Software Reliability Models and Multi-attribute Utility Function Based Strategic Decision for Release Time Optimization

Vishal Pradhan, Joydip Dhar, and Ajay Kumar

Abstract The software industry is working hard to keep up with these rapid changes by devising methods to increase the pace of their work without compromising software quality and reliability. Various factors, such as the testing environment, testing strategy, and resource allocation, can influence the optimal release time. The choice of whether or not to release a software product would become much more complicated and significant. When a software developer, clients, or end-users face significant potential financial losses, a decision has strategic significance. A software release decision is a trade-off between early release to take advantage of an earlier market launch and product release deferral to ensure reliability. If a software product is released too soon, the software developer must pay for post-release costs to correct bugs. To decide the best software release time, two attributes, reliability and cost, must be combined. This study discusses a realistic approach to determining when to stop software testing that considers reliability and cost. A multi-attribute utility theory-based proposed decision model is analyzed on various separate weighted combinations of utility functions.

Keywords Software reliability growth model · Multi-attribute utility function · Optimal release time · Non-homogeneous poisson process

1 Introduction

Software technologies are the most prevalent human-made technology that impacts our daily lives due to the importance of software applications in recent years. In the last two decades, the penetration of software-based technologies into people's everyday lives has been remarkable. Everything we see around us is dependent on software or has some connection to software systems. There is a requirement for

V. Pradhan (✉) · J. Dhar · A. Kumar
ABV-Indian Institute of Information Technology and Management Gwalior,
Gwalior 474015, India
e-mail: vishal.iiitmg@gmail.com; vishalp@iiitm.ac.in

V. Kumar and H. Pham (eds.), *Predictive Analytics in System Reliability*,
Springer Series in Reliability Engineering,
https://doi.org/10.1007/978-3-031-05347-4_12

highly dependable, secure, and high-quality software development since our social structure has become increasingly dependent on software-based technologies [31]. Reliability can only be accomplished by thoroughly testing the program before it is made available to the public. Program errors are found, identified, and fixed throughout the testing process, improving software reliability [13]. Reliability is an essential statistic for evaluating commercial software quality in the testing and operational phases. Software reliability may be defined as the likelihood of error-free software execution in a particular environment over a predetermined duration [17, 23]. Non-homogeneous Poisson process (NHPP) based growth models are frequently used in software systems to describe stochastic failure behavior and measure growth reliability [3, 5, 7, 9, 33].

NHPP models have also been extensively used in the cost-control analysis, software time-to-market analysis, and resource allocation issues [1, 10, 12, 26, 32]. The correctness and security of a software system can only be improved with sufficient testing time and effort, such as CPU hours and qualified testing specialists [10, 29]. In general, software testing uses around half of the resources for software development. Continuous software testing for a more extended period may obstruct the timely delivery of the software system. Furthermore, it will quickly result in significant development expenses. Simultaneously, shorter testing combined with an insufficient debugging procedure would cause customer disappointment, potentially affecting the growth of the software as well as the software firm's goodwill. In today's market, a software testing budget should be prioritized over its development budget [11, 20]. As a result, software reliability engineering provides a cost-effective compromise between client needs for dependability, accessibility, delivery time, and life cycle [16, 19]. SRGMs are used to optimize testing techniques for increased organizational competitiveness, estimate the amount of required resources, and calculate the overall cost of the development process [22, 34, 39, 40].

The software reliability may be predicted using appropriate software reliability growth models (SRGMs) based on the fault count data obtained during the testing process [21, 36]. The testing phase is the most significant since it is at this step that the fault detection and removal procedure takes place, which is critical for the dependability and quality of any software system. A critical decision point for management is when to end testing and release the software system to the user [30]. This is referred to as the "Software Release Time Problem". Before being released, the software is subjected to a rigorous testing procedure in order to identify flaws that might have devastating effects if not corrected. Several methods of software testing are now in use with the goal of eliminating faults. It's possible that many bugs went undiscovered because of the short testing time and the sudden release [37]. The choice to release software is a complicated one, and there are significant dangers involved with a release agreement that is either too rapid or too delayed [18, 24]. One of the most common applications of SRGMs is to assist developers in determining the optimal timing to deploy software [2, 6, 22, 35].

The main contribution of this work is as follows:

1. Proposed new SRGMs with log-logistic and Burr Type XII distribution as a fault detection rate.
2. This study suggests the multi-attribute utility theory based optimal release time.

