AN ALGORITHM FOR SOFTWARE RELIABILITY GROWTH MODEL SELECTION

by

Hakan Burak Duygulu

B.S., Electrical & Electronics Engineering, Orta Doğu Teknik University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2007

AN ALGORITHM FOR SOFTWARE RELIABILITY
GROWTH MODEL SELECTION

APPROVED BY:

Prof. Oğuz Tosun . . . . . .
(Thesis Supervisor)

Assis. Prof. Ayşe Başar Bener . . . . . .

Prof. Süleyman Özekici . . . . . .

DATE OF APPROVAL: 31.10.2007

ACKNOWLEDGEMENTS

It is a pleasure to thank the many people who made this thesis possible.

Firstly, I would wish to express my sincere thanks to my supervisor Prof. Oğuz Tosun for his invaluable guidance, continuous support and advises during this research.

I would also like to thank Assist. Prof. Ayşe Başar Bener and Prof. Süleyman Özekici for their comments and suggestions as the members of my thesis committee.

Special thanks to Oya Ünlü, for her valuable comments during this research and revising the thesis. I would also thank her for her support and understanding throughout my study. Her encouragement and patience helped me to complete this research.

And finally, I would like to express my gratitude to my family for their moral support and patience during my academic life and this research. I owe them a debt of thanks for everything in my life.

ABSTRACT

AN ALGORITHM FOR SOFTWARE RELIABILITY GROWTH MODEL SELECTION

Software plays a critical role in our daily life since it is used in almost every area. Since the software is so important and used widely; it is important to have reliable software. Thus it is very important to use methods to measure and control the reliability of the software. To measure the reliability, a large number of software reliability models have been proposed in the literature. Despite that there exist a large number of software reliability models, there does not exist a model that can be used in all cases and universally recommended. For this reason, recent works have been focused on selecting the reliability growth model among the available models, which best describes the software environment.

In this thesis, we propose an algorithm for software reliability growth model selection. The proposed algorithm is aimed to be a guideline for potential users who wants to evaluate software reliability. The proposed algorithm is tested with the publicly available data and performed satisfactory for reliability model selection.

Moreover, a detailed literature survey on the software engineering process, existing software reliability models and software reliability model selection methods is presented.

ÖZET

GELİŞEN YAZILIM GÜVENİLİRLİĞİ İÇİN MODEL SEÇİM ALGORİTMASI

Yazılım, günlük yaşantımızda hemen hemen her alanda kullanıldığı için kritik bir rol oynar. Yazılım bu kadar önemli olduğu ve çok kullanıldığı için, güvenilir bir yazılıma sahip olmak önemlidir. Bu nedenle, yazılım güvenilirliğini ölçen ve kontrol eden yöntemleri kullanmak çok önemlidir. Güvenilirliği ölçmek için, literatürde çok sayıda yazılım güvenilirlik modeli önerilmiştir. Çok sayıda yazılım güvenilirlik modeli olmasına rağmen, her türlü şartta kullanılabilecek ve evrensel olarak tavsiye edilecek bir model bulunmamaktadır. Bu nedenle, son zamanlarda ki çalışmalar; mevcut modeller arasından yazılım ortamını en iyi tanımlayan yazılım güvenilirlik modelini seçmek üzerine yoğunlaşmıştır.

Bu tezde, gelişen yazılım güvenilirliği için bir model seçim algoritması öneriyoruz. Önerilen algoritmanın, yazılım güvenilirliğini değerlendirecek olası kullanıcılar için kılavuz olması amaçlanmıştır. Önerilen algoritma herkese açık veriler ile test edilmiş ve yazılım güvenilirlik model seçimi konusunda başarılı bulunmuştur.

Ayrıca, yazılım mühendisliği süreci, mevcut yazılım güvenilirlik modelleri ve mevcut yazılım güvenilirlik model seçim metodları üzerine detaylı bir literatür araştırması sunulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

R(t)	Reliability Function.
Z(t)	Hazard Rate Function
$\mu(t)$	Mean Value Function
$\lambda(t)$	Failure Intensity Function
CASE	Computer Aided Software Engineering
CDF	Cumulative Distribution Function
F.C.	Failure Counts
F.N.	Failure Number
GOF	Goodness Of Fit
K-S	Kolmogorov-Smirnov
LS	Least Squares
ML	Maximum Likelihood
N.S.P.	Next Step Prediction
PDF	Probability Density Function
PL	Prequential Likelihood
PLR	Prequential Likelihood Ratio
S.L.F.	Seconds Last Failure
SREP	Software Reliability Engineering Process
SRGM	Software Reliability Growth Model
SSE	Sum of the Squared Errors
T.B.F.	Time Between Failures

1. INTRODUCTION

Software plays a critical role not only in scientific or business applications; but also in embedded applications in telephones, automobiles, televisions etc., which affects our daily life. Since the software is so important and used widely; it is important to have reliable software. Because software generally manages the overall system functions; the faults in the software may cause critical problems such as human death, injuries or financial loss.

Thus it is very important to use methods, which measure and control the reliability and quality of the software in use. Note that, software reliability measurement and prediction are useful approaches to quantify software quality [1]. To measure and control the reliability, more than hundred models have been proposed in the literature [2]. Although there exists more than hundred software reliability models, there does not exist a model that can be used in all cases and universally recommended to users. Also there does not exist a guideline with high confidence level for the potential users that they can select and use any particular model.

In this thesis, a software reliability model selection algorithm is proposed. The proposed algorithm is aimed to be a guideline for the potential users who want to evaluate the software reliability. The proposed algorithm is tested with publicly available data sets and test and performance results are presented.

In the remaining part of this thesis, there are six more chapters. In chapter 2, we discuss the entire software reliability engineering process and describe the components forming the process in a detailed manner. Also, we discuss some of the existing reliability model selection algorithms. Chapter 3 discusses the proposed algorithm. In chapter 4, we present the performance study of the proposed algorithm. Chapter 5 gives concluding remarks on the proposed algorithm and test results. Appendix A gives an overview of the random variables and important properties of the random variables. Appendix B discusses a CASE tool, named CASRE, which is used in the performance studies.

2. BACKGROUND INFORMATION

2.1. Software Reliability Theory

In today's world, people become more dependent on computers in their daily lives. Computers are diffused to every area of modern society. For example, they are embedded in telephones, home appliances, automobiles, aircrafts, etc. Therefore, most of the industries, such as automotive, banking, telecommunication, etc. are highly dependent on computers for their basic functions. The most significant aspects of the customers' needs in these areas are, the quality of the software, the time of delivery and the cost of the product. Quantitative measures have been existed for a long time, but the quantification of the quality of the software has been more recent. McCall [3] defines the elements of software quality as:

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability
- Maintainability
- Testability
- Flexibility
- Portability
- Reusability
- Interoperability

Since the software reliability is the most easily quantifiable attribute of the software quality, there have been considerable amount of research about software reliability in the literature.

2.1.1. Definitions

Software Reliability is the probability that a system functions without failure for a specified time in a specified environment [4]. Where, a failure is a departure system behavior in execution from user needs, it is a user-oriented concept. A fault is the defect that causes the failure when executed and it is a developer-oriented concept. Fault is caused by an error, which is an incorrect or missing action by a person [5].

Reliability theory can be defined as the application of probability theory to the modeling of failures and the prediction of success probabilities. Where, the random variable is the time of failure, T, and we are interested in knowing the probability that the time to failure T is in some interval $(t,t + \Delta t)$,

$$P(t \leq T \leq t+ \Delta t) \equiv \text{probability that } t \leq T \leq t + \Delta t \tag{2.1}$$

One can relate the probability to the density and distribution functions as follows,

$$P(t \leq T \leq t+ \Delta t) = f(t)\, \Delta t = F(t + \Delta t) - F(t) \tag{2.2}$$

where F(t) and f(t) are the cdf (or the cumulative distribution function) and pdf (or the probability density function), respectively. For details of the F(t) and f(t), please refer to the Appendix A. From (2.2), we have;

$$F(t) = P(0 \leq T \leq t) = \int_0^t f(x)dx \tag{2.3}$$

2.1.2. Reliability Function

The reliability function, R(t), is the probability of success at time t or in other words it is the probability the software has not failed by time t. Then, it is calculated as [6]:

$$R(t) = P(T > t) = 1 - F(t) = \int_t^\infty f(x)dx \tag{2.4}$$

2.1.3. Failure Rate

Failure rate is the probability that a failure per unit time occurs in the interval, say [t; t + Δt], given that a failure has not occurred before t [6]. That is, the failure rate is the rate at which failures occur in [t, t + Δt] [7], and then it is calculated as;

$$\begin{aligned} \text{Failure rate} &\equiv P(t \le T < t + \Delta t \mid T > t) / \Delta t \\ &= P(t \le T < t + \Delta t) / \Delta t P(T > t) \\ &= F(t + \Delta t) - F(t) / \Delta t R(t) \end{aligned} \tag{2.5}$$

2.1.4. Hazard Rate

The hazard rate is defined as the limit of the failure rate as the interval approaches zero, that is, $\Delta t \rightarrow 0$. Thus, we obtain the hazard rate at time t as [7]

$$z(t) = \lim_{\Delta t \to 0} F(t + \Delta t) - F(t) / \Delta t R(t) = f(t) / R(t) \tag{2.6}$$

The hazard rate is an instantaneous rate of failure at time t, given that the system survives up to t. In particular, the quantity z(t)dt represents the probability that a system of age t will fail in the small interval t to t + Δt [7].

2.1.5. Failure Intensity & Mean Value Function

Failure intensity is the instantaneous rate of change of the expected number of failures with respect to time and denoted as λ(t). It is the derivative of the mean value function, μ(t). Let M(t) be the random process denoting the cumulative number of failures by time t. Then, its mean value is given as [7];

$$\mu(t) = E[M(t)] \tag{2.7}$$

Then, from (2.7) we can obtain the failure intensity function as;

$$\lambda(t) = d\mu(t) / dt = d/dt\ (E[M(t)]) \qquad (2.8)$$

2.2. Relationship between Hardware and Software Reliability

Software reliability is similar to hardware reliability in that both are stochastic processes and can be described by probability distributions [6]. However, software does not wear out, burn out, or deteriorate (See figure 2.2 and 2.3) as hardware. Moreover, software generally has reliability growth during testing and operation since software faults can be detected and removed when software failures occur [6]. On the other hand, software may be modified in operation usage and this may cause reliability to decrease.

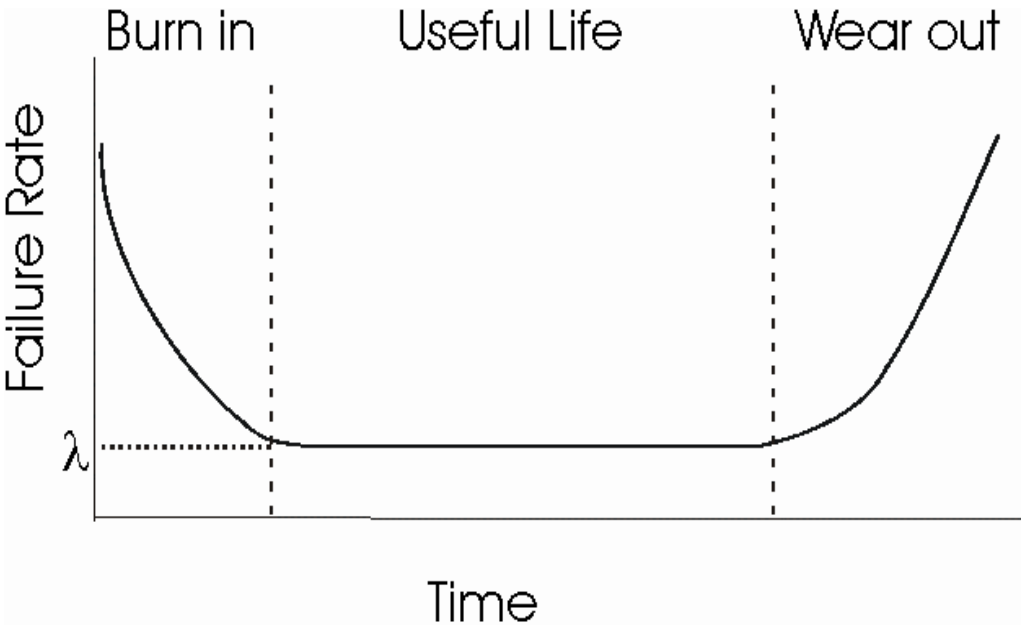

Figure 2.1. Hardware reliability

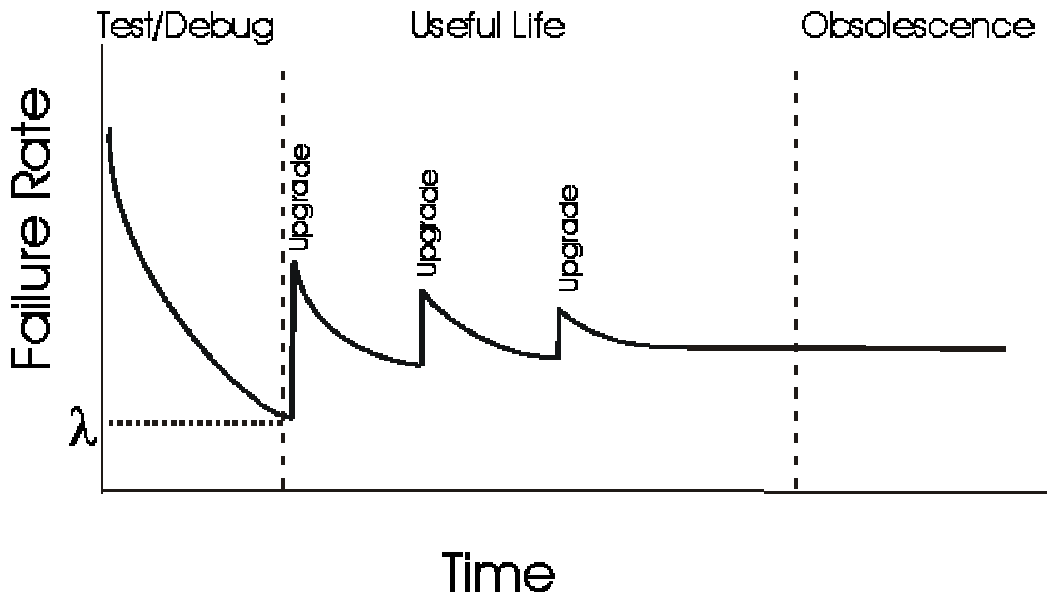

Figure 2.2. Software reliability

Despite these differences, hardware and software can be defined in the same way, so that hardware and software component reliabilities can be combined to get the system reliability. Therefore, software reliability theory has generally been developed in a way that is compatible with hardware reliability theory, so that system reliability figures may be computed using standard hardware combinatorial techniques ([8], [9]). But, note that hardware faults that are mostly physical faults, whereas software faults are design faults, which are harder to detect, and correct. This makes software reliability a challenging problem that requires an employment of several methods to attack [6]. In the next section, we will describe the general software reliability process, which includes those methods.

