Int J Syst Assur Eng Manag (June 2023) 14(3):878–893
https://doi.org/10.1007/s13198-023-01887-3

ORIGINAL ARTICLE

Investigating the impact of effort slippages in software development project

Rajat Arora[1] · Rubina Mittal[2] · Anu Gupta Aggarwal[1] · P. K. Kapur[3]

Received: 27 November 2022 / Revised: 4 February 2023 / Accepted: 22 February 2023 / Published online: 1 April 2023

Abstract In today's highly competitive market, it is very crucial for the software development team to provide on-time delivery of its product. To achieve this, development teams spend considerable time in planning the schedule for software product incorporating stated requirements but practically they are persistently plagued by schedule slippage. The objective of this study is to inspect the progress of the software project and to examine its schedule status on regular basis. The present paper integrates the concept of slippage and management evaluation into an effort-based Software Reliability Growth Model incorporating the application characteristics such as complexity of code, testing environment etc. The study also investigates the optimal release policies for the software. Our research assumes that the review process is scheduled by the management team during testing. This crucial evaluation assist in providing critical information regarding the additional effort required to meet the reliability objective within scheduled time or by keeping the effort expenditure fixed, reschedule the delivery of the project. For theoretical validation of results, numerical illustration is presented on a real-life software fault dataset under perfect debugging and imperfect debugging environment. The results obtained are beneficial in decision making for both the development team and the managers. Our study has relevance in wide range of industries handling diverse projects.

Keywords Software Reliability Growth Model · Testing effort · Software development project · Penalty cost · Schedule slippage · Management evaluation · Optimal release policy

✉ Anu Gupta Aggarwal
anuagg17@gmail.com

Rajat Arora
arorarajat87@yahoo.com

Rubina Mittal
drmittal@keshav.du.ac.in

P. K. Kapur
pkkapur1@gmail.com

[1] Department of Operational Research, University of Delhi, Delhi 110007, India

[2] Keshav Mahavidyalaya, University of Delhi, Delhi 110034, India

[3] Amity Center for Interdisciplinary Research, Amity University, Noida, Uttar Pradesh 201301, India

1 Introduction and literature overview

1.1 Motivation

In last four decades, numerous research studies have been carried out in areas of Software reliability engineering for reliability assessment of software applications, their cost modeling, optimal release planning under perfect and imperfect debugging environment, resource allocation to name a few (Li and Pham 2017; Verma et al. 2020). But most of these studies are based on the assumption of on-time delivery of the software. This implies each stage of software development is covered as per planned schedule by utilizing only the planned effort expenditure in terms of budget, execution hours, man hours etc. However, in real life situation, this assumption sounds very unrealistic as many factors during development of software product do not allow the process to move in a planned way. This may be attributed to code complexity, unexpected additional time taken for debugging, optimistic schedule estimates, unforeseen issues with technology, changing requirements and specifications, environment mismatch and the list is endless. The present

research aims to investigate the ongoing progress of the project through regular review of the development process and to identify the direction in which rework needs to be done to avoid slippage in time and resource schedules. Adherence to time schedules may be achieved by spending additional effort whereas resource schedules may be saved either by extending the duration of project or revising the quality aspirations. To understand the requirement for additional efforts or the additional time requirements to avoid slippage in development process, we make use of an analytical model integrating management evaluation and development process to design future policies and make release decisions. The software failure process is described through NHPP based SRGM incorporating application characteristics.

1.2 Literature overview

1.2.1 Testing effort based SRGMs

The growing use of software in diverse fields necessitates continual improvement in technologies by introducing innovative features as per demand. SRGMs are used to describe the failure and fault removal phenomenon of a software application. These models provide appropriate estimates about the fault content level, fault removal rate, reliability levels achieved, time to stop debugging etc. [Kapur et al. (2011a, b)]. The initial models describing reliability growth assuming homogeneity of faults were proposed by Goel and Okumoto (1979), Jelinski and Moranda (1972). Later, Yamada et al. 1984 introduced an inflexion S-shaped SRGM assuming the dependent nature of faults. Pachauri et al. (2015) introduced S-shaped Fault Reduction Factor (FRF) for multi-release software systems. Chatterjee and Shukla (2016) proposed SRGM under imperfect debugging incorporating concept of fault dependency and fault reduction factor. Aggarwal et al. (2019) proposed the dual concept of FRF and error generation in SRGM for multi-release software models. The earlier SRGMs assumed constant rate of consumption of testing effort. Later, it was realized that the consumption pattern of resources is not uniform throughout and it was imperative to track the progress in reliability of the software system with respect to testing effort expenditure.

Various models have been proposed in the reliability literature integrating the dynamic theories of testing and debugging. The relationship between testing time, effort spent and number of faults discovered was exploited in SRGMs proposed by Yamada et al. (1993) and Huang et al. (2007) to name a few. Inoue and Yamada (2018) discussed in detail the testing effort expending problems. With time, research in testing effort dependent SRGM incorporated concept of fault reduction factor (Arora and Aggarwal 2020; Verma and Anand 2020). Kapur et al. (2019) proposed joint release and testing stop time policy with testing effort and change point. Verma et al. (2022) proposed a unified framework for software reliability assessment and release policy in presence of fault reduction factor and fault removal efficiency.

During testing and debugging of the software, the goal of team is to identify, isolate and remove maximum number of faults. But this process is not always perfect. Instances occur when some faults remain hidden in the software while some other faults are introduced during debugging process. This scenario is termed as imperfect debugging. Many researchers have proposed SRGM under imperfect debugging scenario. Ohba and Chou (1989) proposed an exponential SRGM with a fixed rate of error generation. Recent researches considering imperfect debugging include two phase SRGM with fault dependency and imperfect debugging (Zhu and Pham 2018). Kumar and Sahni (2020) presented a software model in which errors are corrected at prespecified debugging times. These SRGM's have practical utility in cost optimization and determining optimal release policies for the software product.

1.2.2 Software release planning model

The developers aim to minimize development costs and gain advantage over its competitors by making first entry into the market whereas users demand faster delivery, affordable, and reliable product. This requires trade-off between conflicting objectives of users and developers. The performance of the system during operational phase is largely dependent on time and effort spent during testing. Larger the effort spent during testing, better is the performance. Effort based release policy was discussed by Majumdar et al. (2019).The cost of fixing a bug during testing is much less than fixing an error during operation. Kapur et al. (2021) studied whether testing be continued after release of software. But, on the contrary, large time spent on testing cause project slippage and incurs extra cost. Here, we will club together these conflicting goals into a cost model and determine the optimum release time and optimum testing duration.

Various cost components constituting the total development cost of the software include cost of testing, cost of detecting and removing faults before release and post-release and market opportunity cost. The cost of testing is directly proportional to its duration. The opportunity cost is a penalty for not delivering the software product on time. Lai et al. (2011) studied the effect of imperfect debugging on development cost. Kapur and Garg (1989) incorporated the risk cost for late delivery. Practically, the constraints and objectives on software release are decided by the management depending on the prevailing market conditions, available resources and other competitive factors. Cao et al. (2020) proposed a continuous time stochastic control approach for optimal selection and release problem in software testing process.

1.2.3 Software development project

Efficient management of resources and time are the key prerequisites for successful development of a software product. Software projects frequently suffer from unanticipated modifications, reworks and subsequent delays. Project scheduling is a typical work for the managers and plays an important role in managing the development process in order to meet the deadlines and budget (Iqbal and Shahzad 2006). Later, Chen and Shen (2016) formulated a multi-objective project scheduling problem incorporating uncertainties during the development. The scheduling process integrates various tasks viz. identification of activities, their dependencies, estimate resources to be allocated to the constituent activities so as to meet the required objectives of project subject to given constraints (Luna et al. 2014; Minku et al. 2013). Few factors affecting consumption of resources and schedule planning include learning rate of team, project development environment, testing environment and scheduled deadline. Padberg (2006) did his work in determining optimal schedules for sample software projects and studied how the project characteristics have influence on optimal schedule decisions.