2 Software Reliability Modeling

The NHPP based SRGM is used in this work. The NHPP is a method of calculating the total number of faults found throughout the testing procedure. In this technique, SRGMs such as exponential [8], delayed S-Shaped [41], inflected delayed S-shaped [27], and power function have been used to anticipate potential bugs laying latent in the program. Let $N(t)$ be the total number of defects discovered at time t, and $m(t)$ be the expected number of faults. The failure intensity $\lambda(t)$ is therefore linked as follows:

$$m(t) = E[N(t)] = \int_0^t \lambda(s)ds \ , \tag{1}$$

where $N(t)$ has a Poisson probability mass function with parameter m(t), which is as follows:

$$Pr\{N(t) = n\} = \frac{m(t)^n.e^{-m(t)}}{n!} \ , \quad x = 0, 1, 2, ... \tag{2}$$

Various time-dependent models that describe the stochastic failure process of an NHPP have been published in the literature. The failure intensity function $\lambda(t)$ differs across these models, and therefore $m(t)$. In the case of finite failure NHPP models, let "Λ" indicate the estimated total number of faults that would be identified given infinite testing time.

One of the main goals of testing is to identify software faults to fix them. Once the software code has been written, testing can begin. Before the software is released to the public, the software testing team thoroughly tests it to ensure that the software contains the least number of bugs. Despite the fact is that it is almost impossible to eliminate all the software bugs. As a result, when the testing team tests the software, there's a probability they'll only find a finite number of problems in the code (less than the total number of faults).

2.1 *Assumption*

i. NHPP models the failure observation/fault removal phenomenon.
ii. The software system is susceptible to failure at any time due to errors that have remained in the system.

iii. There are a finite number of bugs present in the software.
iv. When a failure occurs, it is instantly removed.
v. The severity level of all faults is the same.
vi. The perfect debugging environment is taken into account.
vii. All remaining software faults have an equal impact on the failure rate.
viii. The number of defects discovered throughout the testing process is directly proportional to the number of faults still present in the software.
ix. With a probability distribution function, each occurrence of failure is distributed independently and identically across the software life-cycle.

As a result, finite numbers of bugs are perfectly eradicated, with the mathematical equation. The finite failure NHPP models' differential equation formulated based on the modeling assumption and it expressed as:

$$\frac{dm(t)}{dt} = r(t)[\Lambda - m(t)] \tag{3}$$

When Eq. (3) is solved for the initial condition $m(0) = 0$, the MVF can be calculated as follows:

$$m(t) = \Lambda[1 - e^{-\int_0^t r(v)dv}] = \Lambda[1 - e^{-B(t)}] \tag{4}$$

$$m(t) = \Lambda.F(t)\ . \tag{5}$$

where $F(t)$ is a distribution function.

Various researches assume that fault detection is constant throughout the testing process, but it is not possible in practical behavior. For the detection rate, we know that it is low at the initial stage, and in the mid-stage, it's on the peak; in the later stage, it's again low. So, the FDR is modeled through the specific distribution handling the situation. Therefore, this study proposed the SRGMs with the two most applicable distribution functions for $B(t)$, i.e., Log-logistic and Burr type XII distribution functions.

2.2 *Fault Detection Rate*

2.2.1 Log-Logistic Distribution

The logistic distribution and the log-logistic distribution are closely linked. A probability distribution whose logarithm has a logistic distribution is known as a log-logistic distribution. Log(x) is distributed logistically with mean and standard deviation if x is distributed loglogistically with parameters μ and σ. The log-logistic distribution is a good replacement for the Weibull distribution. It's a hybrid of the Gompertz and Gamma distributions, with the mean and variance values equal to one. The log-logistic distribution has its own status as a life testing model; it is an increasing failure rate model as well as a weighted exponential distribution. The generalized log-logistic distribution refers to several distinct distributions that include the

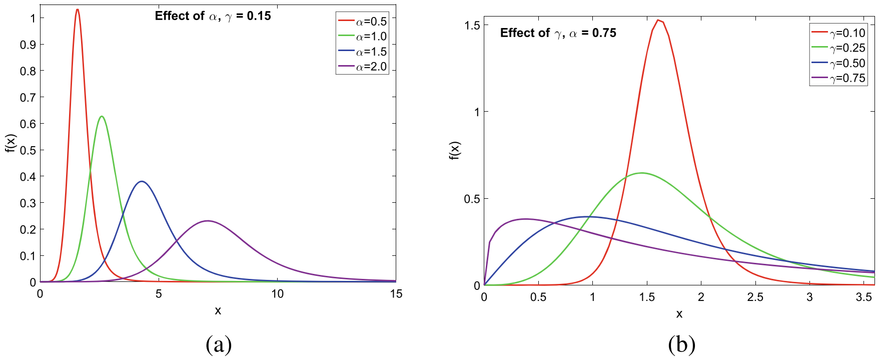


Fig. 1 Log-logistic distributions with effect of **a** shape and **b** scale parameter values

log-logistic as a particular instance. The Burr Type XII distribution and the Dagum distribution, both of which have a second shape parameter, are examples. It's a flexible distribution family that may represent a wide range of distribution types that are shown in Fig. 1. In survival analysis, this distribution is frequently used to simulate events that have an initial rate increase followed by a rate decrease.