2.3. Software Reliability Engineering Process

The key components in the Software Reliability Engineering Process (SREP) are; the reliability objective specification, operational profile determination, reliability modeling and measurement, and reliability validation. Figure 2.3 shows the general SREP Process structure [7].

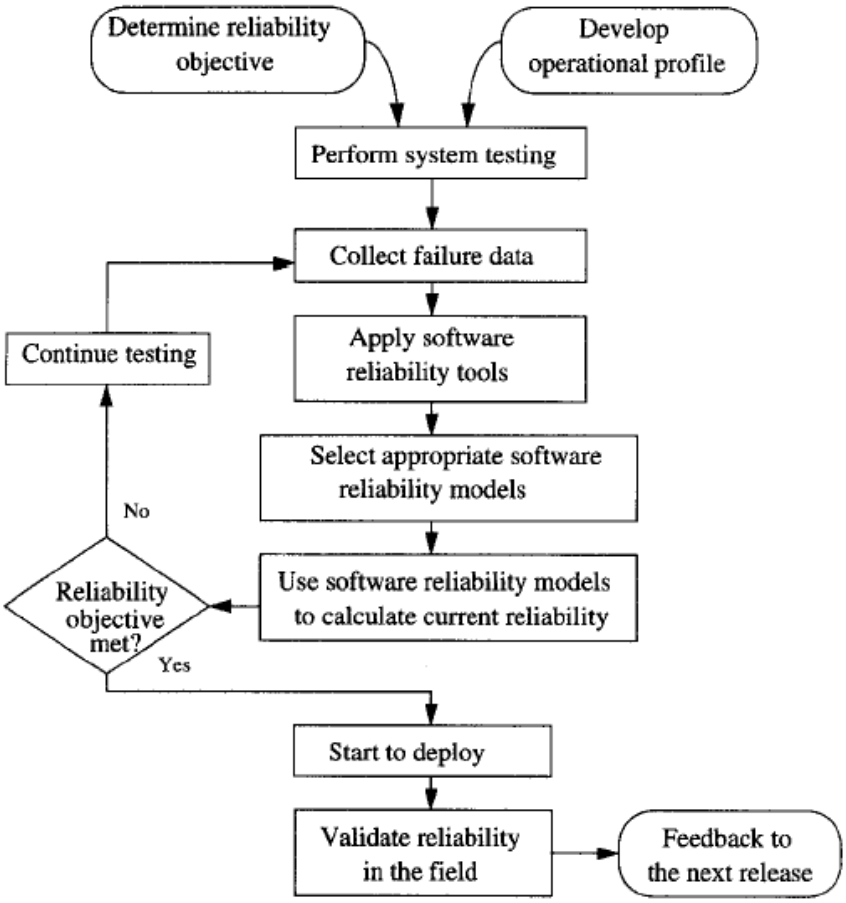

Figure 2.3. Software Reliability Engineering Process Overview

A reliability objective is the specification of the reliability goal of a product from the viewpoint of the customer [7]. With the definition of the reliability objective, it is possible to satisfy customer needs for software reliability and balance the delivery date and cost. An operational profile can be defined as a set of operations with their occurrence probabilities. Operational profile makes testing more realistic and improves the efficiency of the development and testing. Reliability modeling determines if a product meets its reliability objective and is ready for release [7]. It includes selecting a reliability model, collecting data and estimating the reliability of the system. Lastly, the projected field reliability has to be validated by comparing it with the observed field reliability in order to establish benchmarks and confidence levels of the reliability estimates and provide feedback to the SREP for process improvement [7].

Among the four key factors of the Reliability Engineering Process, we will focus on the reliability objective specification and reliability modeling and measurement. In the following sections, we will give more details on the reliability modeling and measurement.

2.4. Model Classification

A model classification scheme proposed in [10] allows relationships to be established for models within the same classification groups and shows where model development has occurred. It classifies models in terms of five different attributes:

- Time domain: Calendar versus execution time.
- Category: The total number of failures that can be experienced in infinite time. This is either finite or infinite, representing two subgroups.
- Type: The distribution of the number of the failures experienced by time t. Two important types are Poisson and Binomial.
- Class: (finite failure category only) functional form of the failure intensity expressed in terms of time.
- Family: (infinite failure category only) Functional form of the failure intensity function expressed in terms of the expected number of failures experienced.

Some important models, which can be categorized according to the above classifications, will be discussed in the following section.

2.5. Review Of The Software Reliability Models

2.5.1. Jelinski-Moranda de-eutrophication model

This model was developed by Jelinski and Moranda [11] in 1972 and was one of the first software reliability growth models. It consists of some simple assumptions:

1. There are fixed number, N, faults in the software code at the beginning of testing, where N is an unknown number.
2. Every fault has the same chance of being encountered. Also, the hazard rate of each fault does not change over time and remains constant at ϕ. That is, hazard rate

remains constant over the intervals between fault occurrences. Figure 2.4 shows the behaviour of hazard rate.

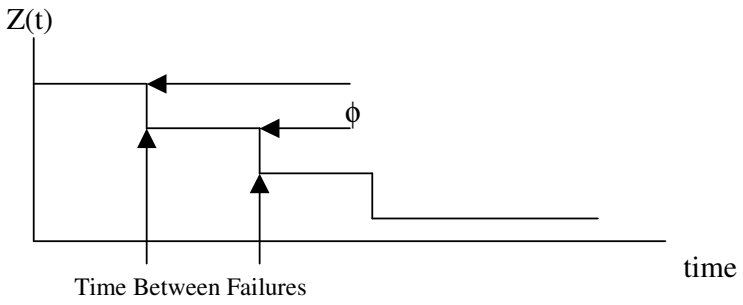

Figure 2.4. Typical plot for hazard rate

According to above assumptions, the program hazard rate after removal of the (i−1)st fault is proportional to the number of faults remaining in the software with the hazard rate of one fault, $z_a(t) = \phi$, being the constant of proportionality [7]:

$$z(\Delta t \mid t_{i-1}) = \phi[N - M(t_{i-1})] = \phi[N - (i - 1)] \tag{2.9}$$

The Jelinski-Moranda model belongs to the binomial type of models. For these models; the failure intensity function and the probability density of the time until activation of a single fault, fa(t), can be related as [7];

$$d\mu(t) / dt = Nfa(t) = N\phi\exp(-\phi t) \tag{2.10}$$

From (2.10), the mean value function is obtained as;

$$\mu(t) = N[1 - \exp(-\phi t)] \tag{2.11}$$

From (2.10) and (2.11), the failure intensity can also be expressed as

$$d\mu(t)/dt = \phi\,[N - \mu(t)] \tag{2.12}$$

Equation (2.12) implies that the failure intensity of the software at time t is proportional to the expected number of faults remaining in the software.

2.5.2. Geometric Model

This model is proposed by Moranda [12] and is a variation of the Jelinski-Moranda model. The time between failures is taken to be an exponential distribution whose mean decreases in a geometric fashion [7]. This model assumes that the discovery of the earlier faults is taken to have a larger impact on reducing the hazard rate than the later ones. In this model, hazard rate decreases in a geometric series as failures occurs. (Figure 2.5 shows this behavior.). As the Figure 2.5 illustrates, the change in the reduction of the function is larger at the beginning and getting smaller as the failures occurs.

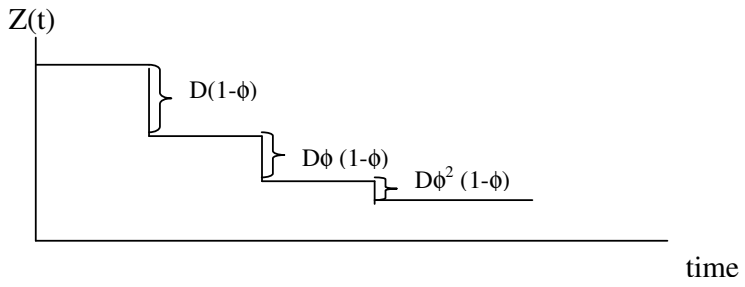

Figure 2.5. Typical plot for hazard rate

Main assumptions of this model are [7]:

1. The fault detection rate forms a geometric progression and is constant between fault detections, that is, $z(t) = D\phi^{i-1}$, where $0 < \phi < 1$ and $t_{i-1} < t < t_i$ being the time of the (i-1)st failure.
2. There is an infinite number of total faults in the system, i.e., $\lim_{t->\infty} \mu(t) = \infty$, where $\mu(t)$ is the mean value function of the process.
3. The time between fault detection follows an exponential distribution.

2.5.3. Goel-Okumoto model

This model is proposed by Goel and Okumoto in 1979 [13] and it has following assumptions [7]:

1. The number of failures, M(t), experienced by time t follows a Poisson distribution with mean value function $\mu(t)$. This mean value function has the boundary conditions $\mu(0) = 0$ and $\lim_{t->\infty} \mu(t) = N < \infty$.
2. The number of software failures that occur in $(t, t+\Delta t]$ with $\Delta t \rightarrow 0$ is proportional to the expected number of undetected faults, $N - \mu(t)$. The constant of proportionality is ϕ.
3. For any finite collection of times $t_1 < t_2 < \cdots < t_n$ the number of failures occurring in each of the disjoint intervals $[(0, t_1), (t_1, t_2), ..., (t_{n-1}, t_n)]$ is independent.

From the assumptions it can be shown that [13]

$$\mu(t) = N (1 - e^{-bt}) \quad (2.13)$$

$$\lambda(t) = Nbe^{-bt} \quad (2.14)$$

where b and N are some constants with b>0 and N>0

Note that, the failure intensity is similar to the Jelinski-Moranda model. However, in this model N is an expected value.

2.5.4. Musa basic execution time model

Musa's basic execution time model [14] was developed in 1975 was the first model, which explicitly requires that the time measurements must be in actual CPU time rather than the calendar (wall clock).

Main model assumptions are [15]:

1. The number of failures that can be experienced in infinite time is finite.
2. The cumulative number of failures by time t, M(t), follows a Poisson process with mean value function $\mu(t) = \beta_0(1 - \exp(\beta_1))$ where β_0 and $\beta_1 > 0$
3. The functional form of the failure intensity in terms of time is exponential.
4. Failure-identification personnel can be fully utilized and computer utilization is constant

5. Failure-correction personnel utilization is established by limitation of error queue length for any debugger. Error queue length is determined by assuming that error correction is a Poisson process and that servers are randomly assigned.

From these assumptions, hazard rate is defined as [15];

$$z(\tau) = fK(N-n) \tag{2.15}$$

where f is taken as the linear execution frequency (average instruction rate divided by the number of instructions in the program) and K is an error exposure ratio which relates error frequency to linear execution frequency. N is the initial number of errors and n is the errors corrected after τ amount time.

2.5.5. Musa-Okumoto logarithmic model

This model is proposed by Musa and Okumoto [16] and is based on failure data measured in execution time. This model assumes a nonhomogenous Poisson process with an intensity function, which decreases exponentially as failures occur. Exponential decrease reflects the idea, which assumes that the earlier failures have greater impact on the reducing the failure intensity than those failures, which were found later.

Main assumptions are as follows [16]:

1. At time $\tau = 0$ no failures have been observed, i.e. $P(M(0) = 0) = 1$.
2. The failure intensity decreases exponentially with the expected number of failures observed, i.e. $\lambda(\tau) = \beta_0\beta_1 \exp(-\mu(\tau)/\beta_0)$, where $\beta_0\beta_1$ is the initial failure intensity and β_0^{-1} is dubbed failure intensity decay parameter.
3. The number of failures observed by time τ, $M(\tau)$, follows a Poisson process.

