Testing resource allocation for software with multiple versions

Adarsh Anand, Subhrata Das* and Ompal Singh

Department of Operational Research,
University of Delhi,
New Delhi-07, India
Email: adarsh.anand86@gmail.com
Email: shus.das@gmail.com
Email: drompalsingh1@gmail.com
*Corresponding author

Vijay Kumar

Department of Applied Mathematics,
Amity Institute of Applied Sciences,
Amity University Uttar Pradesh,
Noida, Uttar Pradesh, India
Email: vijay_parashar@yahoo.com

Abstract: Upgradation has become a mandatory feature for software firms. Companies have now got no time to perform software testing because of coming up of their upgraded products at a fast pace. But even then firms have to test their respective version independently to remove maximum possible number flaws within particular time limit or under testing budget. In the current paper, taking into consideration the aspect of multi-upgradation, different optimisation problems are proposed wherein optimal allocation of testing resources to different versions is discussed. The proposed set of models is solved using dynamic programming approach and is supplemented with numerical illustrations.

Keywords: software reliability; multi-versions; resource allocation; dynamic programming.

Reference to this paper should be made as follows: Anand, A., Das, S. Singh, O. and Kumar, V. (2022) 'Testing resource allocation for software with multiple versions', *Int. J. Applied Management Science*, Vol. 14, No. 1, pp.23–37.

Biographical notes: Adarsh Anand did his Doctorate in the area of Innovation Diffusion Modelling and Software Reliability Assessment. Presently, he is working as an Assistant Professor in the Department of Operational Research, University of Delhi, Delhi (India). He has been conferred with Young Promising Researcher in the field of Technology Management and Software Reliability by Society for Reliability Engineering, Quality and Operations Management in 2012. He is also on the editorial board of *International Journal of System Assurance and Engineering Management*. He has also edited a book entitled *System Reliability Management: Solutions and Technologies*

published by CRC Press, Taylor and Francis group. His research interest includes modelling successive generations in marketing, software reliability growth modelling and Vulnerability Analysis.

Subhrata Das completed her MSc and MPhil degree in Operational Research in 2012 and 2014, respectively from University of Delhi, Delhi (India). She is pursuing her PhD in Operational Research from University of Delhi. Her area of research is software reliability. She is a Lifetime Member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM).

Ompal Singh is currently an Associate Professor in the Department of Operational Research, University of Delhi, Delhi (India). He has been Active Member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM) since 2000. He has published several papers in international/national journals and proceedings. His research interests are in software testing, and software reliability engineering. He has recently developed interest in Innovation diffusion modelling and has published some papers in the same area. He is a Lifetime Member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM).

Vijay Kumar received his MSc degree in Applied Mathematics and MPhil degree in Mathematics from IIT Roorkee, India in 1998 and 2000, respectively. He has completed his PhD degree from the Department of Operational Research, University of Delhi. Currently, he is an Associate Professor in Department of Mathematics, Amity University, Noida, India. He has published more than 40 research papers in the areas of software reliability, mathematical modelling and optimisation in international journals and conferences of high repute. His current research interests include software reliability growth modelling, optimal control theory and marketing models in the context of innovation diffusion theory. He has edited special issues of *IJAMS* and *RIO* journal. He is an Editorial Board Member of *IJMEMS*. He is a Life Member of Society for Reliability Engineering, Quality and Operations Management (SREQOM) and IEEE.

This paper is a revised and expanded version of a paper entitled 'Resource Allocation Problem for Multi Versions of Software System' presented at 'Amity International Conference on Artificial Intelligence (AICAI)', IEEE, Dubai, 4–6 February 2019.

1 Introduction

Development in software engineering has prompted creation of programming for extremely intricate circumstances occurring in industry, research, defence and everyday life. As a result, the reliance of humankind on computer based frameworks is expanding step by step. Failures of software system can be costly not only monetarily but also be a life threat for human beings. Seeing the criticality associated with software usage, guaranteeing the high reliability of software should be a major concern. Therefore, firms must invest heavily in testing phase to obtain the desired level of reliability. Basically testing is a procedure of carrying out a program with the aim of discovering bugs and this stage requires numerical demonstration. The mathematical connection between the count of flaws expelled from a software and the execution time is regarded as Software

