FLEXIBLE SOFTWARE RELIABILITY GROWTH MODELS

P.K. Kapur[1*], A. Gupta[1], V.S.S. Yadavalli[2] and S.J. Claasen[2]

[1]Department of Operational Research
University of Delhi, India
pkkapur1@bol.net.in

[2]Department of Industrial and Systems Engineering
University of Pretoria, South Africa
sarma.yadavalli@up.ac.za, schalk.claasen@up.ac.za

ABSTRACT

Numerous Software Reliability Growth Models (SRGMs) have been discussed in the literature. These models are used to predict fault content and reliability of software. It has been observed that the relationship between testing time and the corresponding number of faults removed is either exponential or S-shaped, or a mix of the two. Another important class of SRGMs, known as flexible SRGMs, can depict both exponential and S-shaped growth curves. The paper introduces a new concept of power logistic learning function that proves to be very flexible, in the sense that it represents various curve types – exponential, Rayleigh, Weibull or simple logistic. The flexible nature of the power logistic function gives the flexible SRGM a higher degree of accuracy and wider applicability.

OPSOMMING

Verskeie voorbeelde van Betroubaarheidsgroeimodelle vir programmatuur word in die literatuur beskryf. Die modelle word gebruik vir die voorspelling van foutinhoud en programmatuurbetroubaarheid. Daar word waargeneem dat die verband tussen toetstyd en die resulterende foutverwydering eksponensiaal of S-vormig of ‘n kombinasie daarvan is. Aanpasbare modelle insluitende diskrete ekwivalente word ook behandel. Die publikasie ontleed vervolgens algemene plooibare maglogistieke leerkromme met wye toepasbaarheid wat slaan op eksponensiële, Rayleigh-, Weilbull- en logistieke funksies. Die plooibaarheid van die model waarborg akkuraatheid en wye toepasbaarheid met die verlangde gehalte van voorspelbaarheid.

[*]This work was carried out during the visit of Prof Kapur to the Department of Industrial & Systems Engineering, University of Pretoria.

1. INTRODUCTION

Due to the incredible growth in information technology and science, computers and computer-based systems have invaded every sphere of human activity – for example, nuclear research, telecommunications, space research programmes, aviation, transportation, banking, education, and health-care. With the world of information technology changing at a staggering rate, not to mention the proliferation of the Internet, more and more systems are being automated, resulting in an increase in people's dependence on computers.

Although this technology revolution has made our lives better, concern for safety and security has never been greater. There are numerous instances where failures of computer-controlled systems have led to colossal loss of human lives and money. With the increased complexity of product design, short development cycles, higher customer demands for quality, and the highly destructive consequences of software failures, a major responsibility lies with those who are active in the areas of software debugging, testing and verification. Moreover, competition in the software market is intense, and as a result of the applications involved, purchasers look for quality and reliability in software. The nature and complexity of software requirements has drastically changed over the last few decades, and users worldwide have become much more demanding in terms of cost and quality.

Computer-based systems typically consist of hardware and software. Quality hardware can now be produced at a reasonable cost, but the same cannot be said of software. Software development consists of a sequence of activities where perfection is yet to be achieved. Thus there is every possibility that faults are introduced and remain in software. These faults can lead to failures that have catastrophic results. So great emphasis is placed on avoiding the introduction of faults during software development and removing latent faults before the product is released for use. The only way to verify and validate the software is by testing. Software testing involves running the software and checking for unexpected behavior in the software output. During the testing phase, test cases that simulate the user environment are run on the software. Any departure from specifications or requirements is called a failure, and immediately an effort is made to remove the cause of that failure (the fault). A successful test can be considered to be one that reveals the presence of latent faults. Therefore, software should be thoroughly tested to expose as many software faults as possible. The testing phase is an extremely important feature of the software development life cycle (SDLC), in which about 50% of the developmental resources are consumed. Therefore, testing software by executing all its statements and paths (even just once) is not practically possible for a large-scale software system. A compromise testing approach is used, which involves dividing the software into blocks and executing each block at least once. Potentially troublesome combinations of blocks are executed many times. However, testing techniques can be categorized into two main groups:

1. Top-down testing involves starting the test at the sub-system level. Then, the modules that comprise the sub-system are tested. The procedure is recursively

repeated until the test reaches the lowest level software component (function, object).