2.5.6. Schneidewinds Model

The main idea is that [17] the current fault rate might be a better predicator of the future behavior than the observed rates in the distant past. To reflect this idea, Schneidewind has three forms of the model that reflect the analyst's view of the importance of the data as functions of time [7].

Suppose there are n intervals, where all of are some fixed length. Then, the three forms of the model are:

- Model 1: Utilize all of fault counts from the n periods. This view assumes that all the data points are equal importance.
- Model 2: Ignore the fault counts from the first through the s-1 periods. This view assumes that early failure intervals contribute little in predicting future behavior.
- Model 3: Use the cumulative fault counts from the intervals 1 to s-1 as the first data interval and the individual intervals for periods s through n.

.

Main assumptions are [7]:

- The cumulative number of failures by time t, M(t), follows a Poisson process with mean value function $\mu(t)$, where $\lim_{t->\infty} \mu(t) = \alpha/\beta < \infty$ for $\alpha,\beta > 0$
- The failure intensity function is assumed to be an exponentially decreasing function of time. Then the failure intensity function is taken to be of the form
  $\lambda(t)= \alpha exp(-\beta t)$
- The number of faults detected in each of the respective intervals are independent.

2.5.7. S-Shaped Reliability Growth Model

The S-shaped (or delayed S-shaped) reliability growth model [18][19] is an example of gamma distribution class model, where the per-fault failure distribution is gamma. According to this model, mean value function $\mu(t)$ is a characteristic S-shaped curve. The main idea is; the software error detection process can be described as an S-shaped growth curve to reflect the initial learning curve at the beginning, as test team members become

familiar with the software, followed by growth and then leveling off as the residual faults become more difficult to uncover [7].

Model assumptions are:

- M(t), cumulative number of failures, follows a Poisson process with the mean value function is of the form $\mu(t)= \alpha[1-(1+\beta t)\exp(-\beta t)]$ for $\alpha,\beta > 0$
- The time between failures of the (i-1)st and ith depends on the time to failure of the (i-1)st
- Failures are removed instantaneously without introducing new faults to the software.

2.5.8. Littlewood-Verrall Reliability

This model views reliability growth and prediction in a Bayesian framework. The previous models allow change in the reliability only when an error occurs. A Bayesian model takes a subjective viewpoint in that if no failure occurs while the software is being observed then the reliability should increase, reflecting the growing confidence in the software by the user. The reliability is therefore a reflection of both the number of faults that have been detected and the amount of failure-free operation [7]. Also Littlewood-Verrall model tries to account for fault generation in the fault correction process by allowing for the probability that the software program could become less reliable than before.

Main assumptions are [7]:

- Successive execution times between failures, that is, X_i, i=1,…,n, are assumed to be independent exponential random variables with parameter ξ_i , i=1,…,n.
- The ξ_i's form a sequence of independent random variables, each with a gamma distribution of parameters α and $\Psi(i)$ where the function $\Psi(i)$ describes the quality of the programmer and the difficulty of the task.

Linear and quadratic forms for the $\Psi(i)$ function are suggested by the Littlewood and Verrall, that is, $\Psi(i)= \beta_0 + \beta_1 i$ and $\Psi(i)= \beta_0 + \beta_1 i^2$ respectively [7]. Then,

$$\lambda_{linear}(t) = (\alpha -1) / \ (\beta_0^2 + 2\beta_1 t(\alpha-1))^{1/2} \quad (2.16)$$

$$\lambda_{quadratic}(t) = (v_1 / (t^2+v_2)^{1/2}).((t + (t^2 + v_2)^{1/2})^{1/3} - (t - (t^2 + v_2)^{1/2})^{1/3} \quad (2.17)$$

where $v_1= (\alpha -1)^{1/3} (18\beta_1)^{1/3}$ and $v_2 = 4\beta_0^3/ (9 (\alpha -1)^2\beta_1)$

2.5.9. Schick-Wolverton Model

This model makes the same assumptions as the Jelinski-Moranda model accept that the hazard rate is assumed to be proportional to the number of faults remaining as well as the time elapsed since last failure. Figure 2.6 illustrates the typical behavior of the hazard rate function over time. Then the hazard rate is defined as [15];

$$z(t'_i \mid t_{i-1}) = \phi [u_0-(i-1)] t'_i \quad (2.18)$$

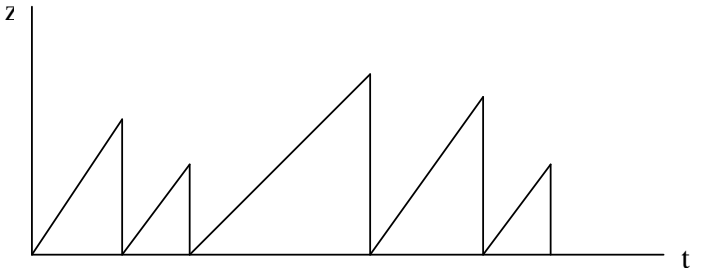

Figure 2.6. A typical hazard rate for Schick-Wolverton model

2.5.10. Generalized Poisson Model

This model can be considered to be analogous in form of to both the Jelinski-Moranda and Schick-Wolverton Models but taken within the error count framework [15].

Main assumptions are;

- The expected number of errors occurring in any interval is proportional to the error content at the time of testing and to some function of the amount of time spent in error testing.

- The errors are corrected at the ends of the testing intervals without introduction of new errors into the program. This model assumes, error corrections made at the end of the testing intervals. However, errors discovered in one testing interval can be corrected in another interval.

Expected number of errors, f_i, in the i'th interval is given as;

$$E\{f_i\} = \phi\,[N - M_{i-1}]\; g_i(x_1,x_2,\ldots,x_i) \tag{2.19}$$

where ϕ is the proportionality constant, N is the initial number of errors, M_i is the number of errors corrected in the interval i, and g_i is some function of the amount of testing time spent previously and currently.

In 2.5 we have presented some reliability models that are commonly introduced in the literature. In the next section, we'll discuss some of the methods for the parameter estimations of the reliability models.

2.6. Estimation Of Parameters

Parameter estimation is a statistical method of estimating model parameters based on failure times or number of failures per time interval. Two common parameter estimation methods will be covered in this part. These methods are: Least Squares Estimation (LS) and Maximum Likelihood (ML) estimation. LS usually requires less computation time to estimate parameters, and always find a parameter estimation. However, sometimes LS parameter estimates are meaningless. ML takes more computation time than LS. However, ML parameter estimates are always meaningful if they are produced. Therefore, it is advisable to use ML first; if it couldn't produce estimates then to use LS.

2.6.1. Least Square Estimation (LS)

In LS it is assumed that a linear law relates to variables, the independent variable x and the dependent variable y [7]:

$$y = ax + b \tag{2.20}$$

The true data relating y and x are of n pairs of points: (x_1,y_1), (x_2,y_2), ..., (x_n,y_n). The error between the true value of the dependent variable and the best fit of a linear function [7] is

$$error_i = y_i - (ax_i + b) \tag{2.21}$$

The sum of the squared errors (SSE) is given by the below equation and least squares estimation method tries to minimizing this equation:

$$SSE = \sum_{i=1}^{n} (error_i)^2 = \sum_{i=1}^{n} (y_i - ax_i - b)^2 \tag{2.22}$$

The best estimates of a and b are the values of a and b that minimize the sum of the squared errors, which is calculated by $\partial SSE / \partial a = 0$ and $\partial SSE / \partial b = 0$. Solving the resulting values of a and b results in;

$$\hat{a} = \sum^{n}_{i=1}(y_i - \bar{y})(x_i - \bar{x}) \; / \sum^{n}_{i=1}(x_i - \bar{x})^2 \tag{2.23}$$

$$\hat{b} = \bar{y} - \hat{a}\,\bar{x} \tag{2.24}$$

where $\bar{x} = (1 / n) \sum^{n}_{i=1} x_i$ and $\bar{y} = (1 / n) \sum^{n}_{i=1} y_i$ and the symbols â and $\hat{b}$ stand for the least-squares estimates of a and b, respectively.

2.6.2. Maximum-Likelihood Estimation (ML)

Suppose that $f(x; \vartheta) = f(x_1, \ldots, x_n; \vartheta)$ denotes the joint probability mass function of the random variables (r.v.) when they are discrete, and their joint probability distribution when they are continuous [20]. Then let,

$$L(\vartheta) = f(x; \vartheta) = f(x_1, , ..., x_n; \vartheta) \tag{2.25}$$

Now $L(\vartheta)$, also called the likelihood function, represents the likelihood that the values $x_1, \ldots, x_n$ will be observed when ϑ is the true value of the parameter [20]. Let $\vartheta_{ML} = s(x_1, \ldots, x_n)$ maximizes the function $L(\vartheta)$; that is,

$$L(\vartheta_{ML}) = \max L(\vartheta) \tag{2.26}$$

Then the maximum-likelihood estimator of ϑ is Θ_{ML} where,

$$\Theta_{ML} = s(X1, \ldots, x,) \tag{2.27}$$

and ϑ_{ML} is the maximum-likelihood estimate of ϑ. Since $L(\vartheta)$ is a product of either pmf s or pdf s, it will always be positive. Thus $\ln L(\vartheta)$ can always be defined, and in determining the maximizing value of ϑ, it is often useful to use the fact that $L(\vartheta)$ and $\ln L(\vartheta)$ have their maximum at the same value of ϑ [20]. Therefore, one can also obtain Θ_{ML}, by maximizing $\ln L(\vartheta)$.

2.7. Prediction Analysis and Recalibration Techniques

We have discussed some important Software Reliability Models, which are introduced in the literature. Although there are too many models that are defined in the literature, no single model can be recommended to a potential user. In fact, the accuracy of the reliability measures arising from the models tends to vary quite dramatically [7].

Since we cannot rely on a single model, we must have some procedures to compare the models and choose which one is superior to others. In this part, we will discuss such important procedures.

2.7.1. Probability Density Function

There are two types of software reliability models. First one predicts time between failures, called Time Between Failures models, and second one predicts the number of failures that will be found in future test intervals, called Failure Counts models. First type

models can be expressed as a probability density function, $\tilde{f_i}(t)$, whose parameters are estimated based on the values of previously observed times between failures $t_1, t_2, \ldots, t_{i-1}$ [21]. This probability density function is used to predict the time of the next failure. Assume that we've observed i-1 times between failures from the beginning of the tests, and we want to predict the time between failure i-1 and failure i, which will be represented by the random variable t. The expected value of t is given by:

$$E[t_i] = \int_0^\infty t\, \tilde{f_i}(t)dt \tag{2.28}$$

For Failure Counts models probability density function $\tilde{f_i}(t)$ is also defined, in which the parameters of $\tilde{f_i}(t)$ are computed based on the failure counts in the previous (i-1) test intervals. Assume that we've observed failure counts in test intervals $f_1, f_2, \ldots, f_i$, and we want to predict what the number of failures will be in interval i+1. Representing this quantity by the random variable x, we'll get our prediction by finding the expected value of x [21]:

$$E[x] = \int_0^\infty x\, \tilde{f_i}(x)dx \tag{2.29}$$

2.7.2. Prequential Likelihood Ratio

Prequential Likelihood is a measure that can tell you how much more appropriate one model is than another and it can be used to discredit one model in favor of another for a particular set of failure data [21]. It is defined as following: assume that we have observed j failures, that is $t_1, t_2, \ldots, t_j$, for predictions of $T_{j+1}, T_{j+2} \ldots, T_{j+n}$, the Prequential Likelihood is [22]

$$PL(j,t) = \prod_{i=j+1}^{j+n} \tilde{f_i}(t_i) \tag{2.30}$$

A comparison of two models, A and B, may be made by forming the Prequential Likelihood ratio $PLRn = PL_n^A / PL_n^B$. The value of the PLRn should be larger than 1 if the predictions of the A is better than B.

Figure 2.7 explains the Prequential Likelihood ratio approach in an informal way [7]. A and B are two models and make predictions at stage j. The true distribution of next time to failure, T_j, is displayed together with the estimates coming from models A and B. As can be observed from the figure, A is better than B.

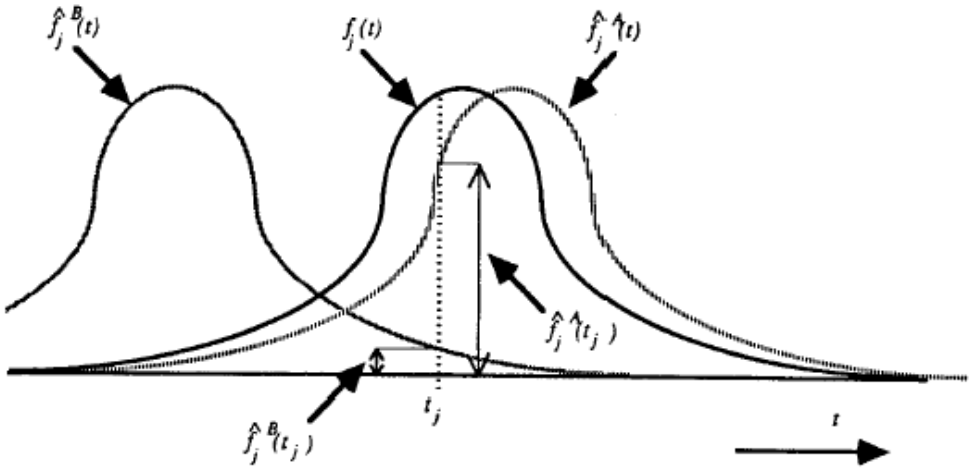

Figure 2.7. True predictive pdf, $f_j(t)$ and estimates of this pdf $\tilde{f}_j^A(t_j)$ and $\tilde{f}_j^B(t_j)$.

