Int J Syst Assur Eng Manag (December 2016) 7(4):427–434
DOI 10.1007/s13198-016-0470-y

ORIGINAL ARTICLE

Testing effort based modeling to determine optimal release and patching time of software

Anshul Tickoo[1] · P. K. Kapur[2] · A. K. Shrivastava[3] · Sunil K. Khatri[4]

Received: 18 March 2016 / Revised: 27 April 2016 / Published online: 14 May 2016

Abstract In this era of information technology, our dependence on software systems is increasing day by day. This dependence on software systems has increased the pressure on software firms to fulfill the customer's demand for highly reliable software. On the other hand, for ensuring high reliability of the software prolonged testing is required, which consumes large amount of resources hence not feasible in the current stiff market competition. Further delay in release can cost a lot in terms of market opportunity. Therefore, to sustain in the market, firms are releasing the software early and removing the remaining number of bugs by updating with patches. A patch is a piece of software designed to update a computer program or its supporting data, to fix or improve it. With such patches usually called bug fixes, firms improve the usability or performance of the software. Providing patches needs extra amount of effort and manpower which costs high. Also early patch release may result in improper removal of bugs and late release can increase the risk of more of failures in the operational phase To overcome the above issues we have proposed a testing effort based cost model to determine the optimal release and patch time of a software so that the total cost is minimized. In the proposed cost model developing team continues removing the faults even after software release. Further, we have taken different distribution function in pre and post release phase (before and after patching) to develop the proposed cost model. Numerical illustration is provided at the end of the paper for validation of the proposed cost model.

✉ Anshul Tickoo
anshultickoo@hotmail.com

[1] Amity School of Engineering, Amity University, Noida, U.P., India

[2] Amity Center for Interdisciplinary Research, Amity University, Noida, U.P., India

[3] Research Development Center, Asia Pacific Institute of Management, Delhi, India

[4] Amity Institute of Information Technology, Amity University, Noida, U.P., India

Keywords Software reliability growth model (SRGM) · Testing effort · Release · Patch · Testing

1 Introduction

Information technology is playing a crucial role in today's fast moving life. Almost everything is more or less dependent on software or driven by software technology. This dependence on software has increased the demand of reliable software in no time. Reliability of software is governed by the amount of testing time and resources (effort) consumed during the software development phase. Software testing is one of the most important phase of software development life cycle as it generally consumes more than 50 % of the total budget. Also release and quality of the software is dependent on the testing phase. Measuring the quality of software is a very difficult job in the software industry. It is important to measure reliability of software before release, also quantifying reliability help software firms in quantifying the behaviour of the system and allocate testing resources. More is the testing greater is the chance of getting a reliable software. But on the other hand to match with the pace of market firms don't want to spend too much of time on software testing. Software failure is said to be occurred if on giving a specific input we don't get the desired output, on the other hand software fault is incorrect code, data definition or statement in a program that has affected the program in such a way so that it will not give the desired output Musa (Musa 2004). During the testing phase of software development life cycle, testers learn and improve their skills. This phenomenon of learning is the result of familiarity of the testers with the software program. Testing team finds faults and developers remove them during testing phase but this phase consumes large amount of testing resources i.e. CPU hours, manpower etc. To find out the relationship between faults detected and removed during testing various testing resources (effort) based software reliability growth models (SRGMs) have been proposed under different set of assumptions in the last four decades (Goel and Okumoto 1979; Huang et al. 2007; Kapur et al. 2011; Kuo et al. 2001; Lin and Huang 2008; Musa 2004; Pham 2006; Chatterjee and Singh 2014). Yamada et al. (1986) first introduced effort based SRGM. Huang and Kuo (2002) developed a SRGM by incorporating logistic testing effort function. Kapur et al. proposed (Kapur et al. 2007, 2008) proposed a SRGM with testing effort based learning process. They further extended their concept to propose a flexible SRGM with testing effort based learning process. Kapur et al. (2009) proposed a unified framework for modelling testing effort based SRGM. Ahmad et al. (2010) proposed a S- shaped SRGM with testing effort under imperfect debugging phenomenon. Inoue and Yamada (2013) proposed a testing effort based lognormal SRGM. Zhao et al. (2005, 2012) proposed a testing effort based SRGM with imperfect debugging. Zhang et al. (2012) proposed a queuing theory based SRGM incorporating testing effort. They further proposed a two-step fault detection and correction model for testing effort based on queuing approach Zhang (2015) and Zhang et al. (2013). Peng et al. (2014) proposed a two stage detection correction based SRGM with testing effort under imperfect debugging environment. Zhang et al. (2014) proposed a unified framework for modelling SRGM with testing effort under the effect of imperfect debugging. Recently Li et al. (2015) incorporated S-shaped testing-effort functions into software reliability model with imperfect debugging. Software reliability modelling with testing effort is not limited to single version only. Researchers are working on to include testing effort in software reliability growth modelling for multiple versions of a software. The study of software reliability modelling with testing effort for multiple version is proposed by Singh et al. (2012). Recently Kapur et al. (2015) proposed generalized framework for a software up-gradation model with testing effort and two types of imperfect debugging.

