

Effort-based Release and Patching Time of Software with Warranty using Change Point

Chetna Choudhary^a, P. K. Kapur^b, Sunil K. Khatri^c, and A. K. Shrivastava^{d,*}

^aAmity School of Engineering and Technology, Amity University, Uttar Pradesh, 201313, India

^bAmity Center for Interdisciplinary Research, Amity University, Uttar Pradesh, 201313, India

^cAmity Institute of Information Technology, Amity University, Uttar Pradesh, 201313, India

^dInternational Management Institute, Kolkata, West Bengal, 700027, India

Abstract

Due to the high dependency on software, the measurement of its performance has become vital. It is a common practice in the software industry to test products exhaustively before release so that the maximum number of faults is detected and removed. Fault detection and the removal rate are governed by many factors such as changes in testing environment, testing strategies, skills, efficiency, etc. The point at which a change in the fault detection rate occurs is known as the change point. Due to the increasing demand of good quality software in a short span and to remain in the market competition, firms are providing warranties on their products to assure reliability. The defects reported during the operational phase are fixed by providing patches. However, delivering patches after release demands extra effort and resources, which is costly and hence not economical for the firms. Considering the above factors, an effort-based cost model with change point and warranty is proposed in this work to determine the optimum release and patch time of a software by minimizing the overall cost. A numerical illustration is provided to validate the proposed cost model.

Keywords: software reliability; test effort; cost; modeling release; testing; patch; change point; warranty

(Submitted on October 3, 2017; Revised on February 27, 2018; Accepted on April 3, 2018)

© 2019 Totem Publisher, Inc. All rights reserved.

1. Introduction

With the emergence of advance techniques in the field of information technology, the unwavering quality of products turns into an essential concern when releasing software. To measure the software performance, we need to identify the factors affecting it. These measuring factors help developers and testers attain a clear understanding of system behavior to identify and fix the faults. Software reliability can be increased by investing more time and effort on testing, which results in higher testing costs.

Several researchers have proposed quantitative methods to establish correlation amongst detected faults and removed faults while performing testing on the basis of different set of assumptions [1-2], commonly identified as software reliability growth models, to frame several software related optimization problems. Yamada et al. [3] first propagated the concept of an effort-based reliability model. They pioneered in the implementation of the effort-based software reliability growth model. Huang and Kuo [4] proposed the logistic-based testing effort function-based software reliability growth model. Kapur et al. [5] extended their work by uniting the learning-based model with effort in SRGM; this approach was further protracted to versatile advancement display with learning process based on testing effort. Kapur et al. [6] developed a unified approach to model testing through the effort-based reliability growth model. Inoue and Yamada [7] suggested that the testing effort built up lognormal SRGM. Zhao et al. [8] discussed the effort-based model considering the imperfect debugging environment. Peng et al. [9] presented a two-stage model using test effort beneath an imperfect environment with respect to detection and correction. Zhang et al. [10] also demonstrated a unified approach used to model the reliability growth model under imperfect debugging with testing effort. Li et al. [11] suggested an S-molded SRGM under the

* Corresponding author.

E-mail address: kavinash1987@gmail.com

imperfect debugging phenomenon by means of testing effort. Kapur et al. [12] exhibited a framework for the upgradation of software, applying effort in testing in addition to the type of imperfect debugging.

Testing is required to remove faults from software, which in turn increases the reliability of the software with the removal of faults. Under specific conditions, software programs are executed to detect and remove faults to update the software. The amount of resources required, testing procedure, running condition, and fault density are various elements that can influence the removal of faults. To ensure that additional numbers of faults are removed inside this smaller timeframe, testers continue to include new methodologies and procedures that have not yet been utilized or perform hazard examination. Henceforth, the rate of fault identification will not be consistently uniform and can be altered at some moment of timeframe τ , known as the change point. Firstly, Zhao [13] consolidated the change point regarding the reliability, i.e., both programming and additional equipment. Meanwhile, Huang et al. [14] incorporated the change point in testing effort capacity to ascertain the software reliability growth modeling.

During testing, phase detection and correction of faults occur. However, testing software leads to the utilization of higher resources such as the workforce and handling hours, which comes at a high cost to organizations. Faults recognized during the operational stage cost more than those recognized during the testing stage [15]. Also, prolonged testing of software gives rise to increased reliability, but it results in high development cost and delay in the release of software. On the other hand, if the testing span is reduced, the development cost is reduced but accompanied by a higher risk of failure due to a higher number of faults lying in the software, which in turn increases the operational cost. Therefore, from the developer's viewpoint, it is fundamental to define the best time for release and to stop testing of the software. Nowadays, software firms continue testing even after software release, with the goal that clients experience failures during the operational stage and provide updates to fix the faults in the software. These updates are called patches. These patches are set guidelines that, when executed, fix the faults. Software firms provide patches to update the software and protect them from unwanted or unexpected functioning, which can cause failure. This in turn increases the software reliability.