2.7.3. The u-plot

The u-plot method, described in this section, is based on a generalization of the simple median check and detects systematic objective differences between predicted and observed failure behavior [7]. It is used to determine whether the predictions, $F_j\tilde{}(t)$ are on average close to the true distributions, $F_j(t)$. Then,

$$u_i = F_i\tilde{}(t_i) \tag{2.31}$$

where t_i is the later-observed realization of the random variable T_i. Thus, u_i is the probability integral transform of the observation using the predictive distribution function. If the sequence of predictions $\{F_i\tilde{}(\,t_i\,)\}$ is good, it is easy to see that the sequence $\{u_i\}$ should look like a random sample from a $U(0\,,1)$ distribution [22]. The u-plot is drawn according to the following rules and can be used to determine any departure from the uniformity:

1. Sort the u_i in ascending order.
2. Suppose that there are n values of u_i. The cdf of the u_i is a step function that increases by 1/(n+1) for every u_i. This means that the cdf for the smallest u_i has a value of 1/(n+1), the cdf for the next smallest u_i is 2/(n+1), and so forth until the largest u_i is reached. The cdf for the largest u_i is n/(n+1).

If the$\{u_j\}$ sequence were truly uniform, this plot should be close to the line of unit slope. Any serious departure of the plot from this line is indicative of non-uniformity, and thus of a certain type of inaccuracy in the predictions [7].

2.7.4. The y-plot

The u-plot can be use to find the bias from the reality. There are other departures from reality, which cannot be detected by u-plot. For example, a data set can have an optimistic early predictions and pessimistic predictions in later predictions. These deviations can be averaged out in the u-plot since the temporal ordering of the u_j's disappears [7]. Therefore, it is necessary to examine the u_j's for trend.

Let U, of u_i be a set. Then, to produce a u-plot, create a new set, U', which contains the values of u_i sorted in ascending order. If the largest values of u_i were at the start of the set U, they would be moved to the end of the set U'. Since the positions of u_i can change when creating U', information relating to the way model bias changes with time can be lost. To preserve the temporal information lost in producing the bias plot, we have to perform two additional transformations after producing the random variable u_i [21]. The transformation sequence is:

$$x_i = -\ln(1-u_i) \text{ and } y_i = \sum_{j=1}^{i} xj \; / \; \sum_{j=1}^{n} xj \qquad (2.32)$$

where n is the total number of failures observed. Like the u-plot, the cumulative distribution function of the y_i is then drawn and called y-plot. The y-plot exposes temporal trends in the u_i, if exists.

2.7.5. Goodness Of Fit Tests

Goodness of fit tests are used to test how well a statistical model fits a set of observations. In this section we will discuss two common goodness of fit tests, Kolmogorov-Smirnov and Chi-square goodness of fit tests.

2.7.5.1. Kolmogorov-Smirnov Goodness-of-Fit Test:

The Kolmogorov-Smirnov test [23] is used to test whether a sample data set comes from a population with a specific distribution. It is based on the empirical cumulative distribution function (ECDF) instead of the empirical probability density function. Given N ordered data points Y_1, Y_2, ..., Y_N, the ECDF is defined as;

$$E_N = n(i) / N \tag{2.33}$$

where n(i) is the number of points less than Yi and the Yi are ordered from smallest to largest value. The Kolmogorov-Smirnov test is defined as:

$$D = \max (F(Y_i) - (i-1)/N , i/N - F(Y_i)) \tag{2.34}$$

where F is the theoretical cumulative distribution of the distribution being tested which should be a continuous distribution. If the test statistic, *D*, is greater than the critical value obtained from a table, where several variations exist in the literature, then the hypothesis regarding the distributional form is rejected.

2.7.5.2 Chi-Square Goodness-of-Fit Test

The chi-square test is used to test whether a sample data set comes from a population with a specific distribution. This test can also be applied to the discrete distributions, whereas Kolmogorov-Smirnov can only be applied to the continuous distribution.

Let's assume X comes from the distribution F(x). Then let, $p_i = F(x_i)-F(x_{i-1})$ where $i = 1,\ldots, k$ and p_i is the probability that an instance fall into bucket i (that is the interval $[x_{i-1},x_i]$). For a random sample of size n, $X_1, X_2, \ldots,X_n$ we form a new random variable to count the number of instances in each bucket [7]:

$$Y_i = \sum_{j=1}^{n} I_{[xi-1\le Xj\le xi]} \quad i = 1,\ldots,k \tag{2.35}$$

where

$$I_{[xi-1\le X\le xi]} = \begin{cases} 1 & x_{i-1}\le X\le x_i \\ 0 & \text{other} \end{cases}$$

Y_i has a joint multidimensional distribution, where the expected instances falling into bucket i is np_i. Furthermore, the sum of error squares divided by the expected numbers

$$q_{k-1} = \sum_{j=1}^{k} (y_i - np_i)^2 / np_i \tag{2.36}$$

is a measure of the closeness of the observed number of instances, y_i, to the expected number of instances, np_i, in bucket i [7]. If q_{k-1} is small, then H_0 is accepted. It can be measured in terms of statistical significance if we treat q_{k-1} as a particular value of the random variable Q_{k-1}. It can be shown that if n is large ($np_i \ge 1$), the distribution of Q_{k-1} is approximately a chi-square distribution with k-1 degrees of freedom, $\chi^2(k-1)$ [7]. If H_0 is true, we expect q_{k-1} to fall into an acceptable range of Q_{k-1}, so that the event is likely to occur. The boundary value, or critical value, of the acceptable range, $\chi^2_\alpha(k-1)$ is chosen such that

$$P[Q_{k-1} > \chi^2_\alpha(k-1)] = \alpha \tag{2.37}$$

where α is called the significance level of the test. Thus, we should reject H_0 if $q_{k-1} > \chi^2_\alpha(k-1)$ [7]. Generally, α is chosen to be either 0.05 or 0.1.

2.8. Software Reliability Model Selection Methods

Today there exists more then hundred software reliability models and more models are being developed every year. But there does not exist any model that can be applied in all cases. Worse than this, models that are good in general are not always the best choice for a particular data set, and it is not possible to know in advance what model should be used in any particular case [24]. Although there does not exist a guideline with high confidence level that we can rely on selecting any particular model; there are some methods proposed for reliability model selection. In the next parts, we will discuss some software reliability model selection methods that are defined in the literature.

2.8.1. The Method of Software Reliability Growth Models Choice Using Assumptions Matrix

In this method, Software Reliability Models' assumptions are analyzed and a matrix is constructed for these assumptions. Then, for choosing SRGM it is offered to use assumptions matrix with taking into account the features of software engineering and testing processes [25].

The software reliability growth model selection procedure, which is proposed in this method, can be explained as following:

1. Analysis of software requirements and software engineering process.
2. Obtaining of the information about assumptions usable to the developed software.
3. SRGM choice using Assumptions Matrix (Table 2.1.).
4. Determination of SRGM input parameters and analysis of their complexing opportunities.
5. SRGM parameters fitting.
6. Calculation of software reliability measures and statistical analysis of obtained data.

This model selection method relies on how much the assumptions of the reliability models are met the reality. It is noted that various assumptions have different degrees of community. Some assumptions may be applied to all SRGMs without exception; and others are applicable to SRGMs of a certain type, category, group or only to individual models. Besides, some basic models are totally defined by assumptions of higher layers and the same assumptions may be used for detailing of different models at different layers. Below, the assumptions are listed with division into four hierarchical layers: [25]

I. SRGM layer – assumptions refer to all SRGMs.

I.1. Software testing and reliability assessment is performed in actual operating conditions.

I.2. Faults are removed immediately.

I.3. New faults are not brought in the process of debugging.

I.4. All faults are occurred independently from each other.

II. Classification scheme layer (Type/Category/ Distribution layer) – assumptions refer to all SRGMs of certain types, categories, classes or families.

II.1. Category layer.

II.1.1. The total number of the failures that can be experienced in infinite time is finite (finite SRGM).

II.1.2. The total number of the failures that can be experienced in infinite time is Infinite (infinite SRGM).

II.2. Type layer.

II.2.1. Poisson SRGM layer.

II.2.1.1.Cumulative number of the failures at a time interval, follows a Poisson process.

II.2.1.2.Current number of software faults is estimated by its possible value - α.

II.2.1.3.Number of faults detected at a time interval is proportional to the current number of undetected faults.

II.2.2. Binomial SRGM layer.

II.2.2.1.Probability of faults occurrence is equal .

II.2.2.2.The number of faults in software is estimated by its fixed number - N.

II.3. Class/Family (Distribution) layer.

II.3.1. The number of detected failures is exponentially distributed (Exponential SRGM).

II.3.2. The number of detected failures follows a Gamma or Weibull's distribution (Weibull & Gamma SRGM).

III. Group layer – assumptions refer to all SRGMs of separate groups of models.

III.1. Frequency (rate) of faults detection remains constant within the interval between occurrence of faults (De-eutrophication SRGM).

III.2. SRGM NHPP group is completely determined by higher layer assumptions.

III.3. Software is divided into several sections (classes) for each of them higher layer assumptions are applied separately (Hyperexponential SRGM).

III.4. The time between two failures depends on the time before the first one (S-shaped SRGM).

III.5. SRGM Weibull group is completely determined by higher layer assumptions.

III.6. Frequency of faults occurrence forms a geometric progression (Geometric SRGM).

III.7. Frequency of faults occurrence is subjected to logarithmic function (Logarithmic SRGM).

IV. Model layer – assumptions refer only to individual models.

IV.1. De-eutrophication SRGM layer.

IV.1.1.Jelinski-Moranda SRGM is completely determined by higher layer assumptions.

IV.1.2.The program size is not substantially changed while testing (Shooman SRGM).

IV.2. NHPP SRGM layer.

IV.2.1.Goel-Okumoto SRGM is completely determined by higher layer

assumptions.

IV.2.2.More late failures influence on software reliability more strongly (Schneidewind).

IV.2.3.Software operation time is expressed as processor time units (Musa SRGM).

IV.3. Hyperexponential SRGM layer.

IV.3.1.Basic Hyperexponential SRGM is completely determined by higher layer assumptions.

IV.3.2.Software is divided into two several sections. (Lapri SRGM).

IV.4. S-shaped SRGM layer.

IV.4.1.Basic S-shaped SRGM is completely determined by higher layer assumptions.

IV.4.2.Software is divided into several sections (classes) for each of them higher layer assumptions are applied separately. Similarly to the III.3. (Hyperexponential S-shaped SRGM).

IV.4.3.Frequency of failures depends on the ratio of the number of detected failures to their initial number in software (S-shaped Ohba SRGM).

IV.4.4.Frequency of failures depends on applied test efforts (S-shaped Test Effort SRGM).

IV.5. Weibull SRGM layer.

IV.5.1.Shick-Wolverton SRGM is completely determined by higher layer assumptions

IV.5.2.Duane SRGM is completely determined by higher layer assumptions.

IV.6. Geometric SRGM layer.

IV.6.1.1-st Moranda Geometric SRGM is completely determined by higher layer assumptions

IV.6.2.Index of geometric progression is the ordinal number of test interval (2-nd Moranda Geometric SRGM).

IV.6.3.Index of geometric progression is total number of faults detected before

beginning of a current test interval (Lipow Geometric SRGM).

IV.7. Logarithmic SRGM layer.

IV.7.1. Musa-Okumoto SRGM is completely determined by higher layer assumptions.

Systemization of SRGM assumptions makes the process of model choosing more purposeful. The choice can be divided into preliminary (choice of a group of SRGMs) and final stages (choice of single or several models for immediate study of software reliability). Then the main task is correct choice of models subgroup. For preliminary choice of models, one should successively test software for conformity to assumptions of two higher layers. The choice problem is solved with the use of a binary search tree, its top nodes corresponding to assumption layer, whereas branches correspond to alternative subset of models at this layer. Then one or several SRGMs must be chosen to be used for reliability evaluation. As the model groups contain a small number of SRGM, principally all of them may be used for software reliability evaluation [25].