Testing is done to remove faults from the software and hence ensuring reliability. But it consumes large amount of resources such as manpower, CPU hours etc. which incurs cost to the developer. On the other hand if faults are not removed in the testing phase, then users will face failures during the operational phase and the cost of debugging faults in the operational phase is much higher than that in the testing phase (Musa 2004; Singh et al. 2015). Prolonged testing increase the software reliability but increases the software development cost and also delays the software release which causes loss in terms of market opportunity. However spending insignificant amount testing time for error removal reduces the software development cost, but increases the risk of higher number of failures in the operational phase. Hence determining the optimal release time is very important from developer's perspective. Literature available in this field, concentrates mainly on minimizing the total expected cost, satisfying the reliability requirement at the lowest total cost as well as sensitivity analysis of the optimal release time (Huang and Lin 2010; Huang and Lyu 2005; Kapur et al. 2011; Kapur and Garg 1990; Li et al. 2010; Lo et al. 2005; Pasquini et al. 1996; Pham 2006; Zachariah 2015).

Although it is impossible for a testing team to come out with software which is completely free of errors. But in order to ensure that users face least number of errors during operation phase, now software firms continue removing the faults even after release (Apple 2015; HP 2015). During this post release phase software firms update the software by providing patches to users. A patch, sometimes also called a fix, is a small program of software that is used to fix errors that successfully deceived the testing team during pre-release testing phase of software. After software release, there is either a decrease in the failure rate function or it remains constant, depending on whether there is a growth in software reliability during the operational phase or not (see Figs. 1, 2). This phenomenon can be understood

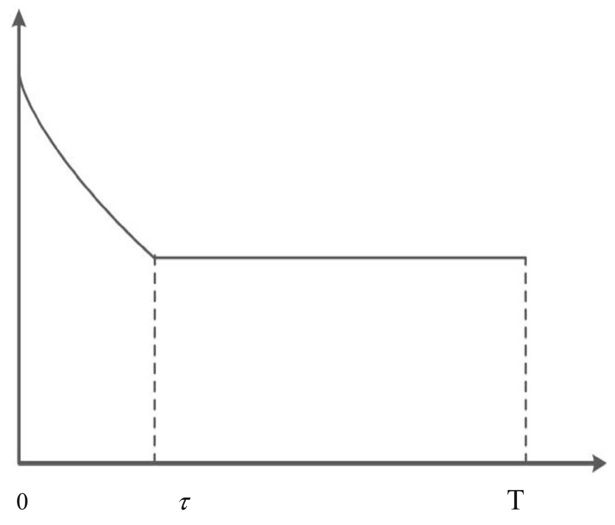

Fig. 1 Software reliability growth without patching

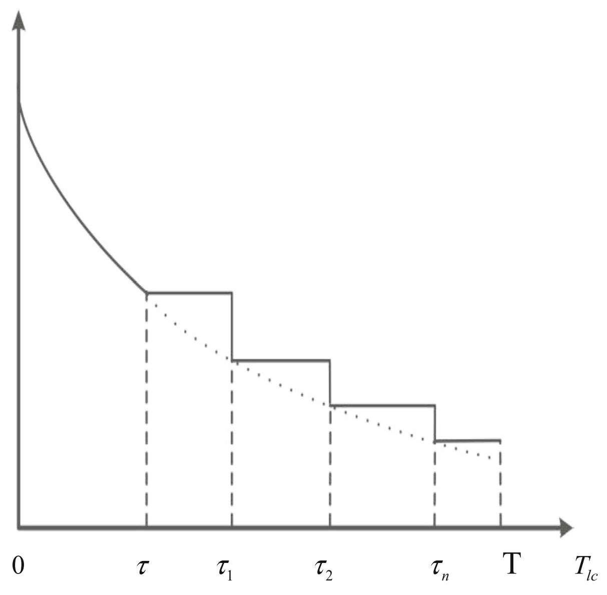

Fig. 2 Software reliability growth with patching

as follows. If the removal of detected faults was not possible during the operational phase then the failure rate remains constant. On the other hand if the detected faults during testing phase are removed and similar failures do not occur again, then the software reliability increases. Yang and Xie (2000) emphasize that the latter is more likely in practice where errors encountered are corrected, bundled as a software patch and presented to the user.