A project is characterized by fixed duration, finite resources, and uniqueness and is evaluated by its performance and on-time delivery. SDP are assemblies of large programs with various interactions and dependencies. The presence of uniqueness element in each project gives rise to uncertainty. Also there is a vast difference between planned and actual progress primarily due to continuously changing requirements, dynamic market conditions. Changes in planned design impact the resource consumption directly in terms of CPU time and memory consumed (Seacord 2014). Besides these phases, other factors affecting SDP includes complexity and size measured in terms of lines of codes. The development of software is a complex process due to the presence of uncertainty involved in inputs, environment and estimates. Hence the team usually is unable to meet the deadlines. This makes it mandatory for management to track and review the progress of the project on regular basis and communicate their evaluation report to development team for meeting the project deadlines. The project manager and the technical team work consistently to face the challenges that the project entails.

Mtsweni and Maveterra (2018) presented a report on various issues affecting application of tacit knowledge in SDP. Subriadi et al. (2019) developed a cost model for SDP. Akgün (2020) introduced the concept of team wisdom in SDP and its impact on project performance.

1.2.4 Slippage and rescheduling

In organizations concerned with the development of project, slippage may be defined as the act of missing the scheduled deadline.

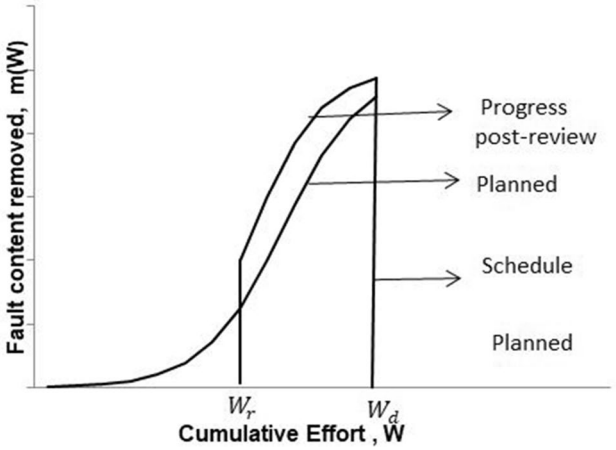

Fig. 1 S-curve incorporating management evaluation

Slippage management needs to be properly integrated with different phases of SDP. S-curves are graphical tools that are used to track the progress of the projects and update them accordingly by adjusting the effort utilization. The relation between effort spent and fault content removed is depicted by S-curve in Fig. 1.

In this paper, an algorithm for overcoming project slippage is presented. It makes use of rescheduling the work in progress by estimating additional amount of resources required in case the software project lags behind the planned development schedule.

Here, we present two possible solutions to tackle slippage. Let us consider a situation where software is tested for time $T_r(\textit{time of review}) < T_d(\textit{scheduled time of delivery})$. At this time, the testing efforts consumed and the reliability level achieved by testing team are analyzed to review the progress of the project. If the debugging process is inefficient and slow and in case the manager, after review, is not satisfied with the progress of the testing then, the testing process needs to be accelerated by putting in extra efforts in terms of CPU execution time, man-hours and skilled personnel. At this stage, we need to determine how much surplus efforts are needed to achieve the stated level of reliability in a pre-stated time interval. On the other hand, if the availability of testing effort is fixed and can't be increased, the delivery of software product is rescheduled with a risk that competitors may offer early release and prospective revenue is lost. The purpose of review is to safeguard on-time delivery (and target reliability) by working out additional resources, if required or to reschedule the delivery keeping the effort consumption at the same rate.

1.2.5 Significance of management evaluation

In real life scenario, for testing of the software product, managers are given targets for the desired reliability and bug content to be corrected within the specified scheduled time. The managers are also faced with other challenges to be dealt with during testing namely fierce market competition, competitive pressures, satisfactory level of quality, changing requirements of customer. But due to constraints on cost and time, it is very difficult to continuously inspect the progress. Therefore, reviews are done at certain time points during testing phase. Evaluation of the project is done by the management to note down the flaws that might have crept in at any stage of development process. Based on the review comments, suggestions are incorporated and rescheduling is done, if needed. Kluender et al. (2017) studied the relevance of team meetings on regular basis for software development. Kabeyi (2019) discussed in detail the significance of project monitoring and evaluation by the management team. Recently, the mediating role of Big data analytics between project performance and management was studied by Mangla et al. (2021). Managerial Reviews at regular intervals during testing are mandatory to deal with the stated challenges and for efficiently managing, tracking and gauging the progress of testing.

These reviews are helpful in making the development process as time, effort and cost efficient, by tackling the flaws at early stage, ensuring quality and incorporating required changes before final delivery of the product.

1.2.6 Novelty in our proposed model

In this paper, a software reliability growth model incorporating application characteristics, modeled by power function of effort expenditure is used. The SRGM considers practical aspects viz. error generation and internal characteristics of software such as complexity of code, testing environment etc. The process of introduction of errors during fault removal process in testing phase is referred to as error generation. This is one of the significant factors affecting reliability growth and is extensively used by researchers (Kapur et al. 2011a, b). In this study, we have compared models under perfect and imperfect debugging environment with constant rate of error generation.

In real life, the progress of the project does not confirm with the planned schedule because of various real life challenges leading to schedule slippage or effort slippage. This gives rise to the need for regular assessment of the progress of project by management team to inspect whether the project is moving as per expectations within given time frame and resources. The suggestions and review of management team further assist developers to take actions accordingly to meet the desired objective. In our study, we will elaborate on the concept of slippage and management evaluation taken to counter this critical problem faced by software industry. We focus our attention on the significance of managerial review on tracking the projects recurrently by the inspection team and giving their inputs to manager. This paper proposes a rescheduling model incorporating feedback for proper resource management for addressing troubles and unexpected events faced during SDP. Incorporating the idea of management evaluation into optimization model aids in making decisions regarding additional effort required to meet reliability objective within stipulated time or rescheduling the project delivery time. Using the model we further elaborate on economic effects by investigating diminishing returns to the testing time and efforts employed.

1.2.7 Research questions

Q1. What is the significance of management evaluation in achieving SDP goals?

Q2. To what extent SDP can deviate from initial plan with respect to cost and schedule?

Q3. To analyze the returns to scale for reliability improvement with respect to effort consumption?

Q4. What are theoretical and managerial implications of increased resources or delayed delivery?

The remainder of the paper is structured as follows. The assumptions and notations used in the suggested model are discussed in Sect. 2. The comprehensive optimization model for release policy based on effort based SRGM is presented in Sect. 3. Section 4 discusses in detail the rescheduling model employed by the management team followed by a numerical illustration in Sect.5. Managerial and theoretical implications are presented in Sect. 6. Next, we discussed threats to validity in Sect. 7. Section 8 concludes the paper followed by limitations and scope for future research in Sect. 9.

2 Model development

Basic assumptions and notations are described in the following subsections.

2.1 Assumptions (for SRGM model incorporating application characteristics)

The model considered in this study is based on the following set of assumptions.

(1) The occurrence of faults in the software, its corresponding correction and detection is modeled by NHPP.
(2) A variable quantifying application characteristics of the software is incorporated explicitly.