Table 2.1. Extended SRGM Database

Attributes				Finite												Infinite				
Category																				
Type				Binomial		Poisson									Bino-mial	Poisson				
Family/Class				Exponential						Weibull & Gamma						Exponential				
Group				1. De-eutrophication		2. NHPP			3. Hyper-exponential		4. S-shaped				5.Weibull	6. Geometric				7. Logarithmic
Model				Jelinski-Moranda	Shooman	Goel-Okumoto	Schneide-wind	Musa	Basic	Lapri	Basic	Hyper-exponential	Ohba	Test Effort	Shick-Wolverton	Duane	1-st Moranda	2-nd Moranda	Lipow	Musa-Okumoto
				1.1	1.2	2.1	2.2	2.3	3.1	3.2	4.1	4.2	4.3	4.4	5.1	5.2	6.1	6.2	6.3	7.1
Assumptions				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
I	I.1																			
	I.2			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	I.3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	I.4			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
II	II.1	II.1.1		1	1	1	1	1	1	1	1	1	1	1	1					
		II.1.2														1	1	1	1	1
	II.2	II.2.1	II.2.1.1			1	1	1	1	1	1	1	1	1		1	1	1	1	1
			II.2.1.2			1	1	1	1	1	1	1	1	1		1	1	1	1	1
			II.2.1.3			1	1	1	1	1	1	1	1	1		1	1	1	1	1
		II.2.2	II.2.2.1	1	1										1					
			II.2.2.2	1	1										1					
	II.3	II.3.1		1	1	1	1	1	1	1							1	1	1	1
		II.3.2									1	1	1	1	1	1				
III	III.1			1	1															
	III.2					1	1	1												
	III.3								1	1										
	III.4										1	1	1	1						
	III.5														1	1				
	III.6																1	1	1	
	III.7																			1
IV	IV.1	IV.1.1		1																
		IV.1.2			1															
	IV.2	IV.2.1				1														
		IV.2.2					1													
		IV.2.3						1												
	IV.3	IV.3.1							1											
		IV.3.2								1										
	IV.4	IV.4.1									1									
		IV.4.2										1								
		IV.4.3											1							
		IV.4.4												1						
	IV.5	IV.5.1													1					
		IV.5.2														1				
	IV.6	IV.6.1															1			
		IV.6.2																1		
		IV.6.3																	1	
	IV.7	IV.7.1																		1

2.8.2. An Empirical Method For Selecting Software Reliability Growth Models

This selection method applies SRGMs to weekly cumulative failure data to determine how well the method predicts the expected total number of failures.

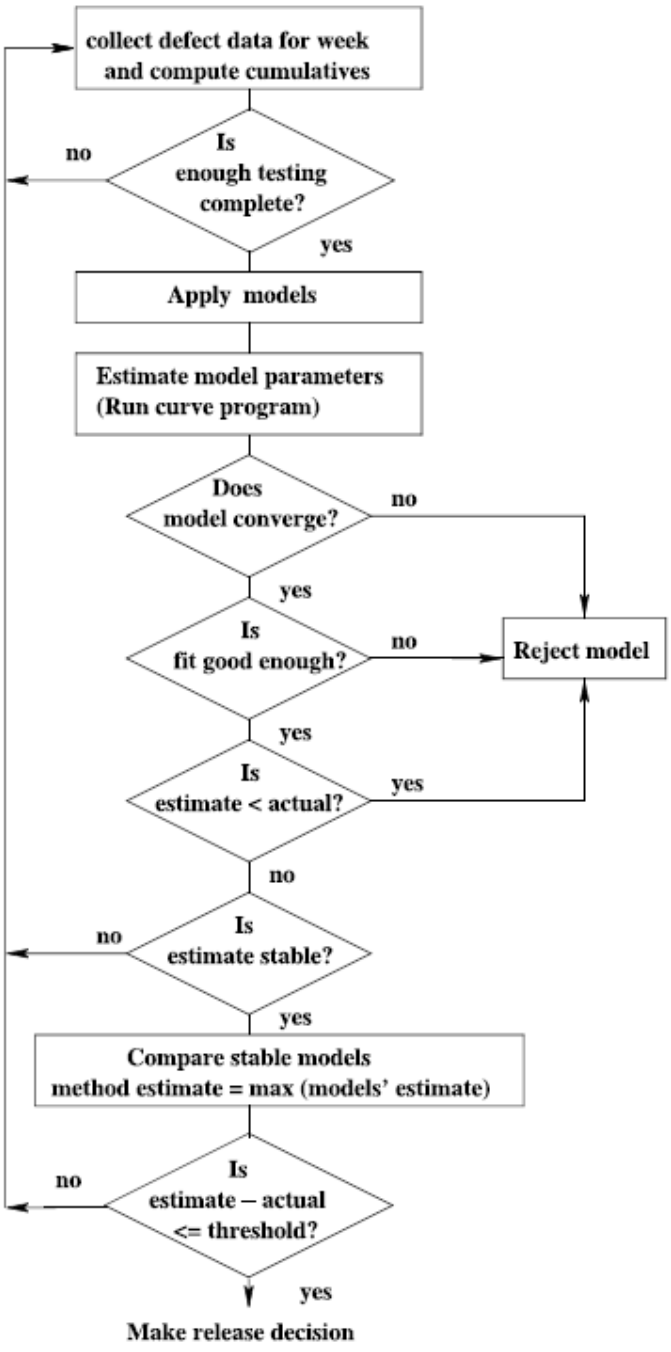

Figure 2.8. Model selection flowchart

Figure 2.8 shows the steps of the approach. The steps are described in more detail as follows [26]:

1. Record the cumulative number of failures found in system test at the end of each week.
2. Determine whether enough of the test plan has been executed to contemplate stopping testing based on applying a model. This requires setting a threshold that determines when to start applying the models to the defect data. It is advised that at least 60 per cent of planned testing must be completed before applying the SRGMs.
3. The curve fit program estimates a model's parameters by attempting to fit the model to the data.
4. If a fit cannot be performed, a model is said to diverge. This situation occurs if the model is not appropriate for the data or not enough data has been collected yet. If this situation occurs after the threshold set for model application, then the model is probably not appropriate for the data and should not be considered in future weeks. If none of the models converges, then this method fails and some other approach should be used to estimate quality or make release decisions. If a curve can be fit to the data for a model, GOF is evaluated based on the R^2 - value. The program also returns the estimate for the expected number of total failures.
5. Using the R^2-value as an indicator of GOF requires setting a threshold for the R-value that is good enough for the model to be selected as acceptable. What is considered a good fit is somewhat up to the user's judgment. Writer chooses a threshold of R = 0.95. This means that the R^2 value will be over 0.90.
6. The curve fit program provides an estimate for the parameter *a*, which is an estimate for the expected number of total failures. If a model's predictions for expected number of total failures are lower than the actual number of failures already found and have been consistently so in prior weeks, the model chosen is inappropriate for the data and should not be used in future weeks.
7. If no model has a stable prediction for the current week, that is within the stability threshold defined additional testing effort is required. System test continues and collects failure data for another week.
8. If there is at least one stable model, then the method estimate is taken to be the

maximum estimate of all stable models. System test determines whether additional testing effort is required by comparing the method estimate to the actual number of failures found. If the method estimate is much higher than the actual number of failures found, additional testing effort may be considered for at least another week. If the difference between the prediction and the actual number of failures found is lower than the acceptability threshold, the decision to stop testing may be considered. This acceptability threshold is set through subjective judgment.

9. System test should apply the models that were not eliminated in previous weeks at the end of system test to estimate the number of remaining failures that could be reported after release. The number of failures after release may be estimated by subtracting the number of known failures in system test from the predicted total number of failures.

2.8.3. An Approach for Software Reliability Model Selection

In this study, the software reliability models are classified as shown Figure 2.9 [27]. Using this classification, a software reliability model selection algorithm shown in Figure 2.10 [27] is proposed. The method identified nine criteria that can help in the process of software reliability model selection, namely; Life Cycle Phase, Output Desired By The User, Input Required By Model, Trend Exhibited by the Data, Validity Of Assumptions According To the Data, Nature Of Project, Structure Of Project, Test Process, Development Process.

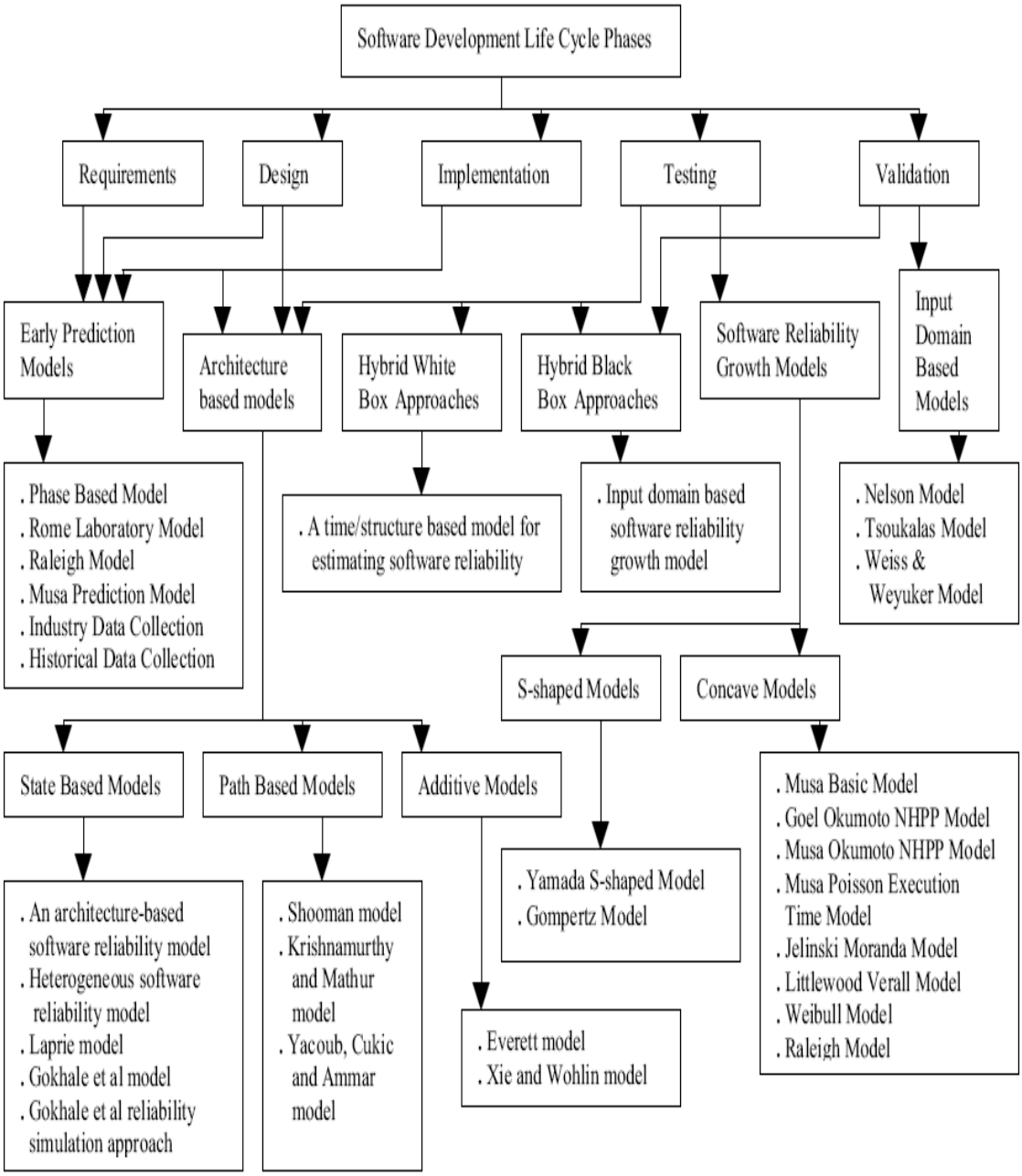

Figure 2.9. Classification of software reliability models

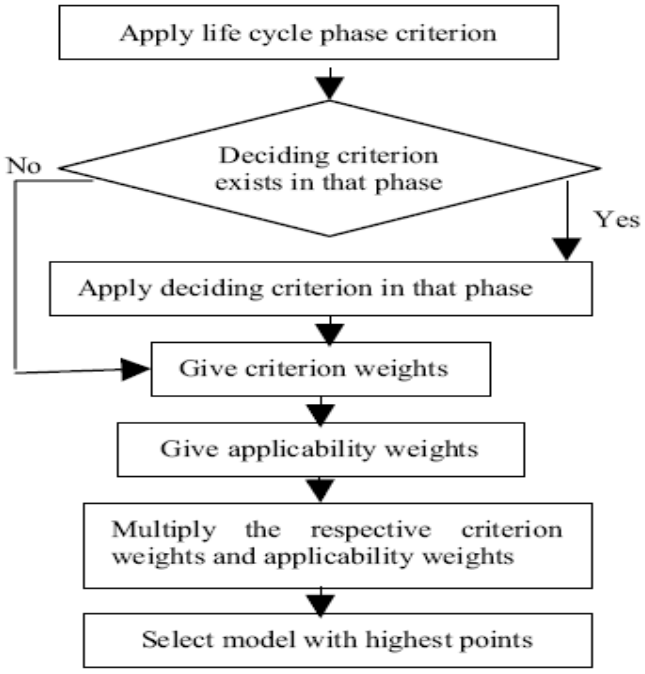

Figure 2.10. Model selection flow graph

3. AN ALGORITHM FOR SOFTWARE RELIABILITY GROWTH MODEL SELECTION

In this chapter, proposed algorithm is presented. The proposed algorithm takes advantages of the algorithms, named "The Method Of Software Reliability Growth Models Choice Using Assumptions Matrix" and "An Empirical Method For Selecting Software Reliability Growth Models" which were discussed in 2.8.1 and 2.8.2.

