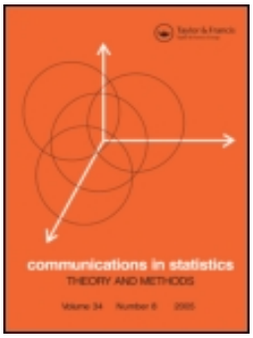

Communications in Statistics - Theory and Methods

ISSN: 0361-0926 (Print) 1532-415X (Online) Journal homepage: http://www.tandfonline.com/loi/lsta20

Determining the optimal allocation of testing resource for modular software system using dynamic programming

M. G. M. Khan, N. Ahmad & L. S. Rafi

To cite this article: M. G. M. Khan, N. Ahmad & L. S. Rafi (2016) Determining the optimal allocation of testing resource for modular software system using dynamic programming, Communications in Statistics - Theory and Methods, 45:3, 670-694, DOI: 10.1080/03610926.2013.834455

To link to this article: http://dx.doi.org/10.1080/03610926.2013.834455

Accepted author version posted online: 30 Oct 2015.
Published online: 30 Oct 2016.

Submit your article to this journal

Article views: 47

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=lsta20

Download by: [University of the South Pacific], [Dr Mohammad Khan] **Date:** 09 March 2016, At: 22:22

COMMUNICATIONS IN STATISTICS—THEORY AND METHODS
2016, VOL. 45, NO. 3, 670–694
http://dx.doi.org/10.1080/03610926.2013.834455

Taylor & Francis
Taylor & Francis Group

Determining the optimal allocation of testing resource for modular software system using dynamic programming

M. G. M. Khan[a], N. Ahmad[b], and L. S. Rafi[c]

[a]School of Computing, Information and Mathematical Sciences, The University of the South Pacific, Suva, Fiji; [b]University Department of Statistics and Computer Applications, T. M. Bhagalpur University, Bhagalpur, Bihar, India; [c]College of Foundation, The University of the South Pacific, Suva, Fiji

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

ABSTRACT

Reliability is a major concern in the process of software development because unreliable software can cause failure in the computer system that can be hazardous. A way to enhance the reliability of software is to detect and remove the faults during the testing phase, which begins with module testing wherein modules are tested independently to remove a substantial number of faults within a limited resource. Therefore, the available resource must be allocated among the modules in such a way that the number of faults is removed as much as possible from each of the modules to achieve higher software reliability. In this article, we discuss the problem of optimal resource allocation of the testing resource for a modular software system, which maximizes the number of faults removed subject to the conditions that the amount of testing-effort is fixed, a certain percentage of faults is to be removed and a desired level of reliability is to be achieved. The problem is formulated as a non linear programming problem (NLPP), which is modeled by the inflection S-shaped software reliability growth models (SRGM) based on a non homogeneous Poisson process (NHPP) which incorporates the exponentiated Weibull (EW) testing-effort functions. A solution procedure is then developed using a dynamic programming technique to solve the NLPP. Furthermore, three special cases of optimum resource allocations are also discussed. Finally, numerical examples using three sets of software failure data are presented to illustrate the procedure developed and to validate the performance of the strategies proposed in this article. Experimental results indicate that the proposed strategies may be helpful to software project managers for making the best decisions in allocating the testing resource. In addition, the results are compared with those of Kapur et al. (2004), Huang and Lyu (2005), and Jha et al. (2010) that are available in the literature to deal the similar problems addressed in this article. It reveals that the proposed dynamic programming method for the testing-resource allocation problem yields a gain in efficiency over other methods.

ARTICLE HISTORY
Received 24 February 2013
Accepted 8 August 2013

KEYWORDS
Dynamic programming technique; Inflection S-shaped models; Non linear programming problem; Resource allocation problem; Software reliability growth model.

MATHEMATICS SUBJECT CLASSIFICATION
Primary 90-08; Secondary 90B32.

1. Introduction

Software is broadly classified as operating system and applications software, and it plays a vital role in modern life. Thus, the reliability of software becomes a crucial feature of the computer systems as its breakdown in the system could result in fiscal, possessions, and human loss.

CONTACT M. G. M. Khan khan_mg@usp.ac.fj School of Computing, Information and Mathematical Sciences, The University of the South Pacific, Suva, Fiji.

© 2016 Taylor & Francis Group, LLC

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Although measuring or predicting software reliability is a difficult task, it is important for the assessment of the performance of the underlying software systems. Much research has been conducted over the past decades to study software reliability and many software reliability growth models (SRGM) have been proposed (Xie, 1991; Lyu, 1996; Musa et al., 1987). In addition to software reliability measurement, SRGMs can also help to predict the fault detection coverage in the testing phase.

A software development process consists of four phases: specification, design, coding, and testing (Myers, 1976). Testing, the final phase, where the software is tested to detect and correct software faults, involves the following consecutive stages: module testing, integration testing, system testing, and installation testing. A software consists of several modules that are tested independently during the module testing, and the interfaces among these modules are tested in the integration testing. Lastly, the complete software system, where the modules are interconnected, is tested under the stimulated user environment in the system testing.

Moreover, all the testing activities of different modules should be completed with a limited resource, approximately 40–50% of the total amount of software development resources (Yamada et al., 1995). When resources available are limited for the testing of software, it is important to allocate the testing resources efficiently among all the modules so that the maximum reliability of the complete system is achieved. As the reliability of software is proportional to the number of faults removed, the problem of achieving maximum software reliability may be treated as the problem of the maximum number of faults to be removed. Many authors have addressed this problem to determine the optimal resource allocation over the years, including Ohtera and Yamada (1990), Yamada et al. (1995), Huang et al. (2002, 2004), Huang and Lyu (2005), Huang and Lo (2006), Lo et al. (2002), Lyu et al. (2002), Leung (1997), Kapur et al. (2004, 2009, 2010), Jha et al. (2010), Rajan (2006), Xie and Yang (2001), Li et al. (2009), Wang et al. (2008, 2010), and Chaudhary et al. (2012).

In this article, the problem of optimal resource allocation during module testing is studied and a general formulation of the problem is presented as a non linear programming problem (NLPP). The formulated NLPP maximizes the total number of faults expected to be removed during module testing satisfying the limited testing resources. Sometimes management may want a desired level of reliability to be achieved or/and a certain percentage of the number of faults to be removed. These requirements are also adopted in the NLPP so that the results are more acceptable to the customers. Further, we have also discussed three special cases of the allocation problem, which are formulated as NLPPs in case either or both of the requirements are not necessarily to be met. These NLPPs are modeled by the inflection S-shaped SRGM based on a non homogeneous Poisson process (NHPP) with exponentiated Weibull (EW) testing-effort function. Treating a testing module as a stage, the NLPPs are considered as multi-stage decision problems and solved by developing a solution procedure using a dynamic programming technique. It is shown that the optimal allocation of testing-resources among software modules can improve software reliability. Finally, several numerical examples using three software failure data sets are given to illustrate the procedure developed in this article and the results are compared with those of Kapur et al. (2004), Huang and Lyu (2005), and Jha et al. (2010). This comparison reveals that the proposed dynamic programming method within our framework yields a gain in efficiency over other methods.

The article is organized as follows. In Sec. 2, inflection S-shaped SRGM with EW testing-effort function based on an NHPP is reviewed. In Sec. 3, the problems of testing-resource allocation are discussed and the details of the formulation of the problems as NLPPs are provided. The solution procedure for the problems using dynamic programming technique is discussed in Sec. 4. The computational details of the solution procedure are illustrated with

numerical examples in Sec. 5. Finally, in Sec. 6, an investigation is carried out to compare the results obtained by the authors' proposed dynamic programming approach with that of Kapur et al. (2004), Huang and Lyu (2005), and Jha et al. (2010).

2. Review of SRGM with EW testing-effort function

A SRGM is a mathematical expression of the software error occurrence and the removal process. Numerous SRGMs have been developed during the last three decades and they can provide very useful information about how to improve software reliability (Musa et al., 1987; Xie, 1991; Lyu, 1996). Among these models, the exponential growth model and the delayed S-shaped growth model have been shown to be very useful in fitting software failure data. Actually, the exponential growth model is sometimes insufficient and inaccurate in analyzing real software failure data for reliability assessment. Hence, Ohba (1984) proposed the inflection S-shaped growth model by modifying the logistic curve model. He described increases in the fault removal rate with time and assumed the presence of two types of errors in the software. Many authors incorporated the concept of testing-effort into SRGM based on the NHPP to get a better description of the fault detection phenomenon. Later, Kapur et al. (2004) modified the inflection S-shaped model and incorporated the testing-effort in an NHPP model. Recently, Bokhari and Ahmad (2007), and Ahmad et al. (2008a, b, 2010) also proposed exponential-type and inflection S-shaped SRGM with the EW testing-effort function to predict the behavior of failure and fault of a software product. The past results indicate that this model is suitable for software reliability application.

The inflection S-shaped SRGM with EW testing-effort function is formulated on the following assumptions (Ohba, 1984; Yamada and Osaki, 1985; Yamada et al., 1986, 1993; Kapur et al., 1999, 2004; Kuo et al., 2001; Huang and Lyu, 2005; Huang and Lo, 2006; Bokhari and Ahmad, 2007; Ahmad et al., 2008a, 2010, 2011):

1. The software system is subject to failures at random times caused by errors remaining in the system.
2. Each time a failure occurs, the error causing that failure is immediately removed and no new errors are introduced.
3. The error removal phenomenon in software testing is modeled by NHPP.
4. The mean number of errors detected in the time interval $(t,\ t+\Delta t)$ by the current testing-effort expenditures is proportional to the mean number of detectable errors in the software.
5. The proportionality increases linearly with each additional error removal.
6. Testing-effort expenditures are described by the EW testing-effort function.

Let $m(t)$ be the mean value of the expected number of errors detected in time $(0, t]$. As the expected number of errors detected at any time is finite, $m(t)$ is an increasing function of t and $m(0) = 0$.