The log-logistic distribution with positive scale parameter γ and shape parameter α is described as follows:

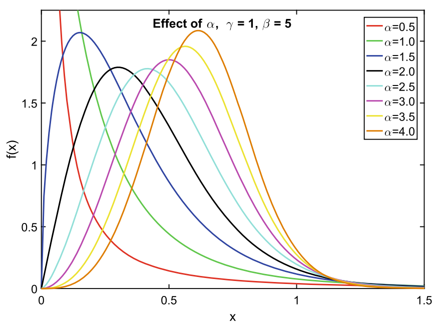
$$B(t) = \frac{(\frac{t}{\gamma})^{\alpha}}{1 + (\frac{t}{\gamma})^{\alpha}}\ , \tag{6}$$

and the density function is:

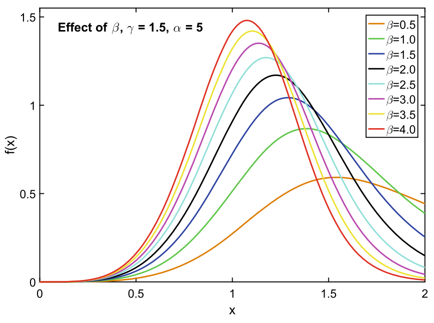
$$b(t) = \frac{(\frac{\alpha}{\gamma})(\frac{t}{\gamma})^{\alpha-1}}{[1 + (\frac{t}{\gamma})^{\alpha}]^2} \tag{7}$$

2.2.2 Burr Type XII Distribution

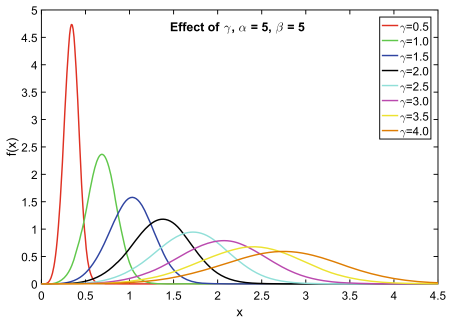
The Burr distribution can fit a wide range of empirical data. The parameters' various values span a wide range of skewness and kurtosis. As a result, it is used to represent a range of data types in diverse disciplines such as finance, hydrology, and reliability. The Burr type XII distribution generalizes Burr distribution with additional scale parameters. It is a three-parameter family of positive real-line distributions. It's a versatile distribution family that may represent a variety of different distribution forms that are shown in Fig. 2. Many widely used distributions, such as gamma, log-normal, log-logistic, bell-shaped, and J-shaped beta distributions, are included, overlapped, or have the Burr distribution as a limiting case (but not U-shaped). The Burr distribution is also found in several compound distributions. A Burr distribution is created by compounding a Weibull distribution with a gamma distribution for the scale parameter. There are two asymptotic limiting cases for the Burr distribution: Weibull and Pareto Type I.



(a)



(b)



(c)

Fig. 2 Burr Type XII distributions with effect of **a** shape, **b** shape and **c** scale parameters

The Burr distribution's cumulative distribution function (cdf) is:

$$B(t) = 1 - \frac{1}{\left[1 + \left(\frac{t}{\gamma}\right)^{\alpha}\right]^{\beta}} , \tag{8}$$

$$b(t) = \frac{\frac{\alpha\beta}{\gamma}(\frac{t}{\gamma})^{\alpha-1}}{[1 - (\frac{t}{\gamma})^{\alpha}]^{\beta+1}} \tag{9}$$

2.3 Software Reliability Growth Models

$$m(t) = \Lambda[1 - e^{-B(t)}] \tag{10}$$

2.3.1 Model-1

The MVF of Log-logistic FDR based SRGM is defined as:

$$m(t) = \Lambda \left[1 - e^{-\frac{(\frac{t}{\gamma})^{\alpha}}{1+(\frac{t}{\gamma})^{\alpha}}} \right]. \tag{11}$$

2.3.2 Model-2

The MVF of Burr type XII FDR based SRGM is defined as:

$$m(t) = \Lambda \left[1 - e^{-\left(1 - \frac{1}{\left[1+\left(\frac{t}{\gamma}\right)^{\alpha}\right]^{\beta}}\right)} \right]. \tag{12}$$

One of the most common applications of SRGMs is to assist developers in determining the optimal timing to deploy software. This study formulated a cost model to estimate the best software release timing in the latter portion of this paper. This field of study is strongly connected to the wider software reliability research.