The topic of security patch management is closely related to general software update management, which has received attention. Cavusoglu and Zhang (2008) studied the security patch release and management problems from the perspective of both vendor and the firm. A game-theoretic model was developed by them to study how the interaction between vendor and firm balances the cost and benefits of patch management. Subsequently, Okamura et al. (2009) and Luo et al. (2015) considered a non-homogeneous bug-discovery process and studied both the periodic and aperiodic patch/update management models. Recently a cost model for finding optimal release policies for security patch management was developed by Dey et al. (2015). Arora et al. (2006, 2008) proposed a cost model in which the trade-off between releasing a buggy software product and investments in patching it later is shown. They also show the benefit of releasing the software early and patching it later. Jiang et al. (2012) also proposed a software scheduling policy where they show the advantage of releasing the software early and to continue testing after the release. The idea was to increase the testing base from a limited number of testers to significantly large number of testers which contributes to customer side testing of the software. Kapur and Shrivastava (2015) proposed a generalized framework to determine the optimal release and testing stop time and showed the benefit of early release and continuing testing even after release.

Practically it is not feasible to release an update every time a fault is reported by the end user. Therefore an important problem for the developer is to determine how and when to release updates and stop testing to remedy faults in its software. Also the existing literature in software release time problems (Kapur et al. 2011; Peng et al. 2014) did not take account the difference between the user's operational environment and the software testing phase for determining optimal release time of software. However several authors have worked on the reliability modelling in different phases of a software and various reliability assessment methods have been proposed for the operational phase (Huang et al. 2005; Huang and Lin 2010; Jain and Priya 2005; Pasquini et al. 1996). A generalized framework was proposed by Huang et al. (2000) to prove that fault detection rate changes from testing to operational phase. A SRGM was proposed by Zhao et al. (2012) to show that due to environmental changes and inherent fault detection rate there is a change in software reliability from testing to operational phase.

The above literature review shows that academic research on software update/patch management is still at an early stage. However, the current software industry has started implementing the concept of releasing early and updating the software by providing patch release. The biggest advantage of early release is capitalizing market opportunity while post release testing ensures higher software reliability. Although software testing and updating activities are very common in practice, not many attempts can be found on modeling their effects on total cost incurred during the lifecycle of software in an integrated manner. In this paper we have taken the first step towards filling this gap by considering both software testing and updating activities jointly to propose a generalized framework for developing a testing effort based cost model to determine the optimal release and patching time of software to minimize the total expected cost. Rest of the paper is organized as follows. Section 2 deals with model assumptions, notations and formulation of the proposed cost model. Section 3 deals with the modeling of the testing effort based cost function with patching. In Sect. 4, numerical example is provided using a real life data set to validate the proposed work. Finally conclusion has been drawn in Sect. 5.

2 Model formulation

This section deals with model formulation. The applied SRGM and its assumptions are firstly described to provide a basis for the model of software failure process. Then, the software testing procedure and updating policies are described in order to minimize the total expected cost. Section 2.1 describes the notations that are used in modeling the proposed framework.

2.1 Notations

m(W(t))	Expected number of faults removed in the time interval (0,t]
W(t)	Cumulative testing effort in the interval (0.t]
w(t)	Current testing-effort expenditure rate at testing time t. i.e. $\frac{d}{dt}W(t) = w(t)$
b	Constant
b_i	Fault detection rate in ith phase
a	Constant, representing the number of faults lying dormant in the software at the beginning of testing
v, k	Parameter of Weibull distribution
F(W(t))	Failure distribution function
F_i(W(t))	Failure distribution function in ith phase
$\overline{W}$	Amount of testing-effort eventually consumed
$W_i(t)$	Amount of testing effort consumed in ith phase
$m(T_{lc})$	Number of faults removed during the lifecycle of the software
τ	Release Time of the software
τ_i	ith patch release time
c_1	Cost of testing per unit testing effort expenditure
c_2	Market opportunity cost
c_3	Cost of detection and removal of faults by testers and developers respectively before the release of the software
c_4	Cost of removing faults reported by user after software release in pre patching period
c_5	Cost of removing faults reported by user after patch release in post patching period

2.2 Assumptions

The proposed model is based upon the following basic assumptions:

1. The process of fault removal follows non-homogeneous Poisson process(NHPP) throughout the software lifecycle.
2. The software system might fail at random owing to the faults lying in it.
3. The expected number of faults detected in the time interval $(t, t + \Delta t)$ is directly proportional to average number of faults remaining in the software.
4. Fault detection leads to its immediate correction.
5. The fault detection/correction rate with respect to testing effort intensity follows independent and identically distribution function, where

$$F(W(t)) = \int_0^{W(t)} f(x)dx$$

6. All faults are removed perfectly.
7. Total numbers of faults lying dormant in the software are finite.
8. Lifecycle of the software is finite.
9. Cost of patching is negligible.
10. Market opportunity cost which is assumed to be monotonically increasing, twice continuously differentiable convex function ofτ. Since the qualitative conclusion of the study is not much affected by the actual functional form of market opportunity cost, therefore we will use the form used by Jiang and Sarkar (Jiang et al. 2012).

2.3 Effort based software reliability growth model

The SRGM developed in this paper considers the time dependent variation in the consumption of testing resources. To elucidate the testing effort in this study, we used Weibull function. The underlying assumption is described as, "The testing effort rate is proportional to the testing resources available".

$$\text{i.e.} \frac{dW(t)}{dt} = v(t)\left[\overline{W} - W(t)\right] \quad (1)$$

where v(t) is the time dependent rate of consumption of testing resources, with respect to remaining available resources.

If $v(t) = v.k \cdot t.^{k-1}$, we get Weibull function as:

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

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

where $b(W(t)) = {f(W(t))}/{(1 - F(W(t))}$ is the failure detection (fault removal) rate.On solving the above equation under the initial conditions of m(t = 0) = 0 and W(t = 0) = 0 we get

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

Here $F(W(t))$ is the testing effort dependent probability distribution function for fault correction times. It can be noted that $F(W(t))$ so defined satisfy all the properties of probability distribution functions.

1. In this paper, we have used Weibull type testing effort function which satisfies the property that, at

   $t = 0,\ W(t) = 0$ and $F(W(t)) = 0$.
2. For $t > 0,\ W(t) > 0$ and $F(W(t)) > 0$.
3. As t increases, $W(t)$ also increases indicating monotonically increasing nature of $F(W(t))$. Similarly the continuity of $F(W(t))$ can also be explained.
4. As testing continues for an infinitely large time i.e. $t \to \infty,\ W(t) \to \overline{W}$, the corresponding value of distribution function $F(W(t))$ is $F(\bar{W})$. Here $\overline{W}$ is very large positive number representing the upper bound on the availability of the amount of testing resources available. Therefore, $F(\bar{W})$ can be assumed to be of order 1.

3 Effort based cost model for single patching

In this case lifecycle of the software is divided into three phases viz. pre release testing phase [0, τ], post release pre patching phase [τ, τ_1] and operational phase after patch release [τ_1, T_{lc}] as shown in Fig. 3 below. Also it is to be noted that since we have assumed that fault detection process follows NHPP throughout the software lifecycle, hence it will follow NHPP in each phase.

Here $\tau_1 = \tau + \tau_1'$, where τ_1' is the time duration between release and first patch time.

In this section we will discuss the total cost incurred for detecting/removing '*a*' number of bugs which were found during the software lifecycle. The total cost is sum of testing cost, market opportunity cost and the cost incurred for removing faults in each phase which are described below.

Testing cost Testing cost associated with testing effort expenditure is given by

$$c_1 W(\tau) \tag{4}$$

Market opportunity cost The market opportunity cost refers to the market related cost-or the opportunity loss cost due to late entry into the market.

Market opportunity cost is given by

$$c_2 \tau^2 \tag{5}$$

where $\tau^2$is the functional form of market opportunity cost c_2 as in Jiang et al. (2012).

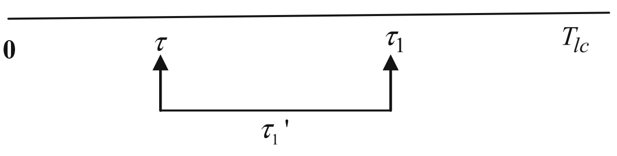

Fig. 3 software cycle with single patching

Phase 1: [0, τ] (Pre-release testing phase) In this phase testing team works to detect failure/fault, i.e. this is the testing period before the release of the software. The tester detects the faults and notifies about it to the developing team for removal process. The total number of faults removed in this interval is given by

$$m(W(\tau)) = a F_1(W_1(\tau)) \tag{6}$$

Cost incurred due to detection and removal of the faults in this phase it is given by

$$c_3\, m(W(\tau)) \tag{7}$$

where $F_1(W_1(\tau))$is the rate by which the faults are removed from the software in the interval [0, τ].