Reliability Growth Model (SRGM) in literature (Goel and Okumoto, 1979). Many researchers have tried to model the real testing condition through SRGMs (Yamada et al., 1986). SRGMs have been utilised to anticipate and forecast the bug count, consistency level and optimal software launch time. These models likewise have been utilised to quantify the fault removal phenomenon under the testing stage and speak on reliability of the software. Testing of the software require resources which can be manpower, CPU time etc. Moreover, software developing firms always have an upper limit for investing the resources under testing. Thus calls for an optimal way out for obtaining the threshold reliability under assigned budget. . It was Ohtera and Yamada (1990) who first studied the utility of testing the software modularly during its development and considered two problems to attain desired reliability levels. The testing effort-based SRGM derive the functional relationship between testing resources applied in detection of software fault for a module testing problem. The allocation problems described by Yamada et al. (1995) and Ohtera and Yamada (1990) considered the problem of minimising the resources when the count of pending faults in each module is equal to the desired level and minimising the pending faults when the resources are constrained. In the above optimisation scenarios only the significant numbers of pending bugs were considered in each module. However, Leung (1997) developed a resource allocation problem to test the modules and suggested a decrease in the variance. He considered the detection of bugs in each module as the testing is preceded and re-estimated the values of the model variables by adjusting testing resources dynamically. Further, Xie and Yang (2001) discussed an allocation problem for modular software by maximising the operational reliability. Another resource application approach suggested by Huang et al. (2004) considered the minimisation of cost of testing resources as an alternative of testing resource consumption. Later, Jha et al. (2006) deduced a coverage-based optimisation model build under the SRGM based on imperfect debugging environment. Some researchers also have worked on resource allocation problem in the area of cloud computing (Cao et al., 2017; Wei et al., 2018). Kapur et al. (2010a) proposed a model to allocate the resources and minimise the testing cost during the testing phase under dynamic condition. Kapur et al. (2013) discussed the problem to minimise the cost and maximise the reliability during the testing phase. 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. Kumar and Sahni (2016) proposed a resource allocation model assuming that different budget for detection and correction process. Kumar and Sahni (2016) used optimal control theory to solve resource allocation problem. Kumar et al. (2016) proposed a two dimensional SRGM for multi-release software considering detection and correction as two different processes. Entire pool of research in the field of resource allocation focused on module wise allocation, little or no importance was given to allocating resources for different versions of the software.

Many a times, software firms keep investing their resources in upgrading the software in order to maintain competitive scenario. Upgradation refers to feature enhancement in the software, making it more attractive and reliable in contrast to existing version (Anand et al., 2014). However, it cannot always be claimed that upgradation increases reliability as many cases exists where they may turn hazardous. In May 2017, British Airways faced their sixth global IT failure of the year which caused all flights from Heathrow and Gatwick to be cancelled (Charlotte and Thomas, 2018). The failure adversely affected British Airways call centres, website, mobile app and over a 1000 flights. These failures could have been avoided had hundreds of their IT staff not been made redundant in 2016.

A similar mishap occurred on 12 December 2017 when a software glitch caused a server in Swanwick running on UK's National Air Traffic System (NATS) to crash and the flight data processing system with it. This resulted in over 150 flights from London to be cancelled and numerous others to be delayed (Charlotte and Thomas, 2018). The crash was not the first technical failure at Swanwick and many such breakdowns had occurred since the base was set up here. A software failure in airbag's sensory detectors has resulted in two reported accidents. The affected cars were not able to detect the presence of an adult in the car's passenger seat and hence the airbags would not inflate when required. Following this Nissan had to recall over a 1 million cars over a span of two years to rectify it. Recently about 600 nodes of Bitcoin Unlimited (BU) crashed from 720 to 180. It rose slightly till 458 bringing the company hash rate share to 41%.The network was affected by a software bug which caused memory leak and instantaneously crashed 400 nodes (Charlotte and Thomas, 2018). Still market demand calls for upgradations which are reliable and in doing so, firms invest their resources majorly in testing phase.

Various researchers worked in this field to model the MVF for software whose different versions have been developed. Starting from Kapur et al. (2010b), they focused on modelling the cumulative number of flaws debugged in a particular release depending on all preceding releases for a multi-upgraded software whereas, Singh et al. (2012a) considered that the bugs present in any version depends on leftover bugs coming from just previous version and bugs due to addition of some newly written piece of code. Likewise researchers studied this concept and several proposals came up based on severity of faults, bugs in current release due to upgradation and leftover bugs from installation step of former releases, randomness, imperfect debugging environment and distribution environment etc. (Anand et al., 2014; Anand et al., 2015; Singh et al., 2009, 2011). Moreover, Singh et al. (2015) developed a MVF for multi-releases of software in which they have considered the debugging rate as a function of testing resources utilised and faults severity. But no attention was given to the manner in which resources are to be allocated or amount of resources that firm should invest in each release of the software. Further, it is always advisable to debug a considerable count of defects from the software. However, it is believed that reliability of the software increases with increase in number of faults debugged. Thus, attaining the maximum threshold of reliability is similar to removing maximum count of flaws from the system. The resources required in testing are limited and has to be used for all versions of the software, making it to important to bifurcate resources among different versions judiciously.

Basically prior to software development, firms have a clear picture regarding their offerings. A vague idea regarding the software functionalities and its performance is decided in advance. Subsequently, they might have also planned about either they are going to provide the complete offerings or will it be updated as time progresses with some functionalities. If firms wish to offer several version of the same software they need to plan their resources accordingly i.e. it has to be closely examine when the next version will be released, amount of resources that will be needed, target reliability level and so on. Making it important to decide about when the first version should be released simultaneously plan for the succeeding versions. The pattern that exists in planning of several versions development is that as soon as a new version is released its preceding version only receives support while its succeeding version is still in testing phase. On the other hand planning about different versions in advance help in capturing the market to larger extent and in gaining competitive advantage. In this paper we have examined and

comprehended such an administration issue of assignment of testing assets among different versions of the software with the help of multi-release software reliability growth model.