3 Numerical Illustration

The practical applicability of the suggested problem is demonstrated in this section using historical fault discovery data as an example. The fault count data set was used for the numerical illustration. The non-linear least square estimation (LSE) method is used to estimate model parameters. The estimated model parameter findings for detected faults throughout the testing period are shown in Table 1.

The behavior of actual defects data for software release is observed in the graph and most of them are in S-shaped form. This is further supported by the usage of the log-logistic and Burr Type XII FDR function to detect software faults. As evidenced by the values of several comparison criteria, model-1 and model-2 provide a perfect fit. Table 2 present the estimated values of proposed and existing models for DS-1 to DS-6. Table 3 present a comparative analysis of the proposed and existing models. The comparison criteria used here are the sum of square error (SSE), coefficient of determination (R^2) and Adjusted R-square (R^2_{adj}) Fig. 3 illustrate a graphical representation of estimated vs. real cumulative failures over time for a better understanding. Based on these findings, we can conclude that the proposed

Table 1 Datasets from the existing literature

Dataset (DS)	Testing time	Detected faults	Remark
DS-1 [38]	18 Weeks	176	Failure data of large medical record system
DS-2 [38]	17 Weeks	204	Failure data of large medical record system
DS-3 [42]	21 Weeks	43	System test data for a telecommunication system
DS-4 [25]	30 Days	289	Real software project failure data
DS-5 [8]	20 Weeks	100	Computer Programming Center of NTDS data
DS-6 [28]	19 Weeks	328	Reported from Ohba 1984 test data

Table 2 Estimated values of SRGMs parameters for all six datasets

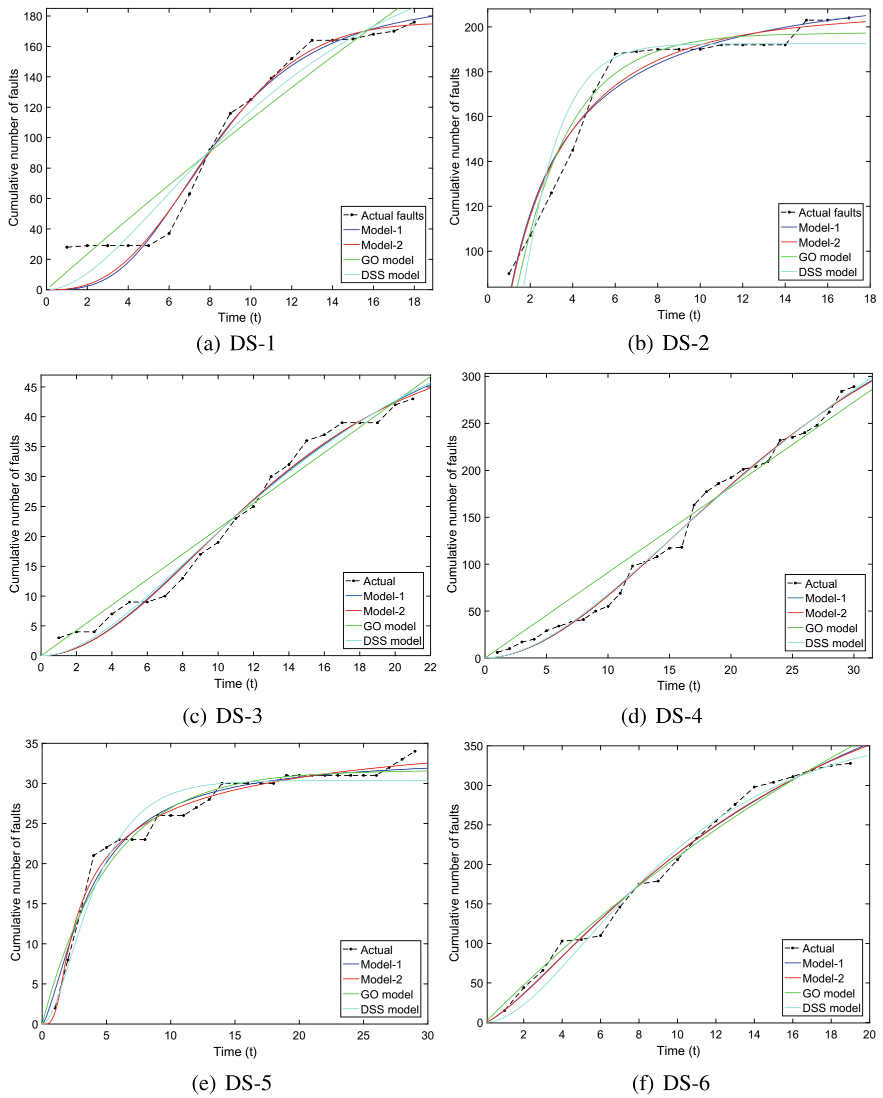
DS	Model-1			Model-2				GO model		DSS model	
	Λ	γ	α	Λ	γ	α	β	Λ	b	Λ	b
DS-1	305.9	9.707	3.082	277.7	171.5	2.672	1805	985.9	0.9243	226.1	0.1741
DS-2	358.9	3.111	0.980	325.2	2.3E+4	0.842	1500	197.4	0.3985	192.5	0.8814
DS-3	106.7	19.63	1.917	82.74	661.8	1.903	987.2	1.6E+4	1.3E+4	62.30	0.1185
DS-4	831.0	35.76	1.880	651.3	171.0	1.841	21.30	6.2E+4	1.4E+4	495.7	0.0645
DS-5	52.85	5.29	1.448	67.19	1.334	4.198	0.083	31.66	0.1906	30.35	0.4601
DS-6	979.8	23.5	1.311	741.9	2768	1.301	626.4	760.5	0.0323	374.1	0.1977

