Annals of Operations Research (2022) 312:305–332
https://doi.org/10.1007/s10479-019-03483-w

S.I.: STATISTICAL RELIABILITY MODELING AND OPTIMIZATION

Joint optimization of software time-to-market and testing duration using multi-attribute utility theory

P. K. Kapur[1] · Saurabh Panwar[2] · Ompal Singh[2] · Vivek Kumar[2]

Published online: 2 December 2019

Abstract
An optimal software release strategy is a well-investigated issue in software reliability literature. Comprehensive testing is expected before releasing the software into the market to enhance the reliability and security of the software device. In recent years, few analysts have recommended the scheme for software projects that support releasing the software early in the market and continue the testing process for an added period in the field environment even after the software is distributed. These studies are based on one common assumption that the efficiency of the software engineers in detecting the faults occurs at a consistent rate throughout the testing phase. However, bug-identification rate may experience discontinuity at the software release time. In software engineering, the time-point at which fault detection rate changes is termed as change-point. Consequently, an alternative software release policy is proposed in the present paper, which offers a generalized framework for fault detection phenomenon using the unified approach. An extensive analysis of software time-to-market and testing duration based on cost-efficiency and reliability measures is discussed by considering the change in tester's fault detection rate. A multi-criteria decision making technique known as multi-attribute utility theory is applied to optimize the software release policy under *field-testing* (FT) and *no field-testing* (NFT) frameworks. The relevance of the optimization problem is illustrated using a numerical example, comprising both the exponential and S-shaped bug-detection process.

Keywords Software reliability · Field-testing · Software reliability growth models (SRGMs) · Change-point · Multi-attribute utility theory (MAUT) · Testing termination point · Software distribution time · Bug-identification rate

✉ P. K. Kapur
pkkapur1@gmail.com

Saurabh Panwar
saurabhpanwar89@yahoo.com

Ompal Singh
drompalsingh1@gmail.com

Vivek Kumar
vivekrajput.du.aor@gmail.com

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

[2] Department of Operational Research, University of Delhi, Delhi, India

1 Introduction

Software products are the most pervasive human generated equipment, which affects our everyday activity. Software have become an indispensable part of enterprises and are applied in various technologies such as electrical gadgets, automobiles, aircraft, telecommunication services, home appliances, etc. Due to the growing dependency of our social system on software-based devices, there is a prerequisite for developing highly reliable, secure, and good quality software. Reliability of software is achieved by the continuous testing process before distributing the software. Testing process along with debugging of the software systems are the primary function of the Software Development Life Cycle (SDLC). The software companies spend about half of their overall development cost on software testing (Myers 1976). Therefore, testing of the software is acknowledged as one of the most substantial stages of SDLC. It assists in achieving the desired reliability and in developing robust and high-quality software.

Software testing identifies errors that are underlying dormant in the software during its development life cycle. Comprehensive testing removes these faults during the debugging process and can escalate the quality of a product, which increases the client's trust in the product. Software testing also cuts down maintenance expenses both for the users and for the analysts. Thus it results in a more reliable, dependable, and failure-free software (Kapur et al. 2011b). Although no software can be perfect and completely free from errors, the testing process can lower the risk of failure associated with the software by reducing the number of bugs present in the software. In addition, the detected faults can be utilized as means for feature improvement. Thus, testing can be viewed as a corrective and innovative aspect of SDLC. During the testing phase, not only faults are debugged, but also development cost and time-to-market of the software is kept in check (Subburaj and Kapur 2014). Therefore, extensive testing by software engineers is imperative to produce highly reliable software and ensure the delivery of the software on time.

In practice, it is highly troublesome for the developers to control the reliability and the quality of the software product. Therefore, the mathematical tools known as software reliability growth models (SRGMs) are rigorously used in software reliability engineering to measure the software reliability and to quantify the testing process (Yamada 2014). These models provide aid for enhancing the reliability of the software products. SRGMs facilitate software engineers to measure the fault levels, failure rates, mean time between failures (MTBF), and accuracy during the coding and verification process (Huang and Lyu 2011). SRGMs are based on the conjecture that the fault content in the software system is defined at the beginning of the testing process. Through testing and debugging processes, these faults are removed from the device, and the reliability of the system is improved at the end of the testing phase. Goel and Okumoto (1979) suggested the initial software reliability model. They made use of Non-homogeneous Poisson Process (NHPP) to analyze the reliability of the software. Their model is based on the assumption that fault detection curve is distributed exponentially over time. To describe the S-shaped growth of fault detection process, Yamada et al. (1983) proposed the modified SRGM to investigate S-shaped fault detection data. Soon after, a series of NHPP-based SRGMs have been designed for the assessment of reliability growth of software during software development handling (Musa and Okumoto 1983; Ohba and Yamada 1984; Yamada and Osaki 1985; Kapur and Garg 1992; Pham et al. 1999; Huang et al. 2003; Huang 2005b; Kapur et al. 2008a, b, 2011a, 2012; Inoue et al.

2016; Zhu and Pham 2018). NHPP models deal with the software failure process as an arbitrary process and fault-observation phenomenon as a random variable. These are widely accepted models due to their easy use and convenience in understanding.

A vast portion of the software literature also focuses on the software time-to-market (Kapur et al. 1999; Zhang and Pham 2002). Specifically, an appropriate time to release the software in the market remains a critical research topic for the software analysts. Testing the system for a protracted period may impede the software release time. Moreover, this may further result in excessively high development cost. On the other hand, releasing software very quickly without sufficient amount of testing may result in user's dissatisfaction, which ultimately affects the advancement of the software device and the company's reputation in the global market. Therefore, the software reliability and availability prediction hold vital importance to establish a trade-off between the quality of software product, development cost and release time during the software development lifecycle. Owing to the extreme usefulness of estimating the software time-to-market and testing duration, the study of optimal release policy has gained tremendous importance in the software literature. In the past, researchers have used conventional SRGMs to model the optimization problem for evaluating the most favorable software release time. The initial study was conducted by Okumoto and Goel (1980). They have formed the unconstrained optimization problems: cost minimization and reliability maximization based on the exponential SRGM. Later, Dalal and Mallows (1988) examined the optimal testing stop time rule, which is established by balancing the cost of continued testing and the probable losses due to the bugs that still exist in the distributed system. Afterward, Yamada and Osaki (1987) considered the constrained optimization problem to calculate the software time-to-market using different SRGMs. Kapur and Garg (1991) further discussed the release time policy using testing effort based SRGMs. They considered the maximization problem of the gain function under failure intensity constraint. After that, Huang (2005a), and Huang and Lyu (2005) determined the release time policies by assessing the testing effort expenditure. Subsequently, many other studies were carried out to obtain the optimal testing stop time and related release policies (Inoue and Yamada 2008; Chiu et al. 2009; Kapur et al. 2013, 2014; Singh et al. 2015; Minamino et al. 2016).

The underlying assumption of all the above-cited studies is that the testing process terminates at the release time of the software system. Although, the reliability aspiration level of the device governs the testing period but prolong testing without releasing the software may cause the excessive development cost and loss in market share due to the competitive environment. Therefore, testing stop time and software release time should be treated essentially different time points (Arora et al. 2006). The software should be released early to capture the market and testing must be continued for an added period until the definitive reliability level is attained. Keeping this aspect in mind, few researchers have examined the optimization problem by choosing testing termination time and software time-to-market as two separate decision variables (Jiang et al. 2012; Majumdar et al. 2017; Kapur et al. 2017). These studies facilitate the post-testing phenomenon wherein both the developers and the users identify the bugs in the software system. The post-release testing is also referred as field-testing in which users also address the problems they encounter and report it to the testing team. Testers, on the other hand, will forward software patches to the customers after they rectify the issue. Software companies are effectively using the user's bug reports to identify flaws and enhance the quality of the software system. Thus, customers can play a prominent part in accelerating the reliability and security of the software device.

The drawback of previous studies incorporating post-release testing is that they have considered the rate of failure-observation of the developers to be constant during the entire testing period. However, in the testing phase after the software is released the efficiency of the testing team may alter. When defective applications run in the field, many issues can emerge, such as, high maintenance cost, user's dissatisfaction, loss of firm's goodwill and eventually decline in market share. Therefore, to avoid failures at the user's end, testers update their testing strategy during the field-testing to intensify the fault detection process. To incorporate this realistic phenomenon, the present paper provides an alternative release time policy and optimal testing termination time using the concept of change-points. Change-point specifies the testing-time at which the software failure-occurrence pattern changes (Zhao 2003; Kapur et al. 2008a, b; Inoue et al. 2015; Tickoo et al. 2016). This change occurs owing to the change in the testing environment, e.g., modification in the testing-effort expenditures, variation in the fault target, the modification in resource allocation, and other random factors.

Another limitation of the previous studies on post-release testing is that their optimal policies are based only on the exponential SRGM. No other failure-occurrence functions such as S-shaped learning function are used. To address this limitation, the present paper proposes an optimal policy using generalized distribution function. To the best of our knowledge, the proposed study is the first attempt that jointly optimizes the software release time and testing stop time for S-shaped bug-detection distribution functions also. An NHPP based unification scheme has been applied to estimate the reliability and availability of the software system. Furthermore, a release policy with *field-testing* (FT) is developed that explicitly considers the phenomenon where developers continue to test the software after its release for some specific period. Joint optimization of software time-to-market and testing stop time is further solved using multi-criterion decision-making (MCDM) technique known as multi-attribute utility theory (MAUT). The results of the problem are compared with the release policy with *no field-testing* (NFT). This is a conventional release time policy wherein in-house testing continues until the required reliability aspiration level is achieved. After that, the testing process terminates, and software is delivered in the market.