The primary optimisation problem proposed (P1) talks about the maximum number of shortcomings anticipated that would be removed, when required testing assets are available. The firms regularly aims for some reliability level that can be deciphered in terms of count of flaws debugged. The second optimisation problem (Prob2) includes an imperative in (Prob1) in terms of least count of issues sought to be expelled from every release of the software. For computation of (Prob2), dynamic programming strategy is utilised. This is the first time that dynamic programming has been used for resource under the concept of multi-upgradations of the software. This approach is quite flexible and easy to implement for obtaining the global solution of an optimisation problem. The technique examined in the paper has been demarcated through numerical illustrations.

The article is further structured as: in Section 2 provides list of notations is supplemented. Section 3 reviews mathematical formulation for multi-release software system. Proposed resource allocation problem of multi-version has discussed in Section 4. The proposition has been validated through numerical illustration in Section 5. At last conclusion is given in Section 6.

2 Notations

Following is the list of notations that has been used throughout the paper:

N	: Number of versions pertaining to a software system.
$m_i\left(X_i(t)\right)$	: Cumulative amount of bugs debugged using resources in $(0,t]$ in *i*-th version
a_i	: Number of bug count present in the software $(i=1,...,N)$.
b_i	: Fault removal rate $(i=1,...,N)$.
$x_i(t)$	: Testing effort expenditure at time t for *i*-th version
X_i, Z	: The quantity of testing resources to be allocated to the *i*-th version
T	: Total testing time.
X_i^*	: Optimal value of $X_i, (i=1,...,N)$
$g_n(Z)$	: Optimal number of bugs removed upto *n*-th version (i.e. corresponding *n*-th to stage in dynamic programming algorithm)
a_{i0}	: Aspiration level of the *i*-th release (i.e. number of bugs desired to be removed from *i*-th version)
p_i	: The minimum proportion of total bugs to be removed from *i*-th version
a_i^*	: Number of bugs both due to upgradation and left over of earlier version $(i=1,...,N)$

3 Bug modelling process for a software system with different versions

3.1 Testing resource based fault removal phenomenon

Software engineering literature has many research proposal based on testing effort which have been utilised for modelling the fault removal process for multi-upgraded software version. It is being assumed that mean count of flaws debugged based on testing resources utilised at a particular instant of time is directly proportional to expected count of pending fault count in the particular version, which can be mathematically expressed as follows (Kapur et al., 2003):

$$\frac{\frac{d}{dt}m_i(t)}{x_i(t)} = \frac{f_i(t)}{1-F_i(t)}\left(a_i^* - m_i(t)\right); i = 1,...,N \tag{1}$$

Solving equation (1) under initial set of conditions $t = 0, X_i(t) = 0, m_i(t) = 0$, we have:

$$m_i\left(X_i(t)\right) = a_i^*.F_i\left(X_i(t)\right); \quad i = 1,...,N \tag{2}$$

Various forms of testing effort have been described in literature, but in this study exponential form of testing effort has been used which is considered the postulation that the rate of effort is directly proportional to available amount of testing resources (Kapur et al., 2003). Its mathematical form can be stated as follows:

$$X_i(t) = \gamma_i.\left(1 - e^{-r_i.t}\right); \quad i = 1,...,N \tag{3}$$

where γ_i is total amount of available resources and r_i is rate by which resources will consumed in *i*-th versions. Above set of equation will be utilised for studying the resource allocation under testing for different versions of the software.

3.2 Testing resource-based multi-upgradation fault removal phenomenon

In multi-upgradation process, the first release is the most important release as it decides the market captured by the firm. If the first version has many bugs and leads to failure then it may affect firm's goodwill. So the testing phase is considered to be the most essential stage for software developers. As soon as the coding phase ends, software testing starts and is carried out even when the software is in operational phase (Anand et al., 2014). Prior to release of the software for usage, testers wish to remove maximum number of bugs or reduce chances of occurrence of any failure. The vast literature on multi-upgradation portrays two different scenarios for modelling the fault removal phenomenon. Basically, the first scenarios laid emphasis on modelling the Mean Value Function (MVF) of present release to be dependent on all its preceding releases, whereas in the other scenario the emphasis was given to depict that the MVF of current release depends upon its just preceding release. Seeing the wide acceptance and preference given to second scenario, in this paper we have considered that the bugs present in the current version is dependent on its just preceding version.

The mathematical formulation for first release of the software as given by Singh et al. (2012b):

$$m_1\left(X_1(t)\right) = a_1 \,.F_1\left(X_1(t)\right) \qquad 0 \le t \le t_1 \tag{4}$$