According to the assumptions stated in (2) and (5), we obtain the following differential equation:

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = \phi(t)\,(a - m(t)), \tag{1}$$

where

$$\phi(t) = b\left[r + (1 - r)\frac{m(t)}{a}\right],$$

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

r is the inflection rate and represents the proportion of independent errors present in the software, $w(t)$ is the current testing-effort expenditure at time t, a is the expected number of errors in the system, and b is the error detection rate per unit testing-effort at time t.

Solving (1) with the initial condition that at $t = 0$, $W(t) = 0$, $m(t) = 0$, the following is obtained:

$$m(t) = \frac{a\left[1 - e^{-bW(t)}\right]}{1 + ((1 - r)/r)e^{-bW(t)}}. \tag{2}$$

It is noted that the current testing-effort expenditure is given as

$$w(t) = \frac{dW(t)}{dt}.$$

During the software testing phase, much testing-effort is consumed. The consumed testing-effort indicates how the errors are detected effectively in the software and can be modeled by different distributions (Yamada et al., 1986, 1993; Musa et al., 1987; Kapur et al., 1999). Actually, the software reliability is highly related to the amount of testing-effort expenditures spent on detecting and correcting software errors. Ahmad et al. (2008a) and Bokhari and Ahmad (2007) have shown that EW testing-effort function is a very suitable and more flexible testing resource for assessing the reliability of software products. The cumulative testing-effort expenditure consumed in (0,t] is depicted in the following:

$$W(t) = \alpha(1 - e^{-\beta t^{\delta}})^{\theta}, \alpha > 0, \beta > 0, \delta > 0, \theta > 0, \tag{3}$$

and the current testing-effort consumed at testing time t is

$$w(t) = \alpha.\beta.\delta.\theta.t^{\delta-1}.e^{-\beta t^{\delta}}(1 - e^{-\beta t^{\delta}})^{\theta-1}, \tag{4}$$

where α is the total amount of testing-effort expenditures; β is the scale parameter, and δ and θ are shape parameters.

When $\delta, \theta > 1$, the EW testing-effort function, $w(t)$, reaches its maximum value at the time

$$t_{\max} = \left[\frac{2(\delta\theta - 1)}{\beta\delta(\theta + 1)}\right]^{1/\delta}.$$

We have the following five special cases:

- Yamada exponential curve (Yamada et al., 1986): For $\theta = 1$ and $\delta = 1$, there is an exponential testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t}), \alpha > 0, \beta > 0.$$

- Yamada Rayleigh curve (Yamada et al., 1986): For $\theta = 1$ and $\delta = 2$, there is a Rayleigh testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t^2}), \alpha > 0, \beta > 0.$$

- Yamada Weibull curve (Yamada et al., 1993): For $\theta = 1$, there is a Weibull testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t^{\delta}}), \alpha > 0, \beta > 0, \delta > 0.$$

- Generalized exponential curve (Quadri et al., 2006): For $\delta = 1$, there is a generalized exponential testing-effort function, and the cumulative testing-effort consumed in time

$(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t})^{\theta}, \alpha > 0, \beta > 0, \theta > 0.$$

- Burr Type X curve (Ahmad et al., 2009): For $\delta = 2$, there is a Burr Type X testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t^2})^{\theta}, \alpha > 0, \beta > 0, \theta > 0.$$

Furthermore, we describe a flexible SRGM with the EW testing-effort expenditure as

$$m(t) = \frac{a\left[1 - e^{-b\alpha(1-e^{-\beta t^{\delta}})^{\theta}}\right]}{1 + ((1 - r)/r)e^{-b\alpha(1-e^{-\beta t^{\delta}})^{\theta}}}.$$

On the other hand, if we let the inflection rate $r = 1$ in the above equation, we have an SRGM proposed by Ahmad et al. (2008a)

$$m(t) = a(1 - e^{-b\alpha(1-e^{-\beta t^{\delta}})^{\theta}}).$$

In addition, the expected number of errors to be detected eventually is

$$m(\infty) = \frac{a\left[1 - e^{-b\alpha}\right]}{1 + ((1 - r)/r)e^{-b\alpha}}.$$

The software reliability representing the probability that no failures occur in the time interval $(t, t + \Delta t)$ given that the last failure occurred at testing time Δt is given by

$$R(\Delta t \,|t) = e^{-[m(t+\Delta t)-m(t)]}.$$

Taking the logarithm on both sides of the above equation, we obtain

$$\ln R(\Delta t \,|t) = -\left[m(t + \Delta t) - m(t)\right].$$

Now, we define another measure of reliability as a ratio of the cumulative number of detected errors at time t to the expected number of initial errors (Huang and Lyu, 2005; Huang and Lo, 2006):

$$R(t) \equiv \frac{m(t)}{a} = \frac{\left[1 - e^{-bW(t)}\right]}{1 + ((1 - r)/r)e^{-bW(t)}}.$$

2.1. *Method of parameter's estimation*

In order to validate the proposed model, experiments on actual software failure data will be performed. Maximum likelihood estimation (MLE) and least square estimation (LSE) techniques are used to estimate the model parameters (Musa et al., 1987; Lyu, 1996). Using the method of LSE, the parameters α, β, δ, and θ in EW testing-effort function can be estimated. These parameters are determined for n observed data pairs in the form $(t_k,\ W_k)$ $(k = 1, 2, \ldots, n;\ 0 < t_1 < t_2 < \cdots < t_n)$, where W_k is the cumulative testing-effort consumed in time $(0,\ t_k]$. The least square estimators $\hat{\alpha}$, $\hat{\beta}$, $\hat{\delta}$, and $\hat{\theta}$ can be estimated by minimizing the following:

$$S(\alpha, \beta, \delta, \theta) = \sum_{k=1}^{n} [W_k - W(t_k)]^2$$

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

$$= \sum_{k=1}^{n} \left[\ln W_k - \ln \alpha - \theta \ln(1 - e^{-\beta t_k^{\delta}}) \right]^2. \tag{5}$$

Once the estimates of α, β, δ, and θ are known, the parameters of the SRGM for the module can be estimated through MLE method. The estimators of a, b, and r are determined for the n observed data pairs in the form (t_k, y_k) $(k = 1, 2, \ldots, n;\ 0 < t_1 < t_2 < \ldots\ldots < t_n)$ where y_k is the cumulative number of software errors detected up to time t_k or $(0, t_k]$.

The likelihood function for that module is given by

$$\begin{aligned} L'(a, b, r) &\equiv P\left\{N(t_i) = y_i, i = 1, 2, \ldots, n\right\} \\ &= \prod_{k=1}^{n} \frac{[m(t_k) - m(t_{k-1})]^{(y_k - y_{k-1})}}{(y_k - y_{k-1})!} \cdot e^{-[m(t_k) - m(t_{k-1})]}, \end{aligned} \tag{6}$$

where $t_0 \equiv 0$ and $y_0 \equiv 0$.

Taking logarithm both sides in (6), we have

$$\begin{aligned} L = \ln L' = &\sum_{k=1}^{n} (y_k - y_{k-1}) \ln [m(t_k) - m(t_{k-1})] \\ &- \sum_{k=1}^{n} [m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n} \ln \left[(y_k - y_{k-1})!\right]. \end{aligned}$$

Consequently, the maximum likelihood estimates a, b, and r can be obtained by solving

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial r} = 0$$

as discussed in the Appendix.

3. Problem of testing-resource allocation

In module testing various kinds of testing-resource allocation problems can be considered to make the best use of a specified total testing-resource. The resource has to be allocated appropriately to each software module, which is tested independently and simultaneously, so that the maximum reliability of the complete system is achieved. In this section, we will consider the resource allocation problems for flexible SRGM with EW testing-effort functions.

Assumptions (Xie and Yang, 2001; Huang and Lyu, 2005; Huang and Lo, 2006; Yamada et al., 1995; Ohtera and Yamada, 1990; Khan et al., 2008a):

1. The software system is composed of N independent modules, and the software modules are tested independently. The number of software faults remaining in each module can be estimated by the SRGM with EW testing-effort function.
2. The total amount of testing-resource expenditures for the module testing is specified.
3. The manager has to allocate the specified total testing-resource expenditures to each software module wisely so that the software faults to be removed in the system may be maximized and the desired software reliability level is achieved.

3.1. *Formulation of resource allocation problems as NLPPs*

Suppose that a software system has N modules to be tested independently for removing faults in the whole testing phase. Let a_i and b_i be the expected number of initial faults content and the

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

error detection rate, respectively, in ith module ($i = 1, 2, \ldots, N$), which are already estimated by the SRGM discussed in Sec. 2. Also, let W_i be the amount of resources to be used in ith module during a fixed testing time T, where W_i is given as

$$W_i = \alpha_i \left(1 - e^{-\beta_i t^{\delta_i}}\right)^{\delta_i}, \quad \alpha_i > 0,\ \beta_i > 0,\ \delta_i > 0,\ \theta_i > 0.$$

Then, from Sec. 2, the SRGM with EW testing resource for ith module can be written as

$$m_i = \frac{a_i\left(1 - e^{-b_i W_i}\right)}{1 + \left[(1 - r_i)/r_i\right] e^{-b_i W_i}}, \tag{7}$$

where m_i be the number of faults expected to be removed from the ith module during a fixed testing time T and r_i be the proportion of independent faults.

It can be noticed that (7) is a non linear function of W_i and may be expressed as

$$m_i = \frac{a_i\left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}}, \tag{8}$$

where

$$\lambda_i = \frac{(1 - r_i)}{r_i}. \tag{9}$$

Sometimes in the resource allocation procedure one or more modules may not get any resources, to which the management may not agree where the modules are not tested any further. Moreover, during module testing it is also expected that each module is tested satisfactorily so that a certain percentage of the error content is desired to be removed from each module of the software.

Let p_i be the minimum proportion of initial faults to be removed in ith module, that is

$$m_i \geq p_i a_i.$$

This, along with (8), gives

$$W_i \geq A_i, \tag{10}$$

where A_i is the minimum amount of resources required to be allocated to ith module to remove p_i proportion of initial faults and is given by

$$A_i = -\frac{1}{b_i} \ln\left(\frac{1 - p_i}{1 + p_i \lambda_i}\right). \tag{11}$$

The management may also wish that the allocation of testing resource should achieve at a certain level of reliability for each of the modules.

