Decision Science Letters 5 (2016) 143–156

Contents lists available at GrowingScience

Decision Science Letters

homepage: www.GrowingScience.com/dsl

An effort allocation model considering different budgetary constraint on fault detection process and fault correction process

Vijay Kumar[a*] and Ramita Sahni[b]

[a]*Department of Mathematics, Amity School of Engineering and Technology, New Delhi-110061, India*
[b]*Department of Applied Mathematics, Amity Institute of Applied Sciences, Amity University, Sector-125, Noida, India*

C H R O N I C L E

Article history:
Received January 16, 2015
Received in revised format:
June 12, 2015
Accepted July 21, 2015
Available online
July 21 2015

Keywords:
Fault detection process
Fault correction process
Optimal control theory
Resource allocation
Release policy

A B S T R A C T

Fault detection process (FDP) and Fault correction process (FCP) are important phases of software development life cycle (SDLC). It is essential for software to undergo a testing phase, during which faults are detected and corrected. The main goal of this article is to allocate the testing resources in an optimal manner to minimize the cost during testing phase using FDP and FCP under dynamic environment. In this paper, we first assume there is a time lag between fault detection and fault correction. Thus, removal of a fault is performed after a fault is detected. In addition, detection process and correction process are taken to be independent simultaneous activities with different budgetary constraints. A structured optimal policy based on optimal control theory is proposed for software managers to optimize the allocation of the limited resources with the reliability criteria. Furthermore, release policy for the proposed model is also discussed. Numerical example is given in support of the theoretical results.

1. Introduction

In recent years, software has become a driving force and there has been an increase in human dependence on computer systems. We use computer for the simplest calculations and, in any organization, the role of computer system has become a necessity. Thus, the failure of computer system troubles the social life leading to loss of resources, time and money, i.e., failure of a software leads to great loss in terms of money and time. Nowadays, more investment is accomplished on testing software rather than on building software. Testing plays a very important role in defining the quality of software. The success of testing lies in detecting bugs, removing the detected bug and consequently, improving the reliability of the software. Every software company aims to minimize the testing cost, give high quality product and meet the target deadline.

Software reliability modeling and prediction during software development process is an area that is getting more focus from software firms. The use of software reliability growth models (SRGMs) plays

* Corresponding author.
E-mail address: vijay_parashar@yahoo.com (V. Kumar)

doi: 10.5267/j.dsl.2015.7.002

a key role in improving reliability and also used to establish the relation among failure observation and fault removal process. Many software reliability growth models (SRGMs) have been proposed during the last fifty years to minimize the cost and to maximize the reliability during testing phase of a software development life cycle (SDLC). Goel and Okumoto (1979) are believed to be the pioneer whose models describe the failure phenomenon by an exponential curve. Yamada et al. (1983) described the S-shaped failure pattern. While some of the SRGMs are flexible in the sense that they can be described by both exponential and S-shaped failure patterns (Ohba, 1984). Numerous proposed SRGMs only consider the fault detection process and are based on the assumption that a fault is removed instantly as soon as it is detected. However, this theory may not be logical as the detected faults are rarely corrected instantly (Gokhale et al., 1998; Schneidewind, 1975, 2003; Ohba, 1984; Xie & Zhao, 1992). Generally, whenever a fault is detected, it is reported first, then diagnosed and finally it is removed. Therefore, the delay time between detection of a fault and correction of a fault should not be neglected during the modeling.

Schneidewind (1975) proposed the model in which FCP was modeled as a separate process followed the FDP with a constant time-lag. Later, Xie and Zhao (1992) used time dependent delay function, and proposed a model to find the expected time lag between detection and correction of a fault. Yamada et al. (1983) proposed a model with two stage processes using time lag concept. Later, Huang and Lin (2006) in their pioneer work discussed the dependency of faults and time- lag between the failure observation and fault removal. Based on FDP/FCP, Xie et al. (2007) and Wu et al. (2007) proposed several paired FDP and FCP models through incorporating other variants of debugging delay. Later, Huang and Pham (2009) proposed a generalized NHPP model considering quasi-renewal time-delay fault removal. Jia et al. (2010) developed a Markovian software reliability model considering the fault correction process.

However, the influence of testing effort during the modeling cannot be ignored due to its necessity for improvement of the modeling. The time-lag is less if more testing effort is allocated during the period between detection and correction of the fault. Therefore, it is reasonable to integrate the testing effort function into the modeling framework on both FDP and FCP. Also it is well known, testing process consumes a major portion of resources such as manpower and CPU hours which are limited. Optimal allocation of resources may reduce the testing cost. Huang et al. (1997), Pillai and Nair (1997), Kapur et al. (2006) and Kapur et al. (2007) discussed the impact of testing effort in their work. Peng et al. (2014) studied the FDP/FCP with the testing effort function and imperfect debugging.

As discussed above, many SRGMs minimize the cost of software during the testing phase but most of them are under static assumptions. The problem becomes more complicated when the nature of the development process is not static, but dynamic. The detection and correction processes are dependent on the nature and quantity of resources utilized. Su and Huang (2007) proposed a model based on neural network approach to build a dynamic weighted combinational model (DWCM) and applied neural network to predict software reliability by designing different elements of neural networks. Li et al. (2012) discussed two Ada Boosting based combination approaches for improving the parametric SRGMs. In the first approach authors selected several variations of one original SRGM for obtaining the self-combination model and secondly they have selected several various candidate SRGMs for obtaining the multi-combinational model. Kapur et al. (2010) proposed a model to allocate the resources and minimize the testing cost during the testing phase under dynamic condition. Kapur et al. (2013) discussed the problem to minimize the cost and maximize the reliability during the testing phase. Kapur et al. (2013) used exponential shaped SRGM and proposed a technique to find the optimal time to release of software. Kapur et al. (2010, 2013) did not mention the concept of time lag during the modeling. Kumar et al. (2014) proposed a resource allocation model for detection and correction purpose with fixed budgetary constraint with the assumption that detection and correction are two concurrent activities. Also, they have assumed detection effort and correction effort are interdependent. However, practically it may be possible that different activities, i.e., detection and correction can have

different budgetary constraints. Therefore, software manager can have different budget for detection and different budget for correction. Keeping this condition in mind we have proposed a resource allocation model for detection and correction process to minimize the testing cost. We have assumed that detection and correction are concurrent activities and have taken detection effort and correction effort to be independent. Finally, the paper is divided into the following sections: modeling, optimal solution, special cases, numerical analysis, release policy and lastly a conclusion is drawn.