When a product is released, it has been seen that either the failure rate declines or is static, as shown in Figures 1 and 2.

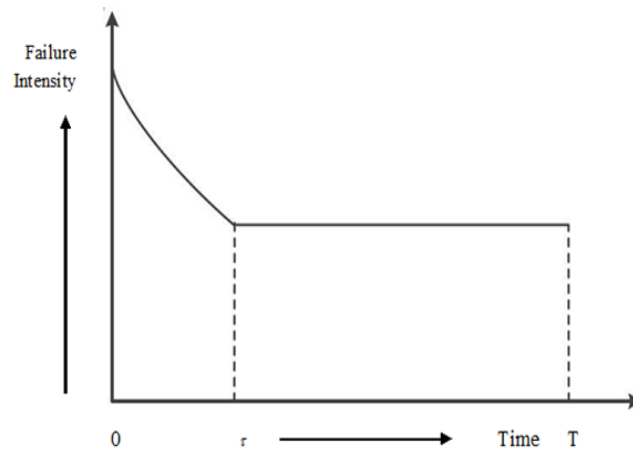


Figure 1. Software failure intensity without patching

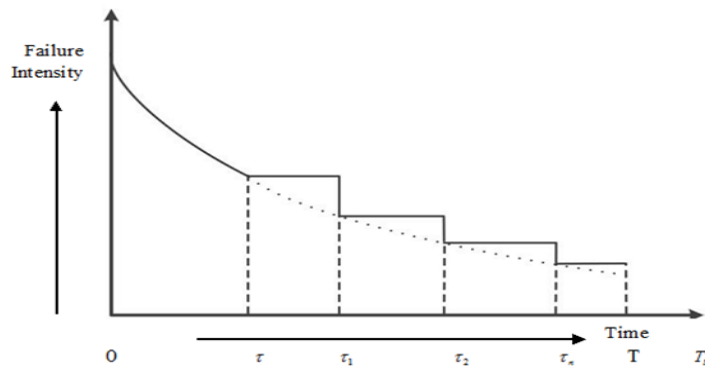


Figure 2. Software failure intensity with patching

Figure 1 presents the software failure intensity graph when no testing is performed after release. In Figure 2, it can be seen that after a time gap, the failure intensity decreases due to the release of patches. This presents the phenomenon that if detected faults are not removed in the operational phase, then a constant failure rate will be experienced by customers. If detected faults during the duration of testing time are distant, then equivalent failures will not take place again. This improves the software reliability, and the failure rate will depend on the usage during the operational phase.

Luo and Okamura [16] further considered a bug revelation process that is non-homogeneous and looks at both sporadic and aperiodic update administration models. Recent studies have put more emphasis on the field of security patch management. Dey et al. [17] presented a model to conclude the optimum release strategies for security patch management. Tickoo et al. [18] proposed resource-based optimal release and patching time of software. In their cost model, they considered that testing continues after release, and firms will provide updates/patches during the warranty period. The concept of providing a patch is similar to multi-upgradations, but there is technical difference between these two terms, i.e., multi-upgradations and patching. In the case of multi-upgradations, firms launch a newer version of the software with added functionalities. In the case of patching, firms provide updates for the same software to increase its reliability and remove the faults reported by the users. As of late, Kapur et al. [15] proposed a comprehensive structure utilized for ideal planning approaches to decide the optimum release and testing stop time of products and minimize the total cost of testing under reliability and budgetary constraints.

Based on the provided literature, we can see that people in industry follow a common practice of prolonged testing of software as a measure of reliability. No research has been conducted on how the testing effort applied affects the optimum release time and time to patch software under warranty with a change point. This has driven us to create a model to search for an answer for the above issue. Under the proposed framework, we have defined an effort-based unified framework to create a system that chooses the optimum release and testing stop time of a product so as to restrict the aggregate of assessed cost. The product testing and operational stages are administered by various distribution functions in various stages, i.e., in the pre-release and post-release stages of our proposed cost model.

The remainder of the paper is organized as follows. In Section 2, we will present the formulation of the model and vital assumptions fundamental to the framework. In Section 3, we will discuss the created cost framework utilizing change point. In Section 4, an outline is provided using an Obha [19] data set to approve the foreseen effort. Finally, Section 5 will present the summarization of work.