2. Bottom-up testing involves reversing the previous process.

Each test technique has its merits and demerits. Choosing the testing approach mainly depends on the software development technique. Software testing is a destructive process, since it aims at forcing the software to behave abnormally under some conditions. For this reason, software programmers subconsciously avoid bringing their product into this stage. Therefore, it is preferable that an independent team tests the software. This test method is called independent validation and verification (IVV), which means that the testing team is functionally independent of the development team, and tests the software from the user's point of view. This method is also called the black box method. The process of locating faults and designing the procedures to remove them is called the debugging process. The process of fault removal (repair) involves rewriting the code if the fault is due to coding and design error, or changing the requirements (which requires major repair). The chronology of failure occurrence and fault removals can be utilized to provide an estimate of the software reliability and the level of fault content. In the light of this, there is a need to develop a tool that can utilize this information to help software engineers and managers to monitor the process of testing. The software reliability model (SRM) is a tool that can be used to evaluate software quantitatively, develop test status and schedule status, and monitor the changes in reliability performance.

Numerous Software Reliability Growth Models (SRGMs), which report the number of failures (faults identified/removed), have been discussed in the literature [7, 10, 13]. These models are used to predict the fault content and reliability of the software. It has been observed that the relationship between the testing time and the corresponding number of faults removed is either exponential [4] or S-shaped [17], or a mix of the two. Another important class of SRGMs, known as flexible SRGMs, exists: depending upon parameter values, these depict both exponential and S-shaped growth curves [7]. In these models the role of the learning process during the testing phase is taken into account, which comes from the experience gained in software testing.

In this paper, section 2 discusses three existing flexible SRGMs developed by Obha [12], Bittanti *et al* [2], and Kapur and Garg [6]; section 3 discusses their discrete equivalents. Two other discrete equivalents of the Kapur and Garg [6] model are also included: these exist in the literature due to Innoue [19] and Satoh [15]. They are subsequently shown to be equivalent to a discrete equivalent of Kapur and Garg [6] that is mathematically and computationally much simpler. The derivation of continuous SRGMs from the equivalent discrete model has also been shown. The interesting part of the paper begins in section 4, with the introduction of a new concept of a power logistic learning function, which proves to be very flexible in the sense that it represents various curve types – exponential, Rayleigh, Weibull or simple logistic. In the same section the concept of generalized SRGMs is taken further by defining the software fault detection rate as the convex combination of the power logistic and Weibull functions. From the numerical illustrations, it is seen that

the newly defined class of generalized SRGMs yields considerably improved results, with a better predictability resulting from a lower MSE and higher coefficient of variation.

2. FLEXIBLE SRGMS

2.1 Inflection S-shaped SRGM Ohba [12]

This model has been developed under the assumption that the more that errors are detected, the more undetected errors become detectable.

$$\frac{d}{dt}m(t) = b(t)\big(a - m(t)\big)$$

where $b(t) = b\varphi(t)$

and $\varphi(t) = r + (1-r)\dfrac{m(t)}{a}$

The solution of the above differential equation with the initial condition $m(t = 0) = 0$ is:

$$m(t) = a\left(\frac{1 - e^{-bt}}{1 + \dfrac{1-r}{r}e^{-bt}}\right) \tag{1}$$

where r, $(0 \le r \le 1)$, is a proportion of independent errors and is the inflection parameter. The model becomes exponential when $r = 1$ and takes the logistic form when $r = 0$. It implies that initially r is almost zero, but as faults are detected and removed it approaches unity. Here $\varphi(t)$ reflects the gradual increase in testing effort and the skill gained by the testers as the testing progresses.

2.2 Flexible SRGM Bittanti *et al* [2]

This model is based on the following differential equation:

$$\frac{d}{dt}m(t) = k(m)\big(a - m(t)\big)$$

where $k(m) = k_i + (k_f - k_i)\dfrac{m(t)}{a}$

Here k_i and k_f are initial and final values of the fault exposure coefficient. If ki $= k_f$, then it reduces to the exponential model. If $k_f >> k_i$; the failure growth curve takes S-

shape. If k_f is very small compared to k_i and is almost equal to zero, the failure growth curve becomes flat at the end.

The solution of the equation with initial condition $m(t = 0) = 0$ is :

$$m(t) = a\left(\frac{1 - e^{-k_f t}}{1 + \frac{k_f - k_i}{k_i} e^{-k_f t}}\right) \tag{2}$$

For different values of k_f and k_i, it describes different growth curves.

2.3 SRGM for an error removal phenomenon Kapur and Garg [6]

This model is based upon the assumption that the detection of errors also results in detection of some of the remaining errors without these errors causing any failure.

The differential equation for this model is given by:

$$m(t) = p\big(a - m(t)\big) + q\left[\frac{m(t)}{a}\right]\big(a - m(t)\big)$$

The solution is:

$$m(t) = a\left(\frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p} e^{-(p+q)t}}\right) \tag{3}$$

where

p : Failure occurrence rate.
q : Fault removal rate of additional removed faults.

This model was developed to account for some additional faults being detected without their causing failure. Here it may be observed that if $q = 0$, then on each failure, only the error causing the failure is removed and it corresponds to the exponential model. The failure growth curve defined by the above model is S-shaped whose nature depends on q/p.