Table 3 Performance comparison of SRGMs for all six datasets

DS	Model-1			Model-2			GO model			DSS model		
	SSE	R^2	R^2_{adj}	SSE	R^2	R^2_{adj}	SSE	R^2	R^2_{adj}	SSE	R^2	R^2_{adj}
DS-1	2544	0.9598	0.9544	2315	0.9634	0.9556	4789	0.9243	0.9196	3246	0.9487	0.9455
DS-2	1034	0.9477	0.9402	910.2	0.9539	0.9433	1210	0.9388	0.9347	3489	0.8234	0.8117
DS-3	59.80	0.9855	0.9839	56.00	0.9864	0.9840	125.8	0.9612	0.9598	62.19	0.9849	0.9841
DS-4	2204	0.9912	0.9905	2194	0.9912	0.9902	9663	0.9612	0.9598	2277	0.9909	0.9905
DS-5	43.80	0.9714	0.9696	26.15	0.9830	0.9810	62.29	0.9596	0.9581	102.4	0.9336	0.9311
DS-6	2111	0.9892	0.9879	2025	0.9897	0.9876	2656	0.9865	0.9857	3205	0.9837	0.9827

model-2 produces good performance and is more realistic when it comes to forecasting the growth behavior of application-based software systems.

In the next section we discuss about the optimal release policy.



(a) DS-1 (b) DS-2

(c) DS-3 (d) DS-4

(e) DS-5 (f) DS-6

Fig. 3 **a–f** The fitting results of SRGMs comparison with actual failure data for DS-1-DS-6

4 Optimal Release Policy

With increasing competition in the software industry, continually changing client expectations, and the usual challenges involved with software maintenance, the timing of a new software release has become increasingly critical for a software vendor's success in the market [15]. Given the fierce competition in the market, deploying software on time has become a vital aspect in deciding the software development team's success. The dynamic release problem in software testing processes is discussed in this work [4]. The process of choosing between alternative courses of action in order to achieve goals and objectives is known as decision-making. Software release time, for example, estimating when it should be completed. Other managerial functions rely substantially on decision-making, such as organizing, implementing, and controlling [14].

If the testing period is extended in the software development process, the developed software will presumably be more reliable, but the testing cost will escalate. If we end testing too soon, the program may have too many flaws, resulting in too many failures during operation and significant losses owing to failure penalties or customer discontent. We may incur a considerable testing expense if we spend too much time testing. If the testing period is too short, the software may not be error-free. As a result, software testing and release are mutually exclusive. The testing procedure should determine the release timing dynamically. As a result, our goal is to come up with an appropriate release policy that reduces the cost and time of software testing while increasing the system's reliability. The ideal release time based on the cost-reliability criterion has been described and evaluated.

4.1 *Cost and Reliability Modeling*

4.1.1 Cost Modeling

1. **Testing cost per unit testing time**: The effort necessary to perform and execute the testing procedure is included in the testing cost. The cost of testing rises linearly with the time of the test. If C_1 is the testing cost per unit time, then the total testing cost is as follows:

$$C_{TCPU} = C_1.T \ . \tag{13}$$

2. **Debugging cost during testing-phase**: This cost includes the testing team's effort to handle failures. The expected number of bugs identified during this time is assumed linearly in software reliability literature. So, in the testing phase, the error-debugging cost is:

$$C_{DCDT} = C_2.m(T) \ . \tag{14}$$

3. **Debugging cost during operational-field**: In the operational phase, it is believed that Debugging cost during operational-field $C_2(T)$ is proportional to the number of software faults that were removed. Thus,