2. Modeling Framework

This section portrays the notation assumptions of the proposed cost model.

2.1. Notations Used

| | |
|-------------|--|
| $m(W(t))$ | Estimated faults removed in the time interval $(0, t]$ |
| $W(t)$ | Cumulative testing effort in the interval $(0, t]$ |
| $w(t)$ | Current testing-effort overheads ratio at testing time t , i.e., $\frac{d}{dt}W(t) = w(t)$ |
| b_1, b_2 | Proportion of fault detection before and after change point |
| a | A constant that signifies the number of faults primarily lying latent in the software |
| ν, k | Parameter of Weibull distribution |
| $F(W(t))$ | Failure distribution function |
| \bar{W} | Quantity of testing-effort ultimately utilized |
| $m(T_{lc})$ | Sum of faults during software lifecycle |
| τ | Change point |
| t | Software release time |
| t_1 | Patch release time |
| t_w | Warranty period |
| c_1 | Cost of testing per unit effort consumed |
| c_2 | Fault debugging cost before change point |
| c_3 | Fault debugging cost after change point prior to software release of the software |

| | |
|-------|--|
| c_4 | Fault debugging cost after release before first patch |
| c_5 | Fault debugging cost after first patch till warranty period |
| c_6 | Fault debugging cost after warranty period |
| c_7 | Debugging failure cost testified by user after testing stop time |

2.2. Assumptions

- The fault detection and removal phenomenon used in the cost model follows the non-homogenous Poisson process (NHPP) throughout the software lifecycle.
- Failures in the product occur because of the faults lying dormant in it.
- Estimated faults removed in the duration $(t, t + \Delta t)$ are directly relational to the remaining number of faults residual in the software.
- Failure detection and correction is an instantaneous process.
- The rate of fault detection and removal by means of testing effort intensity follows an independent and comparable distribution function, with $F(W(t)) = \int_0^{W(t)} f(x)dx$.
- The sum of all the faults lying dormant in the software is finite and removed completely with certainty, and no new faults are generated during removal.
- The software lifecycle is assumed to be finite.

2.3. Effort-based Software Reliability Growth Model

The presented growth model deliberates the variation of time with the utilization of testing resources. Illuminating the testing effort throughout this learning process, the Weibull function is used. The main preposition is represented by the following: "the rate of testing effort is proportional to the amount of available testing resources", i.e.,

$$\frac{dW(t)}{dt} = v(t) [\bar{W} - W(t)] \quad (1)$$

Where $v(t)$ represent the time-dependent rate of utilization of testing assets among the remaining resources.

With the condition $v(t) = v \cdot k \cdot t^{k-1}$, the Weibull function can be obtained as:

$$W(t) = \bar{W} \left(1 - e^{-v t^k} \right) \quad (2)$$

$$\frac{\frac{dm(W(t))}{dt}}{\frac{dW(t)}{dt}} = b(W(t))(a - m(W(t)))$$

Where

$$b(W(t)) = \frac{f(W(t))}{(1 - F(W(t)))}$$

By solving the above equation under the initial conditions $m(t=0) = 0$ and $W(t=0) = 0$, we get the mean value function for fault removal, given by:

$$m(W(t)) = aF(W(t)) \quad (3)$$

For the above developed effort-based SRGM, we have certain assumptions, which are described below.

Assumptions:

- For $t = 0$, $W(t) = 0$, $F(W(t)) = 0$.
- For $t > 0$, $W(t) > 0$, $F(W(t)) > 0$.
- As t increases, $W(t)$ increases, indicating the monotonically increasing nature of $F(W(t))$. The permanence of $F(W(t))$ can also be examined.
- As testing lingers for a considerably large time, i.e., $t \rightarrow \infty$, $W(t) = \bar{W}$, the significance of the distribution function $F(W(t))$ is $F(\bar{W})$. Hence, \bar{W} is a very large positive number, indicating the upper bound on the accessibility of available testing resources. Therefore, $F(\bar{W})$ can be supposed to be an order of 1.

3. Phase-Wise Framework for Fault Removal Model

In the given work, we have subdivided the product lifecycle into five subparts, i.e., the testing stage of the product and software before the change point stage, the product testing stage after the change point, the release of fix, the warranty stage, and the operational stage of software. The interval-wise lifecycle of the software is shown below in Figure 3. It is to be specified that every subdivision of fault detection and removal follows the NHPP phenomenon.