2.4 One-stage equivalent model for (1), (2) & (3)

All three models – Ohba [12], Bittanti [2], and Kapur and Garg [6] – can be written in general form (one-stage process):

$$\frac{d}{dt}m(t) = b(t)\ (a - m(t))$$

with $b(t) = \dfrac{b}{1+\beta e^{-bt}}$

Upon solving we get:

$$m(t) = a\left(\frac{1-e^{-bt}}{1+\beta e^{-bt}}\right) \tag{4}$$

By giving different forms to b and β, the above three models can be derived.

Here, in the alternate formulation, it may be observed that the fault detection rate for the above specified models is a logistic function, and $b(t) \to b$ as $t \to \infty$. Here β is the learning factor, and it represents the skills and experience gained by the testers during testing. If $\beta = 0$, then $b(t) = b$, i.e. constant.

3. DISCRETE SRGMS

NHPP-based SRGMs described above are continuous time models that use the execution time (i.e., CPU time) or calendar time. The other group contains models, that use the test cases as a unit of fault removal period. Such models are called discrete time models, since the unit of software fault removal period is countable. A test case can be a single computer test run executed in an hour, day, week or even month. Therefore, it includes the computer test run and length of time spent to inspect the software source code visually. A large number of models have been developed in the first group, while fewer are in the second group owing to the difficulties caused by their mathematical complexity.

Nevertheless, the utility of discrete SRGMs cannot be underestimated. As the software failure data sets are discrete, these models often provide a better fit than their continuous time counterparts. Therefore, in spite of the difficulties caused by mathematical complexity, discrete models are proposed regularly [7, 18].

Therefore we develop flexible discrete SRGMs on the lines of the abovementioned continuous time SRGMs, using a probability generating function (PGF). Two other forms of the model are also given, and it can be shown that the three forms are equivalent. It is further shown how a continuous time SRGM can be derived from the discrete model. Since most of the software failure data sets are discrete, therefore, it is suggested that discrete SRGMs should be used.

3.1 Discrete time flexible SRGM for error removal [6]

Most of the software reliability growth models assume that the fault removal phenomenon also describes the failure phenomenon. This may not always be true. The test team can remove some faults in the software without these faults causing

any failure, although this may involve some additional effort. A fault that is removed consequent to a failure is known as a leading fault. While removing the leading faults, some other faults are removed that might have caused future failures. These are known as dependent faults.

The difference equation under the above assumptions for the SRGM is given by:

$$\frac{m(n+1)-m(n)}{\delta} = p[a - m(n)] + \frac{q}{a} m(n+1)[a - m(n)]$$

where δ is a constant time interval.

The solution can be obtained as follows (using PGF):

$$(1 - \delta q)m(n+1) = (1 - \delta p)m(n) + a\delta p - \delta \frac{q}{a} m(n+1)m(n)$$

Now by multiplying both sides z^n and sum over n from 0 to ∞, we get:

$$(1 - \delta q)\sum_{n=0}^{\infty} z^n m(n+1) = (1 - \delta p)\sum_{n=0}^{\infty} z^n m(n) + a\delta p \sum_{n=0}^{\infty} z^n - \delta \frac{q}{a} \sum_{n=0}^{\infty} z^n m(n+1)m(n)$$

Denote $\sum_{n=0}^{\infty} z^n m(n) = P(z)$ and solving we get:

$$[(1-\delta q) - (1-\delta p)z][zm(1) + z^2 m(2) + ...]$$
$$= a\delta p(z + z^2 + z^3 + ...)$$
$$- \frac{\delta q}{a}(z^2 m(1)m(2) + z^3 m(2)m(3) + ...)$$

Comparing the coefficients of z^n, by mathematical induction we get:

$$m(n) = a\left[\frac{1-(1-\delta(p+q))^n}{1+\frac{q}{p}(1-\delta(p+q))^n}\right] \quad (5)$$

Here as $n \to \infty, m(n) \to a$.

The above discrete SRGM is equivalent to the continuous time SRGM of Kapur and Garg [6] when $\delta \to 0$.

Define $t = n\delta$

as $\delta \to 0$

$$a\left[\frac{1-(1-\delta\,(p+q))^{n}}{1+\frac{q}{p}(1-\delta\,(p+q))^{n}}\right] \rightarrow a\left[\frac{1-e^{-(p+q)t}}{1+\frac{q}{p}e^{-(p+q)t}}\right]$$

3.2 Other discrete SRGMs

Various other discrete SRGMs have been reported in the literature corresponding to the continuous model. One is developed by Yamada and Inoue [19]. It is based on the discrete Riccati equation.