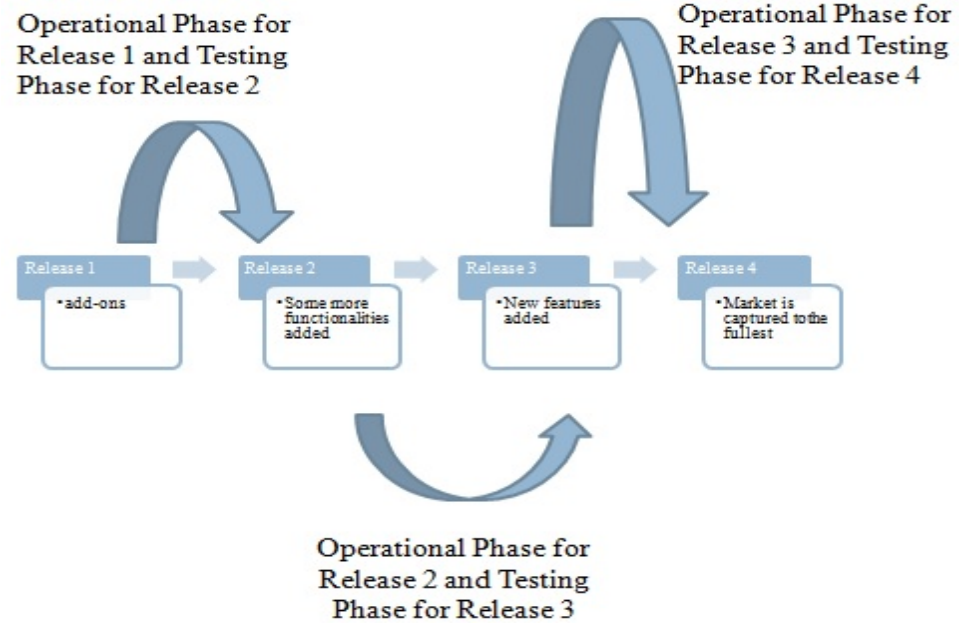
where $a_1 = a_1^*$, as the first version is the initial offering of the firm.

Upgrading the software increases its code complexity which might lead to an increment in the bug content. Thus, adding new functionalities and handling the reported issues leads to evolved and better software. The upgraded version of the software is released in the market and is referred to as the second release of the software. The model differentiates the faults on the basis of their origin i.e. the faults originate from the source code of release 1 or due to the new code added in version 2 (Singh et al., 2012b).

$$\begin{aligned} m_2\left(X_2(t)\right) &= \left(a_2 + a_1\left(1 - F_1\left(X_1(t_1)\right)\right)\right).F_2\left(X_2(t - t_1)\right) \\ &= a_2^*.F_2\left(X_2(t - t_1)\right) \end{aligned} \tag{5}$$

where $a_2^* = \left(a_2 + a_1\left(1 - F_1\left(X_1(t_1)\right)\right)\right)$ accounts for number of newly generated bugs in version 2 and remaining faults of preceding version.

Figure 1 Successive versions of software (Singh et al., 2011)



Generally, when the first version is released into the field (operational phase) it means that vigorous testing for second version has started whereas still the first version is under testing (see Figure 1). For first version testing process is continued till firms doesn't back out its support. And once second version is released, testing for version 3 begins whereas version 1 will receive support from the developer. However, it is quite evident fact that

there can be multiple versions being tested simultaneously only difference being in their intensity level. Similarly, for *n*-th version of the software, the MVF can be given as follows:

$$\begin{aligned} m_n\left(X_n(t)\right) &= \left(a_n + a_{n-1}\left(1 - F_{n-1}\left(X_{n-1}\left(t_{n-1}\right)\right)\right)\right).F_n\left(X_n\left(t - t_{n-1}\right)\right) \\ &= a_n^*.F_n\left(X_n\left(t - t_{n-1}\right)\right) \end{aligned} \tag{6}$$

where $a_n^* = \left(a_n + a_{n-1}\left(1 - F_{n-1}\left(X_{n-1}\left(t_{n-1}\right)\right)\right)\right)$ accounts for newly generated faults in version *n* and leftover bugs of (*n*–1)-th version. For the analytical purpose the fault removal phenomenon is considered to follow exponential distribution.

4 Resource allocation problem for multi-version software

In today's time when companies are following agile technology for software development, they may have no time to wait for finalising their strategies for developing and testing next version product. Moreover, when a firm plans to develop software their aim is to differentiate and build software which can uphold competitive edge. From the beginning firms have a broader vision of developing several versions of the software as the current environment is quite volatile and new versions are launched after few months. The fundamental behind this concept is to start developing the software as and when it is ready it is introduced into the market but before its release, firms start the development process of its next version.