2. Modeling

The fundamental activities of software testing are detection and correction of faults. Once a failure is reported, the fault correction team needs a certain time period to detect the fault and accordingly, they modify/change the codes so as to remove the faults. Thus, it is a very common experience of having a time-lag between both the processes. Certain factors like the number of faults detected, fault complexity, structural complexity of the software and the knowledge and experiences of the correction team influence this time lag. Thus, this correction lag cannot be ignored. Correction lag causes latent faults in the software. These faults are not corrected. They lie dormant in the software and reflect the relationship between the fault detection and correction processes. With the objective of creating a mathematical model for optimal allocation of resources, we have considered detection and correction as simultaneous activities. During software testing life cycle (STLC), we have two separate teams; a team of tester for detection purpose and another team of debugger for correction purpose. Kumar et al (2014) assumed that detection and correction effort are interdependent and they have divided total available resources into two portions for detection and correction.

Contrary to this we have considered the resources available for each activity, i.e., detection and correction to be explicitly independent. This assumption of considering independent activities within their respective budgetary ranges help in better investment for a company. Our approach makes both the variables as two distinct control variables which will influence the overall policy decisions. Thus, making it easier for the software manager to decide the budget of detection and correction separately so as to allocate the resources optimally. This means that the tester and debugger can separately dedicate their resources and time for detection and correction tasks for well controlled expenditure which in turn would minimize the total cost and maximize the reliability in a much better way.

2.1 FDP and FCP Modeling

Resource allocation means division of the resources in an optimal manner. Allocating the resources decides the modeling of any project. To take care of resources during allocation, strategic planning is very important. Generally, the expected number of faults corrected is the same as the expected number of faults detected when the fault removal process is instantaneous and perfect. However, if we consider the time-lag, the expected number of corrected faults at any given point of time will always be less than the expected number of detected faults. To propose fault detection-correction model, we assume that the removal of a fault is done in two stages. In first stage detection is done and in the second stage correction/debugging is done. Therefore, in line with Huang et al. (2007), it is reasonable to assume the following differential equations for detection and correction process

$$x(t) = \frac{dm_f(t)}{dt} = b_1 w_1(t)\left(a - m_f(t)\right) \tag{1}$$

$$y(t) = \frac{dm_r(t)}{dt} = b_2 w_2(t)\left(m_f(t) - m_r(t)\right) \tag{2}$$

where,

$m_f(0) = 0$, $m_r(0) = 0$, b_1 is the fault detection rate and b_2 is the fault correction rate.

Apart from the above notations, we have used following notations in our analysis.

T	The planning period
$w_1(t)$	The portion of total resources utilized for detection purpose at time 't' ($0 \leq w_1(t) \leq 1$)
w_2(t)	The portion of total resources utilized for correction purpose at time 't' ($0 \leq w_2(t) \leq 1$)
m_f(t)	Expected number of faults detected till time 't'
m_r(t)	Expected number of faults removed till time 't'
a	Initial amount of faults present in the software
$c_1\left(m_f(t), w_1(t)\right)$	Per unit detection cost associated with detection efforts $w_1(t)$ and cumulative fault detected is $m_f(t)$ at any time 't'
$c_2\left(m_r(t), w_2(t)\right)$	Per unit correction cost associated with correction efforts $w_2(t)$ and cumulative fault corrected is $m_r(t)$ at any time 't'

2.2 Cost Optimization Modeling

The main objective of any software firm is to minimize the cost. During the cost function modeling we have only considered detection cost and correction cost to reduce the complexity of dynamic model. Thus, the total cost at any point of time 't' during the testing phase of SDLC is the sum of detection cost $c_1(t)x(t)$ and correction cost $c_2(t)y(t)$ neglecting other administrative costs. Therefore, mathematical model can be written as:

$$\min \int_0^T [c_1(t)x(t) + c_2(t)y(t)]\, dt$$

subject to

$$x(t) = \frac{dm_f(t)}{dt} = b_1 w_1(t)\left(a - m_f(t)\right)$$
$$y(t) = \frac{dm_r(t)}{dt} = b_2 w_2(t)\left(m_f(t) - m_r(t)\right)$$
$$\frac{m_r(T)}{a} \geq R$$

where,
$m_f(0) = 0, m_r(0) = 0$
$0 \leq w_1(t)\,, w_2(t) \leq 1$ (3)

SRGMs provide a method to estimate the reliability during the testing progress. Reliability of software is defined as ''The probability that the system will not fail during $(t, t + \Delta t)$ $(\Delta t \geq 0)$ given that the latest failure occurred at t". (Huang et al., 2004) defines software reliability as "the ratio of the cumulative number of detected faults at time *t* to the expected number of initial fault content of the software", which is given by

$$R(t) = \frac{m_r(t)}{a} \quad (4)$$

Using Eq. (4), the reliability of the software at time 't' can be estimated and verified that whether it is equal to the desired reliability objective. Hence, it provides a very useful insight to the software managers whether they meet the desired reliability level as specified by the company. In the above optimization problem described in Eq. (3), $\frac{m_r(T)}{a} \geq R$ implies the criteria for the release of software, where the desired portion i.e. $\frac{m_r(T)}{a}$ of the faults to be removed. It means that with the planning period [0,T], firm aims to achieve a unique number $m_R(= aR)$ for which $m_r(T) \geq m_R$.

