journal of Systems and Software, 77:139–155, 2005.

[14] C. Huang, S. Kuo, and M. R. Lyu. Optimal Software Release Policy Based on Cost and Reliability with Testing Efficiency. International Computer Software and Applica- tions Conference, (COMPSAC), pages 468–473, 1999.

[15] C. Huang and M. R. Lyu. Optimal Release Time for Soft- ware Systems Considering Cost, Testing-Effort, and Test Efficiency. IEEE Transactions on Reliability, 54(4):583– 591, 2005.

[16] Z. Jelinski and P. B. Moranda. Software Reliability Research. In Statistical Computer Performance Evaluation (Ed.) W. Freiberger, 465–484, 1972.

[17] D. R. Jeske and X. Zhang. Some Successful Approaches to Software Reliability Modelling in Industry. The Journal of Systems and Software, 74:85–99, 2005.

[18] P. C. Jha, D. Gupta, B. Yang and P. K. Kapur. Optimal testing resource allocation during module testing consider- ing cost, testing effort and reliability. Journal of Computers & Industrial Engineering, 57:1122–1130, 2009.

[19] J. Juran and F. Gryna. Quality Control Handbook, 4th edition. McGraw-Hill, 1988.

[20] P. K. Kapur and R. B. Garg. Cost-reliability Optimum Re- lease Policies for a Software System Under Penalty Cost. International Journal of Systems Science, 20(12):2547– 2562, 1989.

[21] Palak Raina, Hitali Shah. (2017). A New Transmission Scheme for MIMO - OFDM using V Blast Architecture.Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal, 6(1), 31–38. Retrieved from https://www.eduzonejournal.com/index.php/eiprmj/article/view/628

[22] Raina, Palak, and Hitali Shah."Security in Networks." International Journal of Business Management and Visuals, ISSN: 3006-2705 1.2 (2018): 30-48.

[23]

[24] P. K. Kapur and R. B. Garg. Optimal Software Release Policies for Software Growth Model Under Imperfect Debugging. Researche Operationelle/Operations Research (RAIRO), 24:295–305, 1990.

[25] P. K. Kapur and R. B. Garg. A Software Reliability Growth Model for Error Removal Phenomenon. Software Engineering Journal, 7:291–294, 1992.

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

[27] P. K. Kapur, V. B. Singh, S. Anand, and V. S. S. Yadavalli. Software Reliability Growth Model With Change-point and Effort Control Using a Power Function of the Testing Time. International Journal of Production Research, 46(3):771–787, 2008.

[28] P. K. Kapur, M. Xie, R. B. Garg, and A. K. Jha. A Discrete Software Reliability Growth Model With Testing Effort. 1st International Conference on Software Testing, Reliability and Quality Assurance, 1994.

[29] P. K. Kapur, S. Younes, and S. Agarwala. A General Discrete Software Reliability Growth Model. International Journal of Modelling and Simulation, 18(1):60–65, 1998.

[30] M. Kimura, T. Toyota, and S. Yamada. Economic Analysis of Software Release Problems with Warranty Cost and Re- liability Requirement. Reliability Engineering and System Safety, 66:49–55, 1999.

[31] Hitali Shah.(2017). Built-in Testing for Component-Based Software Development. International Journal of New Media Studies: International Peer Reviewed Scholarly Indexed Journal, 4(2), 104–107. Retrieved from https://ijnms.com/index.php/ijnms/article/view/259

[32] H. S. Koch and P. Kubat. Optimal Release Time of Computer Software. IEEE Transactions on Software En- gineering, SE-9:323–327, 1983.

[33] Y. W. Leung. Optimum Software Release Time with A Given Cost Budget. The Journal of Systems and Software, 17:233–242, 1992.

[34] K. D. Levin and O. Yadid. Optimal Release Time of Improved Versions of Software Packages. Information and Software Technology, 32(1):65–70, 1990.

[35] C. Lin and C. Huang. Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. The Journal of Systems and Software, 81:1025-1038, 2008.

[36] C. T. Liu and Y. C. Chang. A Reliability-constrained Soft- ware Release Policy Using A Non-Guassian Kalman Filter Model. Probability in the Engineering and Informational Sciences, 21:301–314, 2007.