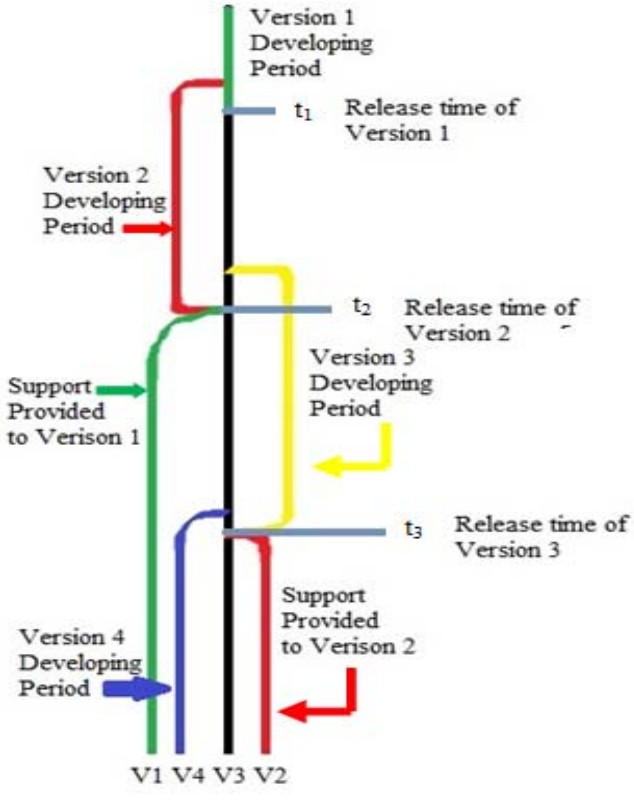
It can be visualised from Figure 2 that at time say 't_1' the software firms wish to launch the first version of the software product so before or at time 't_1' firms will start working on its second version. If firm releases the second version of the software at time 't_2', they will be providing support to version 1. Developers may not be updating version 1 in this period but the idea and development of verison-3 might have begun. Similarly, assuming that version 3 is introduced at time point 't_3' at the same time firm will be providing support to versions 1 and 2. At the same time developers will start the building and testing process for version 4 and this cycle continues.

In line with above explanation, there can be set of firms who may not focus on providing support to earlier version but remain focused on developing and testing new versions of the software. Considering all the aspects there is at least one version which will be termed as master version (called as market driving version) for which the developers will be providing updates (in terms of both feature enhancement and bug fixers), some can be in their initial phases of development or testing and some versions for only support (bug fixers) will be released. Thus, in totality firms may have three to four versions to look after and for which firms need to allocate the available amount of resources.

Example can be quoted from Google's current offering Android for smart phone (www.android.com). There are certain numbers of handsets having the previous versions of Android i.e. either version 6 or version 7 as there Operating System (OS) which was launched in 2015–2017 period. In mid of 2017, verion-8 of android was launched and certain products with this OS are also available. At the same time its succeeding version 9 termed as Android Pie is under beta testing phase. Thus, it become wise to

say that currently version 8 is market driving version where as versions 6 and 7 are just receiving support and version 9 is in its initial phase before its launch. Similarly, we can also look for Microsoft Windows having similar pattern (www.microsoft.com). At present, only few users are employing Windows Vista and Windows 8. On contrary a large pool of users are inclined towards the use of Windows 10 and thus can be termed as market driving version. For Windows 10 developers are working releasing its monthly and yearly updates as well. There can be some users who are still using Windows XP but the developers have now stopped providing support to this version of the software. Wherein we can claim that at some point firms have at least three four versions being simultaneously worked upon be its support, fault debugging, feature enhancement and so on. With this objective, we are considering a scenario in which firms need to allocate resources among multiple versions of the software which are in different phases of their lifecycle.

Figure 2 Infusion strategies for multi-upgraded software system



Enormous amount of resources are invested in developing software, starting from day it is conceived till the day when it is delivered but majority of resources are consumed in testing phase of SDLC. Therefore it calls for an optimal way of allocating resources among testing phase of different versions of the software. In view of the fact that each version has some predetermined time at which it will be released. Here, we have considered the case of the software having four releases which are being tested for

removing faults. Moreover, resources always have an upper threshold limit which is specified by the firms. Considering all the aspects and assuming that firms wish to debug maximum number of bugs present, the optimisation problem can be formulated as follows:

$$\left.\begin{array}{l} \text{Max} \quad \sum_{i=1}^{N} m_i \\ \\ \textit{Subject to} \\ \sum_{i=1}^{N} X_i \le Z \ , \ X_i \ge 0, i = 1,...,N \end{array}\right\} \qquad \text{(Prob1)}$$

where m_i is the expected count of faults to be debugged using X_i amount of resources for *i*-th version of the software and 'Z' being threshold budget as specified by firms for testing phase. Substituting the form for MVF as given in equation (2) we can rewrite (Prob1) as:

$$\left.\begin{array}{l} \text{Max} \quad \sum_{i=1}^{N} m_i\left(X_i\right) = \sum_{i=1}^{N} a_i^*\left(1-e^{-b_i X_i}\right) \\ \\ \textit{Subject to} \\ \sum_{i=1}^{N} X_i \le Z \ , \ X_i \ge 0, i = 1,...,N \end{array}\right\} \qquad \text{(Prob2)}$$

There exists many ways to obtain the optimal allocation of resources for different versions of the software. Here, for solving (Prob2) we made use of Dynamic Programming approach and formulated recursive functions. It is a mathematical optimisation technique that usually works in simplifying a decision by breaking the complex problem into a series of decisions taken in stepwise manner. By making use of Optimality Principle as given by Bellman's following are the forward recursive functions (Hadley, 1964):

$$\text{For } 'i = 1' \text{ we have } g_1\left(Z\right) = \max_{X_1 = Z_1}\left\{a_1\left(1-e^{-b_1 X_1}\right)\right\} \qquad (7)$$

And for '$i = n$' we have

$$g_n\left(Z\right) = \max_{0 \le X_n \le Z}\left\{a_n^*\left(1-e^{-b_n X_n}\right) + g_{n-1}\left(Z - X_n\right)\right\}; n = 2,...,N \qquad (8)$$

Using equations (7) and (8), we can allocate resources to different versions of the software with the help of theorem as given by Kapur et al. (2003) and compute the optimal X_i^*. As resources are allocated in a sequential fashion there can be certain instances when the software version having lower detectability index $\left(a_i^* b_i\right)$ has not been allocated any resources for testing. Though allocation of resources is optimal still some version might not be allocated enough resources to maintain their reliability level and will not be tested at all. This will have an impact on overall performability of the software versions. Occurrence of this situation will not be treated as healthy one.