Phase 2: [τ, τ_1] (Post release phase before patching) This period refers to the post release phase before patching where bugs are reported by the users and fixed by the developing team of the software company. Total number of faults detected in this phase is given by

$$\begin{aligned} m(W(\tau_1 - \tau)) &= (a - m(W(\tau)))\, F_2(W_2(\tau_1 - \tau)) \\ &= a(1 - F_1(W_1(\tau)))\, F_2(W_2(\tau_1 - \tau)) \end{aligned} \tag{8}$$

where $a(1-F_1(W_1(\tau)))$ represents the remaining number of faults that are left undetected in pre-release testing phase and $F_2(W_2(\tau_1 - \tau))$ is the fault removal rate in post release phase i.e. in the interval [τ, τ_1].

Cost incurred in this phase is given by

$$c_4\, m(W(\tau_1 - \tau)) \tag{9}$$

Phase 3: [τ_1, T_{lc}] (Post patching phase) This interval refers to the post patching period till the life cycle of the software. Here we have considered a situation where firm signs a bond with the user that they will remove the bugs throughout the software lifecycle. In this interval, users face the failure in the software due to remaining number of faults which remain undetected even after patching period. Total number of faults detected in this phase is given by

$$\begin{aligned} m(W(T_{lc} - \tau_1)) &= (a - m(W(\tau_1)))\, F_3(W_3(T_{lc} - \tau_1)) \\ &= a(1 - F_1(W_1(\tau)))\, (1 - F_2(W_2(\tau_1 - \tau))) \\ &\quad \times F_3(W_3(T_{lc} - \tau_1)) \end{aligned} \tag{10}$$

where, τ_1 denotes the optimal patch release time and T_{lc} refers to the life cycle of the software. Here $a(1-F_1(W_1(\tau)))(1-F_2(W_2(\tau_1-\tau)))$ represents the remaining number of faults which are left undetected in first and second phase and $F_3(W_3(T_{lc} - \tau_1))$ is the fault removal rate in post patching phase i.e. in interval [τ_1, T_{lc}].

Cost incurred in this phase is given by

$$c_5\, m(W(T_{lc} - \tau_1)) \tag{11}$$

It may be noted that we do not release patch to the users during the last interval [τ_1, T_{lc}], since at some point management needs to release a new version of the software. On combining cost incurred in each phase we get total cost incurred is given by

$$C(\tau, T_{lc}) = c_1 W(\tau) + c_2 \tau^2 + c_3 m(W(\tau)) + c_4 m(W(\tau_1 - \tau)) + c_5 m(W(T_{lc} - \tau_1)) \tag{12}$$

4 Numerical example

Based on our assumption that all the faults are identically and independently distributed we get that the cumulative fault distribution function follows exponential distribution in each phase given by $F_i(W_i(t)) = 1 - e^{-b_i W_i(t)}$, where i denotes the ith phase of software lifecycle. This analysis can be similarly carried out using any other distribution function. Also the effort function follows Weibull distribution given by $W(t) = \overline{W}\left(1 - e^{-vt^k}\right)$. For estimation of parameter of the effort function and mean value function, we have used data set provided by Obha (1984). This data set contains 328 number of faults which were removed in 19 weeks by consuming 47 h of CPU time. The parameter estimation values obtained by using statistical package for social sciences (SPSS) software are described below.

Total amount of testing effort eventually consumed $(\overline{W}) = 715.7$, scale parameter (v) = 0.003, shape parameter (k) = 1.116, total number of faults $(a) = 570.9$, fault detection rate (b) = 0.017, software lifecycle $(T_{lc}) = 100$. It is important to note here that $W(T_{lc}) = \overline{W}$. For numerical illustration purpose we have assumed that the fault detection rate in each phase is equal i.e. $b_i = b$ and testing effort function has equal rate in each phase i.e. $W_i(t) = W(t)$. Based on the experience of the testing team we assume cost parameters as follows: Cost of testing per unit testing effort expenditure $c_1 = 5$, market opportunity cost $c_2 = 1$, Cost of debugging during testing phase $c_3 = 3$, Cost of debugging during patching phase $c_4 = 8$ and cost of debugging a fault after patching (operational phase) $c_5 = 8$. These values can be changed for different scenarios. In general cost per unit fault during a given period is directly proportional to the ratio of resources consumed to the fault removed in that period. It is to be noted that since support cycle of software is much longer as compared to any other phases of software lifecycle, hence it consumes high amount of resources. Also the number of faults removed during this period is low, as maximum number of faults is debugged during testing and patching phase. Hence cost per unit fault removal during post release period is highest. Based on the similar arguments magnitude of cost per unit fault removal in different phases can be described. Now based on the above assumptions we will describe the phase wise fault detection with the associated cost.