3. Optimal Solution

We have applied optimal control theoretic technique to solve the dynamic optimization problem proposed in section2.2. It is a technique, widely accepted in many areas of engineering, management, economics etc. to solve the dynamic optimization problems. In order to solve the above control problem, we start with the Hamiltonian (Sethi & Thompson, 2005; Kapur et al., 2012; 2013) which can be written as

$$H\big(m_f(t), m_r(t), \lambda(t), \mu(t), w_1(t), w_2(t), t\big) = -[c_1(t)x(t) + c_2(t)y(t)] + \lambda(t)x(t) + \mu(t)y(t)\,, \qquad (5)$$

where, $\lambda(t)$ & $\mu(t)$ are the adjoint variables, which satisfy the following differential equations:

$$\frac{d\lambda(t)}{dt} = \dot{\lambda} = -H_{m_f} \qquad (6)$$

For the terminal condition at t=T, $\lambda(T) = 0$.

Similarly

$$\frac{d\mu(t)}{dt} = \dot{\mu} = -H_{m_r} \qquad (7)$$

For the transversality condition, $\mu(T) \leq 0$ $(= 0\ if\ m_r(T) > m_R)$.

The adjoint variable $\lambda(t)$ represents per unit change in the objective function for a small change in $m_f(t)$ i.e. $\lambda(t)$ can be interpreted as marginal cost of faults detected at time t. Similarly, $\mu(t)$ can be interpreted as marginal cost of fault removed at time't'. $\lambda(t) and\ \mu(t)$ in software testing context can be defined as follows: $\lambda(t)$ stands for future cost of detection incurred as one more fault is detected at time 't'. $\mu(t)$ stands for future cost of correction incurred as one more fault is corrected by the debugger at time 't'. Thus, the Hamiltonian is the sum of total current cost $(c_1x + c_2y)$ and the total future cost $(\lambda x + \mu y)$. In short, H represents the instantaneous total cost of the firm at time 't'. The necessary conditions are given by:

$H_{w_1} = 0\ ;\ H_{w_2} = 0$

$$\Rightarrow -c_{1w_1}(t)x(t) - [c_1(t) - \lambda(t)]x_{w_1}(t) = 0 \qquad (8)$$

$$\Rightarrow -c_{2w_2}(t)y(t) - [c_2(t) - \mu(t)]y_{w_2}(t) = 0 \qquad (9)$$

Solving Eq. (8) and Eq. (9) we have

$$w_1(t) = \frac{-[c_1(t) - \lambda(t)]}{c_{1w_1}(t)} \qquad (10)$$

$$w_2(t) = \frac{-[c_2(t) - \mu(t)]}{c_{2w_2}(t)} \qquad (11)$$

Other optimality conditions are $H_{w_1w_1} \leq 0$ and $\begin{vmatrix} H_{w_1w_1} & H_{w_1w_2} \\ H_{w_2w_1} & H_{w_2w_2} \end{vmatrix} \geq 0$

where,

$$H_{w_1w_1} = -c_{1w_1w_1}x(t) - 2c_{1w_1}(t)x_{w_1}(t) - [c_1(t) - \lambda(t)]x_{w_1w_1}(t) \leq 0$$

$$\Rightarrow c_{1w_1w_1}x(t) + 2c_{1w_1}(t)x_{w_1}(t) + [c_1(t) - \lambda(t)]x_{w_1w_1}(t) \geq 0 \quad (12)$$

$$H_{w_1w_2} = 0 \quad (13)$$

$$H_{w_2w_1} = 0 \quad (14)$$

$$H_{w_2w_2} = -c_{2w_2w_2}y(t) - 2c_{2w_2}(t)y_{w_2}(t) - [c_2(t) - \mu(t)]y_{w_2w_2}(t) \leq 0$$

$$\Rightarrow c_{2w_2w_2}y(t) + 2c_{2w_2}(t)y_{w_2}(t) + [c_2(t) - \mu(t)]y_{w_2w_2}(t) \geq 0 \quad (15)$$

Thus $\begin{vmatrix} H_{w_1w_1} & H_{w_1w_2} \\ H_{w_2w_1} & H_{w_2w_2} \end{vmatrix} \geq 0$ (16)

Taking time derivative of Eq. (8) and Eq. (9) we get,

$$\dot{w_1} = \frac{\left[-c_{1w_1}(t)x_{m_f}(t) - c_{1w_1m_f}(t)x(t) - (c_1(t) - \lambda(t))x_{w_1m_f}(t) - c_{1m_f}(t)x_{w_1}(t)\right]\dot{m}_f}{c_{1w_1w_1}(t)x(t) + 2c_{1w_1}(t)x_{w_1}(t) + \left(c_1(t) - \lambda(t)\right)x_{w_1w_1}(t)} \quad (17)$$

$$\dot{w_2} = \frac{\mathrm{A\dot{m}_f - B\dot{m}_r}}{2\mathrm{c_{2w_2}(t)y_{w_2}(t) + y(t)c_{2w_2w_2}(t) + \left(c_2(t) - \mu(t)\right)y_{w_2w_2}(t)}} \quad (18)$$

where,

$$A = \left[-\mathrm{c_{2w_2}(t)y_{m_f}(t) - c_2(t)y_{w_2m_f}(t) + \mu(t)y_{w_2m_f}(t)}\right]$$

$$B = \left[\mathrm{c_{2w_2}(t)y_{m_r}(t) + c_{2w_2m_r}(t)y(t) + \left(c_2(t) - \mu(t)\right)y_{w_2m_r}(t) + c_{2m_r}(t)y_{w_2}(t)}\right]$$

where the subscript on a variable denotes the partial derivative with respect to that variable, i.e.

$$c_{1w_1w_1} = \frac{\delta^2 c_1}{\delta w_1^2}, \quad c_{2w_2w_2} = \frac{\delta^2 c_2}{\delta w_2^2}, \quad c_{1w_1} = \frac{\delta c_1}{\delta w_1}, \quad c_{2w_1} = \frac{\delta c_2}{\delta w_1}$$

$$x_{w_1w_1} = \frac{\delta^2 x}{\delta w_1^2}, \quad y_{w_2w_2} = \frac{\delta^2 y}{\delta w_2^2}, \quad x_{w_1} = \frac{\delta x}{\delta w_1}, \quad y_{w_2} = \frac{\delta y}{\delta w_2}$$