Therefore, it is always expected that a certain level of reliability should be achieved in each version before it is being released. As a part of assumption it has been considered that, maximum number of faults fixed before release maximum will be level of reliability achieved. Accordingly, (Prob2) needs to be modified with an addition of constraint that will specify a lower limit on number of bugs that needs to be fixed. The appended problem can be stated as follows:

$$\left.\begin{aligned} &\text{Max} \quad \sum_{i=1}^{N} m_i\left(X_i\right) = \sum_{i=1}^{N} a_i^*\left(1-e^{-b_i X_i}\right) \\ &\textit{Subject to} \\ &\sum_{i=1}^{N} X_i \le Z \;\;,\; X_i \ge 0, i = 1,...,N \\ &m_i = a_i^*\left(1-e^{-b_i X_i}\right) \ge p_i a_i^* = a_{i0} \;, i = 1,...,N \end{aligned}\right\} \quad \text{(Prob3)}$$

where p_i is the least expected proportions of faults and a_{i0} is the minimum number of faults that needs to be fixed prior to its release. (Prob3) can be solved with the help of Dynamic Programming by using the Lagrange multiplier or making use of substitution process (Hadley, 1964). Here we have specifically focused on later one in many instances the former approach leads to increased complexity at each stage with large number of cases that needs to be executed before taking an optimal decision. Basically, our aim is to convert (Prob3) into similar pattern as (Prob2) so that optimal solution can be directly achieved. For that very purpose the add-on constraint has been modified as follows:

$$m_i\left(X_i\right) \ge a_{i0} \Rightarrow a_i^*\left(1-e^{-b_i X_i}\right) \ge a_{i0} \tag{9}$$

Above equation can be further rewritten as:

$$X_i \ge -\frac{1}{b_i}\log\left[1-\frac{a_{i0}}{a_i^*}\right] = Z_i \;\; (\text{say}),\;\; i = 1,...,N \tag{10}$$

Making use of equation (10) in (Prob3) we have the amended optimal problem as:

$$\left.\begin{aligned} &\text{Max} \quad \sum_{i=1}^{N} m_i\left(X_i\right) = \sum_{i=1}^{N} a_i^*\left(1-e^{-b_i X_i}\right) \\ &\textit{Subject to} \\ &\sum_{i=1}^{N} X_i \le Z \;\;,\; X_i \ge 0, i = 1,...,N \\ &X_i \ge Z \;\;, i = 1,...,N \end{aligned}\right\} \quad \text{(Prob4)}$$

In order to convert (Prob4) to a simpler form we considered $Y_i = X_i - Z_i$; $\left(i = 1,2,...,N\right)$ which will reduce (Prob4) to (Prob5) given as follows:

$$\left.\begin{aligned}
&\text{Max} \quad \sum_{i=1}^{N} m_i = \sum_{i=1}^{N} \overline{a_i}\left(1-e^{-b_i Y_i}\right) \\
&\textit{Subject to} \\
&\sum_{i=1}^{N} Y_i \le Z - \sum_{i=1}^{N} Z_i \; = \overline{Z}\,(\text{suppose}) \\
&Y_i \ge 0, i = 1,.....,N \\
&\overline{a_i} = a_i^* - a_{i0}, i = 1,...,N
\end{aligned}\right\} \qquad \text{(Prob5)}$$

where, $Z - \sum_{i=1}^{N} Z_i \; = \overline{Z}$ has been assumed. Moreover, obtained (Prob5) is same as (Prob2) thus optimal solution is achieved by using theorem as given by Kapur et al. (2003). In the next section, aforesaid models have been evaluated.

5 Numerical illustration

In order to have better understanding, a case of software having its four versions has been presented and optimal resource allocation has been examined by using the estimated values for parameters a_i and b_i as supplemented in Table 1. Here, the considered data has four versions out of which it might be the case that versions 1 and 2 are getting support whereas verison-3 is under usage (i.e. timely updates are released) and for version 4 its vigorous testing has been started (testing under development period). The resources allocated to each version are in accordance with the initial fault content i.e. the version containing more bugs will be allocated more resources. We have assumed that aggregated assets available for testing process $Z = 600\$$. Initially (Prob2) was solved for optimal values of X_i^* by making use of equations (7) and (8) and obtained values are represented in Table 1 along with percentage (%) of flaws fixed and leftover faults in each version of software (Thiriez, 2000). It can be observed that out of 653.35, 459.25 numbers of faults were fixed accounting for 70.29% of faults from all releases of software; whereas in 4th version 29.94% of faults were removed. Thus, in 4th version of the software, the remaining bugs can prompt and disrupt smooth running in operational stage. Clearly this won't be acceptable to developers and will aim for remove at least a 60% of faults from each release before launching it into the field $\left(p_i = 0.6\right)$ for $\left(i = 1,2,3,4\right)$. Since latent bugs in software depicts the quality of the software, more or close to half the number of faults debugged enhances the reliability of the software which can be added as an extra constraint while allocating resources (Prob3).