The remainder of the paper is organized as follows. In Sect. 2, the modeling framework of fault occurrence phenomenon and various costs are formulated along with various assumptions and notations on which the model is based. Following that, two optimal release policies, specifically release policy with *field-testing* (FT) and with *no field-testing* (NFT) are suggested in Sect. 3. Also, multi-criterion decision-making technique, known as MAUT is applied to solve the optimization problem, which is also explained in Sect. 3. In Sect. 4, the findings from the current research are illustrated using a numerical example, and in Sect. 5, sensitivity analysis is further evaluated to understand the effects of the critical parameters on the optimal results. Finally, concluding remarks along with managerial implications and direction for future studies is detailed in Sect. 6.

2 Model development

This section describes the reliability growth modeling using unification scheme. A generalized SRGM fault classification model is developed to measure the number of faults identified during the software product lifecycle. The fault detection process is developed under two scenarios. According to first scenario, software release time and testing duration are considered two distinct time points. Second, when testing release time and testing stop time coincides. Figure 1 pictorially describes the software lifecycle under two different

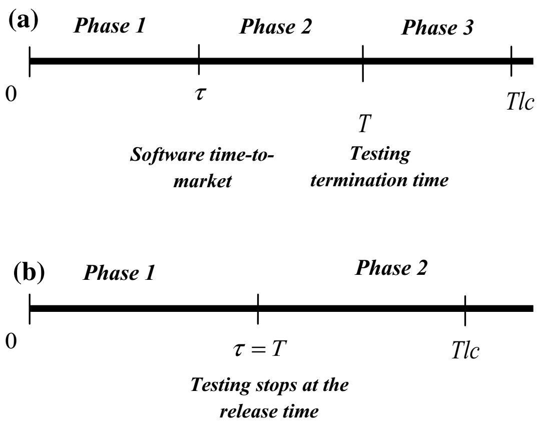

Fig. 1 Different phases of failure occurrence phenomenon under **a** *field-testing* release policy and **b** *no field-testing* release policy

scenarios. Figure 1a represent the scenario when software lifecycle is divided into three phases: system testing phase or pre-release testing phase, field-testing phase or post-release testing phase, and post-testing phase (after testing termination). In addition, Fig. 1b depicts the situation when software lifecycle comprises of two phases: testing phase and operational phase.

2.1 Notations

a	Total number of faults present in the software system before testing process
$m_{t_1}(t)$	Expected number of faults observed by time t during pre-release testing phase $[0,\ \tau)$
$m_{t_2}(t-\tau)$	Expected number of faults identified by testers during field-testing phase $[\tau,\ T)$
$m_{u_1}(t-\tau)$	Expected number of faults observed by users during field-testing phase $[\tau,\ T)$
$m_{u_2}(t-T)$	Expected number of faults discovered by users during post-testing phase $[T,\ T_{lc}]$
τ	Software time-to-market, which also acts as a change-point for tester's fault discovery rate
T	Testing duration of the software product; $T > \tau$
T_{lc}	The total lifecycle of the software product
τ_{FT}	Optimal software release time under *field-testing* release policy
T_{FT}	Optimal software testing duration under *field-testing* release policy
τ_{NFT}	Optimal software time-to-market under *no field-testing* release policy
$F_{t_1}(t)$	The cumulative fault detection distribution function of testers before change-point τ
$F_{t_2}(t)$	The cumulative fault detection distribution function of testers after change-point τ
$F_u(t-\tau)$	The cumulative fault discovery distribution function of users
C_1	Testing cost per unit time during the testing duration
C_2	Market opportunity cost depending on the software release time
C_3	Debugging cost of a single fault during pre-release testing phase (before change-point)
C_4	Debugging cost of a fault detected by the tester during field-testing phase (after change-point)
C_5	Debugging cost of a fault detected by users during field-testing
C_6	Debugging cost of a fault detected by users during the post-testing period

2.2 Assumptions

The growth models for fault detection phenomenon are based upon the following assumptions:

1. Debugging process is modeled using a non-homogenous poison process (NHPP).
2. Every failure in the software is independently and identically distributed over the life-cycle of the system with probability distribution $F(t) = P(X \leq t) = \int f(x)dx$, where $P(\cdot)$ denotes the cumulative probability of failure occurrence.
3. Faults causing failure are removed immediately, i.e., the fault correction time is negligible.
4. The amount of initial faults in the software is finite and fixed.
5. When the fault is identified, it is removed perfectly without generating any additional faults.
6. During field-testing, some portion of the faults lying dormant in the system is observed by the developers, and the users discover the remaining faults.
7. After detecting the faults, customers immediately report it to the testing team. Testers then send a patch to users once they rectify the problem.
8. Cost of providing a patch to the customers in the form of the software update is considered trivial.

2.3 Modeling framework under *field-testing* (FT) release time policy

2.3.1 Phase 1: System-testing or pre-release testing period $[0, \tau)$

During the system-testing period, developers comprehensively debug faults to release a reliable and dependable software device in the market. The expected number of faults observed at any time is directly proportional to the number of faults remaining in the software device at that time. Mathematically, the differential equation for fault identification at any instant of time is given as:

$$\frac{dm_{t_1}(t)}{dt} = \frac{f_{t_1}(t)}{1 - F_{t_1}(t)}\left(a - m_{t_1}(t)\right) \tag{1}$$

where $\frac{f_{t_1}(t)}{1-F_{t_1}(t)}$ is the hazard rate describing the conditional probability of fault detection at time t given that no failure is occurred due to the respective fault before time t; $\left(a - m_{t_1}(t)\right)$ signifies the remaining number of faults to be removed at time t.

Equation (1) can be further solved using the initial condition at $t = 0$, $m_{t_1}(t) = 0$, to obtain the expected number of cumulative faults detected by time t:

$$m_{t_1}(t) = aF_{t_1}(t); \quad 0 \leq t < \tau \tag{2}$$

where a is the initial fault content in the software. Besides, faults are immediately removed as soon as they are identified.

From the past software error detection data, it has been established that the growth curve of fault discovery process follows either concave or S-shaped pattern. Thus, there are two principal categories of modeling software reliability growth process: exponential and S-shaped functions. Consequently, in software engineering literature, researchers have developed different SRGMs by incorporating various testing and operational aspects to depict these two growth curves accurately. In the present study, different non-decreasing functions can be applied to model the fault-detection process. However, the current research utilizes three distinguished and well-acknowledged SRGMs to demonstrate the mean value function for phase 1. The concave fault-identification function is described using Goel–Okumoto exponential intensity function (Goel and Okumoto 1979), and for expressing an S-shaped learning process, Yamada delayed S-shaped (Yamada et al. 1983) and Kapur–Garg logistic distribution (Kapur et al. 2011b) is applied.

- *SRGM with exponential distribution function* (Goel and Okumoto 1979)
  When NHPP based failure-observation follows an exponential distribution function, i.e.$F_{t_1}(t) = (1 - e^{-b_1 t})$, then the mean value function of fault identification in phase 1 takes the following form:
  $$m_{t_1}(t) = a\left(1 - e^{-b_1 t}\right) \tag{3}$$
  where a is the total faults at the start of the testing period and b_1 is the fault identification parameter during system testing (before change-point).
- *SRGM with delayed S-shaped distribution function* (Yamada et al. 1983)
  The NHPP model with delayed S-shaped growth curve for fault identification process with distribution function $F_{t_1}(t) = \left(1 - (1 + b_1 t)e^{-b_1 t}\right)$ is expressed as:
  $$m_{t_1}(t) = a\left(1 - (1 + b_1 t)e^{-b_1 t}\right) \tag{4}$$
  where b_1 is the rate parameter denoting the tester's error detection rate before change-point τ.
- SRGM with logistic distribution function (Kapur et al. 2011b)
  Considering error detection phenomenon as an NHPP with logistic learning function i.e. $F_{t_1}(t) = \left(\frac{1-e^{-b_1 t}}{1+\beta_1 e^{-b_1 t}}\right)$, the expected fault detection during system testing becomes:
  $$m_{t_1}(t) = a\left(\frac{1 - e^{-b_1 t}}{1 + \beta_1 e^{-b_1 t}}\right) \tag{5}$$
  where b_1 is the fault detection parameter and β_1 is the learning parameter during system testing (before change-point).

2.3.2 Phase 2: Field-testing or post-release testing period $[\tau, T]$

Although the software is released in the market at the time τ, the testing of the software continues for an added period to enhance the quality of the software product. During this phase, both developers and customers identify the faults remaining in the system. Moreover, it is believed that the testing team modifies the testing efforts to debug the software faults with more intensity after its release, which changes the fault detection rate of the testers. Therefore, τ also represents the change-point for the developers debugging process.

Now, after phase 1, $aF_{t_1}(\tau)$ faults have been removed by the testing team from the software. Thus, the remaining faults after testing period is $a - aF_{t_1}(\tau)$ (or $a(1 - F_{t_1}(\tau))$). Clearly, $a(1 - F_{t_1}(\tau))$ represents the total faults at the beginning of phase 2 (field-testing). Now, out of these undetected faults, it is assumed that a fixed proportion say 'λ' will be detected by the testers and remaining (1 – λ) will be identified by the users who then immediately report it to the developers for correcting it. Therefore, the expected number of faults discovered by the testers at time t during the field-testing period is given by:

$$\frac{dm_{t_2}(t-\tau)}{dt}=\frac{f_{t_2}(t)}{1-F_{t_2}(t)}\left(\lambda a(1-F_{t_1}(\tau))-m_{t_2}(t-\tau)\right);\ \ \tau<t\le T \tag{6}$$

where $\frac{f_{t_2}(t)}{1-F_{t_2}(t)}$ is the tester's fault detection rate after change-point τ. On further solving the Eq. (6) by applying initial condition i.e. at $t=\tau,\ \ m_{t_2}(t-\tau)=0$, the expected number of faults detected by the testing team during phase 2 becomes:

$$m_{t_2}(t-\tau)=\lambda a\left(1-F_{t_1}(\tau)\right)\left[1-\frac{\left(1-F_{t_2}(t)\right)}{\left(1-F_{t_2}(\tau)\right)}\right];\ \ \ \tau<t\le T \tag{7}$$

Equation (7) represents the expected bugs detected by the testing team in time duration $(t-\tau),\ \ \tau\le t<T$, i.e. during post-release testing period.