"An Empirical Method For Selecting Software Reliability Growth Models" algorithm provides guidelines on how to select among the SRGMs to decide on the best model to use as failures are reported during the test phase [26]. The method applies various SRGMs iteratively during system test. They are fitted to weekly cumulative failure data and used to estimate the expected remaining number of failures in software after release. If the SRGMs pass proposed criteria, they may then be used to make release decisions.

We found the approach of iteratively collecting data and evaluating the models useful. However, we believe that it would be better to replace some parts of this algorithm to improve the success of the algorithm. First of all, this model selects the initial set of models intuitively. In [26] four common SRGMs are selected since they comply with a range of assumptions. But intuitive model selection approach may not always select the correct set of models and may cause best models to never be evaluated. Therefore, we decided to use a more quantitative method to select initial set of models. Secondly, this method decides to keep or discard the models according to results of the goodness of fit tests. However, as discussed in [28], the goodness-of-fit test is not sensitive enough to make fine distinctions among models. Therefore, we decided to change selecting the best model approach. The aim of this method is to predict the number of remaining failures to help release decision. But some times, estimation of next failure of time after the release, for example to check whether the software is likely to survive a mission [29], is more important than the number of remaining failures. This information could be obtained from the Time Between Failures models. However, this method suggests using Failure Counts models. For this reason, we also decided to supply both of the TBF and FC models to have a more general algorithm. Lastly, we proposed a novel approach, which we called local

threshold, to help release decision. Local threshold is used to ensure that the best model, whose estimations will be used in release decision, is giving meaningful estimations locally.

"The Method Of Software Reliability Growth Models Choice Using Assumptions Matrix" algorithm offers to use assumptions matrix taking into account the features of software engineering and testing processes [25]. Assumptions matrix, which is given in Table 2.1, is a table in which rows correspond to various assumptions and columns to SRGMs [25].

This method is useful if the time to select a reliability model is limited and the reliability requirement is not so high. But to achieve a high reliability and find out a model, which represents the test data best, assumptions matrix alone is not enough. Because assumptions are often violated in practice [26]. Examples of common assumption violations are: performing functional testing rather than testing an operational profile, varying test effort due to holidays and vacations, imperfect repair and introduction of new errors, calendar time instead of execution time, defect reports instead of failure reports, partial or complete scrubbing of duplicate defect reports, and failure intervals that are not independent of each other, etc [26]. Also it is not reasonable to use some of the assumptions. For example, one cannot easily assume the number of failures in the system is finite or infinite. Although it has some disadvantages, assumptions matrix could be used in obtaining a model set by using only those valid assumptions for software under test.

By taking account into the above discussion, we've proposed a new Software Reliability Growth Model Selection Algorithm. In section 3.1 the modified assumptions matrix, which is used in the proposed algorithm, is discussed. In section 3.2, the proposed software reliability model selection algorithm is presented. Lastly in section 3.3, requirements to use the proposed algorithm is discussed.

3.1. Modified Assumptions Matrix

We use assumptions matrix [25] to choose initial set of reliability model/models. However, before using the assumptions matrix we had to modify it since some of the

models that are shown in Table 2.1 do not exist in the tool, CASRE that we are using in our studies. Moreover, some of the models that the tool supports do not exist in Table 2.1. Therefore, we rearrange the assumptions matrix to provide the models that the tool is supporting. The modified assumptions matrix is shown in Table 3.1.

We have also added new assumptions to the assumptions matrix and modified the models according to these new assumptions. New assumptions are listed below;

IV.2.2 Schneidewind Model Layer

IV.2.2.1 Combination of the first s-1 period is indicative of the failure rate process during the later stages. This model requires a cutoff interval, s, to be determined. (Schneidewind Cum 1st)

IV.2.2.1 Early time periods contribute little if anything in predicting future behavior. This assumption may be used for example to eliminate a learning curve effect by ignoring the first few time periods. This model requires a cutoff interval, s, to be determined. (Schneidewind Last n)

IV.5.3 Error corrections are made at the end of the testing intervals. However, errors discovered in one testing interval can be corrected in another interval. (Generalized Poisson Model)

V. Failure Data Format Layer

V.1 Failure data format is time between failures (T.B.F.).

V.2 Failure data format is failure counts (F.C.) per test interval.

We use the assumptions matrix to select a set of reliability models, which are suitable for the software under test. Then we find the best model with other methods discussed in later parts. We recommend to software practitioners use only those assumptions that they are certain about fitting their software. We believe that using all the assumptions, where some are not suitable for the software under concern, should not result in a single model selection. We show that using higher-level assumptions to select a set of models then selecting the best model with proposed algorithm is better than using assumption matrix alone.

We use the power of the assumptions matrix in narrowing the reliability model set to work with a small set of models. Without assumptions matrix approach one could decide to work with all of the reliability models, which is definitely not feasible due to having more than hundreds of models in the literature. Or one could be select the initial set of models intuitively, as in the case of [26]. But intuitive selection may result in best models not to be considered and never be tested.

Table 3.1. Modified Assumptions Matrix

Attributes				Finite								Infinite	
Category				Finite								Infinite	
Type				Binomial	Poisson				Binomial			Poisson	
				Exponential					Weibull & Gamma			Exponential	
Group				De- eutrophication	NHPP				S- Shaped	Weibull		Geometric	Logarithmic
Model				Jelinski Moranda	Goel Okumoto	Schneidewind Cum 1st	Schneidewind Last n	Musa Basic	Yamada S-Shaped	Schick - Wolverton	Generalized Poisson	1-st Moranda (Geometric Model)	Musa - Okumoto
				1.1	2.1	2.2.1	2.2.2	2.3	4.1	5.1	5.3	6.1	7.1
Assumptions				1	1	1	1	1	1	1	1	1	1
I	I.1												
	I.2			1	1	1	1	1	1	1		1	1
	I.3			1	1	1	1	1	1	1	1	1	1
	I.4			1	1	1	1	1	1	1	1	1	1
II	II.1	II.1.1		1	1	1	1	1	1				
		II.1.2										1	1
	II.2	II.2.1	II.2.1.1		1	1	1	1	1			1	1
		II.2.1	II.2.1.2		1	1	1	1	1			1	1
		II.2.1	II.2.1.3		1	1	1	1	1			1	1
		II.2.2	II.2.2.1	1						1	1		
		II.2.2	II.2.2.2	1						1	1		
	II.3	II.3.1		1	1	1	1	1				1	1
		II.3.2							1	1	1		
III	III.1			1									
	III.2				1	1	1	1					
	III.4								1				
	III.5									1	1		
	III.6											1	
	III.7												1
IV	IV.1	IV.1.1		1									
	IV.2	IV.2.1			1								
		IV.2.2	IV.2.2.1			1							
			IV.2.2.2				1						
		IV.2.3						1					
	IV.4	IV.4.1							1				
	IV.5	IV.5.1								1			
		IV.5.3									1		
	IV.6	IV.6.1										1	
	IV.7	IV.7.1											1
V	V.1			1	1			1				1	1
	V.2				1	1	1		1	1	1		

3.2. Proposed Algorithm

Proposed algorithm is mainly composed of five parts; choosing initial reliability model set, deciding stopping thresholds, collecting failure data, ranking models and

making release decision. These parts are described in the following sections. Then details of the algorithm are presented step by step.

3.2.1. Choosing Initial Reliability Growth Model Set

The modified assumptions matrix approach is used for selecting initial software reliability model set. Details of this approach are discussed in part 2.8.1 and 3.1. In this step, our main goal is reducing the number of models, which will be used later in the algorithm. Therefore we recommend using assumptions of layer I, II and III, or in other words SRGM layer, Category layer and group layer, which are described in 2.8.1 unless there is an obvious reason to use layer IV.

3.2.2. Deciding On the Thresholds Criteria

An important step in the software development is to decide when to stop testing. We proposed two thresholds to help release decision. First one is the release threshold. With this threshold one can conclude that reliability of the software under test is high enough for deployment. Second threshold is called local threshold. Local threshold reflects the idea that if most recently estimation is not satisfactory then one should not attempt to check the release decision and continue collecting test data. In some situations, although the release threshold is good enough, most recently estimation may be much different than the reality. In case of F.C. data difference between the most recently estimation and actual failure numbers can be too high. The difference may be due to several of reasons. For example, test team might be changed, new code might be deployed, environment might be changed, etc. Whatever the reason it may be good idea to continue testing until the model results are good enough for both local and release thresholds.

3.2.3. Collecting failure data

Before applying the reliability models we have to collect failure data. The studies in [30] and [31] indicates that the software reliability growth models typically do not become stable until 60 per cent of testing is complete. These studies suggested us to complete 60 per cent of testing until to apply the reliability growth models. Note that, this is the

minimum amount to wait. If there is no sensible way to stop testing before 90 per cent of the test plan is complete (e.g. when new functionality is tested), one would usually wait until that time before applying the models. [26]

3.2.4. Ranking Models

We applied the following procedure suggested in [28] to rank the models;

- Apply a goodness of fit test to determine if the model results fit the input data to a specified significance level.
- If a number of models pass the fitness test:
  - Choose the most appropriate model(s) based on the Prequential likelihood.
  - In the event of a tie, use the model bias, then model bias trend to break the tie.
- If only one model provides a good fit to the data then choose that model.
- If no models provide a good fit to the data:
  - Choose the most appropriate model(s) based on the Prequential likelihood.
  - In the event of a tie, use the model bias, then model bias trend to break the tie.

3.2.5. Making Release Decision

In section 3.2 we have proposed two thresholds, namely local and release thresholds. Before making the release decision, first we propose to check whether the local threshold is satisfied. Thus we compare the previous week predictions with actual collected data. If the difference between the first ranked models' prediction and the actual is larger than the local threshold, continue testing. Next we propose to check, whether the release threshold is satisfied. To do this, we compare the first ranked models' prediction and the actual results. If the difference is lower than the release threshold, then release may be decided. If the difference is higher than the threshold then the testing must continue.

3.2.6. Proposed Algorithm Details

Figure 3.1 shows the proposed software reliability selection algorithm. Below the steps of the algorithm are presented.

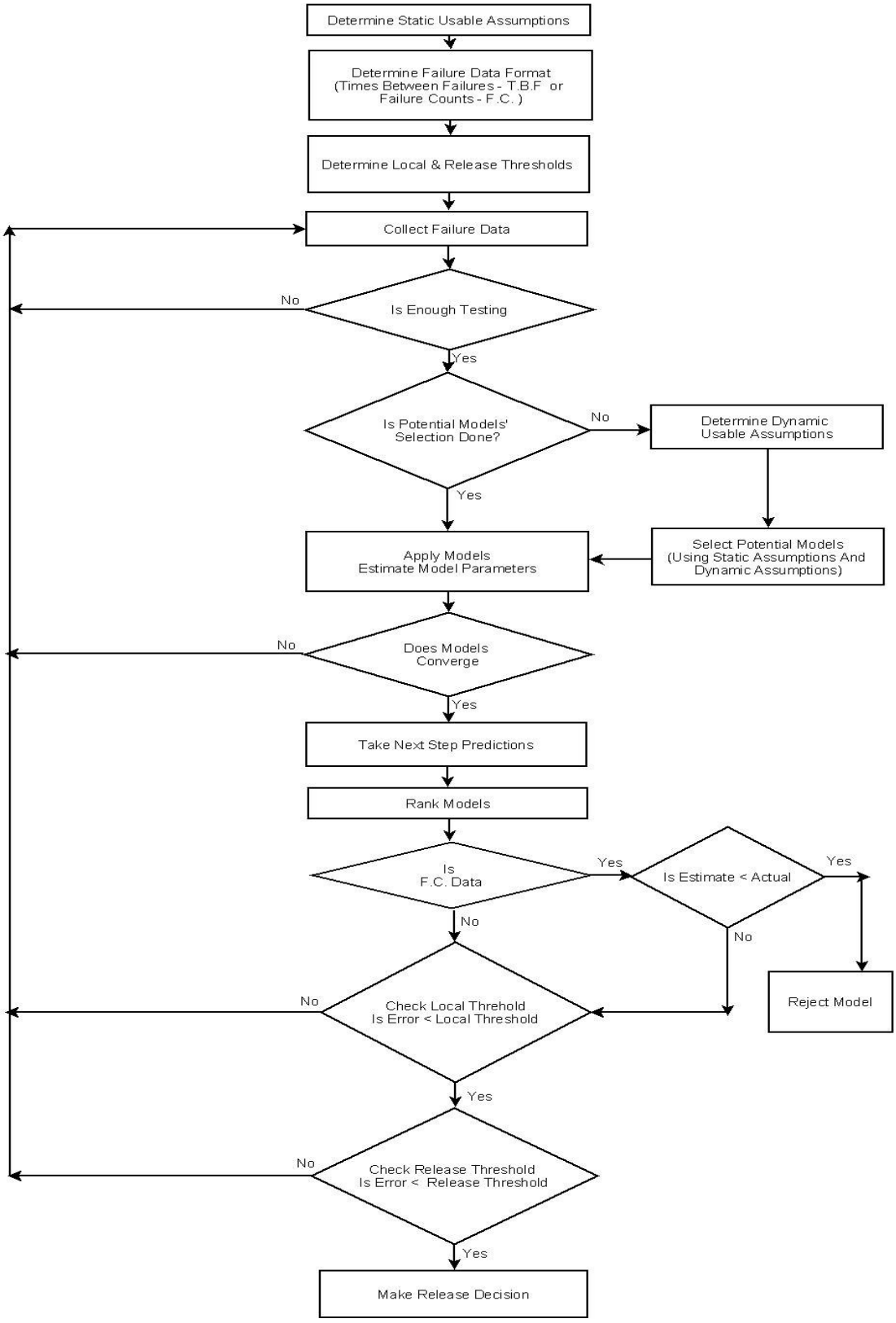

Figure 3.1. Proposed algorithm

3.2.6.1. Determine Static Usable Assumptions. Analyze the test and development environment. Then select the static usable assumptions, which are most suitable for the software environment. Among all the assumptions, which are listed in part 2.8.1 and 3.1, static assumptions are; I.1, I.2, I.3, I.4, II.1.1, II.1.2, III.3, IV.2.2, IV.2.3, IV.3.2, IV.4.2, IV.5.3, V.1 and V.2.