Let R_0 be the minimum level of reliability to be achieved for the modules, that is

$$R_i \geq R_0,$$

where

$$R_i(t) = \frac{a_i\left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}}.$$

This gives

$$W_i \geq B_i, \tag{12}$$

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

where B_i is the minimum amount of resources required for ith module to achieve R_0 level of reliability of removing faults from the modules and is given by

$$B_i = -\frac{1}{b_i} \ln\left(\frac{1 - R_0}{1 + R_0\lambda_i}\right). \tag{13}$$

If the software project manager's aim is to determine the optimum allocation of the testing resources to each module that maximizes the number of faults removed in the system, then a reasonable criterion is to maximize the sum of the weighted total number of faults removed, that is

$$\text{Maximize } Z = \sum_{i=1}^{N} v_i m_i = \sum_{i=1}^{N} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}},$$

where v_i is the positive weight assigned to ith module according to its relative importance of fault removal. If $v_i = 1$ for all $i = 1, 2, \ldots, N$, the objective is to maximize the total number of faults removed.

One may choose these weights such that v_i is proportional to the number of initial faults a_i. The basis of this choice is the fact that the given system may be heterogeneous with respect to a module's initial faults a_i. If a module has a large a_i, then m_i of the module should also be expected to be large. Therefore, a way to maximize Z is to assign more weight to m_i, which could be done by using the given conjecture.

With $\sum_{i=1}^{N} v_i = 1$, this choice gives

$$v_i = \frac{a_i}{\sum_{i=1}^{N} a_i}; \quad i = 1, 2, \ldots, N. \tag{14}$$

If C be the total testing-resource expenditures available for the whole testing phase, then with a linear resource function "$\sum_{i=1}^{N} W_i$" the problem of finding the optimum allocation of the testing resources to each of module for a fixed C may be formulated as the following NLPP:

$$\begin{aligned}
&\text{Maximize } Z = \sum_{i=1}^{N} v_i m_i = \sum_{i=1}^{N} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}};\\
&\text{Subject to } \sum_{i=1}^{N} W_i \le C;\\
&W_i \ge A_i;\\
&W_i \ge B_i;\\
&A_i \ge 0 \text{ and } B_i \ge 0; \quad i = 1, 2, \ldots, N.
\end{aligned} \tag{15}$$

The restrictions (10) and (12) are added obviously to meet the desired proportion of faults to be removed as well as the desired level of reliability to be achieved from each of the modules.

Also note that the NLPP (15) has a feasible solution if and only if $\max(\sum_{i=1}^{N} A_i, \sum_{i=1}^{N} B_i) \le C$. Otherwise, an additional resource of amount $\max(\sum_{i=1}^{N} A_i, \sum_{i=1}^{N} B_i) - C$ is required to meet the restrictions.

If there is no restriction on the proportion of the faults to be removed and/or on the level of reliability to be achieved, NLPP (15) may be reduced to the following three particular cases.

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

3.1.1. *Case-1*

When there is no restriction on the proportion of the faults to be removed, that is, if restriction (10) is not imposed by the management, the problem of determining the optimum resource allocation to each of module for a fixed C that maximizes the removal of the faults in the software subject to the desired level of reliability to be achieved may be expressed as the following NLPP:

$$\begin{aligned} &\text{Maximize } Z = \sum_{i=1}^{N} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}}, \\ &\text{Subject to } \sum_{i=1}^{N} W_i \le C, \\ &W_i \ge B_i; \\ &B_i \ge 0; \quad i = 1, 2, \ldots, N. \end{aligned} \tag{16}$$

It can be seen that the NLPP (16) has a feasible optimum solution if and only if $\sum_{i=1}^{N} B_i \le C$. Otherwise, an additional resource of amount $\sum_{i=1}^{N} B_i - C$ is required to achieve R_0 level of reliability.

3.1.2. *Case-2*

Similarly, if there is no restriction on the level of reliability to be achieved but on the proportion of the faults to be removed, then the NLPP (15) needs to be modified by excluding the restriction (12) and can be expressed as

$$\begin{aligned} &\text{Maximize } Z = \sum_{i=1}^{N} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}}, \\ &\text{Subject to } \sum_{i=1}^{N} W_i \le C, \\ &W_i \ge A_i; \\ &A_i \ge 0; \quad i = 1, 2, \ldots, N. \end{aligned} \tag{17}$$

Note that this NLPP too has a feasible solution if and only if $\sum_{i=1}^{N} A_i \le C$. Otherwise, the management needs additional resources of amount $\sum_{i=1}^{N} A_i - C$.

3.1.3. *Case-3*

Suppose that the software project manager's only concern is to determine the optimum allocation of the testing resources to each module that maximizes the number of faults removed in the system for a fixed C. Then omitting the restrictions (10) and (12), the NLPP (15) reduces to

$$\begin{aligned} &\text{Maximize } Z = \sum_{i=1}^{N} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}}, \\ &\text{Subject to } \sum_{i=1}^{N} W_i \le C, \\ &W_i \ge 0; \quad i = 1, 2, \ldots, N. \end{aligned} \tag{18}$$

In the following section an algorithm using a dynamic programming technique is proposed for solving the NLPP (15) and subsequently for its special cases in (16)–(18).

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

4. Solution procedure using a dynamic programming technique

A dynamic programming technique is a powerful computational method for solving a sequential (or multi-stage) interrelated decision problem. Such a technique was developed by Richard Bellman and is based on a basic principle known as the principle of optimality. It provides a solution with much less effort than exhaustive enumeration. The computational savings are enormous for larger problems. Dynamic programming starts with a small portion (subproblem) of the original problem and finds the optimum solution of this smaller problem. It then gradually enlarges the problem, finding the current optimum solution from the preceding one, until the original problem is solved in its entirety. A major advantage of dynamic programming over the classical optimization techniques is that it determines the global optimum of the objective function (Bellman, 1957; Amini et al., 1990; Khan et al., 2003, 2008b; Hillier and Lieberman, 2010).

The NLPP (15) is a multi-stage decision problem in which the objective function and the constraints are separable functions of W_i; $i = 1, 2, \ldots, N$. A function $f(x_1, x_2, \ldots, x_N)$ is said to be separable if it can be expressed as $f(x_1, x_2, \ldots, x_N) = \sum_{j=1}^{N} f_j(x_j)$. Due to this separability and the nature of the problem, the dynamic programming technique may be used to solve the NLPP (see Hadley, 1970; Khan et al., 1997, 2003, 2008a, b).

A dynamic programming technique solves basically a functional recursive equation that links the different stages of the problem in a manner that guarantees that each stage's optimum feasible solution is also optimum and feasible for the entire problem. In actual practice, the functional recursion equation is derived by using the principle of optimality as follows:

Consider the kth subproblem of NLPP (15), involving the first k modules:

$$\begin{aligned}
&\text{Maximize } Z_k = \sum_{i=1}^{k} \frac{v_i a_i \left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}},\\
&\text{Subject to } \sum_{i=1}^{k} W_i \le C_k,\\
&W_i \ge A_i;\\
&W_i \ge B_i;\\
&A_i \ge 0 \text{ and } B_i \ge 0; \quad i = 1, 2, \ldots, k,
\end{aligned} \tag{19}$$

where $C_k < C$ denotes the amount of resources available for the first k modules. Note that $C_k = C$ for $k = N$.

Also note that $C_k = W_1 + W_2 + \ldots\ldots\ldots + W_k$.

Thus, the transformation functions are given by

$$C_{k-1} = W_1 + W_2 + \ldots\ldots + W_{k-1} = C_k - W_k; \quad i = 1, 2, \ldots, k. \tag{20}$$

Let $z_k(C_k)$ denote the maximum value of the objective function of the problem (19), then

$$z_k(C_k) = \max\left\{ Z_k \;\middle|\; \sum_{i=1}^{k} W_i \le C_k;\quad W_i \ge A_i;\quad W_i \ge B_i;\quad i = 1, 2, \ldots, k \right\}. \tag{21}$$

With the above definition of $z_k(C_k)$, the NLPP (19) is equivalent to finding $z_N(C)$ recursively by finding $z_k(C_k)$ recursively for $k = 1, 2, \ldots, N$ and for all feasible C_k, $0 \le C_k \le C$.

Further, (21) can be written as

$$z_k(C_k) = \max\left\{ \frac{v_k a_k\left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + \sum_{i=1}^{k-1} \frac{v_i a_i\left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}} \;\middle|\; \sum_{i=1}^{k-1} W_i \le C_k - W_k; \right.$$
$$\left. W_i \ge A_i;\ W_i \ge B_i;\ i = 1, \ldots, k \right\}.$$

For a fixed value of W_k, $\max[A_k, B_k] \le W_k \le C_k$, $z_k(C_k)$ is given by

$$z_k(C_k) = \frac{v_k a_k\left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + \max\left\{ \sum_{i=1}^{k-1} \frac{v_i a_i\left(1 - e^{-b_i W_i}\right)}{1 + \lambda_i e^{-b_i W_i}} \;\middle|\; \sum_{i=1}^{k-1} W_i \le C_k - W_k; \right.$$
$$\left. W_i \ge A_i;\ W_i \ge B_i;\ i = 1, \ldots, k-1 \right\}. \tag{22}$$

By the definition (21) the quantity inside {} in (22) is $z_{k-1}(C_{k-1})$, where $C_{k-1} = C_k - W_k$.

Thus, using Bellman's principle of optimality, we write a forwards recursive equation for kth stage of the dynamic programming technique for solving the NLPP (15) as

$$z_k(C_k) = \max_{\max[A_k, B_k] \le W_k \le C_k} \left(\frac{v_k a_k\left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1}(C_{k-1}) \right),\ k = 1, 2, \ldots, N. \tag{23}$$

We also define $z_k(C_k) = 0$ for $k = 0$ and $z_k(C_k) = 0$ if

$$W_k < \max[A_k, B_k] \text{ or } W_k > C_k; \quad k = 1, 2, \ldots, N. \tag{24}$$

This takes care of the restrictions $W_i \ge A_i$ and $W_i \ge B_i$; $i = 1, 2, \ldots, N$ of the NLPP (15).