Now, as the software enters the market, users also contribute in bug identification. Thus, the expected number of bugs identified by the users of the software system at any instant of time t during field-testing is:

$$\frac{dm_{u_1}(t-\tau)}{d(t-\tau)}=\frac{f_u(t-\tau)}{1-F_u(t-\tau)}\left((1-\lambda)a(1-F_{t_1}(\tau))-m_{u_1}(t-\tau)\right);\ \ \ \tau<t\le T \tag{8}$$

where $\frac{f_u(t-\tau)}{1-F_u(t-\tau)}$ is hazard rate that denotes the combined fault identification rate of the users. Note that the user's origin point will be τ. On further evaluating the Eq. (8) under the initial condition that at $t=\tau,\ \ m_{u_1}(t-\tau)=0,\ \ F_u(t-\tau)=0$, following analytical solution is obtained:

$$m_{u_1}(t-\tau)=(1-\lambda)a\left(1-F_{t_1}(\tau)\right)F_u(t-\tau);\ \ \tau<t\le T \tag{9}$$

where $m_{u_1}(t-\tau)$ express the expected number of bugs detected by the user during field-testing.

- SRGM with exponential distribution function

  When fault observation phenomenon occurs with exponentially decreasing rate, then the mean value function of fault identification for testers and users in phase 2 takes the following respective form:

$$m_{t_2}(t-\tau)=\lambda ae^{-b_1\tau}\left(1-e^{-b_2(t-\tau)}\right);\ \ \ \tau<t\le T \tag{10}$$

  where b_2 is the fault detection parameter for the testers during the field-testing (after change-point)

$$m_{u_1}(t-\tau)=(1-\lambda)ae^{-b_1\tau}\left(1-e^{-b_3(t-\tau)}\right);\ \ \ \tau<t\le T \tag{11}$$

  where b_3 is the fault detection parameter for the users.

- SRGM with delayed S-shaped distribution function

  The expected number of bugs identified by the testers and users respectively when fault observation phenomenon follows delayed S-shaped curve is given as:

$$m_{t_2}(t-\tau) = \lambda a(1+b_1\tau)e^{-b_1\tau}\left[1-\left(\frac{1+b_2t}{1+b_2\tau}\right)e^{-b_2(t-\tau)}\right]; \quad \tau < t \le T \tag{12}$$

where b_2 is the fault detection parameter for the testers during the field-testing (after change-point)

$$m_{u_1}(t-\tau) = (1-\lambda)a(1+b_1\tau)e^{-b_1\tau}\left(1-(1+b_3(t-\tau))e^{-b_3(t-\tau)}\right); \quad \tau < t \le T \tag{13}$$

where b_3 is the users combined fault detection rate.

- SRGM with logistic distribution function
  When error identification follows logistic learning function, then mean value function for fault detection by the developers and users will be:

$$m_{t_2}(t-\tau) = \lambda a\left(\frac{(1+\beta_1)e^{-b_1\tau}}{1+\beta_1e^{-b_1\tau}}\right)\left[1-\left(\frac{1+\beta_2e^{-b_2\tau}}{1+\beta_2e^{-b_2t}}\right)e^{-b_2(t-\tau)}\right]; \quad \tau < t \le T \tag{14}$$

where b_2 is the fault detection parameter and β_2 is the learning parameter for the testers during the field-testing (after change-point)

$$m_{u_1}(t-\tau) = (1-\lambda)a\left(\frac{(1+\beta_1)e^{-b_1\tau}}{1+\beta_1e^{-b_1\tau}}\right)\left(\frac{1-e^{-b_3(t-\tau)}}{1+\beta_3e^{-b_3(t-\tau)}}\right); \quad \tau < t \le T \tag{15}$$

where b_3 is the fault detection parameter and β_3 is the learning parameter for the customers.

2.3.3 Phase 3: Post-testing period $[T,\ T_{lc}]$

Software developers have to stop testing the system after a certain time, say T, when the desired level of reliability has been attained. However, during this phase, customers can encounter failure due to the bugs remaining in the system that was not identified in the previous phases. Users will report the fault to the developers, who then rectify the bug and send the patch to the users. This process of fault removal, therefore, continues until the end of software's lifecycle.

Let the remaining number of faults in the software that were undetected in previous phases be Z_a Then, $Z_a = a - m_{t_1}(\tau) - m_{t_2}(T-\tau) - m_{u_1}(T-\tau)$, i.e.

$$Z_a = a(1-F_{t_1}(\tau))\left(1-\lambda\left\{1-\frac{\left(1-F_{t_2}(T)\right)}{\left(1-F_{t_2}(\tau)\right)}\right\}-(1-\lambda)F_u(T-\tau)\right) \tag{16}$$

Thus, the instantaneous rate of fault detection by the users in phase 3 is given as:

$$\frac{dm_{u_2}(t-T)}{d(t-\tau)} = \frac{f_u(t-\tau)}{1-F_u(t-\tau)}\left(Z_a - m_{u_2}(t-T)\right); \quad T < t \le T_{lc} \tag{17}$$

Equation (17) can be further solved under the condition, $t = T,\ m_{u_2}(t-T) = 0$ to obtain the following solution:

$$m_{u_2}(t-T) = Z_a\left[1-\left(\frac{1-F_u(t-\tau)}{1-F_u(T-\tau)}\right)\right]; \quad T < t \le T_{lc} \tag{18}$$

After substituting the value of Z_a in Eq. (18), the expected number of faults observed during this phase becomes:

$$\begin{aligned} m_{u_2}(t-T) =& a(1-F_{t_1}(\tau))\left(1-\lambda\left\{1-\frac{\left(1-F_{t_2}(T)\right)}{\left(1-F_{t_2}(\tau)\right)}\right\}-(1-\lambda)F_u(T-\tau)\right) \\ & \left[1-\left(\frac{1-F_u(t-\tau)}{1-F_u(T-\tau)}\right)\right] \end{aligned} \tag{19}$$

Equation (19) represents the expected bugs removed by the testers in interval $[T,\ T_{lc}]$

- SRGM with exponential distribution function
  When fault observation phenomenon occurs with exponentially decreasing rate, then the mean value function of fault identification for the users in phase 3 takes the following respective form:

$$m_{u_2}(t-T) = ae^{-b_1\tau}\left(1-\lambda\left(1-e^{-b_2(T-\tau)}\right)-(1-\lambda)\left(1-e^{-b_3(T-\tau)}\right)\right)\left(1-e^{-b_3(t-T)}\right); \quad T < t \le T_{lc}. \tag{20}$$

- SRGM with delayed S-shaped distribution function
  When fault observation phenomenon occurs with exponentially decreasing rate, then the mean value function of fault identification for the users in phase 3 takes the following respective form:

$$\begin{aligned} m_{u_2}(t-T) =& a(1+b_1\tau)e^{-b_1\tau}\begin{pmatrix} 1-\lambda\left[1-\left(\dfrac{1+b_2T}{1+b_2\tau}\right)e^{-b_2(T-\tau)}\right] \\ -(1-\lambda)\left(1-(1+b_3(T-\tau))e^{-b_3(T-\tau)}\right) \end{pmatrix} \\ & \left(1-\left(\frac{1+b_3(t-\tau)}{1+b_3(T-\tau)}\right)e^{-b_3(t-T)}\right); \quad T < t \le T_{lc}. \end{aligned} \tag{21}$$

- SRGM with logistic distribution function
  When error identification follows logistic learning function, then mean value function of fault detection by the users becomes:

$$\begin{aligned} m_{u_2}(t-T) =& a\left(\frac{(1+\beta_1)e^{-b_1\tau}}{1+\beta_1e^{-b_1\tau}}\right)\begin{pmatrix} 1-\lambda\left[1-\left(\dfrac{1+\beta_2e^{-b_2\tau}}{1+\beta_2e^{-b_2T}}\right)e^{-b_2(T-\tau)}\right] \\ -(1-\lambda)\left(\dfrac{1-e^{-b_3(T-\tau)}}{1+\beta_3e^{-b_3(T-\tau)}}\right) \end{pmatrix} \\ & \left(1-\left(\frac{1+\beta_3e^{-b_3(T-\tau)}}{1+\beta_3e^{-b_3(t-\tau)}}\right)e^{-b_3(t-T)}\right); \quad T < t \le T_{lc}. \end{aligned} \tag{22}$$

2.4 Modeling framework under *no field-testing* (NFT) release time policy

In the traditional modeling framework, the software lifecycle is separated into two phases, specifically, testing phase (before software release time) and operational phase (after release time). In such scenario, the release time and testing duration coincides, i.e. $\tau = T$. Therefore, faults are detected only by the testers before release time and it is considered that all the remaining faults after the software release is detected and reported by users, which are then removed by the testing team. In the testing phase $[0,\ \tau)$ the expected number of faults detected by the testers is given as:

$$m_{t_1}(t) = aF_{t_1}(t); \ \ 0 \le t < \tau \tag{23}$$

As in the operational phase $[\tau,\ T_{lc}]$, all the remaining faults will be detected by the users, therefore, the mean value function of faults identified during this period will become:

$$m_u(t-\tau) = a\big(1 - F_{t_1}(\tau)\big)F_u(t-\tau); \quad \tau \le t \le T_{lc}. \tag{24}$$

2.5 Cost modeling

To perform joint optimization of the software distribution time and testing termination time, the following cost components are involved:

(a) Testing cost

Testing cost includes the efforts required to perform and execute the testing process. According to the software engineering literature, the cost of testing linearly increases with the testing duration (Pham and Zhang 1999). Therefore, if C_1 is the testing cost per unit time, then the overall testing cost for the entire testing duration is given as:

$$C_{testing}(t) = C_1 T \tag{25}$$

(b) Market opportunity cost

The delay in software release may result in tangible and intangible losses to the firm. These losses can be represented in the form of market opportunity cost. The opportunity cost is an essential attribute to utilize limited resources such as time, labor, etc. efficiently. This cost is assumed to expand non-linearly with an increase in release time. This is because the delay in software distribution will raise the possibility that competitors may manipulate the market. Specifically, the opportunity cost function is presumed a power law form, which is comparatively flexible in representing the dependence of cost over time (Chiu et al. 2009). As time in releasing software increases, the opportunity cost also increases because of more loss of commercial opportunities. Therefore, the following power law form is considered to model opportunity cost function:

$$C_{market\_opp}(t) = C_2\tau^{\gamma} \tag{26}$$

where γ is a parameter denoting the degree of opportunity loss in time and its value can be achieved from the assessment of experts and prior empirical studies.