Integrating Eq. (6) with the terminal condition, the future cost of detecting one more fault from the software is given by

$$\lambda(t) = -\int_t^T \left[\frac{\delta c_1}{\delta m_f}x(t) + \left(c_1(t) - \lambda(t)\right)\frac{\delta x}{\delta m_f} + (c_2(t) - \mu(t))\frac{\delta y}{\delta m_f}\right]dt \quad (19)$$

Similarly, integrating Eq. (7), the future cost of removing one more fault from the software is given by

$$\mu(t) = \mu(T) - \int_t^T \left[\frac{\delta c_2}{\delta m_r}y(t) + (c_2(t) - \mu(t))\frac{\delta y}{\delta m_r}\right]dt \quad (20)$$

4. Special Cases

The following two scenarios depict the behavior of the proposed model. We have assumed the different functional forms of detection cost and correction cost to analyze behavior of the control model and related optimal policies:

4.1 Case 1

In this case we have consider

$$c_1(t) = c_1\,(a\ constant) \& \ c_2(t) = c_2(a\ constant)$$

The Hamiltonian for this case is given by

$$H = -[c_1 x(t) + c_2 y(t)] + \lambda(t)x(t) + \mu(t)y(t) \tag{21}$$

where, $\lambda(t)$ satisfies the following differential equation:

$$\frac{d\lambda(t)}{dt} = \dot{\lambda} = -H_{m_f} = -(c_1 - \lambda)b_1 w_1(t) + (c_2 - \mu)b_2 w_2(t) \tag{22}$$

with the terminal condition at $t = T$, $\lambda(T) = 0$

Solving Eq. (22) together with the terminal condition, we get

$$\lambda(t) = \int_t^T \left[\big(c_1 - \lambda(t)\big)b_1 w_1(t) - \big(c_2 - \mu(t)\big)b_2 w_2(t)\right]dt \tag{23}$$

Similarly $\mu(t)$ satisfies the following differential equation:

$$\frac{d\mu(t)}{dt} = \dot{\mu} = -H_{m_r} = -\big(c_2 - \mu(t)\big)b_2 w_2(t) \tag{24}$$

Solving Eq. (24) together with the terminal condition we get

$$\mu(t) = \mu(T) + \int_t^T \left[\big(c_2 - \mu(t)\big)b_2 w_2(t)\right]dt \tag{25}$$

For optimal policy, let us assume the following:

$$\alpha(t) = [-c_1 + \lambda(t)]b_1\left(a - m_f(t)\right)$$

$$\beta(t) = [-c_2 + \mu(t)]b_2\left(m_f(t) - m_r(t)\right)$$

Thus, Hamiltonian becomes

$$H = \alpha(t)w_1(t) + \beta(t)w_2(t) \tag{26}$$

Since Hamiltonian is linear function in control variables $w_1(t)$ & $w_2(t)$. Therefore, we have the following optimal policies for $w_1(t)$ & $w_2(t)$ which maximize the objective function.

Table 1

The optimal policy for detection effort and correction effort for various values of α and β

Subcases	Condition on $\boldsymbol{\alpha}$ & $\boldsymbol{\beta}$		Optimal Controls		Characterization
1	$\alpha = 0$	$\beta = 0$	${w_1}^* = n.d\,,$	${w_2}^* = n.d$	Bang-Bang
2	$\alpha > 0$	$\beta > 0$	${w_1}^* = 1\,,$	${w_2}^* = 1$	Bang-Bang
3	$\alpha < 0$	$\beta < 0$	${w_1}^* = 0\,,$	${w_2}^* = 0$	Bang-Bang
4	$\alpha < 0$	$\beta > 0$	${w_1}^* = 0\,,$	${w_2}^* = 1$	Bang-Bang
5	$\alpha > 0$	$\beta < 0$	${w_1}^* = 1\,,$	${w_2}^* = 0$	Bang-Bang
6	$\alpha > 0$	$\beta = 0$	${w_1}^* = 1\,,$	$0 \le {w_2}^* \le 1$	Singular
7	$\alpha = 0$	$\beta > 0$	$0 \le {w_1}^* \le 1\,,$	${w_2}^* = 1$	Singular
8	$\alpha < 0$	$\beta = 0$	${w_1}^* = 0\,,$	$0 \le {w_2}^* \le 1$	Singular
9	$\alpha = 0$	$\beta < 0$	$0 \le {w_1}^* \le 1,$	${w_2}^* = 0$	Singular

$n.d = not\ defined$

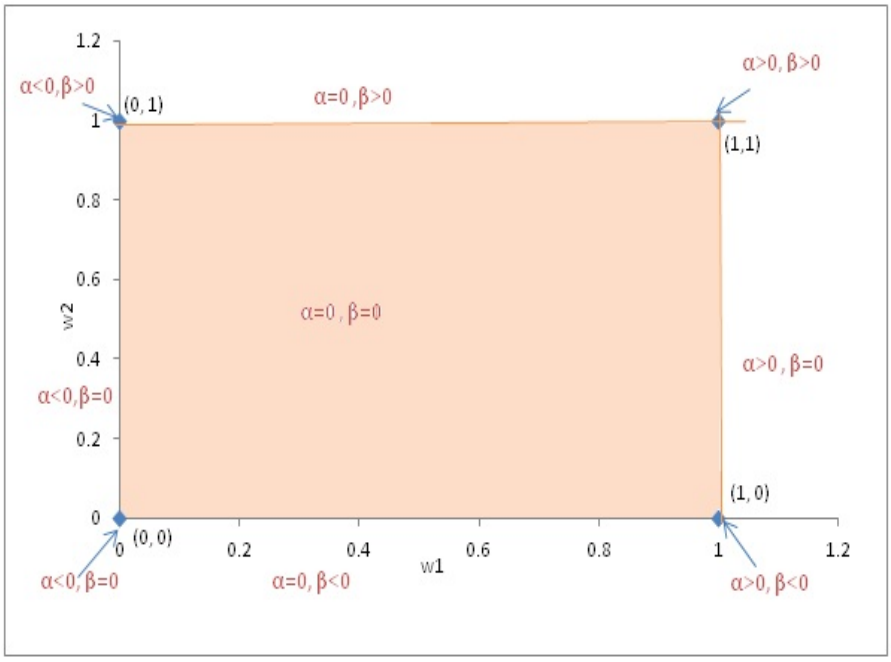


Fig. 1. Graph showing optimal policy of detection effort (w_1) and correction effort (w_2) at any time 't'.