$$C_{DCDO} = C_3.[m(T_{LC}) - m(T)] \ . \tag{15}$$

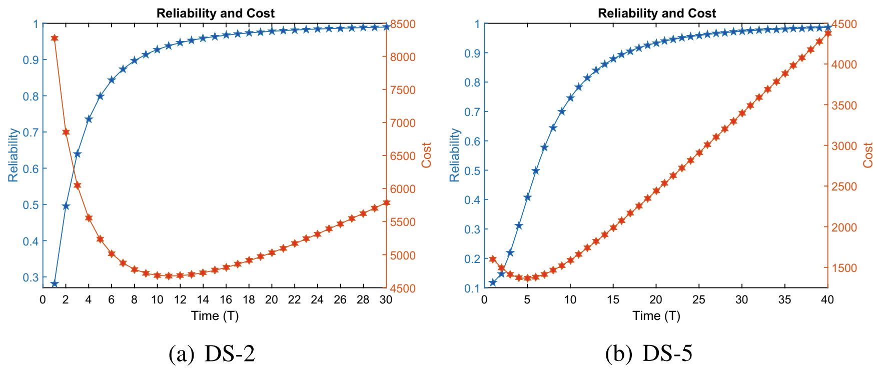


Fig. 4 Cost versus testing time and reliability vs testing time for DS-2 and DS-5

Because C_2 represents the deterministic cost to remove each fault per unit time during testing, and $C_3(T)$ represents the cost of eliminating a fault during the operational phase, C_3 is typically more than C_2, i.e., $C_3 > C_2$.

They presented a three-part software cost model structure: testing cost per unit time, debugging cost in the testing phase, and debugging cost in the operational phase. The mathematical version of the overall cost model is:

$$C(T) = C_{DCDT} + C_{DCDT} + C_{DCDO} \ . \tag{16}$$

$$= C_1 T + C_2 m(T) + C_3[m(T_{LC}) - m(T)] \ . \tag{17}$$

4.1.2 Reliability Modeling

$$R(\Delta t/T) = e^{-[m(T+\Delta t) - m(T)]} \tag{18}$$

Cost and reliability analysis with time is shown in Fig. 4.

4.1.3 Release Time Problem Using MAUT

When a sequence of possibilities is presented, the goal is to obtain a conjoint measure indicating how desirable one conclusion is in comparison to the others. It is a classical multi-objective optimization technique that addresses the optimization problem by applying weights and utility functions to determine which objectives should be prioritized [30]. The following is the formula for the multi-attribute utility function (MAUF), a weighted sum of single utility functions. It is defined as follows:

$$U(x_1, x_2, ...x_n) = f(u(x_1), u(x_2), ...u(x_n)) = \sum_{i=1}^{n} \theta_i u_i(x_i) \tag{19}$$

This work uses MAUT to construct a new decision model for software release schedule determination that trades off two conflicting objectives at the same time.

The process of determining the utility value consists of four steps.

1. Selection of Attributes.
2. Evaluate the utility function for a single attribute.
3. Allocation of credit and preference for trade-offs.
4. Single attribute to multi attribute utility function transformation.

1. **Attribute selection** Reliability is a necessary attribute that influences optimal software time-to-market and testing length selections. As a result, the proposed optimization problem's first attribute is reliability (R). The second attribute is overall software development cost (C), because no company wants to spend more than it can afford. We take the R and C as two attributes in this study. Our initial goal is to strike a compromise between these two goals by maximizing reliability while minimizing total software development costs:

$$max: \quad R(T) = e^{-[m(T+\Delta t)-m(T)]} , \tag{20}$$

$$min: \quad C(T) = \left[\frac{C(T)}{C_b}\right], \quad \frac{C(T)}{C_b} \leq 1 . \tag{21}$$

The total budget available to the testing team is denoted by C_b.

2. **Single attribute utility function**
Each attribute's aim is represented by a utility function applied to each attribute. The single-attribute utility theory (SAUF) expresses the level of satisfaction of management concerning each of the attributes. There are many different functional forms of the utility function, such as linear, exponential, and so on. The utility function of two qualities, namely, reliability and cost function, is used in this study. The linear (additive) form $u(x) = y_1 + y_2 x$ should be employed if they are equivalent to each other because management is risk-neutral. The proposed framework is illustrated as follows:

$$u(R) = l_r R(T) + k_r , \tag{22}$$

$$u(C) = l_c C(T) + k_c . \tag{23}$$

where, k_r, l_r, k_c, l_c are constants.