In software engineering literature, a quadratic function is viewed as the most straightforward and appropriate functional form of opportunity cost. The study proposed by Jiang and Sarkar (2003), Chiu et al. (2009), Jiang et al. (2012), and Yamada and Yamaguchi (2016) have used quadratic function to describe market opportunity cost. Therefore, to depict a benchmark scenario, the value of parameter γ is drawn upon from the past literature. Thus, the opportunity cost is assumed a quadratic function of release time and therefore, the overall opportunity cost becomes:

$$C_{market\_opp}(t) = C_2\tau^2. \tag{27}$$

(c) Faults debugging cost during pre-release testing period

This cost comprises of the efforts required by the testing team to handle failures. In software reliability literature, it is assumed to depend linearly on the expected number of bugs identified during this phase (Okumoto and Goel 1980). Therefore, the error-debugging cost in phase 1 will be:

$$C_{PhaseI}(t) = C_3 m_{t_1}(\tau). \tag{28}$$

(d) Faults debugging cost during field-testing period

Debugging cost during field-testing is segregated into two components. Firstly, the cost of debugging of the faults detected by the testers and secondly, the cost involved in rectifying the errors identified by the users. This cost is considered as a linear function of the number of observed faults. Thus, the total cost of debugging during phase 2 will be:

$$C_{PhaseII}(t) = C_4 m_{t_2}(T-\tau) + C_5 m_{u_1}(T-\tau). \tag{29}$$

(e) Faults debugging cost post-testing period

This cost includes the debugging cost in operational phase when testing has terminated. During this phase, faults reported by the users are corrected by the developers. This cost depends on the expected number of faults identified by the customers in phase 3. Hence, the faults debugging cost during this phase will be:

$$C_{PhaseIII}(t) = C_6 m_{u_2}(T_{lc} - T). \tag{30}$$

Thus, the overall cost function is given as:

$$\begin{aligned} C(\tau_{FT}, T_{FT}) = {} & C_1 T_{FT} + C_2\tau_{FT}^2 + C_3 m_{t_1}(\tau_{FT}) + C_4 m_{t_2}(T_{FT}-\tau_{FT}) + C_5 m_{u_1}(T_{FT}-\tau_{FT}) \\ & + C_6 m_{u_2}(T_{lc} - T_{FT}). \end{aligned} \tag{31}$$

The above cost structure is when company continues to test the software after its release; the firm has adopted the *field-testing* (FT) release time policy. Therefore, subscript *FT* denotes the field-testing. Now, in the conventional approach the optimal testing termination time and optimal time-to-market the software product are same. So under *no field-testing* (NFT) scenario the cost structure becomes:

$$C(\tau_{NFT}) = C_1\tau_{NFT} + C_2\tau_{NFT}^2 + C_3 m_{t_1}(\tau_{NFT}) + C_6 m_u(T_{lc} - \tau_{NFT}). \tag{32}$$

3 Optimal policies using MAUT

In this section, two software policies, i.e. release policy with *field-testing* (FT) and policy with *no field-testing* (NFT) are described. In FT policy, testing is allowed to continue for a fixed period in the operational phase as well. In contrast, according to NFT policy, the testing stops at the release of the software. For FT policy, a joint optimization problem is developed to calculate software time-to-market and optimal testing stop time using a multi-criterion technique known as MAUT. This technique is further applied for NFT policy to evaluate optimal testing stop time, which is also a software distribution time under this policy.

3.1 Multi-attribute utility theory

Multi-Attribute Utility Theory (MAUT) is a multi-criterion decision-making technique developed by Keeney (1971) to examine the tradeoffs among different objectives. This analysis is a tool to evaluate the alternatives quantitatively. MAUT involves the following steps (Garmabaki et al. 2012; Kapur et al. 2013):

1. Determining the appropriate attributes for the problem.
2. Formulating the Single Utility Attribute Function (SAUF).
3. Measuring the relative importance of utility functions.
4. Establishing the Multi-Attribute Utility Function (MAUF).

Step 1 Attributes determination
Two important attributes associated with the software system and decisively affects its market entry time are *reliability* and *cost*. The functional form of these two critical factors are provided below.

Reliability The reliability of a software device is an essential attribute for both the vendors and the users. It is measured as a probability of failure-free operation performed by the product with desirable output in a specific period of time under certain environmental conditions (Yamada 2014). Reliability is thus an indicator of the quality of the product. Therefore, during the debugging process, maximizing the software reliability is the crucial issue for the testing team. The conditional reliability of software system in a specified time interval $[t,\ t + x]$ is given as:

$$R(x|t) = e^{-[m(t+x)-m(t)]} \tag{33}$$

The value of $R(x|t)$ lies between 0 and 1, more close the value of R to 1, more will be the reliability. Also, $R(x|t)$ possesses following boundary conditions: $R(x|0) = e^{-m(x)}$ and $R(x|\infty) = 1$. So, the first attribute for the given multi-attribute utility problem, the reliability function at the release time (τ) and testing stop time of the software is given as (Kapur et al. 2017):

Under *field-testing* (FT) policy:

$$\text{Maximize} \quad R(x|\tau, T) = e^{-[m(\tau_{FT}+x_1)-m(\tau_{FT})]-[m(T_{FT}+x_2)-m(T_{FT})]} \tag{34}$$

where x_1 and x_2 are small time durations; τ_{FT} is the optimal software release time and T_{FT} optimal testing duration under FT policy Also as discussed above, the fault-identification process before software release is done by the testers and after the release time τ, it is performed either by the testing team or users. Therefore, the mean value function of fault detection by time $(\tau + x_1)$ is given by:

$$\begin{aligned} m(\tau + x_1) =& aF_1(\tau) + \lambda a\left(1 - F_{t_1}(\tau)\right)\left[1 - \left(\frac{1 - F_{t_2}(\tau + x_1)}{1 - F_{t_2}(\tau)}\right)\right] \\ &+ (1-\lambda)a\left(1 - F_{t_1}(\tau)\right)F_u(\tau + x_1 - \tau) \end{aligned} \tag{35}$$

Thus, the number of faults detected in the small interval $[\tau,\ \tau + x_1]$ is given as:

$$m(\tau + x_1) - m(\tau) = \lambda a\left(1 - F_{t_1}(\tau)\right)\left[1 - \left(\frac{1 - F_{t_2}(\tau + x_1)}{1 - F_{t_2}(\tau)}\right)\right] + (1-\lambda)a\left(1 - F_{t_1}(\tau)\right)F_u(\tau + x_1 - \tau) \tag{36}$$

Moreover, the mean value function of fault detection in small interval $[T,\ T + x_2]$ is given as:

$$\begin{aligned} m(T + x_2) - m(T) =& a(1 - F_{t_1}(\tau)) \times \left(1 - \lambda\left\{1 - \frac{\left(1 - F_{t_2}(T)\right)}{\left(1 - F_{t_2}(\tau)\right)}\right\} - (1-\lambda)F_u(T - \tau)\right) \\ &\times \left[1 - \left(\frac{1 - F_u(T + x_2 - \tau)}{1 - F_u(T - \tau)}\right)\right] \end{aligned} \tag{37}$$

Under *no field-testing* (NFT) policy:

$$\text{Maximize} \quad R(x|\tau) = e^{-[m(\tau_{NFT}+x_1)-m(\tau_{NFT})]} \tag{38}$$

where τ_{NFT} is the optimal software release time under NFT policy, which is also the testing stop time

In NFT policy, testers detect faults in the software till time τ, the release time of the software. After that, only users detect and report the faults to the testing team. Therefore, the number of faults detected by time $(\tau + x_1)$ under no *filed-testing* policy is given as:

$$m(\tau + x_1 - \tau) = aF_{t_1}(\tau) + a\left(1 - F_{t_1}(\tau)\right)F_u(\tau + x_1 - \tau). \tag{39}$$

Cost components Cost is the most decisive attribute for the success of any enterprise. In software engineering, the cost function is directly dependent on the testing duration. As software testing increases, the cost also increases. Therefore, the primary objective of software developers is to minimize the cost and debug the faults at the earliest. Moreover, the software analysts ought to spend less than their cost budget. Thus, the second attribute takes the following form:

Under *field-testing* (FT) policy:

$$\text{Minimize} \quad C = \frac{C(\tau_{FT},\ T_{FT})}{C_b} \tag{40}$$

where $C(\tau_{FT},\ T_{FT}) = C_1T_{FT} + C_2\tau_{FT}^2 + C_3m_{t_1}(\tau_{FT}) + C_4m_{t_2}(T_{FT} - \tau_{FT}) + C_5m_{u_1}(T_{FT} - \tau_{FT}) + C_6m_{u_2}(T_{lc} - T_{FT})$ describe the software testing cost and debugging cost and C_b is the total budget available with the developers.