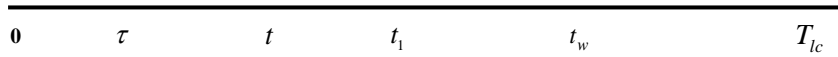


Figure 3. Software cycle with single patching and warranty

Testing cost is defined as the overall cost incurred throughout the testing time of software before it is released by the testing team. As numerous test cases are produced as well as materialized to assess the software, testing resources are also required for this. The cost of effort is computed using the result of testing cost per unit time and the testing effort. Let c_1 indicate the testing effort cost per unit time and W be the product testing effort work according to the framework. Thus, the measure of testing cost is represented by:

$$c_1 \times W(t)$$

Phase 1 $[0, \tau]$ (change point): During this phase, the testing team functions towards the testing strategies for detecting and correcting the fault. The total number of faults perceived during this phase is given by:

$$m(W(\tau)) = a \cdot F_1(W_1(\tau)) \quad (4)$$

The cost acquired in this phase is attributed to the tester, and it is represented by:

$$c_2 \cdot m(W(\tau)) \quad (5)$$

Where $F_1(W_1(\tau))$ is the rate of faults being removed from the product in $[0, \tau]$.

Phase 2 $[\tau, t]$: During this phase, testers detect and correct faults, and the number of faults being removed is represented as:

$$m(W(t - \tau)) = a \cdot (1 - F_1(W_1(\tau))) \left[1 - \frac{1 - F_2(W_2(t))}{1 - F_2(W_2(\tau))} \right] \quad (6)$$

The cost acquired during this stage is represented as:

$$c_3 \cdot m(W(t - \tau)) \quad (7)$$

Where $F_1(W_1(\tau))$ is the rate of faults being removed from the software in $[0, \tau]$ and $\left[1 - \frac{1 - F_2(W_2(t))}{1 - F_2(W_2(\tau))}\right]$ is the rate of faults being removal after the change point.

Phase 3 $[t, t_1]$ (post-release phase with patching): During this phase, update/fix is developed for the remaining number of faults. Therefore, the first update/patch is released at $[t_1]$ release time. Even though faults are being communicated by users, the update is released with the assurance from firms so that there are no costs placed on users. In this stage, users distinguish bugs and faults and give the data to the testing group. The sum of faults detected during this phase is represented as:

$$m(W(t_1 - t)) = a \left[1 - \left\{ 1 - \frac{(1 - F_1(W_1(\tau)))(1 - F_2(W_2(t)))}{1 - F_2(W_2(\tau))} \right\} \right] F_3(W_3(t_1 - t)) \quad (8)$$

The cost incurred is represented as:

$$c_4 \cdot m(W(t_1 - t)) \quad (9)$$

Where $F_3(W(t_1 - t))$ represents the rate of which the residual faults are detected after the release of software in $[t, t_1]$.

Phase 4 $[t_1, t_w]$: This stage is known as the post-fix/patch release phase under defined warranty. During this stage, users experience failures related to the issues not adjusted in the principal fix. Consequently, they report these failures to engineers to investigate the reason. It is verifiable that fix/patch is released toward the end of the guarantee, which encases the remaining faults. Due to the software warranty, firms must remove the failure causing faults. The total sum of faults corrected is represented as:

$$m(W(t + t_w - t_1)) = a \left[1 - \left\{ 1 - \frac{(1 - F_1(W_1(\tau)))(1 - F_2(W_2(t)))}{1 - F_2(W_2(\tau))} \right\} \right] (1 - F_3(W_3(t_1 - t))) F_4(W(t + t_w - t_1)) \quad (10)$$

The cost incurred in this stage can be written as:

$$c_5 \cdot m(W(t + t_w - t_1)) \quad (11)$$

The equation $\left[1 - \left\{ 1 - \frac{(1 - F_1(W_1(\tau)))(1 - F_2(W_2(t)))}{1 - F_2(W_2(\tau))} \right\} \right] (1 - F_3(W_3(t_1 - t)))$ represents the residual sum of faults remaining in the first or second stage/phase. $F_4(W(t + t_w - t_1))$ is the rate of fault removal in the $[t_1, t_w]$ interval.