According to the results of Table 1, we can conclude that optimal policies are the combination of generalized bang-bang and singular controls. In subcase1, we have optimal controls as ${w_1}^* = not\ defined\ , {w_2}^* = not\ defined$ in accordance with the definition of the bang-bang function. Thus, in this case resources are not defined for detection and correction purpose. So we can devoid this case. In subcase 2, we have optimal controls as ${w_1}^* = 1, {w_2}^* = 1$ in accordance with the definition of the bang-bang function. Thus, in this case software manager should utilize their complete resources for detection and correction purpose both simultaneously. In subcase 3, we have optimal controls as ${w_1}^* = 0, {w_2}^* = 0$ according to the definition of bang-bang function. Thus, in this case no portion of resources should be allocated for detection and correction purpose both. In subcase 4, we have optimal controls as ${w_1}^* = 0, {w_2}^* = 1$ in accordance with the definition of bang-bang function. Thus, in this case software manager should allocate total available resources for correction purpose while keeping the detection process idle. Similar interpretation can also be drawn for the sub case 5.

In subcase 6, we have $\alpha > 0\,, \beta = 0$ so optimal controls are ${w_1}^* = 1\,, 0 \le {w_2}^* \le 1$. In this case detection effort should be utilized with full strength while company may utilize none or any portion of

resources for correction purpose. In subcase 8, we have $\alpha < 0$, $\beta = 0$ and so optimal controls are ${w_1}^* = 0$, $0 \le {w_2}^* \le 1$. In this case, company should invest any portion of resources for correction purpose and no resources to be allocated for detection purpose. In both these cases (i.e. sub case 6 & 8) optimal controls are singular with respect to w_2. These cases are termed singular because Hamiltonian maximizing condition does not yield a unique value for the control w_2. In such cases, the optimal controls are obtained by conditions required to sustain $\beta = 0$ for a finite time interval.

In subcase 7, we have $\alpha = 0$, $\beta > 0$ so optimal controls are $0 \le {w_1}^* \le 1$, ${w_2}^* = 1$. In this case company should allocate any portion of resource for detection purpose and all correction resource should be allocated for correction purpose. In subcase 9, we have $\alpha = 0$, $\beta < 0$ so optimal controls are $0 \le {w_1}^* \le 1$, ${w_2}^* = 0$.

Similarly results of sub case 9 may also be interpreted based on sub case 7. In both these cases (subcase 7 & 9) the optimal controls are singular with respect to w_1. To maintain this singular control over a finite time period, we must keep $\alpha = 0$ in the interval.

4.2 Case 2

In contrast with constant detection cost and correction cost discussed in special case 4.1, we have considered a scenario when the detection cost and correction cost are dynamic in nature. We have incorporated Pegels (1969) functional form for detection and correction cost function, which demonstrates the learning curve phenomenon. Kumar et al. (2014) used Pegels (1969) and further applied Genetic Algorithm to find the optimum value of single control variable. Many researcher, scientists and engineers used this functional form to demonstrate the concept of learning curve phenomenon in their pioneer work in the field of management, economics, engineering and science. In the software testing concern, learning curve effect can be visualized as: when a new fault has to be corrected for the first time it is likely that the team involved in the correction will not achieve maximum efficiency immediately. Repetition of the task will make the team more confident and familiar and will finally result in a more efficient and quick. In other words, we can say that more often a task is performed; the lower will be the cost of doing it. We have assumed that each time cumulative volume of fault detected increases, value added costs falls by a constant and predictable percentage. Therefore, functional form of detection cost and correction cost are (Pegels 1969)

$$c_1\left(m_f(t), w_1(t)\right) = c_o(w_1(t))^{m_f(t)}$$

$$c_2\left(m_r(t), w_2(t)\right) = b_o(w_2(t))^{m_r(t)}$$

where c_o & b_o are base detection and base correction cost respectively.

The Hamiltonian for the above case can be written as

$$H = -[c_1 x(t) + c_2 y(t)] + \lambda(t)x(t) + \mu(t)y(t) \tag{27}$$

The following are the necessary condition hold for an optimal solution:

$$H_{w_1} = 0 \; ; \; H_{w_2} = 0 \tag{28}$$

The adjoint variables are given by the following differential equations:

$$\lambda(t) = \int_t^T \Big[\big(c_1(t) - \lambda(t)\big)b_1 w_1(t) - (c_2(t) - \mu(t))b_2 w_2(t) - c_1(t)x(t)\log\big(w_1(t)\big)\Big]dt \tag{29}$$

$$\mu(t) = \mu(T) + \int_t^T \left[(c_2(t) - \mu(t))b_2 w_2(t) - c_2(t)y(t)\log\big(w_2(t)\big)\right]dt \tag{30}$$

Solving Eq. (28) yields

$$w_1(t) = \left[\frac{\lambda(t) - c_1(t)}{c_o(m_f(t))}\right]^{\frac{1}{m_f(t)}} \tag{31}$$

$$w_2(t) = \left[\frac{\mu(t) - c_2(t)}{b_o(m_r(t))}\right]^{\frac{1}{m_r(t)}} \tag{32}$$