(3) The software project is subject to random failure due to presence of hidden faults.
(4) The expected number of faults detected by small amount of testing effort *dw* is proportional to the number of remaining faults.
(5) The model takes into consideration the dependent nature of faults implying exclusion of faults will result in exclusion of various other faults.
(6) Fault once detected is instantaneously removed without any delay.
(7) The operational phase is included in the lifespan of software.
(8) The environment during testing and operational phase is identical.

2.2 Notations

The following notations are used for the SRGM:

Notations	Meaning
a	The fault content in the software application at the outset
$m(W(t))orm(W)$	Expected fault content observed till time t utilizing effort W
$b(W)$	Fault detection rate with respect to effort W
q	Rate at which left over errors are noted
s	Variable measuring application characteristics
W(t) or W	Testing effort consumption by time t
k	Constant
g	Rate of error generation in case of imperfect debugging and is constant

2.3 Mathematical model

This section presents SRGM integrating application characteristics of the software which will be further used for our slippage analysis.

2.3.1 Weibull testing effort model

Since there is always a limit to the amount of resources available for testing, it is reasonable to assume that instantaneous testing effort expenditure is proportional to amount of efforts available to be expended at that time (Yamada et al. 1993).

In our study, the effort utilization is modeled using Weibull curve. The advantage of using Weibull function over others is that it is flexible in nature and can adapt itself to several effort consumption data. It models very nicely to the initial rise and then subsequent decay in effort consumption behavior. The testing effort expended up to time t characterized by Weibull curve is given in equation 1 as follows:

$$W(t) = \overline{W} \times \left(1 - e^{-\alpha t^c}\right) \tag{1}$$

where, $\overline{W}$: effort availability, c, α : shape and scale parameters of Weibull effort function;$\alpha > 0,\ c > 0$

Also, the instantaneous rate of effort consumption is:

$$\frac{d}{dt}W(t) = \overline{W}\alpha c t^{c-1} e^{-\alpha t^c} \tag{2}$$

2.3.2 Testing effort dependent SRGM

Here, we will discuss the proposed model under perfect and imperfect debugging environment.

Case 1. Perfect debugging

Under perfect debugging environment, all the detected faults are removed with certainty without additional errors being introduced in the system. Under this scenario, the rate of change of mean value function with respect to testing effort expended is specified by the following differential equation:

$$\frac{dm(W(t))/dt}{dW(t)/dt} = s(W(t)) * \left(p(a - m(W(t))) + q\frac{m(W)}{a}(a - m(W(t)))\right)$$

For the sake of simplicity of expressions, we will be writing $W(t)$ as W.

So above differential equation may be written as

$$\frac{dm(W)}{dW} = s(W) * \left(p(a - m(W)) + q\frac{m(W)}{a}(a - m(W))\right) \tag{3}$$

Here, the factor $s(W)$ is testing effort dependent function that quantifies the influence of software characteristics namely code size, factors related to development and debugging environment, application type etc.

Taking $s(W)$ as power function of effort represented as

$$s(W) = sW^k \tag{4}$$

On solving (3) using boundary condition $m(0) = 0 and W(0) = 0, we get$:

$$m(W) = a\left[\frac{1 - e^{-(p+q)\frac{sW^{k+1}}{k+1}}}{1 + \frac{q}{p}e^{-(p+q)\frac{sW^{k+1}}{k+1}}}\right] \tag{5}$$

Case 2. Imperfect debugging with error generation

Under imperfect debugging with error generation, each detected fault is removed with certainty but removal may lead to introduction of additional errors in the system. Under this setting, we assume that faults get introduced at a fixed

rate denoted by 'g' and is proportional to the expected number of faults removed with testing effort W. Therefore the fault content is no longer constant but is a function of testing effort W and it may increase with more effort expenditure owing to error introduction. Huang et al. (2000) proposed testing effort based SRGM with constant rate of error generation. Here,

$$a(W) = a + g * m(W) \tag{6}$$

In this case, the rate of change of mean value function with respect to testing effort expended is modified and the equation thus obtained is not solvable. Therefore we will use alternate expression derived in following steps.

Alternatively,

The Fault detection rate can be written as:

$$b(W) = \frac{\frac{d}{dw}(m(W))}{a - m(W)} \tag{7}$$

$$b(W) = \frac{p(p+q)sW^k}{p + qe^{\frac{-(p+q)sW^{k+1}}{k+1}}} \tag{8}$$

$$\frac{dm(W)}{dw} = b(W) * ((a(W) - m(W))) \tag{9}$$

$$= b(W) * ((a + (g-1) * m(W))) \tag{10}$$

Substituting expression for $(b(W))$ from (8) in (10) and solving (10) using boundary condition $m(0) = 0$ and $W(0) = 0$, we get :

$$m(W) = \frac{a}{(1-g)}\left(1 - \left(\left(\frac{p+q}{pe^{\frac{(p+q)sW^{k+1}}{k+1}} + q}\right)^{(1-g)}\right)\right) \tag{11}$$

3 Cost model

The quality and performance of the software product to a great extent, is influenced by time and effort spent during testing. There should be optimum trade-off between testing cost and cost incurred during operational phase. Before delivering the product to the end-users, a critical decision from economic point of view has to be taken whether to stop testing or to continue it.

Any adjustments in the schedule will have subsequent impact on costs. If the schedule is delayed, penalties are faced by developer. This penalty cost is directly proportional and rises exponentially with delivery time.

3.1 Assumptions for cost model

(1) The cost of fixing a fault during testing remains constant throughout the testing period.
(2) The cost of fixing a bug post-release remains constant throughout the operational phase i.e., there is no effect of inflation on costs.
(3) Cost of testing varies linearly with effort spent.
(4) Penalty cost is incurred for not delivering the product as per schedule. This cost includes market opportunity cost, goodwill loss etc.
(5) Penalty cost is a function of the time for which the product is delayed.

3.2 Additional notations for cost model

C_1: Testing cost per unit effort.

C_2: Cost of fixing unit fault during testing phase of SDP.

C_3: Cost of fixing unit fault after the product is released $(C_3 > C_2)$.

C_p: Penalty cost per unit delay for not delivering the software on time.

R_0: Target reliability.

T: Testing duration.

T_d: The scheduled time for delivery of software.

$W^* = W(T)$: Effort expenditure during testing.

$W_d = W(T_d)$: The amount of testing effort spent by scheduled delivery time.

TEC: Total Expected Cost.

I_W : Indicator function defined as:

$$I_W = \begin{cases} 1; & W \geq W_d \\ 0; & otherwise \end{cases}$$

3.3 Cost model formulation

In our research framework, we will consider the cost model incorporating the following four components:

(1) Cost of testing which varies directly with the testing effort expenditure W. It is presumed that testing cost is a linear function of testing effort W. It can be represented as:

$$\text{Expected testing cost } = C_1 W^*$$

(2) Cost of detecting and removing faults during testing phase.

The expected cost of debugging errors using effort W during software testing is given as:

$$\text{Expected cost of fault removal during testing} = C_2 m(W^*)$$

(3) Cost of detecting and removing faults during operational phase.

The expected cost of debugging errors using effort W during operational phase is given as:

$$\text{Expected cost of fault removal post release} = C_3(a - m(W^*))$$

(4) Penalty cost/opportunity cost/risk cost due to delayed delivery of software product.

This cost may be due to several reasons like competitors updated product launch, the software product may become obsolete to name a few.

$$\text{Penalty cost due to product slippage} = I_W C_p\left(W^* - W_d\right)^2$$

The unknown parameters of Mean value function m (W) are obtained using least square methods on SPSS.