Phase 1: [0, τ] This is the testing period before the release of the software. Total number of faults removed in pre-release testing phase [0, τ] are given by

$$m(W(\tau)) = a F_1(W_1(\tau)) = a\left(1 - e^{-bW(\tau)}\right) \tag{13}$$

where $F_1(W_1(\tau))$ is the fault removal rate in pre-release testing phase.

Therefore cost of removing $m(W(\tau))$number of faults is given by

$$c_3 m(W(\tau)) = c_3 a(1 - e^{-bW(\tau)})$$

Phase 2: [τ, τ_1] (Post release phase before patching) This period refers to the post release phase before patching where bugs are reported by the users and fixed by the developing team of the software company. Total number of faults detected in this phase is given by

$$m(W(\tau_1 - \tau)) = a(1 - F_1(W_1(\tau)))F_2(W_2(\tau_1 - \tau)) = a e^{-bW(\tau)}(1 - e^{-bW(\tau_1 - \tau)}) \tag{14}$$

where $F_2(W_2(\tau_1 - \tau))$ is the fault removal rate in post release phase.

Cost incurred in this phase is given by

$$c_4 m(W(\tau_1 - \tau)) = c_4 a e^{-bW(\tau)}(1 - e^{-bW(\tau_1 - \tau)})$$

Phase 3: [τ_1, T_{lc}] This interval refers to the post patching period till the life time of the software.Number of faults removed in [τ_1, T_{lc}] is given by

$$m(W(T_{lc} - \tau_1)) = a(1 - F_1(W_1(\tau)))(1 - F_2(W_2(\tau_1 - \tau))) \times F_3(W_3(T_{lc} - \tau_1)) = a e^{-bW(\tau)} e^{-bW(\tau_1 - \tau)}(1 - e^{-bW(T_{lc} - \tau_1)}) \tag{15}$$

where, τ_1 denotes the optimal patch release time and T_{lc} refers to the life cycle of the software. $F_3(W_3(T_{lc} - \tau_1))$ is the fault removal rate in post patching phase i.e. in interval [τ_1, T_{lc}].

Cost incurred in this phase is given by

$$c_5 m(W(T_{lc} - \tau_1)) = c_5 a e^{-bW(\tau)} e^{-bW(\tau_1 - \tau)}(1 - e^{-bW(T_{lc} - \tau_1)})$$

On combining cost associated in each phase with testing and opportunity costs, we have the total cost function given by

$$\text{Total cost} = C(\tau, T_{lc}) = c_1 W(\tau) + c_2 \tau^2 + c_3 a(1 - e^{-bW(\tau)}) + c_4 a e^{-bW(\tau)}(1 - e^{-bW(\tau_1 - \tau)}) + c_5 a e^{-bW(\tau)} e^{-bW(\tau_1 - \tau)}(1 - e^{-bW(T_{lc} - \tau_1)}) \tag{16}$$

Using the cost parameter values defined above for each phase in Eq. (16) we get total cost function which is a

Table 1 Phase wise faults removed and effort consumed

Phase	Mean value function	No. of faults removed	Effort consumed (in CPU h)
Pre-release phase $[0, \tau]$	$m(W(\tau))$	359	58.2
Post release before patching phase $[\tau, \tau_1]$	$m(W(\tau_1 - \tau))$	30	7.6
Post patching phase $[\tau_1, Tlc]$	$m(W(T_{lc} - \tau_1))$	181	227