It is given by

$$\frac{m(n+1)-m(n)}{\delta} = pa + (q-p)\left[\frac{m(n)+m(n+1)}{2}\right] + \frac{q}{a}m(n).m(n+1)$$

The exact solution of the above difference equation is given by:

$$m(n) = a\left[\frac{1-\left[\frac{\left(1-\frac{1}{2}\delta\,(p+q)\right)}{\left(1+\frac{1}{2}\delta\,(p+q)\right)}\right]^{n}}{1+\frac{q}{p}\left[\frac{\left(1-\frac{1}{2}\delta\,(p+q)\right)}{\left(1+\frac{1}{2}\delta\,(p+q)\right)}\right]^{n}}\right] \tag{6}$$

Similarly, using the Satoh [15] approach, another discrete equivalent of a continuous model (3) can be obtained.

$$\frac{m(n+1)-m(n-1)}{2\delta} = p\left[a-\frac{m(n+1)+m(n-1)}{2}\right]+\frac{q}{a}\left[a\left(\frac{m(n+1)+m(n-1)}{2}\right)-m(n+1).m(n-1)\right]$$

The exact solution as given by Satoh is:

$$m(n) = a\left[\frac{1-\left(\frac{1-\delta(p+q)}{1+\delta(p+q)}\right)^{n/2}}{1+\frac{q}{p}\left(\frac{1-\delta(p+q)}{1+\delta(p+q)}\right)^{n/2}}\right] \tag{7}$$

Again, it can be shown that as $\delta \rightarrow 0$, m(n) given by (6) and (7) converge to the corresponding continuous SRGM given by (3).

3.3 Equivalence of discrete SRGMs given by equation (5), (6) and (7)

Consider (7)

$$m(n) = a\left[\frac{1-\left(\frac{1-\delta(p+q)}{1+\delta(p+q)}\right)^{n/2}}{1+\frac{q}{p}\left(\frac{1-\delta(p+q)}{1+\delta(p+q)}\right)^{n/2}}\right]$$

Here $0 < \delta\,(p+q) < 1$.

Using $(1-x)^n \approx 1-nx$, we get:

$$m(n) = a\left[\frac{1-\left(\frac{1-\frac{\delta}{2}(p+q)}{1+\frac{\delta}{2}(p+q)}\right)^{n}}{1+\frac{q}{p}\left(\frac{1-\frac{\delta}{2}(p+q)}{1+\frac{\delta}{2}(p+q)}\right)^{n}}\right]$$

which is the same as equation (6).

In the above equation we can use the following expansion:

$$\left(\frac{1-\frac{\delta}{2}(p+q)}{1+\frac{\delta}{2}(p+q)}\right)^{n} = \left[1-\frac{\delta}{2}(p+q)\right]^{n}\left[1+\frac{\delta}{2}(p+q)\right]^{-n}$$

$$\approx \left[\left(1-\frac{\delta}{2}(p+q)\right)\left(1-\frac{\delta}{2}(p+q)\right)\right]^{n}$$

$$\approx \left[1-\delta(p+q)\right]^{n} \qquad (8)$$

Since $0<\delta(p+q)<1$,

$$(1+\frac{\delta}{2}(p+q))^{-1} \approx 1-\frac{\delta}{2}(p+q)$$

Using (8), equation (6) can be written as:

$$m(n) = a\left[\frac{1-(1-\delta(p+q))^{n}}{1+\frac{q}{p}(1-\delta(p+q))^{n}}\right]$$

It is the same as equation (5).

A discrete equivalent to (3) given by (5) has been observed to be easier and faster to compute, compared with (6) and (7).

4. A NEW CLASS OF GENERALIZED KAPUR AND GARG MODEL

4.1 Generalized Model-1

An interesting variation of a continuous time SRGM developed by Kapur and Garg [6] is now presented: it adds extra flexibility and wider scope to the already flexible model.

Consider the case when the failure growth phenomenon also depends upon the testing time, in addition to the number of faults remaining in the software, as well as the number of faults already identified [6]. Based on these assumptions, the differential equation for fault identification / removal can be written as:

$$\frac{dm}{dt} = t^k \left(p + q\frac{m}{a} \right)(a - m) \tag{9}$$

Here $k \geq 0$. It can be easily seen that if $k = 0$ it is same as the differential equation for SRGM developed by Kapur and Garg [6].

Solving it with the initial condition $m(0) = 0$ we get:

$$m(t) = a \left(\frac{1 - e^{-(p+q)\frac{t^{k+1}}{k+1}}}{1 + \frac{q}{p} e^{-(p+q)\frac{t^{k+1}}{k+1}}} \right) \tag{10}$$

The solution obtained is capable of capturing different types of variations shown by the curve of $m(t)$, depending upon different values of k. It can be easily shown that if $k = 0$ and $q = 0$ the model simply takes the exponential form, and if $k = 0$ and $q \neq 0$ it reduces to Kapur and Garg [6].