The total expected cost as a function of testing effort W can be obtained by adding the above four components and is represented as:

$$TEC_1(W^*) = C_1 m(W^*) + C_2(a - m(W^*)) + C_3 W^* + I_W C_p\left(W^* - W_d\right)^2 \quad (12)$$

In the following section, we will be presenting constrained and unconstrained optimization models under both Perfect and Imperfect Debugging environment and determine the optimal release policy.

4 Optimization model for release policy

Here, the issue of deciding the best time when testing can be stopped and the software can be delivered to end-users for operational use is taken into consideration. This decision is influenced by various factors such as failure phenomenon and performance criteria used for evaluating the system readiness (Kapur et al. 1999). We address this problem of determining optimal release by considering the effort dependent fault detection rate model of Kapur and Garg incorporating application characteristics based on the criterion of expected cost. An optimum release policy for unconstrained and constrained problem is derived on the basis of reliability criterion and sensitivity analysis is deliberated for the model parameters. The results are separately obtained under perfect and imperfect debugging environment. The outcomes are demonstrated using numerical examples.

Optimal release policy based on reliability criterion

The release policy for our model will be discussed in following two cases:

Case 1: Perfect debugging environment

Taking the expression of $m(W)$ from Eq. (5)

Subcase 1: Unconstrained optimization

$$Minimize\ TEC_1(W^*);\ W \geq 0 \text{ by using Eq.(12)} \quad (13)$$

Subcase 2: Constrained optimization

$$Minimize\ TEC_1(W^*)$$

subjectto

$$R(x/W) \geq R_O \quad (14)$$

$$W \geq 0 \quad (15)$$

where, $R(x/W)$ denotes the reliability of the software and is given by

$$e^{(-m(W+x)-m(W))} \quad (16)$$

Case2: Imperfect debugging environment

Taking expression of $m(W)$ from Eq. (11),

Subcase 1: Unconstrained optimization

$$MinimizeTEC_2(W^*) = C_1 m(W) + C_2(a - (1-g)m(W)) + C_3 W + I_W C_p\left(W - W_d\right)^2;\ W \geq 0 \quad (17)$$

(using equations (6)and (12))

Subcase 2: Constrained optimization

$$Minimize\ TEC_2(W^*) = C_1 m(W) + C_2(a - (1-g)m(W)) + C_3 W + I_W C_p\left(W - W_d\right)^2$$

(using equations (6)and (12))

subjectto

$$R(x/W) \geq R_O \quad (18)$$

$$W \geq 0 \quad (19)$$

5 Rescheduling model

5.1 Notations

In this subsection, we will present the notations used in the rescheduling modeling framework.

W Effort expenditure.

T_r Time point where evaluation by management took place.

W_r Effort expended by time T_r, when the review was done.

T_d Scheduled delivery time pre-decided by management.

W_d Effort utilization till the scheduled delivery of the software.

R_0 Reliability level to be achieved at time of scheduled delivery.

5.2 Assumptions

- The failure phenomenon in SRGM is built on Non-Homogeneous Poisson process.
- The failure phenomenon during testing depends on the remaining fault content and faults identified by current effort level at that time.
- SRGM takes into account factor considering application characteristics represented by power function.
- There is no time lag between detection and correction of faults.
- While removing the faults causing failure, some additional faults are also removed.
- Undetected faults in the software have influence on failure rate.
- In case of imperfect debugging, constant rate of errors are introduced into the system at fixed rate. The no. of faults introduced are directly related to already detected faults ($a + g * m(W)$) by utilizing effort W.

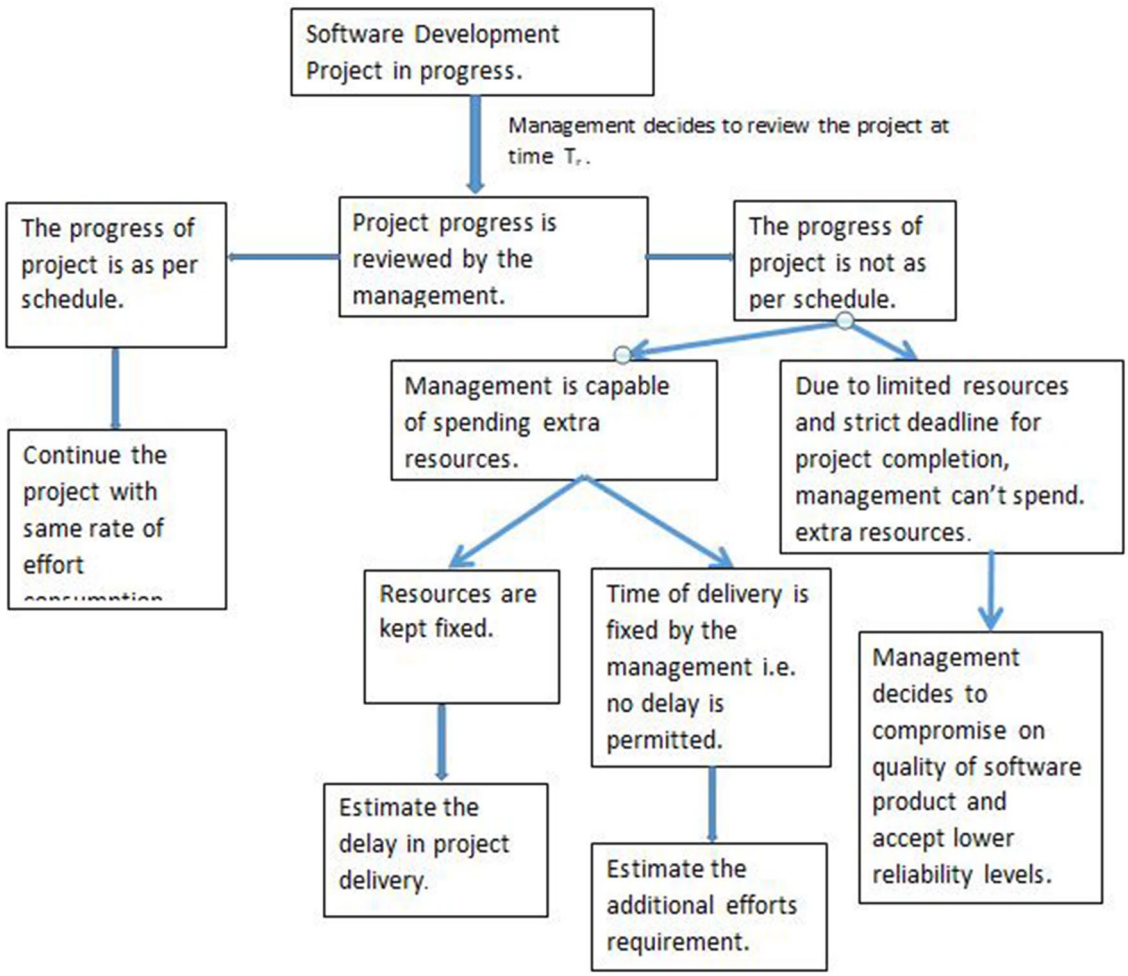

Fig. 2 Flow-diagram depicting methodology followed by management team

Table 1 Description of dataset used

Dataset	Reference	Description	Testing time (Weeks)	Execution time(CPU hours)	Faults
DS-1	Wood (1996)	Tandem computers	20	10,000	100

Table 2 Estimation results for Weibull TEF (14 weeks data)

Parameter estimation	DS [Wood (1996)]
$\overline{W}$	9306.64
α	0.019
c	1.715

- The review is conducted by the management team during testing phase at time T_r.
- Management has pre-decided the scheduled delivery of the software project at time T_d by spending W_d effort.

Figure 2 below demonstrates the flowchart corresponding to the methodology followed by the management team.