5. Numerical Analysis

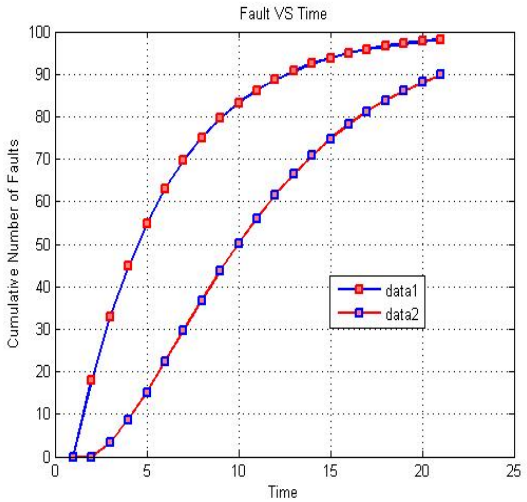
In this section, we numerically illustrate the behavior of cost model proposed in section 4.2 and analyzed the impact of detection effort and correction effort on the cost optimization model. We have conducted various simulations using different values of model parameters. Results converged quickly and became stable. For the numerical illustrations, some base values were considered and then model parameters were varied individually. The base values of parameters are as follows:

$$a = 100,\ b_1 = 0.3,\ b_2 = 0.3,\ w_1 = 0.6,\ w_2 = 0.6,\ \lambda(0) = 100,$$

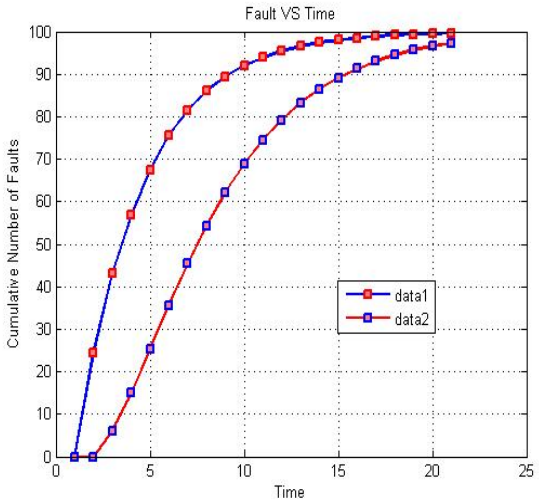
$$\mu(0) = 100,\ m_f(0) = 0,\ m_r(0) = 0,\ c_o = 1000,\ b_o = 1000$$

Initially, we obtain the number of faults detected and corrected for various values of w_1 and w_2 using Eq. (1) and Eq. (2). We have taken four values of w_1 *and* w_2 as 0.6, 0.7, 0.8, and 0.9. The following four graphs in figure 2 show cumulative faults detected and cumulative faults corrected at any time t. Figure 2 shows there is always a time-lag between detection and correction which can be due to the complexity of code, severity of faults and skills of the software engineers. The pattern in Fig. 2 exhibits the same pattern as in Kumar et al. (2014) due the nature of Eq. (1) and Eq. (2) but the values are different from the Kumar et al. (2014). The same is signified by the gap between the two lines, i.e., between the lines showing faults detected and faults corrected.

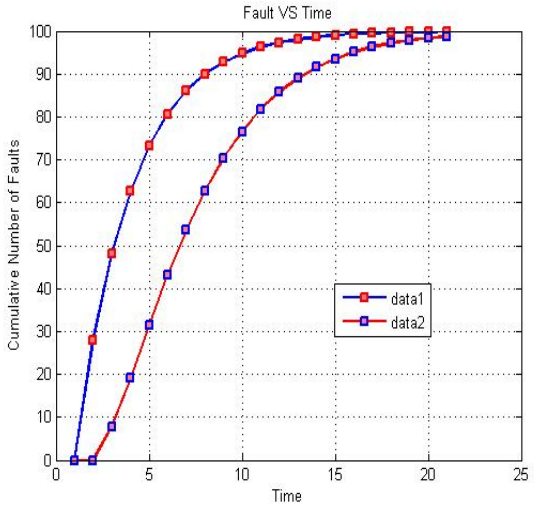
The numerical example reveals that the cumulative number of detected faults decreases as w_1 decreases. The cumulative number of faults corrected increases as the correction efforts w_2 increases and follows S-shaped curve. The S-shaped of curve is due the nature of model in Eq. (1) and Eq. (2).



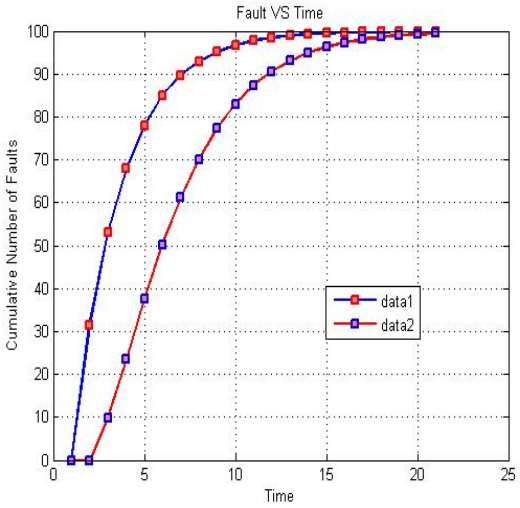
2(a)



2(b)



2(c)



2(d)

Fig. 2. Number of detected and corrected faults vs. Time (a) $w_1 = 0.6\,, w_2 = 0.6$ (b) $w_1 = 0.7\,, w_2 = 0.7$ (c) $w_1 = 0.8\,, w_2 = 0.8$ (d)$w_1 = 0.9\,, w_2 = 0.9$. Here, data 1 represents number of faults detected and data 2 represents number of faults corrected.

Further we have extended the numerical analysis to analyze the behavior of future cost of detection for various values of w_1 and w_2. Figure 3 reveals the pattern of future cost of detection during simulation. The future cost of detection is tending to zero. A similar investigation was done to study the behavior of future cost of correction, which is displayed in figure 4. The following observation was made during the simulation. The future cost of correction is decreasing in nature and approaches to zero with increase in time t. Decreasing the nature of the graph shows that with the increase in correction of faults, lesser number of faults remains to be corrected and consequently, future cost of correction decreases which in turn tends to zero at the end of planning horizon.