At the final stage of the solution, i.e., at $k = N$, $z_N(C)$ is obtained by solving (23) recursively for all C_k. From $z_N(C)$ the optimum value W_N^* of W_N is obtained, from $z_{N-1}(C_{N-1})$ the optimum value W_{N-1}^* of W_{N-1} is obtained and so on until finally we obtain the optimum value W_1^* of W_1. The algorithm of the above solution procedure is summarized by a flowchart in Fig. 1.

The NLPP (16), which is a special case of NLPP (15), can also be solved in a similar way as discussed above and the recursive equation is obtained as

$$z_k(C_k) = \max_{B_k \le W_k \le C_k} \left(\frac{v_k a_k\left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1}(C_{k-1}) \right),\ k = 1, 2, \ldots, N. \tag{25}$$

Define $z_k(C_k) = 0$ for $k = 0$ and $z_k(C_k) = 0$ if

$$W_k < B_k \text{ or } W_k > C_k; \quad k = 1, 2, \ldots, N, \tag{26}$$

which takes care of the restrictions $W_i \ge B_i$; $i = 1, 2, \ldots, N$ of the NLPP (16).

Similarly, the recursive equation for solving the NLPP (17) discussed in Case-2 is

$$z_k(C_k) = \max_{A_k \le W_k \le C_k} \left(\frac{v_k a_k\left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1}(C_{k-1}) \right),\ k = 1, 2, \ldots, N. \tag{27}$$

Define $z_k(C_k) = 0$ for $k = 0$ and $z_k(C_k) = 0$ if

$$W_k < A_k \text{ or } W_k > C_k; \quad k = 1, 2, \ldots, N. \tag{28}$$

This takes care of the restrictions $W_i \ge A_i$; $i = 1, 2, \ldots, N$ of the NLPP (17).

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

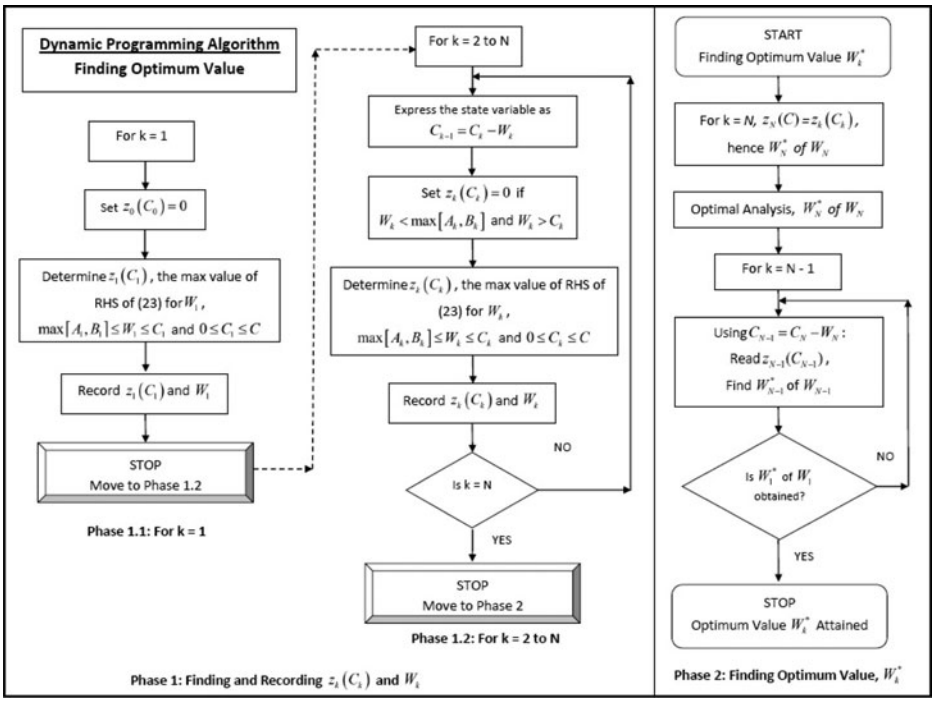

Figure 1. The summary of the dynamic programming algorithm for solving NLPP (15).

Finally, the recursive equation for solving the NLPP (18) discussed in Case-3 is

$$z_k(C_k) = \max_{0 \le W_k \le C_k} \left(\frac{v_k a_k \left(1 - e^{-b_k W_k}\right)}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1}(C_{k-1}) \right), \; k = 1, 2, \ldots, N. \tag{29}$$

Also

$$z_k(C_k) = 0 \text{ for } k = 0 \text{ and } z_k(C_k) = 0 \text{ if } W_k > C_k; \; k = 1, 2, \ldots, N. \tag{30}$$

5. Numerical examples and experimental results

In order to validate and to demonstrate the performance of the proposed strategy for module testing, experiments on three software failure data are performed. We discuss several examples using these data sets to show how the optimal allocation of testing resource to each module is determined. We also compare the accuracy of the proposed strategy of testing software modules with other existing methods/strategies proposed by the authors such as Huang and Lyu (2005), Kapur et al. (2004), and Jha et al. (2010).

Data Set 1. The first set of actual software failure data is taken from a large communication software project (Xie and Wohlin, 1995). The software system has only two modules, which are tested independently. Suppose that the total testing-resource expenditures $C = 10,000$ units are available to allocate between the two modules. We estimate the parameters a_i, b_i, and r_i $(i = 1, 2)$ of (7) for each module by using the MLE method discussed in Sec. 2 (also see, Ahmad et al., 2008b), which are summarized in Table 1.

Suppose that the allocation of testing resource can be made whereby the developer desires simultaneously to remove $p_1 = 0.8$ and $p_2 = 0.4$ proportion of faults and to achieve $R_0 =$

Table 1. Estimated value of SRGM parameters.

Module	a_i	b_i	r_i	λ_i
1	249.22	0.0005946	0.881172548	0.134851514
2	199.48	0.00058379	0.864198757	0.157141216

90% reliability, respectively, from the module 1 and 2, then the optimum allocation of testing resource W_i^* can be determined using NLPP (15).

Since the total amount of testing resource $C \geq \max\left(\sum_{i=1}^{2} A_i, \sum_{i=1}^{2} B_i\right) = \max(3{,}858.49, \quad 8{,}235.93)$, the required restrictions on p_i and R_0 can be met and the optimum allocations W_i^* to modules can be obtained by the solution procedure developed in Sec. 4 using a dynamic programming technique. With the proposed weights $v_1 = 0.6$ and $v_2 = 0.4$ obtained by (14) the recurrence relations (23) are solved using a computer program developed for the solution procedure and the optimum W_i^* to the modules are shown in Table 2. Using (7), the optimum number of faults m_i^* removed along with the percentages for each module is also presented in the table. The total number of faults that can be removed through this allocation is 420 (i.e., 93.6% of the total fault content is removed).

Data Set 2. We consider the second set of software failure data, which is widely used by many authors such as Ohtera and Yamada (1990), Huang and Lyu (2005), Huang and Lo (2006), Huang et al. (2002, 2004), Leung (1997), and Xie and Yang (2001), where the software system consisted of 10 modules. All the parameters a_i, b_i, and r_i $(i = 1, 2, \ldots, 10)$ for each module were estimated similarly by using the MLE method and are summarized in Table 3.

We assume that the management's target is to remove at least a proportion, p_i, of the number of initial faults from ith module with a certain reliability level R_0 as given in Table 4. To satisfy these restrictions, that is, when the restrictions (10) and (12) are imposed, the total amount of testing resource required is $C \geq \max\left(\sum_{i=1}^{10} A_i, \sum_{i=1}^{10} B_i\right) = \max(50{,}167.32, \quad 156{,}936.08)$. Suppose that the management wishes to provide a total resource $C = 165{,}000$, then the problem of optimum allocation for this resource is represented by the NLPP (15).

Table 2. Optimal allocation of testing resource with different proportions, a desired reliability and proposed weights.

Module	p_i	R_0	v_i	A_i	B_i	W_i^*	m_i^*	% of fault removed	% of fault remaining
1	0.8	0.9	0.6	2,879.06	4,065.14	5,829.21	240	96.3	3.7
2	0.4	0.9	0.4	979.44	4,170.79	4,170.79	180	90.2	9.8
Total				3,858.49	8,235.93	10,000	420	93.6	6.4

Table 3. Estimated value of SRGM parameters.

Module	a_i	b_i	r_i	λ_i
1	89	0.00041823	0.986988	0.013184
2	25	0.00050923	0.951585	0.050878
3	27	0.00039611	0.958372	0.043436
4	45	0.00022956	0.997947	0.002058
5	39	2.53E-04	0.96359	0.037785
6	39	1.72E-04	0.938163	0.065913
7	59	8.82E-05	0.975277	0.025349
8	68	7.27E-05	0.9948	0.005227
9	37	6.82E-05	0.978722	0.02174
10	14	1.53E-04	0.88933	0.124442

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Table 4. Proportion, reliability, and weights corresponding to each module.

Module	p_i	R_0	Arbitrary v_i	Equal v_i	Proposed v_i
1	0.8	0.9	0.1	1	0.2
2	0.3	0.9	0.6	1	0.1
3	0.3	0.9	0.7	1	0.1
4	0.5	0.9	0.4	1	0.1
5	0.4	0.9	0.1	1	0.1
6	0.4	0.9	0.2	1	0.1
7	0.6	0.9	0.5	1	0.1
8	0.7	0.9	0.6	1	0.1
9	0.4	0.9	0.1	1	0.1
10	0.2	0.9	0.5	1	0.0

To determine the optimum W_i^* we consider three kinds of weighting vectors v_i. They are: the arbitrary weights as suggested by Huang and Lyu (2005) and Huang and Lo (2006), the equal weights, and the weights as proposed in (14). These weighting vectors are listed in Table 4. Using the recurrence relations (23) we obtain the optimum W_i^* for each module as shown in Table 5. The optimum number of faults m_i^* removed with the percentages for each module is also presented in the table for the different weighting vectors.