5.3 Model for evaluation of testing progress by management team

Before the beginning of software development, the management team fixes the scheduled delivery time based on the feasibility and client's need. Besides that, when testing has been done for a considerable time period, the management evaluates the progress of the project and presents its observations and suggestions to development team to avoid schedule slippage.

6 Numerical illustration

Let's consider a situation where testing has already been done for time T_r (time of review) using effort W_r, and the scheduled delivery time set by management team is T_d. At time T_r, the review is conducted by the management team to track the progress of testing. The failure data is considered for the period $(0, T_r)$ and the Weibull testing effort function is estimated by non-linear regression technique on SPSS. Further using this data, parameters of the proposed SRGM are estimated under perfect and imperfect debugging environment. After parameters are estimated for the proposed model using failure data for time T_r, the additional effort requirement for the period (T_r, T_d) is computed and in the situation of uniform effort rate, the probable delay is examined.

Table 3 Estimated parameters for the two models

Models for comparison	Estimated values of parameters					
	a	p	q	s	k	g
Perfect debugging	100	0.023	0.016	0.008	0.001	–
Imperfect debugging	93	0.215	0.001	0.001	0.001	0.264

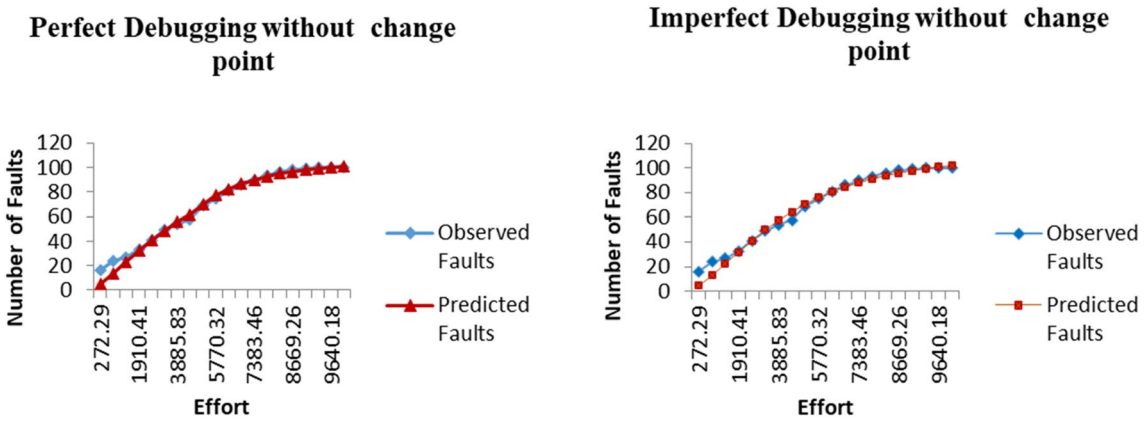

Fig. 3 Goodness of fit curves for the two model

6.1 Estimation of Weibull TEF (data taken till time of review)

For the dataset by Wood (1996) given in Table 1, the time of review is set to 14 weeks i.e., $T_r = 14$. The parameters of effort function are estimated using non-linear regression technique and are presented in Table 2.

6.2 Model validation and performance criteria used

For evaluating the performance of the suggested SRGM on the basis of predicted Weibull effort function, the parameters for the mean value function given by expressions (5) and (11) are estimated. The results obtained through least square estimation on SPSS are provided in Table 3. Also, the goodness of fit curves are presented in Fig. 3.

Performance criteria used are R square, Mean Square Error (MSE), Root mean square error (RMSE), Predictive Ratio risk (PRR) and Predictive power (PP). For both the datasets low MSE and high R^2 indicates a good fit. The results of various performance measures for the two models are shown in Table 4.

6.3 Optimization results

In order to determine the optimal release policy, we will find the optimal level of effort consumption and corresponding minimum cost in the following steps:

Step 1. Consider the optimization problem formulated in sub-Sect. 3.4 and determine the release policy by first ignoring the penalty cost for delayed delivery (by taking $I_W = 0$ in total expected cost function).

Step 2. In the second step include the penalty cost for delayed delivery in the objective function of the optimization problem (by taking $I_W = 1$ in total expected cost function). The results of optimization problem are provided in Table 5. In case of perfect debugging, we have assumed the values of cost coefficients as $C_1 = 60$, $C_2 = 100$, $C_3 = 1500$ and $C_P = 30$. The target reliability level is set at 0.90 and scheduled effort is taken as 10,000 CPU hours. In imperfect debugging environment, the cost coefficients are assumed as $C_1 = 90$, $C_2 = 120$, $C_3 = 2500$ and $C_P = 60$. The cost coefficients in imperfect debugging are greater than perfect debugging owing to the error generation as the introduction of new faults require extra cost for their removal. Also, the research studies in literature have incorporated cost coefficients in the similar manner (Kapur et al. 2008; Verma et al. 2019). Using these estimated parameters, cost coefficients and reliability goal, we obtained the optimal results for unconstrained and constrained optimization problem formulated using expressions (13–19) in Sect. 3.4. On solving the software release problem on Maple software, the results obtained are presented in Table 5 below.

6.4 Estimation of additional testing efforts

Assuming that for the considered dataset, the scheduled delivery is set at time $T_d = 20$ weeks at which point the testing terminates. The goal of management is to release the product at planned time achieving the target reliability level of 0.90. Taking the expression of reliability using Eq. (16), mean value function given by Eqs. (5) and (11) in Sect. 3 and estimated parameters from Table 3, the reliability achieved by time of review, T_r is estimated. If the efforts continue to

Table 4 Performance measures for the two models

Models for comparison	Performance measures				
	R^2	*MSE*	*RMSE*	*PRR*	*PP*
Perfect debugging	0.952	30.2869	5.5033	35.2929	15.362
Imperfect debugging	0.964	22.5573	4.7495	33.6176	18.9364

Table 5 Release policy optimization results

Model		Optimal effort consumption(in CPU Hours)	Minimum expected software cost (in INR)
Ignoring penalty cost	Perfect debugging_ Unconstrained	9863.772	1.453197×10^7
	Perfect debugging_ Constrained	10068.075	1.51083902×10^7
After adding penalty cost	Perfect debugging_ Unconstrained	9975.001	1.498754×10^7
	Perfect debugging_ Constrained	12337.820	1.8247497×10^8
Ignoring penalty cost	Imperfect Debugging_ Unconstrained	9875.18	2.469996×10^7
	Imperfect debugging_ Constrained	14106.607	3.52747×10^7
After adding penalty cost	Imperfect debugging_ Unconstrained	9979.168	2.4982094×10^7
	Imperfect debugging_ Constrained	14106.6075	1.0471278×10^9

Table 6 Additional effort requirement

Reliability = $R(x/W)$	Effort required, W	Additional effort requirement from time of review	Additional effort requirement per unit increase in reliability
Perfect debugging			
0.86	8839.2227	1177.316	–
0.89	9721.2374	2059.331	314.3924
0.92	10843.0439	3181.137	410.2655
0.95	12440.6575	4778.751	613.389
0.98	15449.1974	7787.291	1319.6133
Imperfect debugging			
0.76	8137.3719	–	–
0.79	9086.5997	949.2278	328.4855
0.8	9428.795	1291.4231	342.1953
0.84	10967.7024	2830.3305	414.2513
0.88	12902.1307	4764.7588	533.5974
0.92	15563.6502	7426.2783	766.9879
0.94	17420.7707	9283.3988	992.5146

Table 7 Estimation of probable delay in software delivery

R_O (aspired level of reliability to be achieved by time T_d)	$W^{\#}$ additional effort requirement from time of review T_r)	Probable delay using Weibull function (in weeks)
Perfect debugging		
0.86	0	0
0.89	882.0147	2.6270
0.92	2003.8212	4.4141
0.95	3601.4348	6.6476
0.98	6609.9747	11.4249
Imperfect debugging		
0.76	0	0
0.79	949.2278	2.7482
0.84	2830.3305	5.5813
0.88	4764.7588	8.3087
0.92	7426.2783	13.2601
0.94	9283.3988	28.6458

be expended at the same existing rate till scheduled delivery time,T_d, then the reliability level attained at the release time is estimated. But if management is aspiring for higher reliability, then additional efforts need to be expended. In order to obtain the values of additional effort requirement, $W^{\#}$ to attain desired reliability target, we use the following algorithm. Tables 6 and 7 shows the estimated values of additional effort required corresponding to different levels of reliability.