Table 4 Optimal release time by MAUT for DS-2 and DS-5

Attribute weights		Release time (T^*)	
w_r	w_l	DS-2	DS-5
0.4	0.5	14	11
0.5	0.6	15	13
0.6	0.4	16	15
0.7	0.3	18	17
0.8	0.2	20	21
0.9	0.1	25	27

3. **Weight parameter estimation**
The management decision determines the relative value of each attribute. In this study, we perform various weight combinations values for each attribute. The weight parameter has a value between 0 and 1, with a value closer to 1 denoting greater significance. Furthermore, the sum of the weight parameters must equal 1, i.e.,
$$w_r + w_c = 1 \; . \tag{24}$$
where w_r and w_c are weight for the reliability and cost respectively.

4. **Formulation of MAUT**
The MAUT function is created by multiplying all of the single utility functions by their corresponding weights. The MAUT function with the maximizing objective for the given problem is:
$$Max : \quad U(R, C) = w_r.u(R) - w_c.u(C) \; . \tag{25}$$
where
$$u(R) = 2R - 1 \; , \tag{26}$$
$$u(C) = 2C - 1 \; . \tag{27}$$
$U(R, C)$ is a max function that has been written in terms of R and C. From the manager's perspective, R should be maximized while C should be minimized. where, $T_{LC} = 1000, C_1 = 100, C_2 = 10, C_2 = 50$. For DS-2, $C_b = 8500\$$, $\Delta t = 0.025$ and for DS-5, $C_b = 8500\$$, $\Delta t = 0.4$. With the different combination of weights to reliability and cost based optimal release time is shown in Table 4.

5 Conclusions

It is also possible to optimize software release and testing times by maximizing utility. The results show that a corporation should publish software early to achieve a competitive edge. The solution to the problem can also assist software firms design efficient release and testing procedures. In this work, we propose an effort-based optimum decision model that takes into account the cost of detection during testing and operational phases separately using MAUT. SRGMs provide a statistical foundation for determining optimal software testing release time. A decision model based on MAUT is suggested to make wise decisions on optimal test runs before software release. This study optimizes cost and reliability using multi-attribute utility theory and gets optimal release time. These models may help the software industry anticipate software system dependability and release time.

Acknowledgements One of the authors, Mr. Vishal Pradhan, would like to thank to Ministry of Education (earlier MHRD) for financial support provided. The support is gratefully acknowledged.

References

1. Chatterjee S, Chaudhuri B, Bhar C, Shukla A (2020) Optimal release time determination using FMOCCP involving randomized cost budget for FSDE-based software reliability growth model. Int J Reliab Qual Saf Eng 27(01):2050004
2. Chatterjee S, Shukla A (2017) An ideal software release policy for an improved software reliability growth model incorporating imperfect debugging with fault removal efficiency and change point. Asia-Pac J Oper Res 34(03):1740017, e2150
3. Chatterjee S, Shukla A (2019) A unified approach of testing coverage-based software reliability growth modelling with fault detection probability, imperfect debugging, and change point. J Softw Evol Process 31(3):e2150
4. Choudhary C, Kapur PK, Khatri SK, Muthukumar R, Shrivastava AK (2020) Effort based release time of software for detection and correction processes using MAUT. Int J Syst Assur Eng Manage 11(2):367–378
5. Dhaka R, Pachauri B, Jain A (2021) Two-dimensional SRGM with delay in debugging by considering the uncertainty factor and predictive analysis. Reliab Theory Appl SI 2(64):82–94
6. Dhaka R, Pachauri B, Jain A (2022) Two-dimensional software reliability model with considering the uncertainty in operating environment and predictive analysis. In: Data engineering for smart systems, pp 57–69. Springer
7. Goel AL (1983) A guidebook for software reliability assessment. Technical report, SYRACUSE UNIV NY
8. 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
9. Hsu C-J, Huang C-Y, Chang J-R (2011) Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor. Appl Math Model 35(1):506–521
10. Huang CY, Lo JH, Kuo SY, Lyu MR (2004) Optimal allocation of testing-resource considering cost, reliability, and testing-effort. In: 10th IEEE Pacific Rim international symposium on dependable computing proceedings, pp 103–112. IEEE