Table 5 reveals that the maximum number of total faults is removed by the use of proposed and equal weighting vectors and the total number of faults removed is 409 (92.5%), whereas the arbitrary weighting procedure removes 405 (91.6%) of faults.

Data Set 3. The third data set is from the study by Kapur et al. (2004). We assume that parameters a_i, b_i, andr_i $(i = 1, 2, \ldots, 8)$ summarized in Table 6 have been estimated by MLE using the failure data for a software system consisting of eight modules (Kapur et al., 2004; Khan et al., 2008a). The total resource taken for the testing goes to 110,000 units.

Table 5. Optimal allocation of testing resource with different weights.

	Arbitrary v_i			Equal v_i			Proposed v_i		
Module	W_i^*	m_i^*	% removed	W_i^*	m_i^*	% removed	W_i^*	m_i^*	% removed
1	6,253.36	82	92.1	9,336.30	87	97.8	10,737.20	88	98.9
2	6,660.95	24	96.0	5,832.40	24	96.0	5,608.13	24	96.0
3	8,437.43	26	96.3	6,766.73	25	92.6	6,490.51	25	92.6
4	11,793.00	42	93.3	11,357.76	42	93.3	10,886.09	41	91.1
5	9,233.31	35	89.7	10,095.69	36	92.3	9,666.95	36	92.3
6	13,722.18	35	89.7	13,722.18	35	89.7	13,722.18	35	89.7
7	26,362.17	53	89.8	26,362.17	53	89.8	26,362.17	53	89.8
8	32,747.82	62	91.2	31,736.98	61	89.7	31,736.98	61	89.7
9	34,046.37	33	89.2	34,046.37	33	89.2	34,046.37	33	89.2
10	15,743.42	13	92.9	15,743.42	13	92.9	15,743.42	13	92.9
Total	165,000.00	405	91.6	165,000.00	409	92.5	165,000.00	409	92.5

Table 6. Estimated value of SRGM parameters.

Module	a_i	b_i	r_i	λ_i
1	45	0.000412932	0.85412	0.1707957
2	13	0.000319987	0.885236	0.1296423
3	16	0.000264216	0.889588	0.1241159
4	35	0.000150122	0.789985	0.2658468
5	14	8.95707E-05	0.795781	0.2566271
6	21	5.87323E-05	0.754922	0.3246402
7	20	3.20165E-05	0.58921	0.6971878
8	11	3.15115E-05	0.578634	0.7282082

Table 7. Weights corresponding to each module.

Module	Arbitrary v_i	Equal v_i	Proposed v_i
1	1.0	1	0.3
2	0.5	1	0.1
3	0.6	1	0.1
4	0.4	1	0.2
5	0.9	1	0.1
6	0.2	1	0.1
7	0.4	1	0.1
8	0.6	1	0.1

Let us assume that there is no restriction on the number of faults to be removed as well as on the reliability to be achieved, that is, the restrictions (10) and (12) are not imposed. Then the problem of finding optimum resource allocation is represented by the NLPP (18) as discussed in Case-3. With the three different weighting vectors as listed in Table 7, the recurrence relations (29) of NLPP (18) are solved and the optimum allocation of testing resource W_i^* to the modules is determined as shown in Table 8. The table also shows the results of the optimum number of faults m_i^* removed along with the percentages for each module.

Table 8 reveals that the maximum number of total faults is removed by the use of the proposed weighting vector and the total number of faults removed from the software through this allocation is 136; that is, 77.7% of the fault content is removed whereas, the arbitrary and equal weighting procedures remove 133 (76%) and 135 (77.1%) of faults, respectively.

In all three resource allocations in Table 8, it is noticed that the percentage of the remaining faults is higher than the fault removed for module 7 and none of the faults is removed in module 8 as no testing resource is allocated. Hence, the result may not satisfy the management. Suppose the management desires to set the restriction that at least some percentage of initial faults p_i $(i = 1, 2, \ldots, 8)$ given in Table 9 should be removed. With this additional restriction, the required problem of determining optimum resource allocation is NLPP (17) as given in Case-2.

To solve the problem (17), the A_i, the minimum amounts of resources required for the ith module to remove p_i proportion of initial faults, are calculated using (11) and are presented in Table 9. Since $\sum_{i=1}^{8} A_i = 99{,}659.48$ is less than $C = 110,000$, the NLPP does not have any infeasible solution. Therefore, solving the NLPP (17) with the proposed weighting vector through the recurrence relation (27), the optimum testing-resource allocation W_i^* to the modules is obtained. This result along with the expected number of faults m_i^* removed, the percentages of faults removed, and the faults remaining for each module are given in Table 9. The total number of faults that can be removed through this allocation is 124 (70.9%).

Table 8. Optimal allocation of testing resource with different weights.

	Arbitrary v_i			Equal v_i			Proposed v_i		
Module	W_i^*	m_i^*	% removed	W_i^*	m_i^*	% removed	W_i^*	m_i^*	% removed
1	12,495.04	45	100.9	10,255.11	44	97.8	12,777.62	45	100.0
2	9,130.09	12	92.3	8,405.59	12	92.3	8,216.75	12	92.3
3	11,800.58	15	93.8	10,225.36	15	93.8	9,996.50	15	93.8
4	20,208.77	33	94.3	20,178.14	33	94.3	24,482.71	34	97.1
5	26,603.27	12	85.7	16,787.11	10	71.4	16,041.29	10	71.4
6	11,047.07	9	42.9	25,753.15	15	71.4	24,576.66	15	71.4
7	18,715.18	7	35.0	18,395.53	6	30.0	13,908.46	5	25.0
8	0	0	0	0	0	0	0	0	0
Total	110,000	133	76.0	109,999.99	135	77.1	109,999.99	136	77.7

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Table 9. Optimal allocation of testing resource with different proportions and different weights.

Module	a_i	p_i	A_i	v_i	W_i^*	m_i^*	% removed	% remaining
1	45	0.8	4,207.74	0.3	8,794.65	44	97.8	2.2
2	13	0.4	1,754.39	0.1	2,837.32	7	53.8	46.2
3	16	0.5	2,851.29	0.1	3,503.39	9	56.3	43.7
4	35	0.7	9,156.80	0.2	13,175.38	29	82.9	17.1
5	14	0.5	9,086.36	0.1	9,086.36	7	50.0	50
6	21	0.6	18,631.25	0.1	18,631.25	13	61.9	38.1
7	20	0.6	39,534.57	0.1	39,534.57	12	60.0	40
8	11	0.25	14,437.08	0.1	14,437.08	3	27.3	72.7
Total	175		99,659.48		110,000	124	70.9	29.1

Further, if the management wants to allocate the testing resource that can remove faults from each module with a certain level of reliability R_0, then the optimum W_i^* is determined using NLPP (16) discussed in Case-1. To achieve R_0 level of reliability of removing faults we calculate B_i, the minimum amount of resources required at ith module, using (13). However, with the given resource $C = 110,000$, it has been seen that the maximum level of reliability that can be achieved is $R_0 = 0.54 = 54\%$ as the minimum total resource required is $\sum_{i=1}^{8} B_i = 109{,}474.13$, which is still less than C (see Table 10). This implies that NLPP (16) results in an infeasible solution, if R_0 is to be achieved at more than 54%. Solving the recurrence relation in (25) the optimum W_i^* to achieve a 54% level of reliability at each module is obtained, which is shown in Table 10.

Moreover, if the software manager desires to achieve higher reliability, say $R_0 = 0.9 = 90\%$, the minimum total resource required is $\sum_{i=1}^{8} B_i = 287{,}138.70$. That is, the extra amount of testing resource is anticipated to be 287,138.7 – 110,000 = 177,138.7 units. With the new total testing resource $C = 110,000 + 177,138.7 = 287,138.7$ the optimum allocation of testing-resource W_i^* to modules through NLPP (16) can be obtained as shown in Table 11.

Table 10. Optimal allocation of testing resource with reliability $R_0 = 54\ \%$.

Module	a_i	R_0	B_i	v_i	W_i^*	m_i^*	% removed	% remaining
1	45	0.54	2,094.17	0.3	2,620.05	28	62.2	37.8
2	13	0.54	2,638.21	0.1	2,638.21	7	53.8	46.2
3	16	0.54	3,184.52	0.1	3,184.52	9	56.3	43.7
4	35	0.54	6,066.22	0.2	6,066.22	19	54.3	45.7
5	14	0.54	10,118.37	0.1	10,118.37	8	57.1	42.9
6	21	0.54	15,971.74	0.1	15,971.74	11	52.4	47.6
7	20	0.54	34,234.20	0.1	34,234.20	11	55.0	45.0
8	11	0.54	35,166.69	0.1	35,166.69	6	54.5	45.5
Total	175		109,474.13		110,000	99	56.6	43.4

Table 11. Optimal allocation of testing resource with reliability $R_0 = 90\%$.

Module	a_i	R_0	B_i	v_i	W_i^*	m_i^*	% removed	% remaining
1	45	0.9	5,922.46	0.3	5,922.46	41	91.1	8.9
2	13	0.9	7,540.75	0.1	7,540.75	12	92.3	7.7
3	16	0.9	9,115.57	0.1	9,115.57	14	87.5	12.5
4	35	0.9	16,767.04	0.2	16,767.04	32	91.4	8.6
5	14	0.9	28,026.83	0.1	28,026.83	13	92.9	7.1
6	21	0.9	43,569.09	0.1	43,569.09	19	90.5	9.5
7	20	0.9	87,130.42	0.1	87,130.42	18	90.0	10.0
8	11	0.9	89,066.54	0.1	89,066.54	10	90.9	9.1
Total	175		287,138.70		287,138.70	159	90.9	9.1

Further allocation of testing resource can also be obtained for this data set using NLPP (15) whereby the developer desires simultaneously to remove a certain proportion of faults and to achieve a level of reliability. Then, in a similar way as discussed for Data Set 2, we can determine a feasible optimum allocation W_i^* to modules by the proposed solution procedure if the total amount of testing resource is $C \geq \max\left(\sum_{i=1}^{8} A_i,\ \sum_{i=1}^{8} B_i\right)$.