[37] J. D. Musa, A. Iannino and K. Okumoto Software Reliabil- ity: Measurement, Prediction, Application. McGraw-Hill, Inc., 1987.K. Okumoto and A. L. Goel. Optimum Release Time for Software Systems Based on Reliability and Cost Criteria. The Journal of Systems and Software, 1:315–318, 1980.

[38] H. Pham and X. Zhang. A Software Cost Model with War- ranty and Risk Costs. IEEE Transactions on Computers, 48(1):71–75, 1999.

[39] H. Pham and X. Zhang. Software Release Policies With Gain in Reliability Justifying the Costs. Annals of Software Engineering, 8:147–166, 1999.

[40] H. Pham and X. Zhang. NHPP Software Reliability and Cost Models with Testing Coverage. European Journal of Operational Research, 145:443–454, 2003.

[41] J. G. Shanthikumar. Software Reliability Models: A Review. Microelectronics Reliability, 23:903–949, 1983.

[42] J. G. Shanthikumar and S. Tufekci. Application of A Software Reliability Model to Decide Software

Release Time. Microelectronics Reliability, 23(1):41–59, 1983.

[43] S. A. Slaughter, E. D. Harter, and M.S. Krishnan. Evalu- ating the Cost of Software Quality. Communication of the ACM, 41(8):67–73, 1998.

[44] G. Xia, P. Zeephongsekul, and S. Kumar. Optimal Software Release Policy With a Learning Factor for Imperfect Debugging. Microelectronics Reliability, 33:81–86, 1993.

[45] M. Xie. Software Reliability Modelling. World Scientific, Singapore, 1991.

[46] M. Xie and B. Yang. A Study of the Effect of Imperfect Debugging on Software Development Cost. IEEE Trans- actions on Software Engineering, 29(5):471–473, 2003.

[47] S. Yamada, J. Hishitani, and S. Osaki. Software-Reliability Growth with a Weibull Test-Effort: A Model & Appli- cation. IEEE Transactions on Reliability, 42(1):100–106, 1993.

[48] Raina, Palak, and Hitali Shah."Data-Intensive Computing on Grid Computing Environment." International Journal of Open Publication and Exploration (IJOPE), ISSN: 3006-2853, Volume 6, Issue 1, January-June, 2018.

[49] Hitali Shah."Millimeter-Wave Mobile Communication for 5G". International Journal of Transcontinental Discoveries, ISSN: 3006-628X, vol. 5, no. 1, July 2018, pp. 68-74, https://internationaljournals.org/index.php/ijtd/article/view/102.

[50] S. Yamada, H. Narihisa, and S. Osaki. Optimum Release Policies for A Software System With A Scheduled Soft- ware Delivery Time. International Journal of Systems Science, 15(8):905–914, 1984.

[51] S. Yamada and S. Osaki. Discrete Software Reliability Growth Models. Applied Stochastic Models and Data Analysis, 1:65–77, 1985.

[52] S. Yamada and S. Osaki. Optimal Software Release Poli- cies for A Non-Homogeneous Software Error Detection Rate Model. Microelectronics Reliability, 26(4):691–702, 1986.

[53] S. Yamada and S. Osaki. Optimal Software Release Policies With Simultaneous Cost and Reliability Require- ments. European Journal of Operational Research, 31:46– 51, 1987.

[54] B. Yang, H. Hu, and L. Jia. A Study of Uncertainty in Software Cost and its Impact on Optimal Software Re- lease Time. IEEE Transactions on Software Engineering, 34(6):813–825, 2008.

[55] M. C. K. Yang and A. Chao. Reliability-estimation and Stopping-rules for Software Testing, Based on Repeated Appearance of Bugs. IEEE Transactions on Reliability, 44(2):315–321, 1995.

[56] P. Zeephongsekul, C. Xia, and S. Kumar. A Software Reliability Growth Model Primary Errors Generating Sec- ondary Errors under Imperfect Debugging. IEEE Transac- tions on Reliability, R-43(3):408–413, 1994.