Step-wise procedure for estimating requirement of additional efforts

Step1. Utilize failure dataset till effort expenditure W_r for estimating the parameters for the Reliability growth model presented in Sect. 3.2.

Step2. Assuming the same testing environment continues, use the above parameter values to estimate the level of reliability achieved by cumulative effort W_r by time T_r. Denote this by $R(T_r)$. In addition, compute the reliability level attained at the scheduled delivery time T_d. Denote this by $R(T_d)$.

Step 3. If the aspired level of reliability time T_d is R_O. Two cases arise:

Case 1: If $R_O < R\,(T_d)$, then nothing to worry about and with same rate of testing the product will be delivered by schedule delivery time.

Case 2: If $R_O > R\,(T_d)$, then there is a need to expedite testing process by employing more effort. Our objective is to estimate the additional testing efforts required in time interval $(T_r,\ T_d)$ in order to attain reliability level R_O at time T_d.

From the dataset used and estimated values of the parameters, we can compute $W^{\#}$ denoting the additional efforts required to avoid slippage, corresponding to different levels of reliability.

In the case of perfect debugging, the reliability level achieved at time of review T_r (after 14 weeks) is R (T_r) = 0.81.If the efforts are continued to be expended at the existing rate then, reliability level achieved at scheduled delivery time T_d is 0.86. If the target reliability level, R_O is greater than 0.86, then additional efforts need to be employed. Taking the value of increment, x as 50 in the definition of reliability given by Eq. (16), the additional effort required are computed on MAPLE software and the results are presented in Table 6 given below.

In the case of Imperfect debugging environment, the reliability level achieved at time of review T_r, (after 14 weeks) is 0.72. If the efforts are continued to be expended at the existing rate then, reliability level achieved at scheduled delivery time T_d is 0.76. If the target reliability level, R_O is greater than 0.76, than additional efforts need to be employed. Taking the value of increment,x as 50 in the definition of reliability given by Eq. (16), the additional effort required are computed on MAPLE software and the results are presented in Table 6 given below. From Table 6, it can be inferred that as the aspired reliability level increases, the effort requirement to achieve the additional unit of reliability also increases.That is, the marginal effort requirement increases with each additional reliability level achieved. This may be regarded as diminishing returns to effort consumption. This may be attributed to the fact that few latent faults are very hard to detect and require exceptionally more resources and time to detect and remove them.

Figure 4 below depict the total effort consumption with respect to reliability improvement estimated at time of review for perfect debugging and imperfect debugging respectively.

6.5 Estimation of probable delay if effort utilization is kept fixed at a uniform rate

Step 1. Determine the surplus efforts to be expended in time interval (T_r, T_d) to achieve level R_O as given by expression (16).

Step 2. Using the Weibull function of testing effort given in expression (1), the surplus effort can yield the corresponding time delay.

The estimated values of probable delay corresponding to different levels of reliability are presented in Table 7 for perfect and imperfect debugging environment respectively. Graphically, their curves are shown in Fig. 5.

Further, from Table 7 and corresponding Fig. 5, it can be observed that as the aspired reliability level increases, if the usage of the effort is kept at the same rate,then there is probable delay in the release of software project and this delay is an increasing function of aspired reliability level. Also the graphs clearly demonstrates the diminishing returns to scale that is, in order to achieve each successive unit of reliability,the delay in time is greater than the previous unit. This may be attributed to the fact that to improve the quality or reliability of the software after a certain level, large amount of resources in terms of manpower, time are needed to debug hard and complex faults.

7 Implications

7.1 Theoretical implications

This study yields various theoretical implications which have noteworthy influences in the field of software development. This paper combines SRGM with management evaluation to appropriately reschedule the development process to avoid schedule slippage. The marginal analysis of effort towards on-time delivery is presented which gives an idea to development team about trade-off between efforts and scheduled delivery of software with respect to reliability level attained during testing.

In Kapur and Garg model, we have taken a factor to incorporate application characteristics of the software which is more practical and suitable to study fault removal process. Moreover, throughout SDP, testing is very crucial and requires timely review for efficient utilization of time and resources. This facilitates developers to reschedule delivery or modify resource consumption to overcome slippage in software projects. Management evaluation are integral component of development standards for SDP specified by International organizations (Suryn et al. 2003). In our research we have studied managerial reviews during testing in SRGM incorporating application characteristics. The application characteristics incorporated in the SRGM assist in achieving the actual failure behavior of software project. For numerical illustration, we have taken failure dataset of Tandem computers (Wood 1996) which shows testing for 20 weeks in removing 100 faults consuming 10,000 CPU hours of effort. The parameters of SRGM are estimated on this dataset when testing has been done for 14 weeks. At this point of time, management evaluation was done. The fault content at this time showed that development process was running slow and needs to be taken care of, otherwise it may lead to opportunity loss and hence profit loss. Using this SRGM in perfect debugging environment, it was observed that 0.81 reliability level is attained in 14 weeks and if the development process continue at the same pace then, the reliability level achieved will be 0.86 end of testing phase i.e., 20 weeks. But, if we desire to achieve higher reliability level, then the process has to be expedited by expending more testing efforts or to postpone the release seeing around

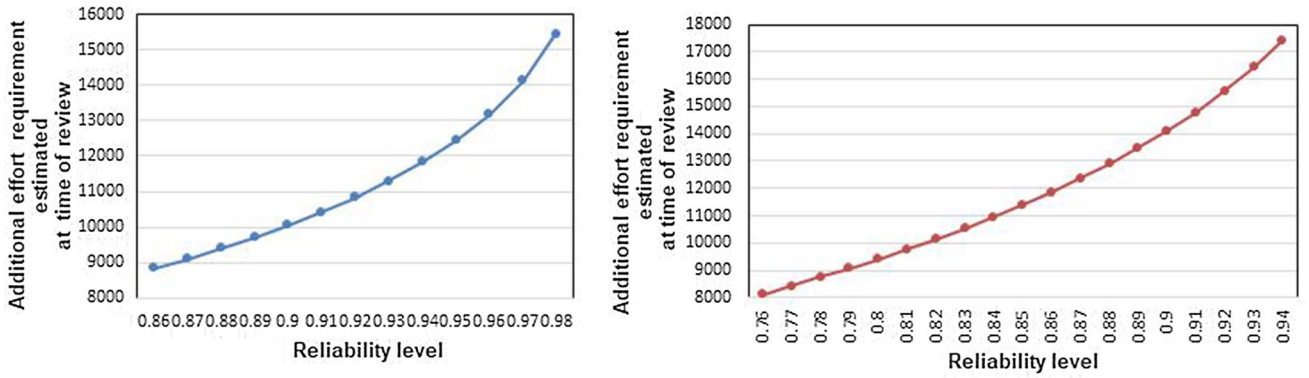

Fig. 4 Additional effort requirement

competitive environment. Similarly, in case of imperfect debugging, the determination of reliability attained at the time of review revealed that the development was lagging.