6. Comparison and discussion

In this section, an investigation is carried out to compare the method of optimum resource allocation proposed in this article to the methods of Kapur et al. (2004), Huang and Lyu (2005), and Jha et al. (2010).

6.1. *The comparison with Kapur et al. (2004)*

When $v_i = 1$ for all $i = 1, 2, \ldots, N$, the NLPP (17) and (18) discussed in Case-2 and Case-3, respectively, are equivalent to the formulations of Kapur et al. (2004). To solve the NLPPs for W_i with a closed explicit form, Kapur et al. treated the problems as the multiobjective fractional programming problems, which are further approximated to equivalent scalarization formulations. These approximate optimization problems are then solved using a dynamic programming approach. The closed form solutions are obtained by differentiating the recurrent relation for each stage and proceeding by mathematical induction. A brief review of Kapur et al. procedure is given below.

Using the method of Dur et al. (2001), Kapur et al. treated the NLPP (18) as the following multi-objective fractional programming problem:

$$\begin{aligned} &\text{Maximize } Z(W) = \left[l_1(W_1)/n_1(W_1),\ l_2(W_2)/n_2(W_2), \ldots, l_N(W_N)/n_N(W_N)\right]^T, \\ &\text{Subject to } \ W \in S = \left\{ W \in R^N \Big/ \sum_{i=1}^{N} W_i \leq C,\ W_i \geq 0,\ i = 1, \ldots N \right\} \end{aligned} \tag{31}$$

where $l_i = a_i\left(1 - e^{-b_i W_i}\right)$ and $n_i = 1 + \lambda_i e^{-b_i W_i}$.

For fixed weight of objective function the problem (31) is further approximated and expressed as Geoffrion (1968)'s equivalent scalarization formulation as

$$\begin{aligned} &\text{Maximize } Z = \sum_{i=1}^{N} \phi_i \left(l_i(W_i) - n_i(W_i)\right), \\ &\text{Subject to } \ \sum_{i=1}^{N} W_i \leq C, \\ &W_i \geq 0; \quad i = 1, 2, \ldots, N \\ &\phi \in \Omega = \left\{ \phi \in R^N \Big/ \sum \phi_i = 1,\ \phi_i \geq 0,\ i = 1, \ldots, N \right\}. \end{aligned} \tag{32}$$

The problem (32) is then solved by using a dynamic programming approach for which the recurrence relations are obtained as

$$f_k(C) = \max_{0 \leq W_k \leq C} \left\{ (a_k - 1) - (a_k + \lambda_k) e^{-b_k W_k} + f_{k-1}(C - W_k) \right\}; \quad i = 1, 2, \ldots, N. \tag{33}$$

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Table 12. Optimum number of faults removed and relative gain due to use of proposed method over Kapur et al.

	NLPP (18)			NLPP (17)		
Module	m^*_{Kapur}	m^*_{Proposed}	RG in %	m^*_{Kapur}	m^*_{Proposed}	RG in %
1	40	44	10.0	33	42	27.3
2	10	12	20.0	8	8	0.0
3	13	15	15.4	9	10	11.1
4	28	33	17.9	23	25	8.7
5	9	10	11.1	7	7	0.0
6	13	15	15.4	11	11	0.0
7	9	6	−33.3	10	10	0.0
8	4	0	−100.0	6	6	0.0
Total	126	135	7.1	107	119	11.2

Differentiating (33) for each stage and proceeding by induction, the closed form solutions are obtained by

$$W_i^* = \frac{1}{b_i + \mu_{i-1}} \left[\mu_{i-1} C - \ln \left(\frac{\mu_{i-1} V_{i-1}}{(a_i + \lambda_i)\, b_i} \right) \right], \quad i = 1, 2, \ldots, N, \tag{34}$$

where $\mu_i = \frac{1}{\sum_{j=1}^{i} 1/b_j}$ and $\mu_i V_i = \prod_{j=1}^{i} [(a_j + \lambda_j) b_j]^{(a_j/b_j)}$, $i = 1, 2, \ldots, N$.

It can be observed that the solution obtained by (34) is obviously an approximate one as the problem (18) is approximated to (32). It has also been seen that Kapur et al. used C, the whole amount of resources, as the state variable at kth stage to obtain the recurrence relation (33), which is constant for all stages. In practice, a state variable for kth is defined by C_k $(< C)$, which is the total amount of resources available at kth stage and is not constant at each stage. The state transformation function from $(k-1)$th stage to kth stage is given by (20). Note that $C_k = C$ for $k = N$.

Kapur et al. also solved the NLPP (17) in a similar way as discussed above and obtained the optimum resource allocation W_i^* and the optimum number of faults removed m_i^* for the numerical example presented in Data Set 3 in Sec. 5. The results are shown in Table 12.

Table 12 shows the optimum number of faults m^*_{Proposed} and m^*_{Kapur} removed by the proposed and Kapur et al. formulations, respectively, for $v_i = 1$ and $p_i = p_0 = 0.5$ $(i = 1, 2, \ldots, 8)$. From this table it can be observed that the proposed method is more efficient as it removes more faults than Kapur et al. in all the modules for both NLPPs (17) and (18), except the modules 7 and 8 in NLPP (18). The table also shows the relative gain (RG) in percentage of faults removed due to the proposed method over Kapur et al. for all the modules, where RG is defined as $(m^*_{\text{Proposed}} - m^*_{\text{Kapur}})/m^*_{\text{Kapur}}$. The overall RG in percentage for the proposed method over Kapur et al. are 7.1% and 11.2%, respectively, for NLPP (18) and (17).

Thus, the results show that the proposed method is more useful in allocating the testing resources to the modules than the method of Kapur et al. Moreover, the Kapur et al. formulation is restricted only to solve the NLPPs (17) and (18) with equal weight $v_i = 1$ to each module. This implies that the Kapur et al. method may not be useful when a production manager wishes to achieve a desired level of reliability as stated in NLPP (16) or to achieve the removal of a certain proportion of faults from each module, as well as the desired level of reliability to be achieved as stated in NLPP (15). Also, the Kapur et al. method does not address the problem when a manager wants to use different weights v_i to the modules when the system is heterogeneous with respect to modules' initial faults a_i.

Table 13. Optimum allocation of resources using Huang and Lyu, and proposed.

Module	Example 1			Example 2			Example 3		
	v_i	$W^*_{i,\text{H\&L}}$	$W^*_{i,\text{Prop}}$	v_i	$W^*_{i,\text{H\&L}}$	$W^*_{i,\text{Prop}}$	v_i	$W^*_{i,\text{H\&L}}$	$W^*_{i,\text{Prop}}$
1	1.0	6,254	6,256	1.0	8,105	8,109	0.5	6,015	6,017
2	1.5	3,826	3,826	0.6	3,547	3,548	0.5	2,833	2,833
3	1.3	4,117	4,117	0.7	4,409	4,510	0.7	4,052	4,052
4	0.5	2,791	2,793	0.4	5,191	5,190	0.4	4,402	4,402
5	2.0	7,825	7,830	1.0	8,145	8,151	1.5	9,030	9,037
6	0.3	0	0	0.2	403	389	0.2	0	0
7	1.7	13,366	13,362	0.5	8,267	8,269	0.6	8,280	8,280
8	1.3	11,820	11,816	0.6	11,833	11,834	0.6	9,343	9,339
9	1.0	0	0	0.1	0	0	0.9	6,046	6,040
10	1.0	0	0	0.5	0	0	0.5	0	0
Total	11.6	49,999	50,000	5.6	49,900	50,000	6.4	50,001	50,000

6.2. *The comparison with Huang and Lyu (2005)*

Huang and Lyu (2005) also considered the problem of determining the optimum resource allocation, which minimizes the number of remaining faults given a fixed amount of testing-effort and a reliability goal to be achieved with arbitrary weights v_i. Since the minimization of the number of remaining faults is the same as the maximization of the number of faults removed, Huang and Lyu's formulation of the problem of resource allocation based on the exponential type SRGM is an equivalent to the proposed formulation stated in NLPP (16) for $r_i = 1; \quad (i = 1, 2, \ldots, 10)$.

Although Huang and Lyu considered the constraint (12) to achieve the reliability goal, it was eliminated while solving the problem by substituting $X_i = W_i - B_i$. Therefore, the formulation is basically reduced to NLPP (18) without addressing the problem of achieving the reliability goal and it is solved by using Lagrange multiplier technique. They presented three examples using the Data Set 2 with total amount of testing resource $C = 50,000$ man-hours and three arbitrary weighting vectors (v_i) given in Table 13.

To compare the proposed allocation method using a dynamic program with that of Huang and Lyu (2005), we obtained optimum, W_i^* for all three arbitrary weighting vectors as used by Huang and Lyu. Table 13 shows the optimum resource allocations $W^*_{i,\text{H\&L}} = X_i^*$ and $W^*_{i,\text{Prop}}$ of Huang and Lyu, and proposed formulations, respectively.

Table 14 shows the total number of faults $\sum_{i=1}^{10} m^*_{i,\text{Prop}}$ and $\sum_{i=1}^{10} m^*_{i,\text{H\&L}}$ removed by the proposed and Huang and Lyu formulations, respectively. The table also shows the RG in percentage of total faults removed due to the use of proposed method over Huang and Lyu for each example. It reveals that the overall RG of the proposed method is more than that of Huang and Lyu for examples 1 and 2. Although, the RG in example 3 is slightly higher for Huang and Lyu, the solution is not useful as it violates the resource constraint $\sum_{i=1}^{N} W_i \leq C$ as the required total resource is 5,001. Thus the proposed method is more efficient than the Huang and Lyu method.

Table 14. Total faults removed and relative gain due to use of proposed method over Huang and Lyu.