Under *no field-testing* (NFT) policy:

$$\text{Minimize} \quad C = \frac{C(\tau_{NFT})}{C_b} \tag{41}$$

where $C(\tau_{NFT}) = C_1\tau_{NFT} + C_2\tau_{NFT}^2 + C_3m_{t_1}(\tau_{NFT}) + C_6m_u(T_{lc} - \tau_{NFT})$ is the total software cost under NFT policy.

Step 2: SAUF formulation

In MAUT, the aspiration level of all attributes is illustrated using utility functions (Li et al. 2012; Minamino et al. 2015). The functional form of these utility functions is either linear, $u(x) = l + mx$ or exponential, $u(x) = l + me^{px}$. The applicability of the particular functional form for the problem is evaluated by interviews with the managers, surveys, and lottery (Kapur et al. 2013). For the proposed problem, linear form is adopted for both the criteria, i.e.

$$u(C) = l_c + u_cC \ \text{ and } \ u(R) = l_r + m_rR \tag{42}$$

Furthermore, each utility function is bounded with the best, $u(x^{best}) = 1$ and the worst, $u(x^{worst}) = 0$ values for every attribute. For the given problem, SAUF is based on the following management policies:

(a) At least 60% of the faults must be identified for the reliable software, and the maximum aspiration level is 100%.
(b) At least 90% of the cost budget is consumed, and the maximum requirement is 100%.

Then, the minimum cost requirement is $C^{worst} = 0.9$, and maximum cost requirement is $C^{best} = 1$. Similarly, the minimum reliability aspiration level is $R^{worst} = 0.6$ and maximum reliability aspiration level is $R^{best} = 1$. Therefore, the SAUF for cost and reliability attributes will be:

$$U(C) = 10C - 9 \ \text{ and } \ U(R) = 2.5R - 1.5 \tag{43}$$

Step 3: Relative importance of attributes

The weight or relative importance to each attribute is assigned based on the management decision. These weights are also referred to as scaling constants (Garmabaki et al. 2012). In the given problem, scaling constants are calculated by evaluating the two choices. Management has given priority to the reliability attribute, the, i.e., weight assigned to reliability is $w_R = 0.6$. Moreover, the sum of the scaling constants is always equal to 1. Therefore, the weight given to cost attribute will be $w_c = 0.4$.

Step 4 MAUF formulation

On adding all the single utility functions by multiplying each with their respective weights, MAUF is formed. For the given problem, the Multi-Attribute Utility Function with maximization objective is given as:

$$Max\ U(R,C) = w_R u(R) - w_C u(C) \tag{44}$$

where $w_R + w_c = 1$

Here the managers intend to maximize the reliability R and minimize the cost C. Therefore, the negative sign is multiplied with the cost-utility. With the help of the optimization problem given in Eq. (44), the optimal software distribution time, τ^* and optimal testing termination time, T^* is evaluated. Using the values from the previous steps, the MAUF takes the following form:

$$Max\ U(R,C) = 0.6 \times (2.5R - 1.5) - 0.4 \times (10C - 9) \tag{45}$$

The maximization problem is also subjected to the following budgetary constraint:

$$\begin{aligned} C(\tau_{FT},\ T_{FT}) &= C_1 T_{FT} + C_2 \tau_{FT}^2 + C_3 m_{t_1}(\tau_{FT}) \\ &+ C_4 m_{t_2}(T_{FT} - \tau_{FT}) + C_5 m_{u_1}(T_{FT} - \tau_{FT}) + C_6 m_{u_2}(T_{lc} - T_{FT}) \le Cb \qquad (under\,FT\,policy) \end{aligned} \tag{46}$$

$$C(\tau_{NFT}) = C_1 \tau_{NFT} + C_2 \tau_{NFT}^2 + C_3 m_{t_1}(\tau_{NFT}) + C_6 m_{u_2}(T_{lc} - \tau_{NFT}) \le Cb \qquad (under\,NFT\,policy). \tag{47}$$

4 Numerical analysis

The prediction accuracy and estimation efficiency of the SRGMs discussed in Sect. 2 is evaluated using the actual failure data. In addition, the applicability of the optimization problem developed in the previous section is exemplified using a numerical example. The fault count data of Tandem computers collected during the system-testing period is used to estimate the parameters of the developed SRGM model for phase 1. The cumulative failure occurrence dataset of the first two releases of Tandem computers are obtained from (Wood 1996). The bug-detection behavior of Tandem computers Release-1 is distributed exponentially, and Release-2 follows an S-shaped pattern. Therefore, exponential SRGM is fitted to the Release-1 (DSI) and delayed S-shaped and logistic SRGMs are fitted to the Release-2 (DSII) of Tandem computers dataset. In Release-1, the software was tested for 20 weeks wherein developers identified 100 faults and for Release-2, debugging process continues for 19 weeks wherein 120 faults were observed. To conduct the regression analysis and parameter estimation, non-linear least square (NLLS) statistical procedure is applied using Levenberg–Marquardt's method (Marquardt 1963) that minimizes the overall sum of squared errors.

The data analysis is performed using procedure SAS PROC MODEL in statistical software known as SAS (SAS/ETS User's Guide 2004). The values of four goodness-of-fit measures: root mean squared error (RMSE), mean absolute error (MAE), coefficient of determination (R-square) and, adjusted coefficient of determination (Adjusted R-square) are summarized in Table 1 for both datasets. From the results, it can be interpreted that Logistic SRGM fit better to the dataset (DSII) than delayed S-shaped SRGM.

Table 1 Performance measure results

SRGM	RMSE	MAE	R-Square	Adj R^2
DSI: Tandem computers I release				
Exponential	3.5928	3.0647	0.9857	0.9849
DSII: Tandem computers II release				
Delayed S-shaped	3.8325	2.40016	0.9900	0.9895
Logistic	2.6696	1.76753	0.9955	0.9949

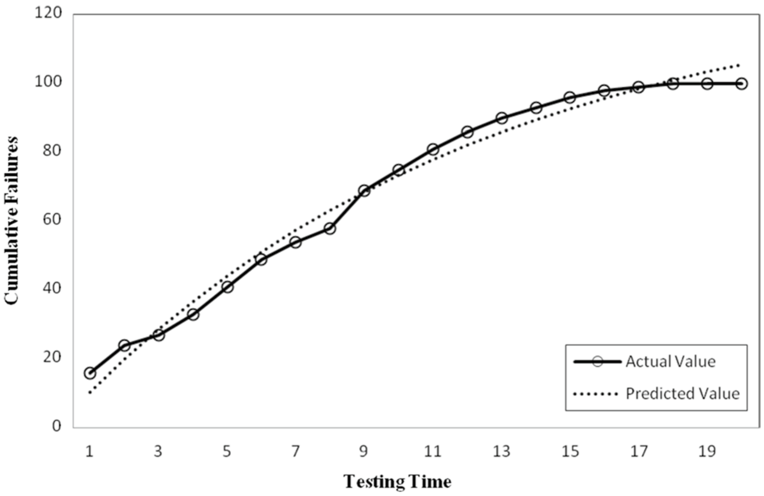

Fig. 2 Goodness-of-fit curve for exponential SRGM for DSI

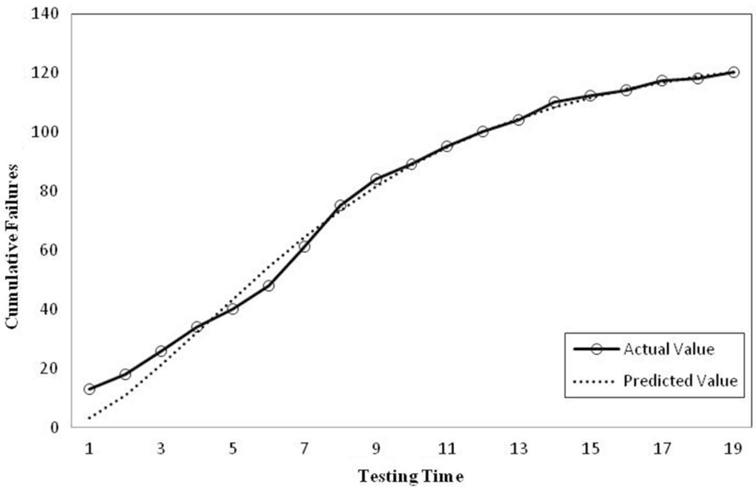

Fig. 3 Goodness-of-fit curve for delayed S-shaped SRGM for DSII

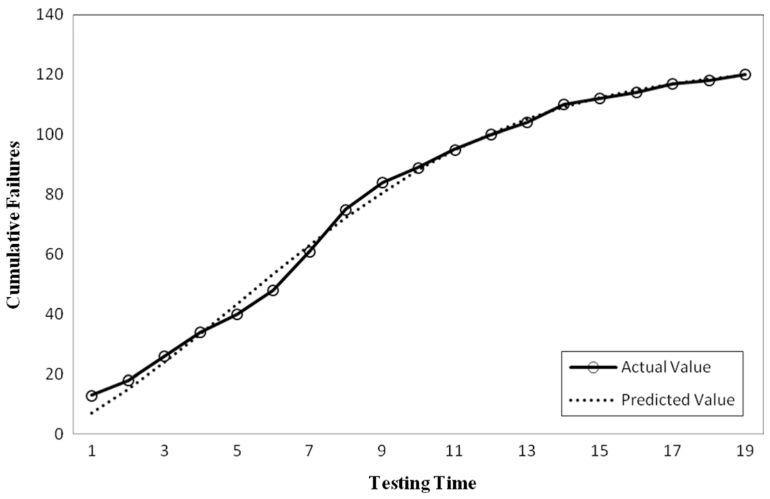

Fig. 4 The goodness-of-fit curve for logistic SRGM for DSII

Further, comparison between the actual value of fault count and the predicted values for three SRGMs are pictorially represented in Fig. 2, Figs. 3, and 4 respectively. In addition, the estimated values of the parameters for exponential SRGM are $a = 130.201$, $b_1 = 0.083$, for Delayed S-shaped SRGM are $a = 127.3989$,$b_1 = 0.241689$ and for logistic SRGM are $a = 124.445$, $b_1 = 0.2535$,$\beta_1 = 3.77849$. Further, it is assumed that the efficiency of testers in discovering the faults increases by 50% during the field-testing period, i.e., after change-point, τ, the intensity with which testers are detecting the faults is escalated. Therefore, for exponential SRGM, parameters are set as: $b_2 = 0.124$, for Delayed S-shaped SRGM: $b_2 = 0.3625335$ and for logistic SRGM: $b_2 = 0.38025$, $\beta_2 = 5.667735$. Besides, the competency of the users in detecting the faults is considered 60% of that of testers. Therefore, for users, parameters are set as $b_3 = 0.0498$ (for exponential distribution), $b_3 = 0.1450134$ (for delayed S-shaped distribution), and $b_3 = 0.1521$,$\beta_3 = 2.267094$ (for logistic distribution). The values of other parameters are set based upon the prior empirical studies and experts evaluation:

$$C_1 = \$100, C_2 = \$16, C_3 = \$40, C_4 = \$70, C_5 = \$120, C_6 = \$150,$$
$$C_b = \$23,500, x_1 = 2, x_2 = 2, \lambda = 0.6, \text{ and } T_{lc} = 100 \text{ weeks}$$

It may be noted that the value of parameter λ is assumed based on the assessment and prediction of the software analysts. It is considered that the efficiency of professional testers in identifying bugs is more as compared to the novice users. Therefore, it is assumed that the 60% ($\lambda = 0.6$) of remaining faults from the software system is detected by the testing team and remaining 40% of bugs are being detected by the users. Moreover, to analyze the significance of parameter λ on the optimal value of decision variables, the sensitivity analysis of λ is also performed.

Furthermore, using the estimated and assumed values of parameters, the optimization problem described in Eq. (45) is evaluated for the discussed SRGMs. For the present study, numeric computational software MAPLE is used for measuring and graphically representing the utility function. The prime objective of the optimization problem is to analyze and

Table 2 Optimal results under *field-testing* (FT) policy

Model under FT policy	$U*(\tau*_{FT}, T*_{FT})$	$\tau*_{FT}$ (in weeks)	$T*_{FT}$ (in weeks)	$(T*_{FT} - \tau*_{FT})$
Exponential SRGM	0.542	8.164	20.928	12.764
Delayed S-shaped SRGM	0.844	10.231	16.864	6.633
Logistic SRGM	0.916	10.478	16.041	5.563

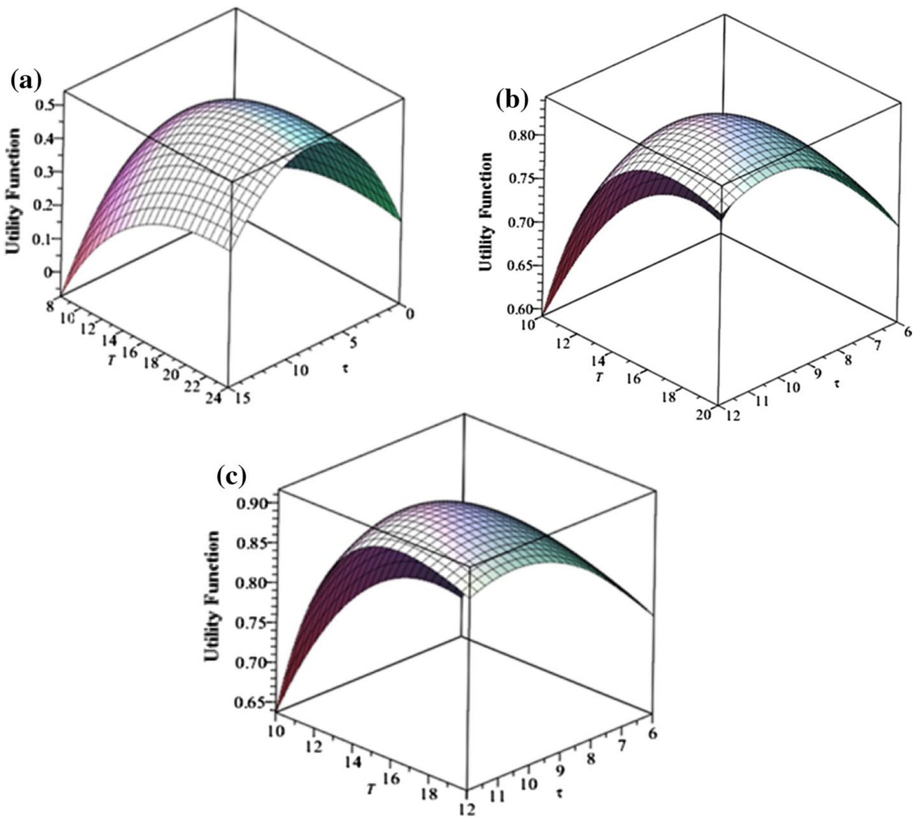

Fig. 5 Concavity plot under FT policy for **a** exponential SRGM, **b** delayed S-shaped SRGM, and **c** logistic SRGM

Table 3 Optimal results under *no field-testing* (NFT) policy

Model under NFT policy	$U(\tau_{NFT})$	$\tau*_{NFT}$ (in weeks)
Exponential SRGM	0.348	11.778
Delayed S-shaped SRGM	0.727	12.500
Logistic SRGM	0.813	12.835

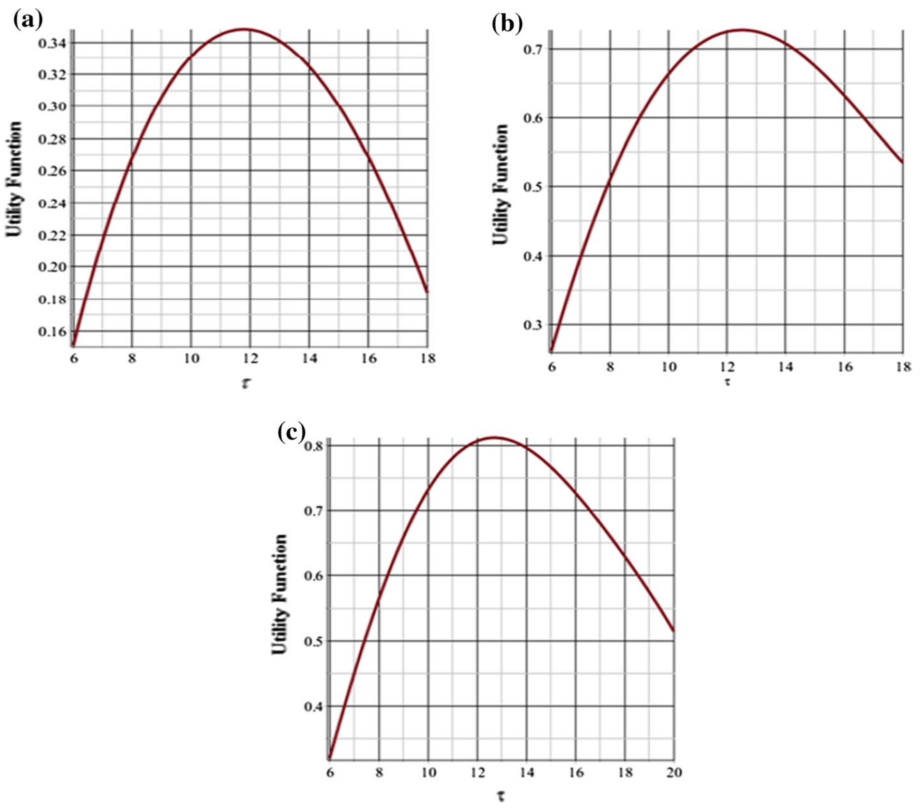

Fig. 6 Concavity plot under NFT policy for **a** exponential SRGM, **b** delayed S-shaped SRGM, and **c** logistic SRGM

compare the two different release time policies. The results of the optimization problem under FT release time policy are summarized in Table 2. The three-dimensional pictorial representation of the concavity of the utility function under FT policy is depicted in Fig. 5a–c for exponential, delayed S-shaped and logistic SRGM, respectively. In addition, the optimal results under NFT policy, wherein it is considered that testing stops at the software release time are listed in Table 3. The combined utility function of reliability and cost attribute under NFT policy is strictly concave, which is shown in Fig. 6a–c for the three SRGMs respectively.