Table 1 Parameter estimates and initial resource allocation (Prob2)

Release	a_i^*	b_i	X_i^*	m_i^*	*% of bugs removed*	*% of bugs remaining*
1	147.36	0.01231	138.7334	120.63	81.86	18.14
2	182.95	0.00979	173.1159	149.35	81.64	18.36
3	187.13	0.008534	185.1525	148.59	79.40	20.60
4	135.91	0.003454	102.9982	40.69	29.94	70.06
Total	653.35		600	459.25	70.29	29.71

On solving (Prob3) we have the results summarised in Table 2, which clearly places a lower capping of debugging at least 60% of bugs from all software versions.

Table 2 Resource allocation from (Prob3)

Release	a_i^*	a_{i0}	Z_i^*	Y_i^*	$m_i(Y_i)$		*% of bugs removed*	*% of bugs remaining*
1	147 m_i^* .36	88.42	74.46	20.39	13.08	101.50	68.88	31.12
2	182.95	109.77	93.59	24.37	15.54	125.31	68.49	31.51
3	187.13	112.28	107.37	14.52	8.72	121	64.66	35.34
4	135.91	81.55	265.28	0	0	81.55	60	40
Total	653.35	392.01	540.71	59.29	37.34	429.35	65.72	34.28

It is quite evident that initially 459.25 numbers of bugs were removed as whole which got reduced to 429.35 once the capping of 60% was placed for all versions. Thus, leading to decrease in the overall percentage of fault removed to 65.72% while the original debugging percentage was around 70%. In order to maintain same overall level of percentage and at the same time putting a constraint for each version may require addition of extra testing resources. For the present case, if firms wish to put amount of resources assuming it to be 60 we can solve (Prob5) and obtain the results as given in Table 3.

Table 3 Allocation after addition of extra resources (Prob5)

Release	a_i^*	a_{i0}	Z_i^*	Y_i^*	$m_i(Y_i)$	X_i^*	m_i^*	*% of bugs removed*	*% of bugs remaining*
1	147.36	88.42	74.46	36.61	21.38	111.08	109.80	74.51	25.49
2	182.95	109.77	93.59	44.76	25.97	138.36	135.74	74.19	25.81
3	187.13	112.28	107.37	37.91	20.69	145.28	132.97	71.06	28.94
4	135.91	81.55	265.28	0.00	0.00	265.28	81.55	60.00	40.00
Total	653.35	392.01	540.71	119.28	68.04	660.00	460.05	70.41	29.59

The results depict that as a whole 70.41% of faults are removed at the same time at least 60% of faults are debugged from each version. On clear examination of all Tables 1, 2 and 3, it can be seen that the resource investment has high influence on faults being removed. Moreover, the decision regarding bifurcation of resources among versions decides the operational capability of particular version of the software.

6 Conclusion

In this paper we have proposed a novel concept and formulated different optimisation problems accounting for resource allocation in several versions of the software. Separate set of optimisation problems are proposed to describe the allocation of resources in upgraded versions of the software based on resource constraint and other with added constraint representing the aspiration level for fault removal in each version of the software. Further by making use of dynamic programming approach and utilising the basic recursion conditions we obtained the optimal resources that should be invested. Putting more resources often increases the reliability, as more number of faults can be fixed. In many cases, during testing process a level is reached where you need to invest more in order to reap more. Thus, putting extra efforts becomes difficult for firms and a trade-off is required which can be in terms of level of reliability the firm is looking for which is presented in form of optimal problems here.

Acknowledgement

The work presented in this article is supported by grants to second author via Rajiv Gandhi National Fellowship from UGC, New Delhi, India.

References

Anand, A., Singh, O. and Das, S. (2015) 'Fault severity based multi up-gradation modeling considering testing and operational profile', *International Journal of Computer Applications*, Vol. 124, No. 4, pp.9–15.

Anand, A., Singh, O., Kapur, P.K. and Das, S. (2014) 'Modeling conjoint effect of faults testified from operational phase for successive software releases', *Proceedings of the 5th International Conference on Life Cycle Engineering and Management (ICDQM'14)*, pp.83–94.

Cao, Z., Lin, J., Wan, C., Song, Y., Zhang, Y. and Wang, X. (2017) 'Optimal cloud computing resource allocation for demand side management in smart grid', *IEEE Transactions on Smart Grid*, Vol. 8, No. 4, pp.1943–1955.

Charlotte, J. and Thomas, M. (2018) *Top Software Failures in Recent History* (accessed on 10 September 2018).

Goel, A.L. and Okumoto, K. (1979) 'Time-dependent error-detection rate model for software reliability and other performance measures', *IEEE transactions on Reliability*, Vol. 28, No. 3, pp.206-211.