The development team must decide appropriately as delayed delivery may have serious consequences including goodwill loss (Verma et al. 2020). It is apparent from the graphs shown in Fig. 4 that unlimited testing and resources can't be employed to make software 100 percent bug free. The diminishing returns to effort and time employed show that after certain point of time, the results achieved are no more profitable. Cost–benefit analysis has to be done simultaneously during project development to determine optimal values of testing time and efforts. Different research studies are carried out to demonstrate diminishing returns in interdisciplinary fields viz. economic analysis (Jordan 2017), project management (Mahmoudi and Feylizadeh 2018) etc. Similar behavior is shown in software development projects. Therefore, SDP needs to be examined cautiously to guarantee optimal use of time and efforts.

7.2 Managerial implications

The outcomes of our present research have great impact on the actions taken by management team of a software organization. The observations that are resulted from outcomes of the study have significant implications for the management as well as development teams in software development organizations. The suggestions given by the management to the development team after examining the progress are valuable in identifying the bottlenecks in the process which may be the cause for the slippage in software projects. These flaws are initially identified and the actions are taken to correct them (Wang et al. 2008). The development process of software is characterized by S-shaped curve signifying that the fault isolation and removal is slow in the beginning and with learning of the testing team, it speeds up in the middle and again follows a slow pace towards the termination of its useful life. By incorporating management evaluation, the delay in the project can be controlled to match the planned one by adjusting the testing effort consumption. Taking care of the competitive conditions and client's urgency, the management team may decide to alter the scheduled delivery by keeping the same effort utilization. But this is not always the feasible option since it may lead to opportunity loss to developers and organization may incur loss due to obsolescence.

The present study assists the managers to take fundamental decision concerning scheduled delivery and effort consumption as the testing evolves. The time for assessing the progress of ongoing project is set during the later stage of testing and before the final delivery. If the progress doesn't match the planned one then either the effort consumption is increased or the scheduled delivery is shifted but each decision has its own pros and cons. Increasing the effort consumption is feasible only if efforts are available and higher costs are affordable, on the other hand delayed delivery is possible only to the extent that it doesn't cause opportunity loss and discontented clients. It is one of the major concerns of management to decide about the optimal testing duration. Figure 4 show the improvement in reliability with respect to efforts and testing duration respectively.

8 Threats to validity

Several types of threats to validity have been described in previous studies depending on the nature of research. Here, we will discuss the threats to validity in our present study. In our study, the Software Reliability Growth model is developed to discuss the fault removal phenomenon under combined effect of effort expenditure and time assuming constant fault detection rate (FDR) but in real life, it may change with time because of learning phenomenon and change in severity of faults. In optimization, we have considered criteria corresponding to cost and reliability but in real-life, other key criteria like code coverage and functional coverage may influence the release planning. Also, in our schedule

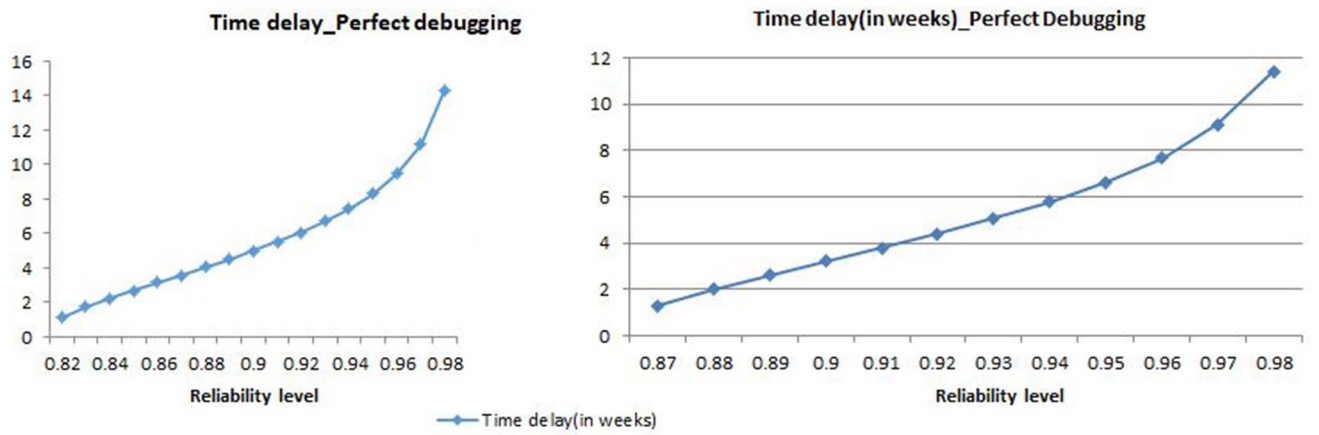

Fig. 5 Probable delay

planning, it has been assumed that the project will continue at the current pace only but practically there are numerous factors related to the testing conditions and environment which may change in the post-review period. The threats to validity include the environmental influences comprising administrative and human factors on the testing process which may not be taken care of by the model parameters.

Last, but not the least, the model has been validated on a single real failure dataset.The threats to external validity may be minimized by validating the model on other datasets.

9 Conclusion

In our research, Kapur and Garg model incorporating testing effort function and factor for application characteristics has been assessed under both perfect and imperfect debugging environment. The SRGM takes into consideration the practical aspects faced by the development team. This works on the assumption that hazard rate is dependent on both the remaining fault content and proportion of faults discovered. Furthermore, the incorporation of constant for error generation improved the performance of model by yielding lower MSE and higher R square. Also, the major contribution of present research is the significance of management evaluation during the later duration of testing phase, when the scheduled delivery of the software product is approaching. The assessment is done to track the progress of the project and analyze additional effort requirement to avoid slippage or to quantify the probable delay if effort consumption remains the same. This worked as a beneficial tool for the manager to plan and workout the requirement for surplus efforts during testing so that the software product is delivered as per schedule. The analysis in this paper provides awareness about current level of development during testing and aids in estimating additional efforts required to achieve the goal. It facilitates the software development organization in retaining its goodwill and clientage, beating the intense competition by delivering the reliable software product on time. The present research demonstrates the additional effort (keeping scheduled delivery time fixed) and possible slippage (keeping effort consumption fixed) in Tables 6 and 7 respectively. Furthermore, the corresponding Figs. 4 and 5 reveal the diminishing returns to efforts employed.

This study addresses the process of rescheduling in SDP. Organizations are confronted with several risks and uncertainties such as unanticipated modification, rework and delays. These have great impact on feasibility and optimality of schedules and thus encourage rescheduling. This paper proposes a procedure to handle slippage. This practice assists the developer in choosing an appropriate response by evaluating the impact on feasibility of schedule and effort available for rescheduling. The process contributes to traditional scheduling frames, models and supports the sensible choice and use of rescheduling approaches in development process.

10 Limitations and future scope

Each research has its limitations which are necessary to be highlighted. This provides route for further research. In our present study, the dynamicity of projects is not taken into consideration as they need to undergo continuous updates and adapt itself to the environment they are exposed to. This may be extended to multi-release scenario. In case of complex software projects, testing usually takes longer than expected and there is possibility that FDR get changed due to change in testing environment, changes in skilled personnel,

number of test cases, new testing tools and techniques, changing strategies etc. In this scenario, the proposed model can be extended to incorporate single or multiple change points.

Also we have considered a case where review is done at one point of time during the later phase of testing. This can be extended to multiple review points during testing.