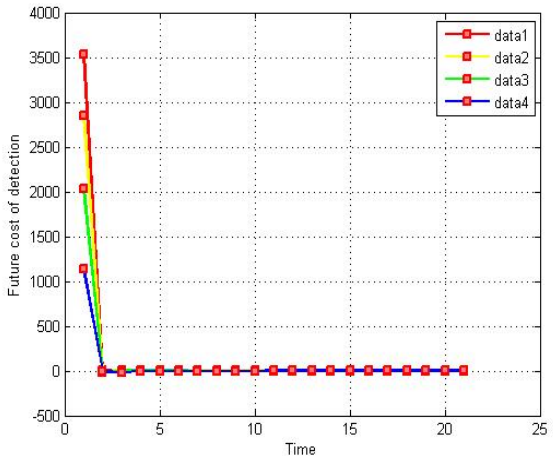


Fig. 3. Shadow cost for detection vs. time

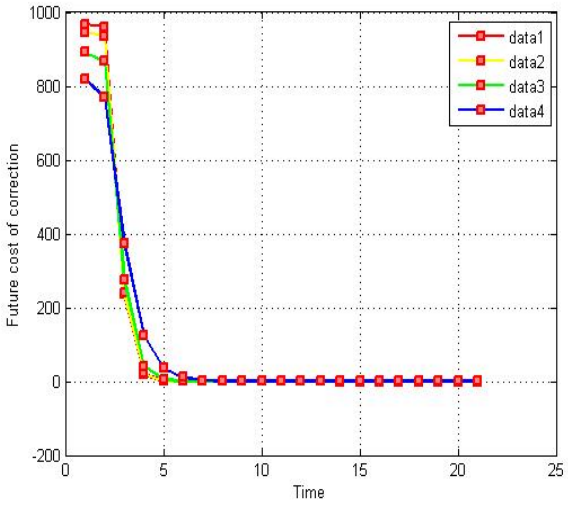


Fig. 4. Shadow cost for correction vs. time

According to Fig. 3 and Fig. 4, data 1 is corresponding to $w_{1=}0.6\,, w_2 = 0.6$; data 2 is corresponding to$w_{1=}0.7\,, w_2 = 0.7$; data 3 is corresponding to $w_{1=}0.8\,, w_2 = 0.8$ and data 4 is corresponding to$w_{1=}0.9\,, w_2 = 0.9$.

6. Release Policy

A very important aspect in software management is the release time of software. Software companies aim at minimizing their cost, maximizing the overall profit and meeting the competitive requirements and on the other hand, software users demand minimum cost and good quality software product. Basically, it is a trade-off between conflicting objectives of software users and the software developers. Therefore, for a software developer, the most important aspect to be kept in mind is when to stop testing and when to release the software system to the users focusing on their necessities. In the software reliability engineering literature, such type of problem is known as software release time decision (SRTD) problem. Timely release of software offers the software at a reasonable price with high quality level whereas the delay in release causes the burden of penalty cost, revenue loss and the product may suffer from obsolescence in the market. Testing activities i.e. detection and correction processes play an important role in the software release policy. So it is very important to balance and integrate several cost incurred due to the detection and correction during the testing phase to find the optimal release time of the software maintaining the desired reliability level under minimal cost criterion.

Software release time prediction has remained a key objective of study for many eminent researchers, engineers and scientists in software engineering field, reliability modeling and optimization over the years. Chang (2004) studied the sequential software release policy based on a state space model, considering a Gamma–Gamma-type invariant conditional distribution to define the state space model. Huang (2005) developed a software cost model to calculate software project cost and discussed the release time based on cost and reliability, considering testing effort and efficiency. Inoue and Yamada (2007) proposed a model to find the optimal time to release of software based on the generalized discrete binomial-process model. Recently, Kapur et al. (2013) developed an optimal control theoretic based cost optimization model and proposed that the optimal release time of the software is the time point where the total cost incurred due to correction coincides with the cost of detection maintaining the strict reliability constraint. Peng et al. (2014) proposed the optimal release policy under different criteria namely reliability, cost and mixed criterion. To determine the release time of software, we have considered the detection cost and correction cost and obtained an optimal time to release the software. Throughout our analysis we have assumed that the software can be released when the desired reliability level (or corrected fault level) $m_R$is achieved. Therefore, in line with Kapur et al. (2013) we have proposed the following theorem.

Theorem: The optimum release time is the time where the total cost of correction coincides with the total cost of detection, maintaining the strict reliability criteria. Else, the firm should wait for an ideal time.

Proof: For the optimization problem (3), the final time T is unspecified, i.e., $\delta T \neq 0$ and the final state $m_r(T)$ is specified, i.e., the time to stop testing depends on the value of the final state $m_r(T)$ at that time. Clearly, $\delta m_r(T) = 0$. Hence, the transversality conditions are $\mu(T^*) \leq 0$ $(= 0\ if\ m_r^*(T^*) > m_R)$. Therefore,

$$H(T^*) = 0,$$

where $H(T^*) = H(x^*(T^*), y^*(T^*), w_1^*(T^*), w_2^*(T^*), \lambda^*(T^*), \mu^*(T^*), T^*) = 0$

$$\text{Hence (5)} \Rightarrow [-c_1(T^*) + \lambda(T^*)]x(T^*) = -[-c_2(T^*) + \mu(T^*)]y(T^*) \quad (33)$$

7. Conclusion and Future Scope

This paper has investigated the optimal resource allocation model considering different budgetary constraints on fault detection and fault correction processes to minimize the testing cost. This means

that the tester and debugger can devote their resources to finish off their tasks separately in a well-controlled optimal manner. We have used optimal control theoretic approach to solve the dynamic optimization model and obtained optimal policies for detection and correction effort considering different cost functions. During the analysis, we have observed that the graphs between faults detected and corrected versus time depict a time gap. This time gap signifies the time it takes to correct a fault after it is detected. For the success of a software product, there is a need to determine the optimal stopping time for testing with a desired level of reliability. Based on theoretical study, we observed that the optimum release time is the time where the total cost of correction coincides with the total cost of detection, maintaining the strict reliability criteria. Else, the firm should wait for an ideal time. A method to calculate the testing cost of the software has been discussed. The results were verified using simulation technique. From the graph of the future cost of detection, we can conclude that due to learning curve phenomenon it eventually reaches zero with time. And consequently, shadow cost of detection tends zero. Also, from the graph of shadow cost of correction we can summarize that the shadow cost decreases with time. Moreover the effect of learning curve is exhibited in tending the shadow cost to zero.