4.2 Alternate formulation for generalized model-1

4.2.1 Alternate formulation –1

The number of failures during testing is dependent upon the number of instructions executed. Here the number of instructions executed is a function of testing time.

Using these assumptions, the failure removal phenomenon can be expressed as:

$$\frac{dm}{dt} = \frac{dm}{de}\frac{de}{dt}$$

Define:

$$\frac{dm}{de} = \left(p + q\frac{m}{a}\right)(a - m)$$

Let the number of instructions executed be a power function of testing time:

$$\frac{de}{dt} = t^k$$

By combining the above two factors, we can write:

$$\frac{dm}{dt} = t^k\left(p + q\frac{m}{a}\right)(a - m)$$

which is the same as (9).

4.2.2 Alternate formulation–2

Another alternate formulation for SRGM given by (10) is described, which will allow definition of a new type of logistic function.

Consider:

$$\frac{dm}{dt} = \frac{bt^k}{1 + \beta e^{-b\frac{t^{k+1}}{k+1}}}(a - m) \qquad (11)$$

Solving it with the initial condition $m(0) = 0$, we get the same result as in (9).

Here $b(t) = \dfrac{bt^k}{1 + \beta e^{-b\frac{t^{k+1}}{k+1}}}$ can be termed a power logistic rate which reduces to the ordinary logistic rate if $k = 0$.

4.3 Generalized model-2

The basic levels of testing are ‘white-box’ testing and ‘black-box’ testing. White-box testing is performed to find faults that lie within the internal structure of the software. It requires the tester to have a detailed knowledge of the internal structure. The main

objective of white-box testing is to ensure that test cases execute every path throughout the code. The importance of white-box testing is expressed in terms of test or code coverage metrics, which measure the fraction of code executed by test cases.

Black-box testing is carried out to judge how well software meets the user's requirements.

Testing coverage is defined as the extent or degree to which software is executed by the test cases. During testing, if few bugs are encountered, it does not necessarily indicate that the coding is of a high quality; on the contrary, it implies the poor design of test cases. To ensure better testing, testing coverage analysis is used to assess the quality of test cases. Testing coverage is actually a structural testing technique in which software performance is judged with respect to the specification of the source code. The basic testing coverage measures are:

[1] Statement coverage: It is defined as the proportion of lines executed in the program. If it is assumed that the faults are uniformly distributed throughout the code, then the percentage of executable statements covered shows the percentage of faults discovered.
[2] Decision / condition coverage: This measure indicates whether Boolean expressions tested in control structures evaluated to both true and false.
[3] Path coverage: This measure shows the percentage of all possible paths existing in the code exercised by the test cases.
[4] Function coverage: This measure indicates the proportion of functions/ procedures influenced by the testing.

Testing coverage provides an important criterion for the optimal release policy, based on available testing resources and the importance of risk-free and safe operations of the software on implementation. Therefore, safety-critical systems have a high coverage objective.

A software reliability growth model that incorporates the concept of testing coverage in the model building is now presented. The concept of convex combination of two different types of functions to represent the testing coverage function is used. This approach helps to capture various types of coverage function simultaneously, depending on the various possible values of the parameters. The goodness of the fit has been tried on two real software failure datasets. The results obtained are fairly accurate, and show considerable improvement and a better fit.

Let $c(t)$ define the proportion of the software executed by the test cases. So, $1 - c(t)$ defines the proportion of the code which is yet to be covered by the test cases. Then, the first order derivative of $c(t)$, denoted by $c'(t)$, represents the testing coverage rate. The ratio of the two – i.e. $\frac{c'(t)}{1-c(t)}$ – will be taken as measure of the fault detection rate.

Let

$$c(t) = 1 - \exp(-k\lambda\, t^{u} - (1-k)(e^{\gamma t^{b}} - 1)) \tag{12}$$

Here $0 \le k \le 1$ and $\lambda, u, b, \gamma > 0$
where *u* and *b* are the shape parameters
λ, γ are the scale parameters.

Using (12)

$$\frac{c'(t)}{1-c(t)} = k\lambda\, u\, t^{u-1} + (1-k)\, b\, t^{b-1} \gamma\, e^{\gamma t^{b}} \tag{13}$$

Consider the model having the fault detection rates as defined above in (13) respectively:

$$\frac{d}{dt} m(t) = \frac{c'(t)}{1-c(t)} (\mathrm{a - m(t)}) \tag{14}$$

It can be seen that failure intensity not only depends on the number of remaining faults in the software, but also on the ratio of the rate at which the remaining faults are covered and the present proportion of uncovered faults.