Acknowledgments This research work was supported by FRP grant received from Institution of Eminence, University of Delhi, India (Ref. No./IoE/2021/12/FRP).

Declarations

Conflict of interest The authors declared that they have no competing interests and have no conflicts of interest among them.

Informed consent All the authors have approved the manuscript and agree with its submission to the International Journal of System Assurance Engineering and Management.

Human and animals rights Human participants and/or animals are not involved in this proposed work.

References

Aggarwal AG, Gandhi N, Verma V, Tandon A (2019) Multi-release software reliability growth assessment: an approach incorporating fault reduction factor and imperfect debugging. Int J Math Oper Res 15(4):446–463

Akgün AE (2020) Team wisdom in software development projects and its impact on project performance. Int J Inf Manag 50:228–243

Arora R, Aggarwal AG (2020) Testing effort based software reliability assessment incorporating FRF and change point. Yugosl J Oper Res 30:273–288

Cao P, Yang K, Liu K (2020) Optimal selection and release problem in software testing process: a continuous time stochastic control approach. Eur J Oper Res 285(1):211–222

Chatterjee S, Shukla A (2016) Modeling and analysis of software fault detection and correction process through Weibull-type fault reduction factor, change point and imperfect debugging. Arab J Sci Eng 41(12):5009–5025

Chen R-M, Shen Y-M (2016) Dynamic search control-based particle swarm optimization for project scheduling problems. Adv Mech Eng 8(4):1687814016641837

Goel AL, Okumoto K (1979) Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Trans Reliab 28(3):206–211

Huang C-Y, Kuo S-Y, Lyu MR (2007) An assessment of testing-effort dependent software reliability growth models. IEEE Trans Reliab 56(2):198–211

Huang C-Y, Kuo S-Y, Lyu MR (2000) Effort-index-based software reliability growth models and performance assessment. In: Paper presented at the proceedings 24th annual international computer software and applications conference. COMPSAC2000

Inoue S, Yamada S (2018) Optimal software testing effort expending problems. System reliability management. CRC Press, Boca Raton, pp 51–64

Iqbal J, Shahzad B (2006) Iterative project scheduling-a time bound technique. In: Paper presented at the 2006 international conference on computing & informatics

Jelinski Z, Moranda P (1972) Software reliability research. Statistical computer performance evaluation. Academic Press, Amsterdam, pp 465–484

Jordan LP (2017) Introduction: understanding migrants' economic precarity in global cities. Urban Geograph 38(10):1455–1458

Kabeyi MJB (2019) Evolution of project management, monitoring and evaluation, with historical events and projects that have shaped the development of project management as a profession. Int J Sci Res 8(12):63–79

Kapur P, Garg R (1989) Cost–reliability optimum release policies for a software system under penalty cost. Int J Syst Sci 20(12):2547–2562

Kapur PK, Kumar S, Garg R (1999) Contributions to hardware and software reliability, vol 3. World Scientific Press, Singapore

Kapur P, Gupta D, Gupta A, Jha P (2008) Effect of introduction of fault and imperfect debugging on release time. Ratio Math 18(1):62–90

Kapur P, Pham H, Anand S, Yadav K (2011a) A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans Reliab 60(1):331–340

Kapur P, Pham H, Gupta A, Jha P (2011b) Software reliability assessment with OR applications. Springer, London

Kapur P, Panwar S, Singh O, Kumar V (2019) Joint release and testing stop time policy with testing-effort and change point. Risk based technologies. Springer, Singapore, pp 209–222

Kapur P, Panwar S, Kumar V (2021) Should software testing continue after release of a software: a new perspective. Handbook of advanced performability engineering. Springer, Cham, pp 709–737

Kluender J, Unger-Windeler C, Kortum F, Schneider K (2017) Team meetings and their relevance for the software development process over time. In: Paper presented at the 2017 43rd Euromicro conference on software engineering and advanced applications (SEAA)

Kumar V, Sahni R (2020) Dynamic testing resource allocation modeling for multi-release software using optimal control theory and genetic algorithm. Int J Qual Reliab Manag 37:1049–1069

Lai R, Garg M, Kapur P, Liu S (2011) A study of when to release a software product from the perspective of software reliability models. JSW 6(4):651–661

Li Q, Pham H (2017) NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. Appl Math Model 51:68–85

Luna F, González-Álvarez DL, Chicano F, Vega-Rodríguez MA (2014) The software project scheduling problem: a scalability analysis of multi-objective metaheuristics. Appl Soft Comput 15:136–148

Mahmoudi A, Feylizadeh MR (2018) A grey mathematical model for crashing of projects by considering time, cost, quality, risk and law of diminishing returns. Grey Syst: Theor Appl 8(3):272–294

Majumdar R, Kapur P, Khatri SK, Shrivastava A (2019) Effort-based software release and testing stop time decisions. Int J Reliab Saf 13(3):179–193

Mangla SK, Raut R, Narwane VS, Zhang Z, Priyadarshinee P (2021) Mediating effect of big data analytics on project performance of small and medium enterprises. J Enterp Inf Manag 34(1):168–198

Minku LL, Sudholt D, Yao X (2013) Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. IEEE Trans Softw Eng 40(1):83–102

Mtsweni ES, Maveterra N (2018) Issues affecting application of tacit knowledge within software development project. Procedia Comput Sci 138:843–850

Ohba M, Chou X-M (1989) Does imperfect debugging affect software reliability growth? In: Paper presented at the proceedings of the 11th international conference on software engineering

Pachauri B, Dhar J, Kumar A (2015) Incorporating inflection S-shaped fault reduction factor to enhance software reliability growth. Appl Math Model 39(5–6):1463–1469

Padberg F (2006) A study on optimal scheduling for software projects. Softw Process: Improv Pract 11(1):77–91

Seacord RC (2014) The CERT C coding standard: 98 rules for developing safe, reliable, and secure systems. Pearson Education

Subriadi AP, Muqtadiroh FA, Dewi RS (2019) A model of owner estimate cost for software development project in Indonesia. J Softw: Evol Process 31(10):e2175

Suryn W, Abran A, April A (2003) ISO/IEC SQuaRE: the second generation of standards for software product quality. Acta Press, Calgary

Verma V, Anand S (2020) Two-dimensional release policy for software systems incorporating FRF, opportunity cost and environment factor. ICDSMLA 2019. Springer, Singapore, pp 879–885

Verma V, Anand S, Aggarwal AG (2019) Software warranty cost optimization under imperfect debugging. Int J Qual Reliab Manag 37:1233–1257

Verma V, Neha N, Aggarwal AG (2020) Software release planning using grey wolf optimizer. Soft computing methods for system dependability. IGI Global, Hershey, PA, pp 1–44

Verma V, Anand S, Kapur P, Aggarwal AG (2022) Unified framework to assess software reliability and determine optimal release time in presence of fault reduction factor, error generation and fault removal efficiency. Int J Syst Assur Eng Manag 13(5):2429–2441

Wang ET, Ju P-H, Jiang JJ, Klein G (2008) The effects of change control and management review on software flexibility and project performance. Inf Manag 45(7):438–443

Wood A (1996) Predicting software reliability. Computer 29(11):69–77

Yamada S, Ohba M, Osaki S (1984) S-shaped software reliability growth models and their applications. IEEE Trans Reliab 33(4):289–292

Yamada S, Hishitani J, Osaki S (1993) Software-reliability growth with a Weibull test-effort: a model and application. IEEE Trans Reliab 42(1):100–106

Zhu M, Pham H (2018) A two-phase software reliability modeling involving with software fault dependency and imperfect fault removal. Comput Lang Syst Struct 53:27–42