One constraint of the proposed work is that our model only applies to projects where the testing activities are planned. Our work does not apply, where testing and debugging activities are hard to separate. Also before implementing the proposed model, software managers must collect all the appropriate information of parameters such as initial fault present in the software, detection rate and correction rate .The value of model parameters i.e. number of faults present in the software, detections rate and correction rate can be estimated outside by the normative model using equation1 and equation2. Further, this study has only analyzed the effect of deterministic models for fault detection and fault correction. Stochastic model for fault detection and correction may be used for future work. Finally, the model can be extended by incorporating the other imperfect debugging models. All these issues are worth further investigation.

Acknowledgement

The authors are thankful to the anonymous referees for their useful suggestions.

References

Chang, Y. C. (2004). A sequential software release policy. *Annals of the Institute of Statistical Mathematics*, *56*(1), 193-204.

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

Gokhale, S. S., Wong, W. E., Trivedi, K. S., & Horgan, J. R. (1998, September). An analytical approach to architecture-based software reliability prediction. In *Computer Performance and Dependability Symposium, 1998. IPDS'98. Proceedings. IEEE International* (pp. 13-22). IEEE.

Huang, C. Y., Kuo, S. Y., & Chen, Y. (1997, November). Analysis of a software reliability growth model with logistic testing-effort function. In *Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on* (pp. 378-388). IEEE.

Huang, Y., Lo, J.-H., Kuo, S.-Y. and Lyu, M.R. (2004). Optimal allocation of testing resources considering cost, reliability, and testing-effort, In the Proc. Pacific-Rim Dependable Computing, Papeete, Tahiti, French Polynesia, pp.103-112.

Huang, C.-Y. (2005). Cost-reliability-optimal Release Policy for Software Reliability Models Incorporating Improvements in Testing Efficiency, Journal of Systems and Software, 77, 139–155.

Huang, C. Y., & Lin, C. T. (2006). Software reliability analysis by considering fault dependency and debugging time lag. *Reliability, IEEE Transactions on*, *55*(3), 436-450.

Huang, C. Y., Kuo, S. Y., & Lyu, M. R. (2007). An assessment of testing-effort dependent software reliability growth models. *Reliability, IEEE Transactions on*, *56*(2), 198-211.

Hwang, S., & Pham, H. (2009). Quasi-renewal time-delay fault-removal consideration in software reliability modeling. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *39*(1), 200-209.

Inoue, S., & Yamada, S. (2007). Generalized discrete software reliability modeling with effect of program size. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *37*(2), 170-179.

Lixin, J. I. A., Bo, Y. A. N. G., Suchang, G. U. O., & Park, D. H. (2010). Software reliability modeling considering fault correction process. *IEICE transactions on information and systems*, *93*(1), 185-188.

Kapur, P. K., Gupta, A., Shatnawi, O., & Yadavalli, V. S. S. (2006). Testing effort control using flexible software reliability growth model with change point. *International Journal of Performability Engineering*, *2*(3), 245-262.

Kapur, P. K., Bardhan, A. K., & Yadavalli, V. S. S. (2007). On allocation of resources during testing phase of a modular software. *International Journal of Systems Science*, *38*(6), 493-499.

Kapur, P. K., Chanda, U., & Kumar, V. (2010). Optimal allocation of testing effort: A control theoretic approach. In *Proc. National Conf. Computing for Nation Development; INDIACOM-2010* (pp. 425-430).

Kapur, P. K., Pham, H., Kumar, V., & Anand, A. (2012). Dynamic optimal control model for profit maximization of software product under the influence of promotional effort. *The Journal of High Technology Management Research*,*23*(2), 122-129.

Kapur, P. K., Pham, H., Chanda, U., & Kumar, V. (2013). Optimal allocation of testing effort during testing and debugging phases: a control theoretic approach. *International Journal of Systems Science*, *44*(9), 1639-1650.

Kumar, V., Khatri, S. K., Dua, H., Sharma, M., & Mathur, P. (2014). An assessment of testing cost with effort dependent FDP and FCP under learning effect: A Genetic Algorithm Approach; *International Journal of Reliability, Quality and Safety Engineering*, 21(6), 1450027.

Li, H., Zeng, M., Lu, M., Hu, X., & Li, Z. (2012). Adaboosting-based dynamic weighted combination of software reliability growth models. *Quality and Reliability Engineering International*, *28*(1), 67-84.

Ohba, M. (1984). Software reliability analysis models. *IBM Journal of research and Development*, *28*(4), 428-443.

Pegels, C. C. (1969). On startup or learning curves: An expanded view. *AIIE Transactions*, *1*(3), 216-222.

Peng, R., Li, Y. F., Zhang, W. J., & Hu, Q. P. (2014). Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction. *Reliability Engineering & System Safety*, *126*, 37-43.

Pillai, K., & Sukumaran Nair, V. S. (1997). A model for software development effort and cost estimation. *Software Engineering, IEEE Transactions on*, *23*(8), 485-497.

Schneidewind, N. F. (1975, April). Analysis of error processes in computer software. In *ACM Sigplan Notices* (Vol. 10, No. 6, pp. 337-346). ACM.

Schneidewind, N. F. (2003, November). Fault correction profiles. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*(pp. 257-267). IEEE.

Sethi, S.P., & Thompson, G.L. (2005). *Optimal Control Theory — Applications to Management Science and Economics*, 2nd ed. (Springer, New York).

Su, Y. S., & Huang, C. Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, *80*(4), 606-615.

Wu, Y. P., Hu, Q. P., Xie, M., & Ng, S. H. (2007). Modeling and analysis of software fault detection and correction process by considering time dependency. *Reliability, IEEE Transactions on*, *56*(4), 629-642.

Xie, M., & Zhao, M. (1992, October). The Schneidewind software reliability model revisited. In *Software Reliability Engineering, 1992. Proceedings., Third International Symposium on* (pp. 184-192). IEEE.

Xie, M., Hu, Q. P., Wu, Y. P., & Ng, S. H. (2007). A study of the modeling and analysis of software fault-detection and fault-correction processes. *Quality and Reliability Engineering International*, *23*(4), 459-470.

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