3.2.6.2. Determine Failure Data Format. Failure data can be in either Failure Counts (F.C) or Time Between Failures (T.B.F.) format. In F.C data format, failure counts per test intervals are recorded. F.C. software reliability models, give estimates on failure intensity; failure counts per intervals and total number of failures in the software. In T.B.F. data, times between successive failures are recorded. T.B.F. software reliability models, give estimates on failure intensity and failure times.

3.2.6.3. Determine Local & Release Threshold. Local threshold will be used before determining release threshold. The idea behind this threshold is, if the previous estimated value for the current time (number of failures for F.C. data, next time of the failure for T.B.F. data) is not satisfactory then do not test for the release threshold and continue with testing. Release threshold will be used in stopping tests and making the release decision. For F.C. data, this threshold may be the difference between the predicted and the actual total number of failures found. For T.B.F. data, time to next failure after release may be used. Note that, failure intensity objective may also be used both for F.C and T.B.F. data.

3.2.6.4. Collect Failure Data. Record the failure counts per test interval for F.C. data or times between failures for T.B.F. data.

3.2.6.5. Is Enough Testing To Apply Models. Determine if the collected data is enough to apply software reliability models. The studies in [30] and [31] indicates that the software reliability growth models typically do not become stable until 60 per cent of testing is complete. This is the minimum amount of time to collect data. If there is an obvious reason, as mentioned in 3.2.3, one can continue collecting data before applying the models.

3.2.6.6. Initial Model Selection Already Done?. Check whether the potential model selection done before. If not; then continue with steps 3.2.6.7 and 3.2.6.8 else continue with step 3.2.6.9.

3.2.6.7. Determine Dynamic Usable Assumptions. Analyze the collected data and select the suitable dynamic assumptions. Among all the assumptions, which are listed in part 2.8.1, dynamic assumptions are; II.2.1, II.2.1.1, II.2.1.2, II.2.1.3, II.2.2, II.2.2.1, II.2.2.2, II.3.1, II.3.2, III.1, III.4, III.6, III.7, IV.1.2, IV.4.3, IV.4.4, IV.6.2, IV.6.3. To determine suitable dynamic assumptions, we've used a data analysis tool. We tried to fit the collected data to a known distribution, such as Binomial or Poisson distribution.

3.2.6.8. Select Potential Models. By using the determined static and dynamic assumptions, select the reliability growth model/models from the modified assumptions matrix showed in Table 3.1.

3.2.6.9. Apply Models, Estimate Model Parameters. Estimate the model parameters and evaluate the models on the test data. In this study, we use a CASE tool, which is discussed in Appendix B, to determine model parameters and evaluate models. In the estimation of model parameters, we use Maximum Likelihood method for T.B.F. data and Least Squares method for F.C. data. Note that, one can use other methods such as regression methods [26] in parameter estimation. We use the data collected between the beginning of the testing and the time at which the potential model selection done to estimate model parameters.

3.2.6.10. Do Models Converge. Continue to collect failure data unless at least one model converges. Sometimes due to the collected data's nature, model parameters cannot be obtained and models diverge. If all the selected models are diverge then continue to collect data. Note that, in order to use reliability growth models, test data must have a reliability growth characteristic. That is, for T.B.F. data, on the average time between successive failures should be increasing and for the F.C. data on the average failures per intervals should be decreasing. If the test data does not exhibit reliability growth then possibly models would not converged.

3.2.6.11. Take Next Predictions. In this step, calculate the next step predictions of the models. For F.C. data, this is the estimated number of failures in the next test interval. For T.B.F. data this is the estimated time of the next failure.

3.2.6.12. Rank Models. Rank models as described in part 3.2.4.

3.2.6.13. Is F.C. Data. If the failure data is of F.C. format, then apply step 3.6.1.14 else continue with step 3.6.1.15.

3.2.6.14. Is Estimate < Actual. Check whether the models' estimate is lower then the actual total failure. If any of the selected reliability models' estimated number of the total failures less than the actual; then model is underestimating the remaining number of failures and give a false sense of security. Then we eliminate this model and continue with other models.

3.2.6.15. Check Local Threshold. The estimation for the current failure/interval was calculated in the previous step. Now, compare the first ranked models previous estimations with actual collected data. For F.C. data, compare estimated number of failures with actual failures found. Whereas, in T.B.F. data compare estimated time to next failure with the actual time to failure. If the error is less than the local threshold then apply step 3.2.1.16 else continue testing.

3.2.6.16. Check Release Threshold. A release threshold was set at the beginning. Now compare the first ranked models' estimation and the actual collected data. For F.C. data, compare the estimated total number of failures with the total number of failures discovered so far. Whereas, in T.B.F. data check the estimated next time of failure after the release. If the difference is lower than the release threshold, then release may be decided. If the difference is higher than the threshold then the testing must continue.

3.2.6.17. Make Release Decision. At this step we know that both the local and release thresholds are satisfied. With this information in hand, the release may be decided by the test manager.

3.3. Using The Proposed Algorithm

The proposed algorithm can be used in a wide range of software environments, which have different reliability requirements. Reliability requirements could be estimating number of remaining failures, time to next failure or achieving a specified failure intensity level.

If the reliability objective is to estimate the number of the remaining failures, then one can use the failure counts data format. Then next step prediction will give the estimated number of failures in the next test interval. Difference between the estimated number of failures and actual number of failures found in the current test interval will be compared with the local threshold. If the local threshold is satisfied then release threshold will be checked. If the first ranked models' estimation on the number of remaining failures is less than the release threshold, then release can be decided.

If the reliability objective is to estimate the next failure time after the release, then one can use time between failures data format. Then next step prediction will give the estimated time of the next failure. Difference between the estimated time of the failure and the actual recorded time will be compared with the local threshold. If the local threshold is satisfied then release threshold will be checked. If the first ranked models estimation on the next failure time after the release is higher than the release threshold, then can be decided.

In some cases, estimating the remaining number of failures and next time of failure after the release may both be important. Also, achieving a specified failure intensity level could be the reliability objective. In such cases, failure counts and time between failures data formats could be used together. However, using both data formats will be resulted in the synchronization problems, since F.C. models evaluated at the end of the intervals and T.B.F. models evaluated after each failure occurrences. In order to overcome the synchronization problems; F.C. and T.B.F. models may be evaluated in parallel at the end of the intervals. For the T.B.F. case, error in the next step predictions could be found by taking the average of the differences between the estimated next failures times and actual times of the failures. So, for T.B.F. data local threshold would be compared with the average error in the previous test interval. Then, both first ranked F.C. models' estimate on

the number of remaining failures and first ranked T.B.F. models' estimate on the next failure time must satisfy the respective reliability objectives. In the case, where the reliability objective is to achieve a specified failure intensity level, again F.C. and T.B.F. data could be used together. Then, average intensity of the first ranked F.C. model and first ranked T.B.F. model could be used as the estimated failure intensity of the software.

4. PERFORMANCE STUDY

In this chapter, test results of the proposed algorithm are presented. Proposed algorithm is tested with various data sets, which were collected from different software environments and publicly available. Both F.C. and T.B.F. data formats are used in the tests.

In T.B.F. tests; results are presented in a table, whose first column is the failure number and second column is the time passed from the last failure. Then for each reliability model; rank of the model, estimated time of the next failure, difference between the estimated times and the actual failure times are presented respectively.

In F.C. tests; results are given in a table, whose first column is the test interval, second column is the number of failures found in that interval and third column is the cumulative number of failures found by that interval. Then for each reliability model; estimated total number of failures, rank of the model, estimated number of failures for the next interval, difference between the estimated number of failures and the actual failures found in the current interval are presented respectively.

As the proposed algorithm suggests, static and dynamic assumptions are used to choose an initial set of software reliability models. In the determination of the static assumptions, only those valid assumptions for software under test are used. To obtain suitable dynamic assumptions, a data analysis tool is used to determine type of the data sets, which have either Poisson or Binomial distribution.

Modified assumptions matrix in Table 3.1 is used to select initial set of reliability models. In Schneidewind models, a detailed knowledge about the software is required to determine the cutoff interval. Since we don't have such detailed information on the public data sets, Schneidewind models are not included in the tests although they exist in Table 3.1. Schneidewind models are finite failure category, Poisson type, Exponential class and NHPP group models. Since the Goel-Okumoto model is also a finite failure category, Poisson type, Exponential class and NHPP group model, it is used instead of the

Schneidewind models. Also, in our tests Generalized Poisson and Schick-Wolverton models give identical results. Since they exist in the same category, type, class and group we included only one of them in the test results.

Kolmogorov-Smirnov GOF test is used for T.B.F. data, whereas Chi-Square GOF test is used for F.C. data. The threshold for Kolmogorov-Smirnov test is set to five per cent. In the Chi-Square test, significance level is set to five per cent and cell combination frequency is set to five, where cell combination is the number of degrees of freedom minus the number of parameters in the model. We've set the parameters of these tests subjectively. In [26] the threshold for the GOF is set to 0.90 and it is accepted as a very high confidence level. The thresholds for the GOF tests can be varied for different software environments, and one can choose other values.

In the following section, the proposed algorithm is tested with a Real Time Command & Control Application data set. Then, in section 4.2 it is tested with a Large Medical Record System Data Set.

4.1. Real Time Command & Control Application

We have chosen a data set from a Real Time Command & Control application named SYS1 from [32]. The application has 21700 delivered object code instructions and 136 failures were recorded in the test phases. This data set is also used in [25] to test the assumptions matrix approach discussed in 2.8.1. Same dataset is used, in order to compare the results of proposed algorithm with the results of the assumptions matrix approach.

We have tested this data set with both F.C. and T.B.F. formats. The results are presented in the following sections.

4.1.1. T.B.F. Test

According to [25], the below assumptions are valid for this data set.

- I.1, I.2, I.3 and I.4 of the first layer assumptions.
- II.1.1, II.2.1.1, II.2.1.2, II.2.1.3 and II.3.1 of the second layer assumptions.

First four assumptions are general assumptions that are applicable to all of the models. II.1.1 assumes that the total number of failures is finite. We believe that assuming the number of failures in the system is whether finite or infinite, is not reasonable. So we've decided not to use this assumption. Assumptions II.2.1.1 through III.3.1 are applicable for the Poisson type and Exponential class models. Besides those assumptions, since the failure format is T.B.F. type, V.1 assumption is also valid for this data set. Therefore, the assumptions that are valid for this data set are:

- I.1, I.2, I.3 and I.4 of the first layer assumptions.
- II.2.1.1, II.2.1.2, II.2.1.3 and II.3.1 of the second layer assumptions.
- V.1 of the data format category.

Due to the above assumptions; Goel-Okumoto NHPP, Musa Basic, Geometric and Musa-Okumoto models are chosen from Table 3.1 as the initial set of models. Since the proposed algorithm suggests to apply models after 60 per cent of the testing completed, we have applied the models after the failure number 110.

Table 4.1 shows the test results and indicates that for most of the failure numbers, except for the failure number 112, ranking of the models are the same. Musa-Okumoto, Geometric, Musa Basic and NHPP models are ranked in the given order from first to last. From Table 4.1, it can be observed that Musa-Okumoto model underestimates time between the successive failures three times at failure numbers; 123, 129 and 133. Whereas, Geometric model underestimates time between the successive failures six times and Musa Basic and Goel-Okumoto NHPP models five times. System test stopped after the failure number 136. Musa-Okumoto model, first ranked model for the failure number 136, predicts that the next failure will occur 5789 seconds later after the release. Whereas, Geometric, Musa Basic and NHPP models predict as 2192, 4249 and 4591 seconds respectively. We don't know the exact time of the failure after the release but we know from [32] that there were no failures 2526 seconds later after the release. Therefore the second ranked model, which is Geometric model, is underestimated the next failure time.