Phase 5 $[t_w, T_{lc}]$ (post-warranty phase): In this stage, as the product guarantee is damaged, firms do not provide any sort of assistance to the failures that have occurred yet. At times, they give confirmation to users of the evacuation of deficiencies directly through the product lifecycle. This has been consolidated while displaying the aggregated cost function. During this period, failures experienced by users are suitable to the remaining sum of issues and left undetected during the post-guarantee stage. The sum of faults detected is represented by:

$$m(W(T_{lc} - (t + t_w))) = a \left[1 - \left\{ 1 - \frac{(1 - F_1(W_1(\tau)))(1 - F_2(W_2(t)))}{1 - F_2(W_2(\tau))} \right\} \right] (1 - F_3(W_3(t_1 - t))) (1 - F_4(W(t + t_w - t_1))) F_5(W(T_{lc} - (t + t_w))) \quad (12)$$

The cost caused in this stage can be communicated as:

$$c_6 \cdot m(W(T_{lc} - t + t_w)) \quad (13)$$

Where $\left[1 - \left\{1 - \frac{(1 - F_1(W_1(\tau)))(1 - F_2(W_2(t)))}{1 - F_2(W_2(\tau))}\right\}\right] (1 - F_3(W_3(t_1 - t)))(1 - F_4(W(t + t_w - t_1)))$ expresses the residual number of faults left undetected in the proceeding intervals, while $F_5(W(T_{lc} - (t + t_w)))$ is the fault removal rate in the interval $[t_w, T_{lc}]$.

The total cost function can be represented as:

$$\begin{aligned} \text{Cost} = & c_1 \cdot w(t) + c_2 \cdot m(W(\tau)) + c_3 \cdot m(W(t - \tau)) + c_4 \cdot m(W(t_1 - t)) + c_5 \cdot m(W(t + t_w - t_1)) \\ & + c_6 \cdot m(W(T_{lc} - (t + t_w))) \end{aligned} \quad (14)$$

4. Case Study

In this section, we present a numerical example for model validation purposes. A data set provided by Obha [19], which consists of 328 faults being removed through 19 weeks with a CPU time consumption of 47 hours, is chosen.

For simplicity and numerical illustration purposes, we use the exponential distribution function for the fault detection and removal process. Based on the assumptions for the fault detection process, we get the fault distribution function $F_i(W(t)) = 1 - e^{-b_i W_i(t)}$, and we also assume that the testing effort function has an identical rate for every phase, i.e., $W_i(t) = W(t)$. The parameter estimation performed using SPSS is presented by:

Sum of testing effort eventually utilized (\bar{W}) = 799.3, parameter for scale (v) = 0.002, parameter for shape (k) = 1.115, initial number of failures (a) = 614.6, rate of fault detection before change point (b_1) = 0.130, rate of fault detection after change point (b_2) = 0.142, and software lifecycle (T_{lc}) = 100. This shows that the software has approximately 615 faults initially, which were detected and removed with the help of testers and with fault detection rates b_1 and b_2 . Generally, users and testers have diverse fault detection rates because of various proficiency. Suppose r_1, r_2, r_3 are fault detection rate ratios in the third, fourth, and fifth interval. On the basis of suppositions, the number of faults detected and removed during each phase can be characterized along with the cost associated. The number of faults identified and removed during $[0, \tau]$ can be depicted as:

$$m(W(\tau)) = a F_1(W_1(\tau)) = a(1 - e^{-b_1 W_1(\tau)})$$

Therefore, from Equation (12), the cost associated while detecting and removing $m(W(\tau))$ of faults is expressed through:

$$c_2 \cdot m(W(\tau)) = c_2 \cdot a(1 - e^{-b_1(W_1(1 - e^{-v\tau^k}))}) \quad (15)$$

The sum of faults detected and removed in $[\tau, t]$ is shown as:

$$\begin{aligned} m(W(t - \tau)) = & a(1 - F_1(W_1(\tau))) \left[1 - \frac{1 - F_2(W_2(t))}{1 - F_2(W_2(\tau))} \right] = a \cdot (1 - (1 - e^{-b_1 W(\tau)})) \left(1 - \frac{(1 - (1 - \exp(-b_2 \cdot W(t))))}{1 - (1 - \exp(-b_2 \cdot W(\tau)))} \right) \\ = & a \cdot e^{-b_1 W_1(1 - e^{-v\tau^k})} \cdot (1 - e^{-b_2 W_1(-b_1 W_1(1 - e^{-v(t-\tau)^k}))}) \end{aligned}$$

Therefore, from Equation (14), the associated cost for detection and removal of $m(W(t - \tau))$ of faults is expressed as:

$$c_3 \cdot m(W(t-\tau)) = c_3 a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot (1-e^{-b_2 w_1 (-b_1 w_1 (1-e^{-v(\tau-\tau)^k}))}) \quad (16)$$