Example	$\sum_{i=1}^{10} m^*_{i,\text{Prop}}$	$\sum_{i=1}^{10} m^*_{i,\text{H\&L}}$	RG in %
1	260.61	260.60	0.0038
2	267.06	266.92	0.0525
3	262.72	262.73	−0.0038

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Table 15. Optimum allocation of resources W_i^* and number of faults m_i^* removed using Jha et al. and proposed.

	Jha et al.			Proposed			
Module	W_i^*	m_i^*	% of fault removed	W_i^*	m_i^*	% of fault removed	RG in %
1	5,093.49	89.00	96.19	7,812.47	85.61	100.00	−3.81
2	3,267.52	23.30	89.92	4,502.36	22.48	93.20	−3.52
3	4,009.02	16.81	86.96	5,146.18	23.48	62.26	39.68
4	6,862.71	34.50	86.09	8,589.70	38.74	76.67	12.29
5	6,137.14	38.83	85.26	7,566.53	33.25	99.56	−14.37
6	8,052.94	36.74	79.33	9,170.05	30.94	94.21	−15.79
7	14,475.53	31.94	72.68	14,714.08	42.88	54.14	34.25
8	17,182.80	14.99	70.99	17,021.12	48.27	22.04	222.01
9	14,106.22	3.02	42.86	8,204.99	15.86	8.16	425.17
10	5,812.63	8.28	29.36	2,272.53	4.11	59.14	−50.36
Total	85,000.00	297.41	67.29	85,000.00	345.62	78.19	16.21

As discussed earlier in comparison to Kapur et al.'s method, the Huang and Lyu method addressed the solution of the NLPPs (16) and (18), that is, it will not be possible to use this method, when a manager wishes to achieve both a certain proportion of faults to be removed from each module and the desired level of reliability to be achieved at stated in NLPP (15). Moreover, as the Huang and Lyu suggested the use of arbitrary weight v_i, there is no guarantee that the method gives the optimum allocation of resources in the absence of choosing the appropriate arbitrary weights.

6.3. *The comparison with Jha et al. (2010)*

Jha et al. (2010) proposed a dynamic resource allocation method by extended the method of Kapur et al. (2004) as discussed in previous section. Their first problem was to determine the optimum resource allocation among the modules that maximize the mean number of detected faults, which is discussed in NLPP (18). When the total time available for testing is $C = 85{,}000$ s, they used the technique for the Data Set 2. The optimum resource allocation W_i^* obtained for equal weights ($v_i = 1$) and the optimum number of faults m_i^* removed are shown in Table 15. The results for the proposed method are also presented in the table.

Table 15 shows that the total number of faults removed by the proposed and Jha et al. methods are 297.41 (67.29%) and 345.62 (78.19%), respectively. The results also reveal that the proposed method is more consistent in removing the faults and the overall RG in using the method over the Jha et al. is 16.21%.

7. Conclusion

In this article we have discussed the optimal testing-resource allocation problem for modular software systems. A general formulation as well as three special cases and the solution using a dynamic programming technique to the problem based on testing-effort dependent NHPP models are provided. The proposed strategies maximized the total number of faults removed with a fixed amount of testing-effort, a certain percentage of faults to be removed and a reliability constraint to be achieved. Numerical examples using three sets of software failure data are described and discussed with three kinds of different weights v_i assigned to ith module. Finally, we compared the experimental results with the results of Kapur et al., Huang and Lyu, and Jha et al. methods. The experimental results show that:

i. For a fixed amount of testing resources and certain reliability, the total number of faults removed can be maximized by the use of proposed and equal weighting vectors (see Table 5).
ii. For a fixed amount of testing resources, the total number of faults removed can be maximized by the use of proposed weighting vector (see Table 8).
iii. For $v_i = 1$ and $p_i = p_0 = 0.5$; $(i = 1, 2, \ldots, 8)$, the overall gain in percentage of the proposed method is more than that of Kapur et al. (see Table 12).
iv. For all cases discussed, the RGs of the proposed strategies are more than when compared to Huang and Lyu (see Table 14).
v. For $v_i = 1$, the overall gain in percentage of the proposed method is more than that of Jha et al. (see Table 15).

Therefore, on the basis of the comparisons made in Sec. 6, we conclude that by using the dynamic programming technique a better solution to the proposed strategies can be achieved as compared to the other methods discussed.

Acknowledgments

The authors are grateful to the Editor in Chief and the learned reviewer for their valuable comments and suggestions to improve the manuscript.

Appendix

Maximum likelihood method

Once the estimates of α, β, δ, and θ are known, the parameters of the SRGMs can be estimated through the MLE method. The likelihood function for the unknown parameters a, b, and r in the NHPP model as given as in (6) is

$$\begin{aligned} L'(a, b, r) &\equiv P\left\{N(t_i) = y_i, ++ i = 1, 2, \ldots, n\right\} \\ &= \prod_{k=1}^{n} \frac{[m(t_k) - m(t_{k-1})]^{(y_k - y_{k-1})}}{(y_k - y_{k-1})!} \cdot e^{-[m(t_k) - m(t_{k-1})]}, \end{aligned} \tag{A1}$$

where $t_0 \equiv 0$ and $y_0 \equiv 0$.

Taking logarithm on both sides in (A1), we have

$$\begin{aligned} L = \ln L' &= \sum_{k=1}^{n} (y_k - y_{k-1}) \ln [m(t_k) - m(t_{k-1})] \\ &- \sum_{k=1}^{n} [m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n} \ln \left[(y_k - y_{k-1})!\right]. \end{aligned}$$

Now

$$m(t_k) - m(t_{k-1}) = \frac{a(1+\lambda)\left(e^{-bW(t_{k-1})} - e^{-bW(t_k)}\right)}{\left(1 + \lambda e^{-bW(t_k)}\right)\left(1 + \lambda e^{-bW(t_{k-1})}\right)}$$

and we have

$$\sum_{k=1}^{n} [m(t_k) - m(t_{k-1})] = m(t_n) = \frac{a\left[1 - e^{-bW(t_n)}\right]}{1 + \lambda e^{-bW(t_n)}}.$$

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Thus,

$$\begin{aligned} L = & \sum_{k=1}^{n} \left(y_k - y_{k-1}\right) \ln a + \sum_{k=1}^{n} \left(y_k - y_{k-1}\right) \ln (1+\lambda) \\ & + \sum_{k=1}^{n} \left(y_k - y_{k-1}\right) \ln \left(e^{-bW\left(t_{k-1}\right)} - e^{-bW\left(t_k\right)}\right) \\ & - \sum_{k=1}^{n} \left(y_k - y_{k-1}\right) \ln \left(1 + \left[(1-r)/r\right] e^{-bW\left(t_k\right)}\right) \\ & - \sum_{k=1}^{n} \left(y_k - y_{k-1}\right) \ln \left(1 + \left[(1-r)/r\right] e^{-bW\left(t_{k-1}\right)}\right) \\ & - \frac{a\left[1 - e^{-bW(t_n)}\right]}{1 + \left[(1-r)/r\right] e^{-bW(t_n)}} - \sum_{k=1}^{n} \ln \left[\left(y_k - y_{k-1}\right)!\right]. \end{aligned} \tag{A2}$$

The maximum likelihood estimates of SRGM parameters a, r and $b$can be obtained by solving the following equations:

$$\frac{\partial L}{\partial a} = \frac{\sum_{k=1}^{n} \left(y_k - y_{k-1}\right)}{a} - \frac{a\left[1 - e^{-bW(t_n)}\right]}{1 + \left[(1-r)/r\right] e^{-bW(t_n)}} = 0,$$

$$\begin{aligned} \frac{\partial l}{\partial r} = & \frac{\sum_{k=1}^{n} \left(y_k - y_{k-1}\right)}{1 + \left[(1-r)/r\right]} \times \left(-\frac{1}{r^2}\right) - \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)}{1 + \left[(1-r)/r\right] e^{-bW(t_k)}} \times \left(-\frac{1}{r^2} e^{-bW(t_k)}\right) \\ & - \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)}{1 + \left[(1-r)/r\right] e^{-bW\left(t_{k-1}\right)}} \times \left(-\frac{1}{r^2} e^{-bW\left(t_{k-1}\right)}\right) \\ & - \frac{a\left[1 - e^{-bW(t_n)}\right]}{\left[1 + \left[(1-r)/r\right] e^{-bW(t_n)}\right]^2} \times \left(-\frac{1}{r^2} e^{-bW(t_n)}\right), \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial b} = & \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)}{\left[e^{-bW\left(t_{k-1}\right)} - e^{-bW\left(t_k\right)}\right]} \times \left(-W\left(t_{k-1}\right) e^{-bW\left(t_{k-1}\right)} + W\left(t_k\right) e^{-bW\left(t_k\right)}\right) \\ & - \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)}{1 + \lambda e^{-bW\left(t_k\right)}} \times \left(-W\left(t_k\right) \lambda e^{-bW\left(t_k\right)}\right) \\ & - \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)}{1 + \lambda e^{-bW\left(t_{k-1}\right)}} \times \left(-W\left(t_{k-1}\right) \lambda\right) e^{-bW\left(t_{k-1}\right)} - \frac{aW\left(t_n\right) e^{-bW(t_n)} \left[1+\lambda\right]}{\left[1 + \lambda e^{-bW(t_n)}\right]^2} = 0. \end{aligned}$$

After some algebraic simplification, the following is obtained:

$$\hat{a} = \frac{y_n \left[1 + \lambda \phi_n\right]}{1 - \phi_n}, \tag{A3}$$

$$\frac{y_n}{r^2 \left[1+\lambda\right]} + \frac{a\phi_n \left[1 - \phi_n\right]}{\left[r^2 \left[1 + \lambda\phi_n\right]\right]^2} = \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right) \phi_k}{r^2 \left[1 + \lambda\phi_k\right]} + \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right) \phi_{k-1}}{r^2 \left[1 + \lambda\phi_{k-1}\right]}, \tag{A4}$$

$$\sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)\left[W\left(t_k\right)\phi_k - W\left(t_{-1}\right)\phi_{k-1}\right]}{\phi_{k-1} - \phi_k} + \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)\left[\lambda W\left(t_k\right)\phi_k\right]}{1 + \lambda\phi_k} \tag{A5}$$

$$+ \sum_{k=1}^{n} \frac{\left(y_k - y_{k-1}\right)\left[\lambda W\left(t_{k-1}\right)\phi_{k-1}\right]}{1 + \lambda\phi_{k-1}} = \frac{aW\left(t_n\right)\phi_n\left[1 + \lambda\right]}{\left[1 + \lambda\phi_{k-1}\right]^2},$$

where $\phi_k = e^{-bW(t_k)}$, $k = 1, 2, \ldots n$, and $\lambda = \frac{1-r}{r}$.