Under FT policy, the optimal market entry time is 8.16 weeks and testing stop time is 20.93 weeks (for Release-1) with the total utility of 54.2% when fault-debugging process follows an exponential SRGM. For release 2, the optimal time-to-market is 10.23 weeks, testing duration is 16.86 weeks with 84.4% overall utility when fault-debugging process follows a delayed S-shaped function, and for logistic distribution, optimal software release time is 10.48 weeks and testing stop time is 16.04 weeks with 91.6% utility. Under NFT, only 34.8% of utility is achieved when software (Release 1) is released after 11.78 weeks of testing. For release 2, 72.7% of utility is obtained for delayed S-shaped SRGM with $\tau_{NFT} = 12.5$ weeks and when logistic distribution function is utilized to illustrate the fault-debugging process, 89.9% of utility is attained with $\tau_{NFt} = 12.83$ weeks. Thus, from the

Table 4 Sensitivity analysis on the parameter λ for exponential SRGM

λ	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
0.3	0.399	9.845	18.124	8.279
0.4	0.441	9.300	19.349	10.049
0.5	0.489	8.743	20.249	11.506
0.6	0.542	8.164	20.928	12.764
0.7	0.598	7.553	21.439	13.886

Table 5 Sensitivity analysis on the parameter λ for delayed S-shaped SRGM

λ	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
0.3	0.762	11.526	15.793	4.267
0.4	0.775	11.280	16.113	4.833
0.5	0.807	10.759	16.595	5.836
0.6	0.844	10.231	16.864	6.633
0.7	0.885	9.661	16.984	7.323

Table 6 Sensitivity analysis on the parameter λ for logistic SRGM

λ	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
0.3	0.834	11.960	14.940	2.980
0.4	0.848	11.647	15.320	3.673
0.5	0.884	10.992	15.843	4.851
0.6	0.916	10.478	16.041	5.563
0.7	0.953	9.905	16.116	6.211

joint optimization of testing termination time and software release time, it can be construed that the maximum utility can be achieved when software is released early and testing continues for an added period in operational phase. The maximum reliability and minimum cost consumption can prominently be attained when field-testing is conducted. When both testing-stop time and software distribution time coincides, the utility function decreases. Without post-release testing, the release of the software has to be delayed to meet the reliability criterion, which will ultimately increase the market opportunity cost. Thus, by releasing the software early, substantial market share can be captured. Moreover, by continuing the testing process after the software release in the market, the maximum reliability of the system can be accomplished, thereby gaining the client's trust. Moreover, from the results summarized in Tables 2 and 3, it can be inferred that the maximum utility in both the release time policy is obtained when bug-detection behavior follows a logistic distribution function.

5 Sensitivity analysis

This section examines the impact of strategic parameters on the optimal solution under FT policy. Sensitivity analysis is done by changing the values of a specific parameter and keeping the values of other parameters fixed.

Table 7 Sensitivity analysis on the parameter a for exponential SRGM

a	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
120.201	0.667	7.761	20.104	12.343
125.201	0.604	7.965	20.524	12.559
130.201	0.542	8.164	20.928	12.764
135.201	0.479	8.358	21.318	12.960
140.201	0.418	8.547	21.694	13.147

Table 8 Sensitivity analysis on the parameter a for delayed S-shaped SRGM

a	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
117.3989	0.946	9.880	16.440	6.560
127.3989	0.844	10.231	16.864	6.633
137.3989	0.744	10.554	17.255	6.701
147.3989	0.646	10.855	17.616	6.761
157.3989	0.549	11.135	17.953	6.818

Table 9 Sensitivity analysis on the parameter a for logistic SRGM

a	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
119.445	0.965	10.301	15.857	5.556
124.445	0.916	10.478	16.041	5.563
134.445	0.820	10.808	16.385	5.577
144.445	0.725	11.109	16.701	5.592
154.445	0.632	11.386	16.995	5.609

(a) Impact of λ—the fraction of bugs detected by developers during the field-testing period

When λ increases, i.e., when the testers are identifying a higher proportion of undetected faults during field-testing, then the optimal software release time ($\tau *_{FT}$) decreases. However, the optimal testing stop time ($T *_{FT}$) and the optimal duration of field-testing ($T *_{FT} - \tau *_{FT}$) increases. The results for the three distribution functions are summarized in Tables 4, 5 and 6 respectively. As software release time acts as a change-point, i.e., the effectiveness of the tester for fault-detection increases after the release of the software, it is optimal to lower the release time. Besides, the testing time and the field-testing period will increase. This is because the software is released early in the market, so more number of faults that are left undetected in the system-testing period. Therefore, it is optimal to increase the field-testing duration to reduce the risk of failure.

(b) Impact of a—The eventual number of faults to be detected

When the initial quantity of bugs in the software is more, either due to system complexity or due to flaws in the development phase, then it is optimal to extend the software

Table 10 Sensitivity analysis on the parameter C_1 for exponential SRGM

C_1(in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$(T *_{FT} - \tau *_{FT})$
80	0.616	8.246	23.004	14.758
90	0.578	8.205	21.905	13.700
100	0.542	8.164	20.928	12.764
110	0.507	8.122	20.049	11.927
120	0.473	8.078	19.251	11.173

Table 11 Sensitivity analysis on the parameter C_1 for delayed S-shaped SRGM

C_1 (in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$(T *_{FT} - \tau *_{FT})$
80	0.903	10.274	17.864	7.590
90	0.873	10.253	17.332	7.079
100	0.844	10.231	16.864	6.633
110	0.816	10.206	16.447	6.241
120	0.788	10.179	16.070	5.891

Table 12 Sensitivity analysis on the parameter C_1 for logistic SRGM

C_1 (in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$(T *_{FT} - \tau *_{FT})$
80	0.972	10.531	16.839	6.308
90	0.944	10.506	16.413	5.907
100	0.916	10.478	16.041	5.563
110	0.889	10.449	15.708	5.259
120	0.863	10.418	15.408	4.990

release time ($\tau *_{FT}$), testing termination time ($T *_{FT}$) and optimal field-testing duration ($T *_{FT} - \tau *_{FT}$) (Tables 7, 8, and 9). This is because for more number of faults, more testing is required before delivering the software to lower the risk of failure in the operational phase. Also, it will take a longer testing period to debug the number of undetected faults to meet the reliability requirement.

(c) Impact of C_1—Software per unit testing cost

The influence of variation in testing cost on the decision variables is contradictory to the consequence of the parameter a. When testing cost escalates, it is optimal to deliver the software quickly and to end the testing process relatively early, i.e., both release time ($\tau *_{FT}$) and testing termination time ($T *_{FT}$) shortens. Additionally, the duration of field-testing ($T *_{FT} - \tau *_{FT}$) will also decrease. The results for three distribution functions are listed in Tables 10, 11, and 12 respectively.

(d) Impact of C_2—Market opportunity cost

Again, when market opportunity cost increases, both release time ($\tau *_{FT}$) and testing termination time ($T *_{FT}$) reduces. The impact of C_2 on decision variable ($\tau *_{FT}$) is

Table 13 Sensitivity analysis on parameter C_2 for exponential SRGM

C_2(in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
12	0.595	9.574	21.262	11.688
14	0.566	8.804	21.078	12.274
16	0.542	8.164	20.928	12.764
18	0.520	7.621	20.802	13.181
20	0.502	7.154	20.695	13.541

Table 14 Sensitivity analysis on parameter C_2 for delayed S-shaped SRGM

C_2 (in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
12	0.923	11.393	17.105	5.712
14	0.882	10.772	16.972	6.200
16	0.844	10.231	16.864	6.633
18	0.810	9.751	16.775	7.024
20	0.779	9.320	16.700	7.380

Table 15 Sensitivity analysis on parameter C_2 for logistic SRGM

C_2 (in \$)	$U(\tau *_{FT}, T *_{FT})$	$\tau *_{FT}$	$T *_{FT}$	$\left(T *_{FT} - \tau *_{FT}\right)$
12	0.999	11.600	16.326	4.726
14	0.956	11.007	16.171	5.164
16	0.916	10.478	16.041	5.563
18	0.881	9.996	15.929	5.933
20	0.848	9.551	15.832	6.281

very significant because the overall market opportunity cost is quadratic increasing with the increase in release time. However, the effect of C_2 on the field-testing duration is opposite to that of $\tau *_{FT}$ and $T *_{FT}$, i.e. $(T *_{FT} - \tau *_{FT})$ increases with increase in market opportunity cost. This is because as software is released early, the number of undetected faults will be higher after its release and therefore, to reduce the risk of failure in operational phase, relatively lengthy field-testing is required (Tables 13, 14, 15).

6 Conclusion, managerial insights, and future research direction

In today's software-driven market, reliability and security of software devices are becoming increasingly significant. Since the cost associated with the failures are mounting and errors are continuously affecting the firm's performance, determination of the failure-occurrence phenomenon is growing into exceedingly necessary tasks for software analysts. It yields highly useful and relevant information to the management team, which help them to optimize their decisions concerning software release time and testing duration. Thus, to increase software reliability and dependability, an effective software reliability engineering techniques are required that will improve business performance and reduce software development cost and introduce software on time in the market.

In this paper, a new fault-discovery paradigm for software products is proposed. The current framework studies the bug-detection process in three phases, namely, pre-release testing phase, field-testing phase, and post-testing phase. In pre-release testing phase, developers meticulously observe the errors, while in the field-testing phase both the testing team and users discover the faults. In the post-testing phase, failures are observed only by the customers. However, in all the phases, bugs are removed immediately from the system after their discovery only by the testing team. It is assumed that the software is distributed in the market early and the testing process continues for some fixed duration in an operational phase as well.

Moreover, it is considered that the developers intensify the fault observation process after the release of the software. Otherwise, software failure during the operational phase may cause the loss of reputation of the firm and their position in the global platform. As a result, software engineers try to identify the bugs quickly to avoid failures at the user's end. Thus, the change-point occurs in the bug-detection behavior at the release time of the software. A unified approach is implemented to model the fault-detection process in different phases. This approach provides a flexible environment as it can model both an exponential NHPP and an S-shaped fault detection pattern.

An optimal release policy with consideration to field-testing (FT) is formulated. Two conflicting attributes, specifically, software reliability and software failure cost are considered, and a trade-off between them is achieved using multi-attribute utility theory (MAUT). The problem aims to maximize the utility of software reliability and to minimize the utility for the cost of software failure. The developed model can assist managers in decision-making about optimal software time-to-market and testing duration. In addition, the results of the proposed release policy are compared with the conventional release policy wherein no field-testing is conducted (NFT). From the findings, it can be concluded that the maximum utility for software reliability and minimum cost of software failures can only be achieved when software is delivered in advance and testing persists for a longer duration until the desired level of reliability is reached. Although the expected number of undiscovered faults is greater at the software release time under FT policy, the mean number of failures in the operational phase is reduced considerably under FT policy. Thus, the results of the proposed research provide vital managerial insights and support managers in regulating the decisive factors associated with the software products. The findings of the sensitivity analysis can provide an aid to the project managers in analyzing the influence of critical parameters on the release policy. With an early release of the software, managers are capturing higher market share, and with a longer testing duration, they are minimizing the risk of software failure in the field. Furthermore, longer testing duration also helps software engineers to offer highly dependable software that improves customer's satisfaction.