11. Huang CY, Lyu MR (2005) Optimal release time for software systems considering cost, testing-effort, and test efficiency. IEEE Trans Reliab 54(4):583–591
12. Jain M, Manjula T, Gulati TR (2014) Prediction of reliability growth and warranty cost of software with fault reduction factor, imperfect debugging and multiple change point. Int J Oper Res 21(2):201–220
13. Kapur PK, Goswami DN, Bardhan A, Singh O (2008) Flexible software reliability growth model with testing effort dependent learning process. Appl Math Model 32(7):1298–1307
14. Kapur PK, Khatri SK, Tickoo A, Shatnawi O (2014) Release time determination depending on number of test runs using multi attribute utility theory. Int J Syst Assur Eng Manage 5(2):186–194
15. Kapur PK, Panwar S, Singh O, Kumar V (2019) Joint release and testing stop time policy with testing-effort and change point. In: Risk based technologies, pp 209–222. Springer
16. Kapur PK, Pham H, Aggarwal AG, Kaur G (2012) Two dimensional multi-release software reliability modeling and optimal release planning. IEEE Trans Reliab 61(3):758–768
17. Kapur PK, Pham H, Gupta A, Jha PC (2011) Software reliability assessment with OR applications. Springer
18. Kapur PK, Shrivastava AK, Singh O (2017) When to release and stop testing of a software. J Ind Soc Probab Stat 18(1):19–37
19. Kapur PK, Singh O, Shrivastava AK (2014) Optimal price and testing time of a software under warranty and two types of imperfect debugging. Int J Syst Assur Engi Manage 5(2):120–126
20. Kumar V, Sahni R, Shrivastava AK (2016) Two-dimensional multi-release software modelling with testing effort, time and two types of imperfect debugging. Int J Reliab Saf 10(4):368–388
21. Kumar V, Saxena P, Garg H (2021) Selection of optimal software reliability growth models using an integrated entropy–technique for order preference by similarity to an ideal solution (topsis) approach. Math Methods Appl Sci
22. Lee DH, Hong Chang I, Pham H, Song KY (2018) A software reliability model considering the syntax error in uncertainty environment, optimal release time, and sensitivity analysis. Appl Sci 8(9):1483
23. Lin C-T, Huang C-Y (2008) Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. J Syst Softw 81(6):1025–1038
24. Majumdar R, Shrivastava AK, Kapur PK, Khatri SK (2017) Release and testing stop time of a software using multi-attribute utility theory. Life Cycle Reliab Saf Eng 6(1):47–55
25. Meeker WQ, Hong Y (2014) Reliability meets big data: opportunities and challenges. Qual Eng 26(1):102–116
26. Mishra G, Kapur PK, Shrivastava AK (2017) Multi release cost model : a new perspective. Int J Reliab Qual Saf Eng 24(06):1740007
27. Ohba M, Yamada S (1984) S-shaped software reliability growth models, pp 430–436
28. Ohba M (1984) Software reliability analysis models. IBM J Res Dev 28(4):428–443
29. Pachauri B, Kumar A, Dhar J (2013) Modeling optimal release policy under fuzzy paradigm in imperfect debugging environment. Inf Softw Technol 55(11):1974–1980
30. Pachauri B, Kumar A, Dhar J (2014) Software reliability growth modeling with dynamic faults and release time optimization using GA and MAUT. Appl Math Comput 242:500–509
31. Pham H (2007) System software reliability. Springer Science & Business Media
32. Pham H, Zhang X (2003) NHPP software reliability and cost models with testing coverage. Eur J Oper Res 145(2):443–454
33. Pradhan V, Dhar J, Kumar A (2022) Testing-effort based NHPP software reliability growth model with change-point approach. J Inf Sci Eng 38(2)
34. Pradhan V, Dhar J, Kumar A, Bhargava A (2020) An S-shaped fault detection and correction SRGM subject to gamma-distributed random field environment and release time optimization. Decis Anal Appl Ind 285–300. Springer
35. Pradhan V, Kumar A, Dhar J (2021) Modelling software reliability growth through generalized inflection s-shaped fault reduction factor and optimal release time. In: Proceedings of the institution of mechanical engineers, Part O: journal of risk and reliability, p 1748006X211033713

36. Saxena P, Kumar V, Ram M (2021) Ranking of software reliability growth models: a entropy-electre hybrid approach. Reliab Theory Appl SI 2(64):95–113
37. Shrivastava AK, Kumar V, Kapur PK, Singh O (2020) Software release and testing stop time decision with change point. Int J Syst Assur Eng Manage 11(2):196–207
38. Stringfellow C, Amschler Andrews A (2002) An empirical method for selecting software reliability growth models. Empir Softw Eng 7(4):319–343
39. Tamura Y, Yamada S (2019) Software reliability model selection based on deep learning with application to the optimal release problem. J Ind Eng Manage Sci 2019(1):43–58
40. Tickoo A, Kapur PK, Shrivastava AK, Khatri SK (2016) Testing effort based modeling to determine optimal release and patching time of software. Int J Syst Assur Eng Manage 7(4):427–434
41. Yamada S, Ohba M, Osaki S (1984) S-shaped software reliability growth models and their applications. IEEE Trans Reliab 33(4):289–292
42. Zhang X, Jeske DR, Pham H (2002) Calibrating software reliability models when the test environment does not match the user environment. Appl Stochast Models Bus Ind 18(1):87–99