Hadley, G. (1964) *Nonlinear and Dynamic Programming*, Addison-Wesley, Reading Mass.

Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R. (2004) 'Optimal allocation of testing-resource considering cost, reliability, and testing-effort', *Proceedings of the 10th IEEE Dependability Computing on Pacific Rim International Symposium on IEEE*, pp.103–112.

Jha, P.C., Gupta, D., Anand, S. and Kapur, P.K. (2006) 'An imperfect debugging software reliability growth model using lag function with testing coverage and related allocation of testing effort problem', *Communication Dependability Quality Management: International Journal*, Vol. 9, No. 4, pp.148–165.

Kapur, P.K., Chanda, U. and Kumar, V. (2010a) 'Dynamic allocation of testing effort when testing and debugging are done concurrently', *An International Journal Serbia Communication in Dependability and Quality Management*, Vol. 13 No. 3, pp.14–28.

Kapur, P.K., Jha, P.C. and Bardhan, A.K. (2003) 'Dynamic programming approach to testing resource allocation problem for modular software', *Ratio Mathematical*, Vol. 14, No.1, pp.27–40.

Kapur, P.K., Pham, H., Chanda, U. and Kumar, V. (2013) 'Optimal allocation of testing effort during testing and debugging phases: a control theoretic approach', *International Journal of Systems Science*, Vol. 44, No. 9, pp.1639–1650.

Kapur, P.K., Tandon, A. and Kaur, G. (2010b) 'Multi up-gradation software reliability model', *Proceedings of the 2nd International Conference on IEEE Reliability, Safety and Hazard (ICRESH'10)*, pp.468–474.

Kumar, V. and Sahni, R. (2016) 'An effort allocation model considering different budgetary constraint on fault detection process and fault correction process', *Decision Science Letters*, Vol. 5, No. 1, pp.143–156.

Kumar, V., Khatri, S., Dua, H., Sharma, M. and 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,* Vol. 21, No. 6. Doi: 10.1142/S0218539314500272.

Kumar, V., Mathur, P., Sahni, R. and Anand, M. (2016) 'Two-dimensional multi-release software reliability modeling for fault detection and fault correction processes', *International Journal of Reliability, Quality and Safety Engineering*, Vol. 23, No. 3. Doi: 10.1142/S0218539316400027.

Leung, Y.W. (1997) 'Dynamic resource-allocation for software-module testing', *Journal of Systems and Software*, Vol. 37, No. 2, pp.129–139.

Ohtera, H. and Yamada, S. (1990) 'Optimal allocation and control problems for software-testing resources', *IEEE Transactions on Reliability*, Vol. 39, No. 2, pp.171–176.

Singh, O., Aggrawal, D., Anand, A. and Kapur, P.K. (2015) 'Fault severity based multi-release SRGM with testing resources', *International Journal of System Assurance Engineering and Management*, Vol. 6, No. 1, pp.36–43.

Singh, O., Kapur, P.K. and Anand, A. (2011) 'A stochastic formulation of successive software releases with faults severity', *Proceedings of the 4th IEEE International Conference on Industrial Engineering and Engineering Management (IEEM'11)*, IEEE, pp.136–140.

Singh, O., Kapur, P.K. and Singh, J.N.P. (2012b) 'Testing-effort based multi up-gradation software reliability growth model', *Proceeding of the Communication in Dependability and Quality Management-An International Journal, Serbia (CDQM'12)*, Vol. 15, No. 1, pp.88–100.

Singh, O., Kapur, P.K., Anand, A. and Singh, J. (2009) 'Stochastic differential equation based modeling for multiple generations of software', *Proceedings of the 4th International Conference on Quality, Reliability and Infocom Technology (ICQRIT), Trends and Future Directions, Narosa Publications*, pp.122–131.

Singh, O., Kapur, P.K., Khatri, S.K. and Singh, J.N.P. (2012a) 'Software reliability growth modeling for successive releases', *Proceeding of the 4th International Conference on Quality Reliability and Infocom Technology (ICQRIT'12)*, pp.77–87.

Thiriez, H. (2000) 'Or software lingo', *European Journal of Operational Research*, Vol. 12, pp.655–656.

Wei, W., Fan, X., Song, H., Fan, X. and Yang, J. (2018) 'Imperfect information dynamic stackelberg game based resource allocation using hidden Markov for cloud computing', *IEEE Transactions on Services Computing*, Vol. 11, No. 1, pp.78–89.

Xie, M. and Yang, B. (2001) 'Optimal testing-time allocation for modular systems', *International Journal of Quality and Reliability Management*, Vol. 18, No. 8, pp.854–863.

Yamada, S., Ichimori, T. and Nishiwaki, M. (1995) 'Optimal allocation policies for testing-resource based on a software reliability growth model', *Mathematical and Computer Modelling*, Vol. 22, Nos. 10/12, pp.295–301.

Yamada, S., Ohtera, H. and Narihisa, H. (1986) 'Software reliability growth models with testing-effort', *IEEE Transactions on Reliability*, Vol. 35, No. 1, pp.19–23.