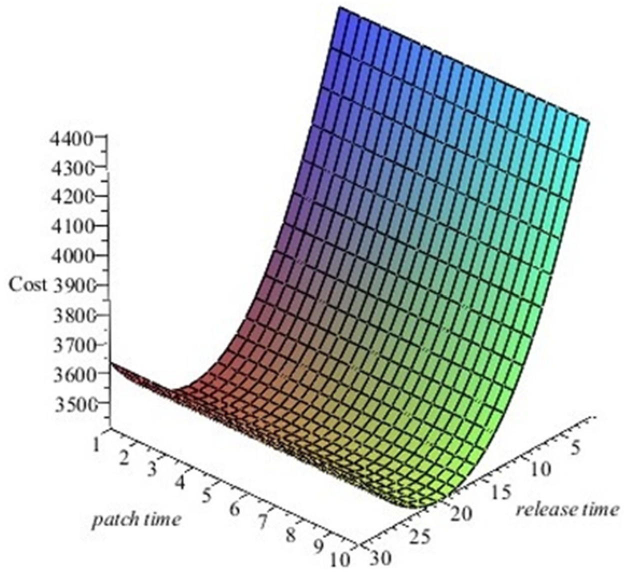

Fig. 4 Graph for optimal release and patching time

function of two variables i.e. τ and τ_1. On optimizing the total cost function by using MAPLE software we obtain the optimal cost incurred is 3411.675, release time $(\tau) = 19.9$ weeks and $\tau_1' = 3.14$ hence, optimal patch time $(\tau_1 = \tau + \tau_1') = 23.04$ weeks. The graph for optimal release and patching time is given below in Fig. 4. According to the testing data set (19 weeks) used for numerical illustration and optimal results we can infer that firm should test for one more week before release of the software and release the first patch after 3 weeks. In Table 1 we have summarized the number of faults removed and effort consumed in each phase by using the above mentioned values.

Table 1 shows that out of 570 faults 359 faults are removed by developing team in pre release phase which is of 19.9 weeks. Out of the remaining number $(570-359 = 211)$ faults, 30 are removed during the post release before patching phase which is of 3 weeks. We can say that the first patch is released for 30 faults. The remaining number $(211-30 = 181)$ of faults are removed during post patching period, this phase is the longest of all as it extends up to software lifecycle. In the current work we have discussed the cost model for single patching but on the similar lines of the proposed model we can extend it for 'n' number of patches. Faults remaining after the release of first patch can be removed by releasing more patches.

5 Conclusion

Decisions related to software release and testing are some of the most important aspects of policy making in software industry. Generally firms release their software after spending significant amount of time on testing so as to ensure that software has achieved a specified reliability level which in turn reduces the chance of software failure in field. But delay in release of software results in loss of market opportunity. In the current market situation firms are releasing early to gain high market potential and fixing the faults later in the post release phase. Keeping this in mind in our proposed work we formulated a generalized testing effort based cost model under different testing environment to determine the optimal release and patch time of software. Based on the proposed policy, project managers can plan the release and testing time of software. This additional flexibility can help firms significantly to gain an advantageous position in a competitive marketplace. Future work may include other factors like change-point, error generation and warranty in the proposed framework to make it more practical.

Acknowledgments Authors express their deep sense of gratitude to The Founder President of Amity Universe, Dr. Ashok K. Chauhan for his keen interest in promoting research in the Amity Universe and always being an inspiration for achieving greater heights.

References

Ahmad N, Khan MGM, Rafi LS (2010) A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging. Int J Qual Reliab Manag 27(1):89–110

Apple (2015) https://www.apple.com/softwareupdate. Accessed 24 Sept 2015

Arora A, Caulkins JP, Telang R (2006) Research note: sell first, fix later: impact of patching on software quality. Manag Sci 52(3):465–471

Arora A, Telang R, Xu H (2008) Optimal policy for software vulnerability disclosure. Manag Sci 54:642–656

Cavusoglu H, Zhang J (2008) Security patch management: Share the burden or share the damage? Manag Sci 54:657–670

Chatterjee S, Singh JB (2014) A NHPP based software reliability model and optimal release policy with logistic–exponential test coverage under imperfect debugging. Int J Syst Assur Eng Manag 5(3):399–406

Dey D, Lahiri A, Zhang G (2015) Optimal policies for security patch management. INFORMS J Comput 27(3):462–477

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

HP (2015) http://www.zdnet.com/hp-to-begin-charging-for-firmware-updates-and-service-packs-for-servers-7000026110. Accessed 24 July 2015

Huang CY, Kuo SY, Lyu MR, Lo JH (2000) Quantitative software modeling from testing to operation. In: Proceedings of 11th international symposium on software reliability engineering, San Jose, California, 9–11 Oct 2000

Huang CY, Kuo SY (2002) Analysis of incorporating logistic testing-effort function into software reliability modeling. IEEE Trans Reliab 51(3):261–270

Huang CY, Lin CT (2010) Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship. IEEE Trans Comput 59(2):283–288

Huang C-Y, Lyu MR (2005) Optimal release time for software systems considering cost, testing-effort, and test efficiency. IEEE Trans Reliab 54:583–591

Huang CY, Lin CT, Su YS (2005) Modeling and prediction of software operational reliability. Int J Technol Eng Educ 2(2):91–100

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

Inoue S, Yamada S (2013) Lognormal process software reliability modeling with testing-effort. J Softw Eng Appl 6:8–14

Jain M, Priya K (2005) Software reliability issues under operational and testing constraints. Asia Pac J Oper Res 22(1):33–49

Jiang Z, Sarkar S, Jacob VS (2012) Post-release testing and software release policy for enterprise-level systems. Inf Syst Res 23(3):635–657