The sum of faults detected and removed in $[t, t_1]$ is represented as:

$$\begin{aligned} m(W(t_1-t)) &= a \left[1 - \left\{ 1 - \frac{(1-F_1(W_1(\tau)))(1-F_2(W_2(t)))}{1-F_2(W_2(\tau))} \right\} \right] F_3(W_3(t_1-t)) \\ &= a \left[1 - \left\{ 1 - \frac{(1-(1-e^{-b_1(W_1(\tau)})))(1-(1-e^{-b_2(W_2(t))}))}{1-(1-e^{-b_2(W_2(\tau))})} \right\} \right] (1-e^{-b_2 r_1(W_3(t_1-t))}) \\ &= a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau-\tau)^k})} \cdot (1-e^{-b_2 r_1 w_1 (1-e^{-v(\eta-\tau)^k})}) \end{aligned}$$

Therefore, from Equation (16), the associated cost during detection and removal of $m(W(t_1-t))$ of faults is represented by:

$$c_4 m(W(t_1-t)) = c_4 a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau-\tau)^k})} \cdot (1-e^{-b_2 r_1 w_1 (1-e^{-v(\eta-\tau)^k})}) \quad (17)$$

The sum of faults detected and removed in $[t_1, t_w]$ is represented by:

$$\begin{aligned} m(W(t+t_w-t_1)) &= a \left[1 - \left\{ 1 - \frac{(1-F_1(W_1(\tau)))(1-F_2(W_2(t)))}{1-F_2(W_2(\tau))} \right\} \right] (1-F_3(W_3(t_1-t))) F_4(W(t+t_w-t_1)) \\ &= a \left[1 - \left\{ 1 - \frac{(1-(1-e^{-b_1(W_1(\tau)})))(1-(1-e^{-b_2(W_2(t))}))}{1-(1-e^{-b_2(W_2(\tau))})} \right\} \right] (1-(1-e^{-b_2 r_1(W_3(t_1-t))}))(1-e^{-b_2 r_2(W(t+t_w-t_1))}) \\ &= a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 r_1 w_1 (1-e^{-v(\eta-\tau)^k})} (1-e^{-b_2 r_2 w_1 (1-e^{-v(t+t_w-\eta)^k})}) \end{aligned}$$

Therefore, from Equation (18), the associated cost for detection and removal of $m(W(t+t_w-t_1))$ of faults is depicted as:

$$c_5 m(W(t+t_w-t_1)) = c_5 a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 r_1 w_1 (1-e^{-v(\eta-\tau)^k})} (1-e^{-b_2 r_2 w_1 (1-e^{-v(t+t_w-\eta)^k})}) \quad (18)$$

The total number of faults detected and removed in $[t_w, T_{lc}]$ is represented by:

$$\begin{aligned} m(W(T_{lc}-(t+t_w))) &= a \left[1 - \left\{ 1 - \frac{(1-F_1(W_1(\tau)))(1-F_2(W_2(t)))}{1-F_2(W_2(\tau))} \right\} \right] (1-F_3(W_3(t_1-t))) \\ &\quad (1-F_4(W(t+t_w-t_1))) F_5(W(T_{lc}-(t+t_w))) \\ &= a \left[1 - \left\{ 1 - \frac{(1-(1-e^{-b_1(W_1(\tau)})))(1-(1-e^{-b_2(W_2(t))}))}{1-(1-e^{-b_2(W_2(\tau))})} \right\} \right] \\ &\quad (1-(1-e^{-b_2 r_1(W_3(t_1-t))}))(1-(1-e^{-b_2 r_2(W(t+t_w-t_1))}))(1-e^{-b_2 r_3(W(T_{lc}-(t+t_w)))}) \\ &= a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 r_1 w_1 (1-e^{-v(\eta-\tau)^k})} \cdot e^{-b_2 r_2 w_1 (1-e^{-v(t+t_w-\eta)^k})} \cdot (1-e^{-b_2 r_3 w_1 (1-e^{-v(T_{lc}-(t+t_w))})}) \end{aligned}$$

Therefore, from Equation (20), the associated cost for the detection and removal of $m(W(T_{lc} - t + t_w))$ of faults is expressed as:

$$c_6 m(W(T_{lc} - t + t_w)) = c_6 a e^{-b_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 r_1 w_1 (1-e^{-v(\tau)^k})} \cdot e^{-b_2 r_2 w_1 (1-e^{-v(\tau)^k})} \cdot (1-e^{-b_2 r_3 w_1 (1-e^{-v(\tau)^k})}) \quad (19)$$