Using the fault detection rate as defined in (13), we get:

$$\frac{d}{dt} m(t) = (k\lambda\, u t^{u-1} + (1-k) b t^{b-1} \gamma e^{\gamma t^{b}})(\mathrm{a - m(t)}) \tag{15}$$

Using the initial condition $m(0) = 0$, we get:

$$m(t) = a\left[1 - \exp(-k\lambda\, t^{u} - (1-k)(e^{\gamma t^{b}} - 1))\right] \tag{16}$$

where $m(t) \to a$ as $t \to \infty$.

If k=1 it is the same as the Weibull model developed by Yamada.[20].

5. MODEL VALIDATION

To check the accuracy of the generalized SRGM-1 given by (10) with power logistic and generalized SRGM-2 given by (16) with convex combination of two different types of function with respect to flexible SRGM of type (4), parameter estimation on real software failure datasets has been done.

Data Set-I (DS-I):

The data was obtained from Musa *et al* [9]. The software was a real time command and control system which was tested for 92 days (21 weeks). The delivered object instructions were 21,700 involving 9 programmers; 136 faults were removed during testing.

The estimation results are given in Table 1.

Data Set – II (DS-II):

The data are cited from Brooks and Motley [3]. The fault data set is for a radar system of size 124 KLOC (kilo lines of code) tested for 35 months, in which 1301 faults were identified.

The estimation results are given in Table-2.

Data Set	Parameter estimation	Models under comparison		
		Flexible Model	Generalized SRGM-1	Generalized SRGM-2
DS-I	a	168	140	142
	b	.33242	.00403	.0105
	β	193.55	8.073	-
	k	-	1.7103	.000025
	λ	-	-	.0595
	u	-	-	4.81
	γ	-	-	.162
Comparison Criterion	R^2	.99310	.99738	.99745
	MSE	16.354	6.21	6.037

Table 1: Estimation results for DS-I

Data Set	Parameter Estimation	Models Under Comparison		
		Flexible Model	Generalized SRGM-1	Generalized SRGM-2
DS-II	a	1331	1322	1305
	b	.20059	.1234	.423
	β	20.16	11.15	-
	k	-	.1678	.0029
	λ	-	-	.177
	u	-	-	2.594
	γ	-	-	.0262
Comparison Criterion	R^2	.99904	.99911	.99909
	MSE	204	190.35	193.86

Table 2: Estimation results for DS-II

5.1 Goodness of fit curves for the proposed generalized SRGMs

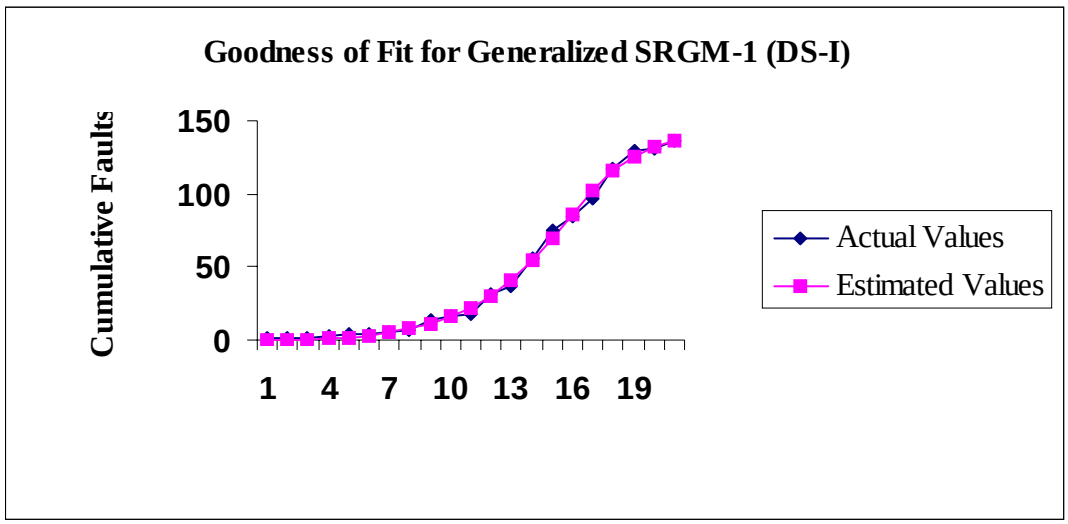

Figure 1: Cumulative faults vs time for SRGM-1(DS-I)