Kapur PK, Garg RB (1990) Optimal release policies for software systems with testing effort. Int J Syst Sci 22(9):1563–1571

Kapur PK, Singh O, Shrivastava AK, Singh JNP (2015) A software up-gradation model with testing effort and two types of imperfect debugging. In: IEEE Xplore conference proceedings of international conference on futuristic trends in computational analysis and knowledge management, held at Amity University Greater Noida Campus, UP on 25–27 Feb 2015, pp 613–618

Kapur PK, Goswami DN, Bardhan A (2007) A general software reliability growth model with testing effort dependent learning process. Int J Model Simul 27(4):340–346

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

Kapur PK, Ompal S, Aggarwal AG, Kumar R (2009) Unified framework for developing testing effort dependent software reliability growth models. WSEAS Trans Syst 4(8):521–531

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

Kapur PK, Shrivastava AK (2015) When to release and stop testing of a software: a new insight, international conference on reliability, Infocom Technology and Optimization (trends and future directions), held during 2–4 Oct 2015 at Amity University, Noida, Uttar Pradesh, pp 1–6

Kuo SY, Huang CY, Lyu MR (2001) Framework for modeling software reliability, using various testing-efforts and fault-detection rates. IEEE Trans Reliab 50(3):310–321

Li X, Xie M, Ng SH (2010) Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. Appl Math Model 34(11):3560–3570

Li Q, Li H, Lu M (2015) Incorporating S-shaped testing-effort functions into NHPP software reliability model with imperfect debugging. J Syst Eng Electron 26(1):190–207

Lin CT, Huang CY (2008) Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. J Syst Softw 81:1025–1038

Lo JH, Huang CY, Chen IY, Kuo SY, Lyu MR (2005) Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. J Syst Softw 76:3–13

Luo C, Okamura H, Dohi T (2015) Optimal planning for open source software updates. Proc IMechE Part O J Risk Reliab. doi:10.1177/1748006x15586507

Musa JD (2004) Software reliability engineering: more reliable software, faster and cheaper, 2nd edn, McGraw-Hill, New Delhi

Obha M (1984) Software reliability analysis models. IBM J. Research Devlopment 28(4):428–443

Okamura H, Tokuzane M, Dohi T (2009) Optimal security patch release timing under non-homogeneous vulnerability-discovery processes. In: Proceedings of the 20th international symposium on software reliability engineering (ISSRE ‘09), Mysuru, India, 2009, pp 120–128

Pasquini A, Crespo A, Matrella P (1996) Sensitivity of reliability-growth models to operational profile errors vs testing accuracy. IEEE Trans Reliab 45(4):531–540

Peng R, Li YF, Zhang WJ, Hu QP (2014) Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction. Reliab Eng Syst Saf 126:37–43

Pham H (2006) System software reliability. Springer, London

Singh J, Singh O, Aggrawal D, Anand A, Singh I (2012) A flexible reliability growth model for various releases of software under the influence of testing resources. J Pure Appl Sci Technol NLSS 2(2):23–35

Singh O, Kapur PK, Shrivastava AK, Kumar V (2015) Release time problem with multiple constraints. Int J Syst Assur Eng Manag 6(1):83–91

Yamada S, Ohtera H, Narihisa H (1986) Software reliability growth models with testing-effort. Reliab IEEE Trans 35(1):19–23

Yang B, Xie M (2000) A study of operational and testing reliability in software reliability analysis. Reliab Eng Syst Saf 70:323–329

Zachariah B (2015) Optimal stopping time in software testing based on failure size approach. Ann Oper Res. doi:10.1007/s10479-015-1959-5

Zhang N (2015) Queue-based FDP and FCP analysis with detection effort and correction effort. J Inf Comput Sci 12(1):21–29

Zhang N, Cui G, Liu H (2012) Considering detection effort and correction effort for software reliability analysis. J Comput Inf Syst 8(19):7991–8000

Zhang N, Cui G, Liu H (2013) Software reliability analysis using queuing based model with testing effort. J Softw 8(6):310–317

Zhang C, Cui G, Liu H (2014) A unified And flexible framework of imperfect debugging dependent SRGMs with testing-effort. J Multimed 9(2):310–317

Zhao J, Liu H, Cui G, Yang XZ (2005) Software reliability growth model from testing to operation. In: Proceedings of the 21st IEEE international conference on software maintenance (ICSM’05), 26–29 Sept 2005, pp 691–694. doi:10.1109/ICSM.2005.82

Zhao Q, Zheng J, Li J (2012) Software reliability modeling with testing-effort function and imperfect debugging. TELKOMNIKA 10(8):1992–1998