After summing all the costs phase-wise with the cost of testing e , the aggregate cost with single fixing is determined by:

$$\begin{aligned} Cost = & c_1 w + c_2 a (1 - e^{-b_1 (W_1 (1-e^{-v\tau^k}))}) + c_3 a \cdot e^{-b_1 (W_1 (1-e^{-v\tau^k}))} \cdot (1 - e^{-b_2 W_1 (1-e^{-v(\tau-\tau)^k})}) + c_4 a \cdot e^{-b_1 (W_1 (1-e^{-v\tau^k}))} \\ & \cdot e^{-b_2 W_1 (1-e^{-v(\tau-\tau)^k})} \cdot (1 - e^{-b_2 W_1 r_1 (1-e^{-v(\tau-\tau)^k})}) + c_5 a \cdot e^{-b_1 (W_1 (1-e^{-v\tau^k}))} \cdot e^{-b_2 W_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 w_1 r_1 (1-e^{-v(\tau-\tau)^k})} \\ & \cdot (1 - e^{-b_2 w_1 r_1 (1-e^{-v(\tau-\tau)^k})}) + c_6 a \cdot e^{-b_1 (W_1 (1-e^{-v\tau^k}))} \cdot e^{-b_2 W_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 w_1 r_1 (1-e^{-v(\tau-\tau)^k})} \cdot e^{-b_2 w_1 r_1 (1-e^{-v(\tau-\tau)^k})} \\ & \cdot (1 - e^{-b_2 w_1 r_3 (1-e^{-v(\tau-\tau)^k})}) \end{aligned} \quad (20)$$

To perform the numerical calculations of the proposed cost model, we have assumed the following values of cost parameters: $c_1 = \$30$, $c_2 = \$20$, $c_3 = \$45$, $c_4 = \$165$, $c_5 = \$165$, and $c_6 = \$325$. These values can vary depending on the different conditions. Additionally, let us assume that organizations provide a user warranty $w = 26$ (weeks) on the product, and the fault detection rate proportions are $r_1 = 0.2$, $r_2 = 0.3$, and $r_3 = 0.5$. It should be noticed that parameters obtained are simply through knowledge. In particular, the cost per unit fault is dependent upon the amount of resources utilized during that period. It is prominent that the post-warranty stage for products is extraordinarily long in contrast to other stages of software; in this way, a large amount of resources are used. Moreover, the quantities of faults removed during this span are lesser, as most extreme quantities of faults are redressed inside testing and in the warranty stage. That is the reason why it costs more per unit fault removed for the time of the post-warranty term. According to the related supposition, the cost per unit fault removed in other stages can be given. The warranty time and release time of patch are after the release of the software, i.e., t , on the grounds that fixes and assurance can be given to the user after release. By substituting the cost values in condition (27) and carrying out enhancement utilizing MAPLE for the aggregate budget work, we secure the optimum result as software release time $t^* = 21.88$ weeks, the ideal time to release fix as $t_1^* = 3.14$ weeks, and the optimal cost as \$17171.33. The chart shown in Figure 4 features the ideal release and fixing/patching time. The amount of resources consumed before the release time is 48.35 hours and 5.70 hours for releasing the first patch. This means the ideal time to release the software based on the given data is approximately 22 weeks, and it takes three weeks to obtain the optimum cost after releasing the first patch.

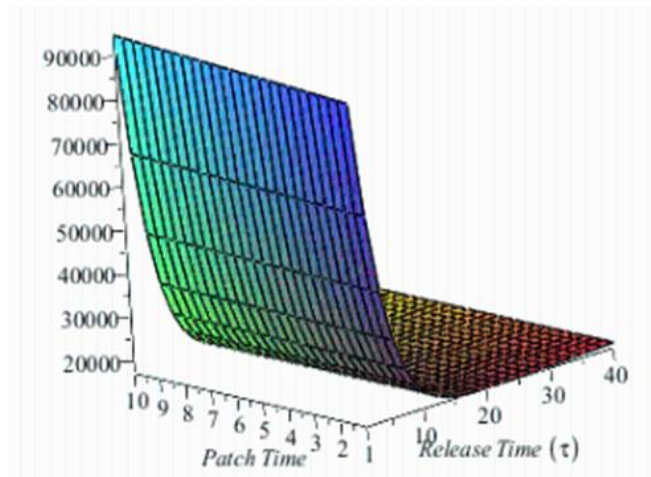


Figure 4. Optimal release and patching time

5. Conclusions and Future Scope

In this paper, we proposed a complete effort-based model by considering various testing conditions for determining the optimum release and fixing time under a given warranty period to minimize the total expected cost. In the present work, we computed the time of single patch release within the warranty period, but we can extend the proposed model for more than

one patch. This model additionally furnishes developers and engineers with the data of the supplementary testing effort required to assess the software and products in the wake of altering the testing methodologies. Because of the monotonous environment, designers can release the ensuing fixes and patches with negligible charges. Based on the proposed methodologies, venture managers can schedule the release and testing period of the software and products and will have the capacity to quantify the execution of the released software and products. This helps organizations achieve a worthwhile position in a focused commercial centre.