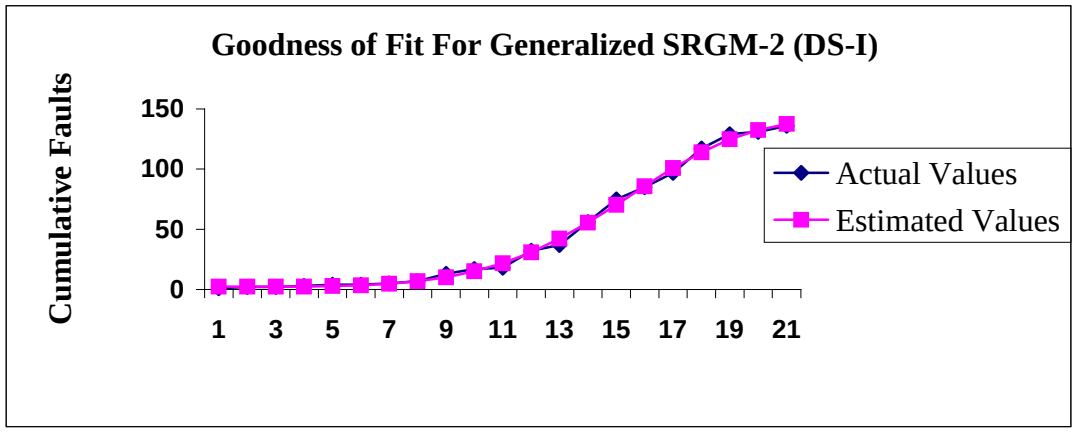

Figure 2: Cumulative faults vs time for SRGM-2(DS-I)

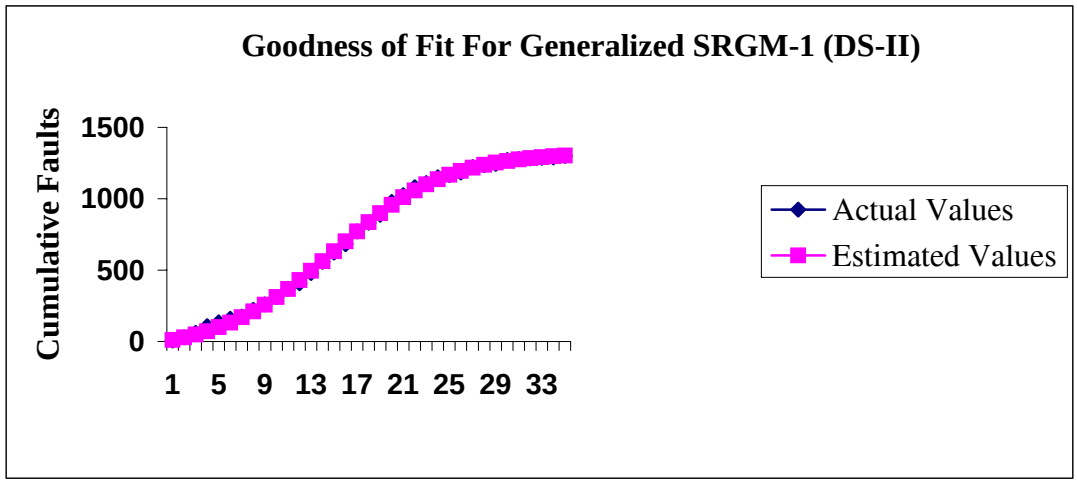

Figure 3: Cumulative faults vs time for SRGM-1 (DS-II)

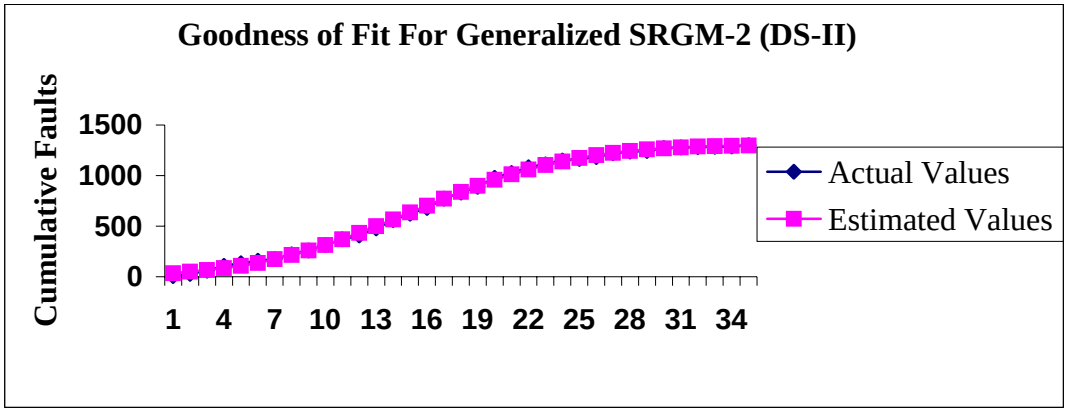

Figure 4: Cumulative faults vs time for SRGM-2 (DS-II)

It has been observed that the generalized SRGM-1 and 2 given by (10) and (16) has always provided results that are the same as or better than the flexible SRGM of type (4). This is due to the flexibility given to the model by the presence of the 'k' factor. The increased accuracy achieved shows the ability of the model to capture different types of failure datasets – e.g. exponential, S-shaped Rayleigh or Weibull types, etc.