Besides, the formulated release policy laid a foundation for future research by incorporating various dimensions to cater different software aspects. Firms continuously update their software products to include a range of features and then release different versions of the software in the market. Therefore, the proposed release policy can be extended into the multi-release framework. Secondly, the developed SRGM has ignored the random fluctuations in the bug-observation behavior. Therefore, in future, stochastic SRGMs can be used to optimize software release time and testing stop time. Moreover, the model is based on the assumption that faults are removed as soon as they are discovered. However, in a realistic scenario, there exists a definite time lag between the software detection and their correction. Thus, the proposed optimization problem can be further formulated using two-stage

SRGMs. Another fundamental assumption on which the SRGMs is based on is that the faults are removed perfectly without causing any further defects in the software system. Therefore, this assumption can be further relaxed by incorporating the concept of imperfect debugging into the modeling framework.

References

Arora, A., Caulkins, J. P., & Telang, R. (2006). Research note—Sell first, fix later: Impact of patching on software quality. *Management Science, 52*(3), 465–471.

Chiu, K. C., Ho, J. W., & Huang, Y. S. (2009). Bayesian updating of optimal release time for software systems. *Software Quality Journal, 17*(1), 99.

Dalal, S. R., & Mallows, C. L. (1988). When should one stop testing software? *Journal of the American Statistical Association, 83*(403), 872–879.

Garmabaki, A. H., Aggarwal, A. G., Kapur, P. K., & Yadavali, V. S. S. (2012). Modeling two-dimensional software multi-upgradation and related release problem (a multi-attribute utility approach). *International Journal of Reliability, Quality and Safety Engineering, 19*(03), 1250012.

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability, 28*(3), 206–211.

Huang, C. Y. (2005a). Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency. *Journal of Systems and Software, 77*(2), 139–155.

Huang, C. Y. (2005b). Performance analysis of software reliability growth models with testing-effort and change-point. *Journal of Systems and Software, 76*(2), 181–194.

Huang, C. Y., & Lyu, M. R. (2005). Optimal release time for software systems considering cost, testing-effort, and test efficiency. *IEEE Transactions on Reliability, 54*(4), 583–591.

Huang, C. Y., & Lyu, M. R. (2011). Estimation and analysis of some generalized multiple change-point software reliability models. *IEEE Transactions on Reliability, 60*(2), 498–514.

Huang, C. Y., Lyu, M. R., & Kuo, S. Y. (2003). A unified scheme of some non-homogenous poisson process models for software reliability estimation. *IEEE Transactions on Software Engineering, 29*(3), 261–269.

Inoue, S., Ikeda, J., & Yamada, S. (2016). Bivariate change-point modeling for software reliability assessment with uncertainty of testing-environment factor. *Annals of Operations Research, 244*(1), 209–220.

Inoue, S., Taniguchi, S., & Yamada, S. (2015). An all-stage truncated multiple change point model for software reliability assessment. *International Journal of Reliability, Quality and Safety Engineering, 22*(04), 1550017.

Inoue, S., & Yamada, S. (2008, December). Optimal software release policy with change-point. In *IEEE international conference on Industrial engineering and engineering management, 2008. IEEM 2008* (pp. 531–535). IEEE.

Jiang, Z., & Sarkar, S. (2003, December). Optimal software release time with patching considered. In *Workshop on Information Technologies and Systems, Seattle, WA, USA*.

Jiang, Z., Sarkar, S., & Jacob, V. S. (2012). Postrelease testing and software release policy for enterprise-level systems. *Information Systems Research, 23*(31), 635–657.

Kapur, P. K., & Garg, R. B. (1991). Optimal release policies for software systems with testing effort. *International Journal of Systems Science, 22*(9), 1563–1571.

Kapur, P. K., & Garg, R. B. (1992). A software reliability growth model for an error-removal phenomenon. *Software Engineering Journal, 7*(4), 291–294.

Kapur, P. K., Goswami, D. N., Bardhan, A., & Singh, O. (2008a). Flexible software reliability growth model with testing effort dependent learning process. *Applied Mathematical Modelling, 32*(7), 1298–1307.

Kapur, P. K., Khatri, S. K., Tickoo, A., & Shatnawi, O. (2014). Release time determination depending on number of test runs using multi attribute utility theory. *International Journal of System Assurance Engineering and Management, 5*(2), 186–194.

Kapur, P. K., Kumar, S., & Garg, R. B. (1999). *Contributions to hardware and software reliability* (Vol. 3). Singapore: World Scientific.

Kapur, P. K., Pham, H., Aggarwal, A. G., & Kaur, G. (2012). Two dimensional multi-release software reliability modeling and optimal release planning. *IEEE Transactions on Reliability, 61*(3), 758–768.

Kapur, P. K., 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 Transactions on Reliability, 60*(1), 331–340.

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

Kapur, P. K., Shrivastava, A. K., & Singh, O. (2017). When to release and stop testing of a software. *Journal of the Indian Society for Probability and Statistics, 18*(1), 19–37.

Kapur, P. K., Singh, V. B., Anand, S., & Yadavalli, V. S. S. (2008b). 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.

Kapur, P. K., Singh, V. B., Singh, O., & Singh, J. N. (2013). Software release time based on different multi-attribute utility functions. *International Journal of Reliability, Quality and Safety Engineering, 20*(04), 1350012.

Keeney, R. L. (1971). Utility independence and preferences for multi attributed consequences. *Operations Research, 19*(4), 875–893.

Li, X., Xie, M., & Ng, S. H. (2012). Multi-objective optimization approaches to software release time determination. *Asia-Pacific Journal of Operational Research, 29*(03), 1240019.

Majumdar, R., Shrivastava, A. K., Kapur, P. K., & Khatri, S. K. (2017). Release and testing stop time of a software using multi-attribute utility theory. *Life Cycle Reliability and Safety Engineering, 6*(1), 47–55.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics, 11*(2), 431–441.

Minamino, Y., Inoue, S., & Yamada, S. (2015). Multi-attribute utility theory for estimation of optimal release time and change-point. *International Journal of Reliability, Quality and Safety Engineering, 22*(04), 1550019.

Minamino, Y., Inoue, S., & Yamada, S. (2016). NHPP-based change-point modeling for software reliability assessment and its application to software development management. *Annals of Operations Research, 244*(1), 85–101.

Musa, J. D., & Okumoto, K. (1983). Software reliability models: concepts, classification, comparisons, and practice. In J. K. Skwirzynski (Ed.), *Electronic systems effectiveness and life cycle costing* (pp. 395–423). Berlin: Springer.

Myers, G. J. (1976). *Softwear reliability: Principles and practices*. New York: Wiley.

Ohba, M., & Yamada, S. (1984). S-shaped software reliability growth models. In *International colloquium on reliability and maintainability, 4th, Tregastel, France* (pp. 430–436).

Okumoto, K., & Goel, A. L. (1980). Optimum release time for software systems based on reliability and cost criteria. *Journal of Systems and Software, 1*(4), 315–318.

Pham, H., Nordmann, L., & Zhang, Z. (1999). A general imperfect-software-debugging model with S-shaped fault-detection rate. *IEEE Transactions on Reliability, 48*(2), 169–175.

Pham, H., & Zhang, X. (1999). Software release policies with gain in reliability justifying the costs. *Annals of Software Engineering, 8*(1–4), 147–166.

SAS, S. (2004). *STAT User guide, Version 9.1.2.*. Cary, NC: SAS Institute Inc.

Singh, O., Kapur, P. K., Shrivastava, A. K., & Kumar, V. (2015). Release time problem with multiple constraints. *International Journal of System Assurance Engineering and Management, 6*(1), 83–91.

Subburaj, R., & Kapur, P. K. (2014). Two practical software reliability growth models for software project management. In P. K. Kapur, et al. (Eds.), *Quality, reliability and Infocomm technology and industrial technological management*. I.K: International Publishing House Pvt. Ltd.

Tickoo, A., Verma, A. K., Khatri, S. K., & Kapur, P. K. (2016). Modeling Two-Dimensional Framework for Multi-Upgradations of a Software with Change Point. *International Journal of Reliability, Quality and Safety Engineering, 23*(06), 1640008.

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

Yamada, S. (2014). *Software reliability modeling: Fundamentals and applications* (Vol. 5). Tokyo: Springer.

Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability, 32*(5), 475–484.

Yamada, S., & Osaki, S. (1985). Software reliability growth modeling: Models and applications. *IEEE Transactions on Software Engineering, 12,* 1431–1437.

Yamada, S., & Osaki, S. (1987). Optimal software release policies with simultaneous cost and reliability requirements. *European Journal of Operational Research, 31*(1), 46–51.

Yamada, S., & Yamaguchi, M. (2016). A method of statistical process control for successful open source software projects and its application to determining the development period. *International Journal of Reliability, Quality and Safety Engineering, 23*(05), 1650018.

Zhang, X., & Pham, H. (2002). Predicting operational software availability and its applications to telecommunication systems. *International Journal of Systems Science, 33*(11), 923–930.

Zhao, M. (2003). Statistical reliability change-point estimation models. In H. Pham (Ed.), *Handbook of Reliability Engineering* (pp. 157–163). London: Springer.

Zhu, M., & Pham, H. (2018). A multi-release software reliability modeling for open source software incorporating dependent fault detection process. *Annals of Operations Research, 269,* 773–790.