References

1. P. K. Kapur, H. Pham, A. Gupta, and P. C. Jha, "Software Reliability Assessment with OR Application," Springer, Berlin, 2011
2. N. Ahmad, M. G. M. Khan, and L. S. Rafi, "A Study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models with Imperfect Debugging," *International Journal of Quality & Reliability Management*, Vol. 27, No. 1, pp. 89-110, 2010
3. S. Yamada, H. Ohtera, and H. Narihisa, "Software Reliability Growth Models with Testing-Effort," *IEEE Transactions on Reliability*, Vol. 35, No. 1, pp. 19-23, 1986
4. C. Y. Huang and S. Y. Kuo, "Analysis of Incorporating Logistic Testing-Effort Function into Software Reliability Modeling," *IEEE Transactions on Reliability*, Vol. 51, No. 3, pp. 261-270, 2002
5. P. K. Kapur, D. N. Goswami, and A. Bardhan, "A General Software Reliability Growth Model with Testing Effort Dependent Learning Process," *International Journal of Modelling and Simulation*, Vol. 27, No. 4, pp. 340-346, 2007
6. P. K. Kapur, S. Ompal, A. G. Aggarwal, and R. Kumar, "Unified Framework for Developing Testing Effort Dependent Software Reliability Growth Models," *WSEAS Transactions on Systems*, Vol. 4, No. 8, pp. 521-531, 2009
7. S. Inoue and S. Yamada, "Lognormal Process Software Reliability Modeling with Testing-Effort," *Journal of Software Engineering and Applications*, pp. 8-14, 2013
8. Q. Zhao, J. Zheng, and J. Li, "Software Reliability Modeling with Testing-Effort Function and Imperfect Debugging," *TELKOMNIKA*, Vol. 10, No. 8, pp. 1992-1998, 2012
9. R. Peng, Y. F. Li, W. J. Zhang, and Q. P. Hu, "Testing Effort Dependent Software Reliability Model for Imperfect Debugging Process Considering Both Detection and Correction," *Reliability Engineering and System Safety*, Vol. 126, pp. 37-43, 2014
10. N. Zhang, G. Cui, and H. Liu, "Considering Detection Effort and Correction Effort for Software Reliability Analysis," *Journal of Computational Information Systems*, Vol. 8, No. 19, pp. 7991-8000, 2012
11. Q. Li, H. Li, and M. Lu, "Incorporating S-Shaped Testing-Effort Functions into NHPP Software Reliability Model with Imperfect Debugging," *Journal of Systems Engineering and Electronics*, Vol. 26, No. 1, pp. 190-207, 2015
12. P. K. Kapur, O. Singh, A. K. Shrivastava, and J. N. P. Singh, "A Software Up-Gradation Model with Testing Effort and Two Types of Imperfect Debugging," in *Proceedings of International Conference on Futuristic Trends in Computational Analysis and Knowledge Management*, 2015
13. M. Zhao, "Change-Point Problems in Software and Reliability," *Communications in Statistics Theory and Methods*, Vol. 22, No. 3, pp. 757-768, 1993
14. C. -Y. Huang and M. R. Lyu, "Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency," *IEEE Transactions on Reliability*, Vol. 54, pp. 583-591, 2005
15. P. K. Kapur, S. K. Khatri, O. Singh, and A. K. Shrivastava, "When to Stop Testing under Warranty using SRGM with Change Point," in *Proceedings of International Conference on IT in Business, Industry & Govt.*, pp. 200-205, 2014
16. C. Luo, H. Okamura, and T. Dohi, "Optimal Planning for Open Source Software Updates," in *Proceedings of the Institution of Mechanical Engineers Part O Journal of Risk and Reliability*, 2015
17. D. Dey, A. Lahiri, and G. Zhang, "Optimal Policies for Security Patch Management," *INFORMS Journal on Computing*, Vol. 27, No. 3, pp. 462-477, 2015
18. A. Tickoo, P. K. Kapur, and A. K. Shrivastava, "Testing Effort based Modeling to Determine Optimal Release and Patching Time of Software," *International Journal of System Assurance Engineering and Management*, Vol. 7, No. 4, pp. 427-434, 2016
19. M. Obha, "Software Reliability Analysis Models," *IBM Journal of Research Development*, Vol. 28, No. 4, pp. 428-443, 1984