6. CONCLUSION

Software reliability engineering is rapidly emerging as an important field of study in the area of information technology. Mathematical models play a significant role in its growth. These models provide quantitative tools to assess the reliability of the developers' software. In this paper, an important class of SRGMs, known as 'flexible SRGMs', has been assessed. New dimensions have been added to the flexible modelling by introducing the concept of the 'k' factor and convex combination, which proves to be more flexible and yields a better predictability and higher degree of accuracy.

The usability of this newly introduced concept is not limited to these SRGMs. It can be extended to improve the results of any SRGM, whether exponential, S-shaped or of any other type – for example, multi-stage models or SRGMs related to distributed environment**.**

7. ACKNOWLEDGEMENT

This research work was initiated while Prof. P.K. Kapur was visiting researcher at the Department of Industrial and Systems Engineering, University of Pretoria, South Africa.

8. REFERENCES

[1] **Bass FM.** 1969 "A new product growth model for consumer durables". *Management Science*; 15(5): pp 215-224.

[2] **Bittanti S, Bolzern P, Pedrotti E, Scattolini R**. 1988. "A flexible modelling approach for software reliability growth". *Software Reliability Modelling and*

Identification. G. Goos and J. Harmanis (eds), Springer Verlag, Berlin, pp 101-140.

[3] **Brooks WD, Motley RW.** 1980. "Analysis of discrete software reliability models – Technical Report (RADC-TR-80-84)". New York: Rome Air Development Center.

[4] **Goel AL, Okumoto K**. 1979. "Time dependent error detection rate model for software reliability and other performance measures". *IEEE Transactions on Reliability*, R-28(3): pp 206-211.

[5] **Jelinski Z, Moranda PB.** 1972. "Software reliability research", in Freiberger W, (ed.) *Statistical Computer Performance Evaluation,* New York: Academic Press: pp 465-497.

[6] **Kapur PK, Garg RB.** 1992. "A software reliability growth model for an error removal phenomenon". *Software Engineering Journal*, 7: pp 291-294.

[7] **Kapur PK, Garg RB, Kumar S**. 1999. *Contributions to hardware and software reliability*. Singapore, World Scientific Publishing Co. Ltd.

[8] **Kapur PK, Shatnawi O and Singh O.** 2002**. "**Discrete imperfect software reliability growth models under imperfect debugging environment", in: Rajaram NJ and Verma AK (eds), *Proceedings of the International Conference on Multimedia and Design*; Arena Multimedia & IIT: Mumbai, Vol (II): pp. 114-29.

[9] **Musa JD.** 1979. "Validity of execution time theory of software reliability". *IEEE Transactions on Reliability,* R-28: pp 181-191.

[10] **Musa JD,** 1999. *Software Reliability Engineering*, McGraw-Hill.

[11] **Musa JD, Iannino A, Okumoto K.** 1987.*Software reliability: Measurement, Prediction, Applications*. New York: Mc Graw Hill.

[12] **Ohba M.** 1984. "Software reliability analysis models". *IBM Journal of Research and Development*; 28: pp 428-443.

[13] **Pham H.** 2000. *Software Reliability*. Springer-Verlag Singapore Pvt. Ltd.

[14] **R. Hirota,** 1979. "Nonlinear partial difference equation v. nonlinear equations reducible to linear equations". *Journal of the Physical Society of Japan,* 46, pp 312-319.

[15] **Satoh D,** 2001. "A discrete bass model and its parameter estimation". *Journal of Operations Research Society of Japan*, 44(1): pp 1-18.

[16] **Xie M**. 1991. *Software reliability modeling*. World Scientific.

[17] **Yamada S, Ohba M., Osaki S**. 1983. "S-shaped software reliability growth modelling for software error detection". *IEEE Trans. On Reliability*, R-32(5): pp 475-484.

[18] **Yamada S and Osaki S**. 1985, "Discrete Software reliability growth models". *Applied stochastic models and data analysis***,** Vol. 1: pp. 65-77.

[19] **Yamada S, Inoue S and Yamamoto T.** 2004. "Software Reliability Growth Modeling Based on Testing-coverage". To appear in *International Journal of Quality, Reliability and Safety Engineering.*

[20] **Yamada, S., J. Hishitani and S. Osaki ,** 1993. "Software Reliability Growth Model with Weibull testing effort: A model and application". *IEEE Trans. on Reliability*, R-42, pp 100-105.