The above can be solved by numerical methods to get the values of $\hat{a}$, $\hat{b}$, and $\hat{r}$.

References

Ahmad, N., Bokhari, M. U., Quadri, S. M. K., Khan, M. G. M. (2008a). The exponentiated Weibull software reliability growth model with various testing-efforts and optimal release policies: A performance analysis. *Int. J. Qual. Reliab. Manage*. 25(2):211–235.

Ahmad, N., Khan, M. G. M., Quaddri, S. M. K., Kumar, M. (2009). Modelling and analysis of software reliability with Burr type X testing-effort and release-time determination. *J. Modell. Manage*. 4(1):28–54.

Ahmad, N., Khan, M. G. M., Rafi, L. S. (2008b, December 29–31). Inflection S-shaped software reliability growth models with testing-effort functions. In: *Proceedings of the VI International Symposium on Optimization and Statistics*, Aligarh Muslim University, Aligarh, India: Department of Statistics & O.R., (pp. 151–172).

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

Ahmad, N., Khan, M. G. M., Rafi, L. S. (2011). Analysis of an inflection S-shaped software reliability model considering log-logistic testing-effort and imperfect debugging. *Int. J. Comput. Sci. Network Secur*. 11(1):161–171.

Amini, A. A., Weymouth, T. E., Jain, R. C. (1990). Using dynamic programming for solving variational problems in vision. *IEEE Trans. Pattern Anal. Mach. Intell.* 12(9):855–867.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bokhari, M. U., Ahmad, N. (2007). Software reliability growth modeling for exponentiated Weibull functions with actual software failures data. *Adv. Comput. Sci. Eng. Rep. Monogr*. 2:390–396.

Chaudhary, K., Manik, P., Bali, S. (2012). Dynamic testing resource allocation of modular software system for SRGM incorporating testing efficiency using differential evolution. *Adv. Intell. Soft Comput.* 130:1011–1023.

Dur, M., Horst, R., Thoai, N. V. (2001). Solving sum-of-ratios fractional programs using efficient points. *Optimization* 41:447–466.

Geoffrion, A. M. (1968). Proper efficient and theory of vector maximization. *J. Math. Anal. Appl.* 22:613–630.

Hadley, G. (1970). *Nonlinear and Dynamic Programming*. Sydney: Addison Wesley.

Hillier, F. S., Lieberman, G. J. (2010). *Introduction to Operations Research*. New York: McGraw-Hill.

Huang, C. Y., Lo, J. H. (2006). Optimal resource allocation for cost and reliability of modular software systems in the testing phase. *J. Syst. Software* 79(5):653–664.

Huang, C. Y., Lo, J. H., Kuo, S. Y., Lyu, M. R. (2002). Optimal allocation of testing resources for modular software systems. In: *Proceedings of the Thirteenth IEEE International Symposium on Software Reliability Engineering* Annapolis, Maryland, 12–15 Nov, Los Alamitos, CA: IEEE Computer Society Press (pp. 129–138).

Huang, C. Y., Lo, J. H., Kuo, S. Y., Lyu, M. R. (2004). Optimal allocation of testing resource considering cost, reliability, and testing-effort. In: *Tenth IEEE Pacific Rim International Symposium on Dependable Computing* Papeete, Tahiti, French Polynesia, 3–5, March 2004, Los Alamitos, CA: IEEE Computer Society Press (pp. 103–112).

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016

Huang, C. Y., Lyu, M. R. (2005). Optimal testing resource allocation, and sensitivity analysis in software development. *IEEE Trans. Reliab.* 54(4):592–603.

Jha, P. C., Arora, R., Kapur, P. K. (2010, February 25–26). Optimal testing time allocation of modular software for exponential SRGM incorporating imperfect debugging. In: *Proceedings of the 4th National Conference on Computing for Nation Development, INDIACom-2010*, New Delhi, India: BVICAM.

Kapur, P. K., Aggarwal, A. G., Kapoor, K., Kaur, G. (2009). Optimal testing resource allocation, and sensitivity analysis in software development. *Int. J. Reliab. Qual. Saf. Eng.* 16(6):495.

Kapur, P. K., Aggarwal, A. G., Kaur, G. (2010). Simultaneous allocation of testing time and resources for a modular software. *Int. J. Syst. Assur. Eng. Manage.* 1(4):351–361.

Kapur, P. K., Garg, R. B., Kumar, S. (1999). *Contributions to Hardware and Software Reliability*. Singapore: World Scientific.

Kapur, P. K., Jha, P. C., Bardhan, A. K. (2004). Optimal allocation of testing resource for a modular software. *Asia-Pac. J. Oper. Res.* 21(3):333–354.

Khan, M. G. M., Ahmad, N., Rafi, L. S. (2008a). Optimal testing resource allocation for modular software based on a software reliability growth model: A dynamic programming approach. *IEEE Proc. Int. Conf. Comput. Sci. Software Eng.* 2:759–762.

Khan, M. G. M., Ahsan, M. J., Jahan, N. (1997). Compromise allocation in multivariate stratified sampling: An integer solution. *Nav. Res. Logist.* 44:69–79.

Khan, M. G. M., Khan, E. A., Ahsan, M. J. (2003). An optimal multivariate stratified sampling design using dynamic programming. *Aust. N. Z. J. Stat.* 45(1):107–113.

Khan, M. G. M., Nand, N., Ahmad, N. (2008b). Determining the optimum strata boundary points using dynamic programming. *Surv. Method.* 34(2):205–214.

Kuo, S. Y., Huang, C. Y., Lyu, M. R. (2001). Framework for modeling software reliability, using various testing-efforts and fault-detection rates. *IEEE Trans. Reliab.* 501(3):310–320.

Leung, Y. W. (1997). Dynamic resource allocation for software module testing. *J. Syst. Software* 37: 129–139.

Li, X., Xie, M., Ng, S. H. (2009). A general formulation of optimal testing-time allocation for modular systems. In: *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM'09)* Hong Kong, 8–11, December, Los Alamitos, CA: IEEE Computer Society Press (pp. 252–256).

Lo, J. H., Kuo, S. Y., Lyu, M. R., Huang, C. Y. (2002). Optimal resource allocation and reliability analysis for component-based software applications. In: *Proceedings of the 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC'2002)* Oxford, England, August, Los Alamitos, CA: IEEE Computer Society Press (pp. 7–12).

Lyu, M. R. (1996). *Handbook of Software Reliability Engineering*. New York: McGraw-Hill.

Lyu, M. R., Rangarajan, S., Van Moorsel, A. P. A. (2002). Optimal allocation of test resources for software reliability growth modeling in software development. *IEEE Trans. Reliab.* 51(2):183–192.

Musa, J. D., Iannino, A., Okumoto, K. (1987). *Software Reliability: Measurement, Prediction and Application*. New York: McGraw-Hill.

Myers, G. J. (1976). *Software Reliability: Principles and Practices*. New York: John Wiley & Sons.

Ohba, M. (1984). Software reliability analysis model. *IBM J. Res. Dev.* 28(4):428–443.

Ohtera, H., Yamada, S. (1990). Optimal allocation and control problems for software-testing resources. *IEEE Trans. Reliab.* 39(2):171–176.

Quadri, S. M. K., Ahmad, N., Peer, M. A., Kumar, M. (2006). Non-homogeneous Poisson process software reliability growth model with generalized exponential testing-effort functions. *RAU J. Res.* 16(1–2):159–163.

Rajan, R., Misra, R.B. (2006). Optimal testing resource allocation models for modular software. In: *Proceedings of the Annual Symposium of Reliability and Maintainability (RAMS'06)* Newport Beach, CA, USA, 14–16, June, Los Alamitos, CA: IEEE Computer Society Press (pp. 104–109).

Wang, Z., Tang, K., Yao, X. (2008). A multi-objective approach to testing resource allocation in modular software systems. In: *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC2008)*, 1–6, June, Los Alamitos, CA: IEEE Computer Society Press, Hongkong, China (pp. 1148–1153).

Wang, Z., Tang, K., Yao, X. (2010). Multi-objective approaches to optimal testing resource allocation in modular software systems. *IEEE Trans. Reliab.* 59(3):563–575.

Xie, M. (1991). *Software Reliability Modeling*. Singapore: World Scientific.

Xie, M., Wohlin, C. (1995). An additive reliability model for the analysis of modular software failure data. In: *Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering* Toulouse, France, 24–27, October, Los Alamitos, CA: IEEE Computer Society Press (pp. 188–193).

Xie, M., Yang, B. (2001). Optimal testing-time allocation for modular systems. *Int. J. Qual. Reliab. Manage.* 18(8):854–863.

Yamada, S., Hishitani, J., Osaki, S. (1993). Software reliability growth with a Weibull test-effort: A model & application. *IEEE Trans. Reliab.* 42(1):100–105.

Yamada, S., Ichimori, T., Nishiwaki, M. (1995). Optimal allocation policies for testing-resource based on a software reliability growth model. *Int. J. Math. Comput. Model.* 22(10–12):295–301.

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

Yamada, S., Osaki, S. (1985). Software reliability growth modeling: Models and applications. *IEEE Trans. Software Eng.* 11(12):1431–1437.

Downloaded by [University of the South Pacific], [Dr Mohammad Khan] at 22:22 09 March 2016