Table 4.1. Test results

Failure Number	Seconds Last Failure	Musa-Okumoto			Geometric			Musa Basic			NHPP (TBE)		
		Rank	Next Step Prediction	Next Step Prediction - Actual	Rank	Next Step Prediction	Next Step Prediction - Actual	Rank	Next Step Prediction	Next Step Prediction - Actual	Rank	Next Step Prediction	Next Step Prediction - Actual
111	729	1	3184	-	2	1354	-	3	1936	-	4	2033	-
112	1897	2	3365	1287	1	1409	-543	3	2109	39	4	2225	136
113	447	1	3329	2918	2	1407	962	3	2029	1662	4	2132	1778
114	386	1	3284	2943	2	1403	1021	3	1945	1643	4	2037	1746
115	446	1	3250	2838	2	1400	957	3	1877	1499	4	1959	1591
116	122	1	3169	3128	2	1384	1278	3	1767	1755	4	1838	1837
117	990	1	3211	2179	2	1402	394	3	1787	777	4	1857	848
118	948	1	3246	2263	2	1417	454	3	1799	839	4	1091	909
119	1082	1	3298	2164	2	1437	335	3	1828	717	4	1899	9
120	22	1	3204	3276	2	1417	1415	3	1714	1806	4	1774	1877
121	75	1	3120	3129	2	1398	1342	3	1618	1639	4	1669	1699
122	482	1	3091	2638	2	1395	916	3	1577	1136	4	1624	1187
123	5509	1	3735	-2418	2	1576	-4114	3	2198	-3932	4	2299	-3885
124	100	1	3646	3635	2	1559	1476	3	2069	2098	4	2155	2199
125	10	1	3546	3636	2	1538	1549	3	1940	2059	4	2012	2145
126	1071	1	3588	2475	2	1556	467	3	1958	869	4	2031	941
127	371	1	3537	3217	2	1548	1185	3	1886	1587	4	1951	1660
128	790	1	3541	2747	2	1554	758	3	1870	1096	4	1932	1161
129	6150	1	4247	-2609	2	1748	-4596	3	2598	-4280	4	2728	-4218
130	3321	1	4581	926	2	1840	-1573	3	2966	-723	4	3139	-593
131	1045	1	4607	3536	2	1856	795	3	2940	1921	4	3106	2094
132	648	1	4579	3959	2	1858	1208	3	2850	2292	4	3001	2458
133	5485	1	5198	-906	2	2019	-3627	3	3623	-2635	4	3879	-2484
134	1160	1	5231	4038	2	2038	859	3	3588	2463	4	3832	2719
135	1864	1	5357	3367	2	2080	174	3	3688	1724	4	3942	1968
136	**4116**	**1**	**5789**	**1241**	**2**	**2192**	**-2036**	**3**	**4249**	**-428**	**4**	**4591**	**-174**

This data set was also used in [25], where Musa Basic model was selected from the assumptions matrix. In our test, Musa Basic model is always ranked third after the Musa-Okumoto and Geometric models. From Table 3.1, these three models are Poisson Type and Exponential Class models. The difference is that, Musa Basic model assumes finite number of failures whereas other two models assume infinite number of failures. We've recommended to the practitioners not to use the assumption on the number of failures unless there is an obvious reason. And the test shows that, not using the number of failures assumptions resulted in more accurate predictions.

Figure 4.1 shows the plot of the cumulative failures of actual data and the first ranked model, which is Musa-Okumoto model. Visual inspection indicates that, Musa-Okumoto model fits the cumulative failures of actual data well, especially at the beginning and end of the tests.

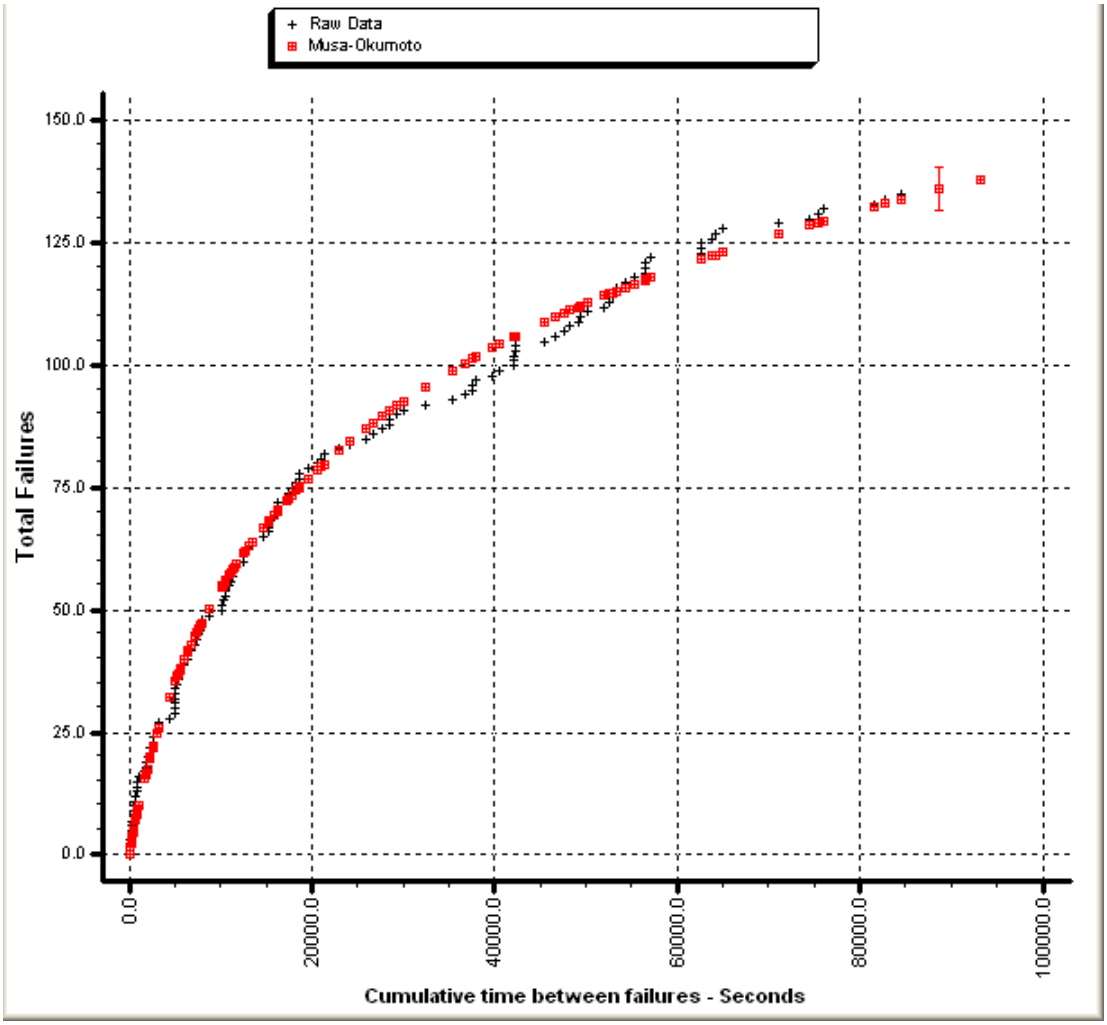

Figure 4.1. Cumulative failures of actual data and Musa-Okumoto model

4.1.2. F.C. Test

In order to use this data set with the proposed algorithm for F.C. data, we have grouped the data in 5000 seconds intervals, which resulted in 18 test intervals. Then, we have applied the models after the interval 10 since the proposed algorithm suggests applying models after 60 per cent of the testing completed.

We've identified the below assumptions as valid for this data set.

- I.1, I.2, I.3 and I.4 of the first layer assumptions.
- V.2 of the data format category.

First four assumptions are general assumptions that are applicable to all of the models. Assumption V.2 is applicable since the failure format is F.C type. We have analyzed the data after the week 10 to determine whether it has Poisson or Binomial distribution. But, we couldn't fit the data to a Poisson or Binomial distribution. Therefore, we have chosen all of the F.C. models from Table 3.1, namely; Goel-Okumoto NHPP, Yamada S-Shaped and Schick-Wolverton models.

Table 4.2 shows the test results and points out that Schick-Wolverton model is always ranked as the first model. NHPP model is ranked as second if it passes GOF test. Yamada S-Shaped model never passes the GOF test so it doesn't participated in the ranking. In interval 11, Schick-Wolverton model predicts the total number of failures as 130, which is 11 per cent higher than the actual cumulative number of failures. Also it predicts two failures in the next interval. In interval 12, Schick-Wolverton model predicts total number of failures as 135, which is again 11 per cent higher than the actual. The error in the interval failure estimation is -3. In interval 13 and 14, Schick-Wolverton model predicts the total number of failures as 144 and 137, which are 13 per cent and 7 per cent higher than the actual, respectively. The errors in the interval failure estimation are –4 in the interval 13 and two in the interval 14. In interval 15, first ranked models' error in the failures per interval is zero and the predicted total number of failures is 138 which is six per cent more than the actual. If the release threshold is higher than eight failures then release decision can be made since the estimated number of remaining failures is eight. In interval 17, Schick-Wolverton model predicts the total number of failures as 142, which is again five per cent higher than the actual. The error in the interval failure estimation is –2. In interval 18, first ranked models' error in the failures per interval is zero and the predicted total number of failures 142 which is 4 per cent higher than the actual. If the release threshold is higher than six, then release decision can be made since the predicted number of remaining failures is six.

Note that, in intervals 14,15,16 and 17, NHPP model could not be ranked since it does not succeed on the GOF test. However, in interval 18 it passes the GOF test and ranked second. If we were eliminating the models due to not converging or passing the GOF test as in [26]; we would not be able to take the results of the eliminated models later. But as shown in Table 4.2, NHPP model gives successful estimates. It predicts the number

of failures per interval correctly in three out of seven intervals, which are intervals 15, 16 and 17. Also it predicts the total number of failures just one higher than the first ranked model at release time. Therefore, these results are justified our decision that is based on not eliminating the models, which do not converge or pass GOF tests in the early test intervals. Because, these models might converge, or pass GOF tests later and might give successful results.

Table 4.2. F.C. test results

Test Interval	Failure Per Interval	Total Failures	Schick-Wolverton				Goel-Okumoto NHPP				Yamada S-Shaped			
			Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual
11	7	117	130	1	2	-	132	2	3	-	120	-	1	-
12	5	122	135	1	2	-3	137	2	3	-2	125	-	1	-4
13	6	128	144	1	2	-4	146	2	3	-3	131	-	1	-5
14	0	128	137	1	2	2	140	-	2	3	130	-	1	1
15	2	130	138	1	1	0	140	-	2	0	132	-	1	-1
16	2	132	139	1	1	-1	141	-	1	0	133	-	0	-1
17	3	135	142	1	1	-2	143	-	1	-2	136	-	0	-3
18	1	136	142	1	1	0	143	2	1	0	137	-	0	-1

Figure 4.2 displays the cumulative failures plot of actual data and Schick-Wolverton model.

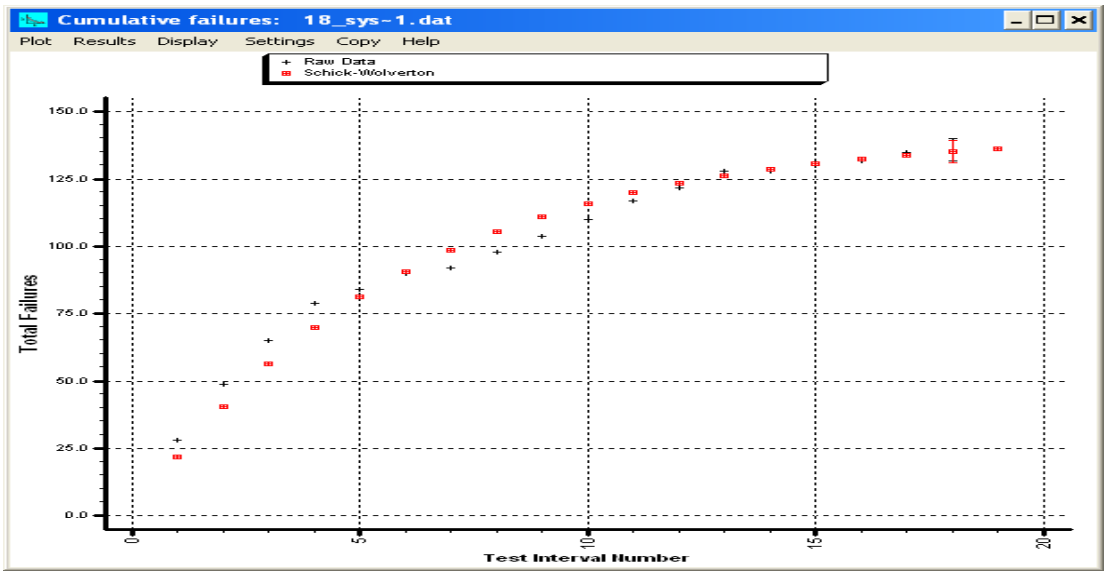

Figure 4.2. Cumulative failures of actual data and Schick-Wolverton model

4.2. Large Medical Record System Data Set

The next data set contains three release results of a large medical record system, which consists of 188 software components. The failure data of the medical system is shown in Table 4.3. Initially, the software was composed of 173 components and 15 components were added to the software after the three releases [26]. However, many of the other components were modified in all three releases as a side effect of the added functionality.

Table 4.3. Failure data

	Failures Per Interval			Total Failures		
Test Interval	Release 1	Release 2	Release 3	Release 1	Release 2	Release 3
1	28	90	9	28	90	9
2	1	17	5	29	107	14
3	0	19	7	29	126	21
4	0	19	7	29	145	28
5	0	26	25	29	171	53
6	8	17	3	37	188	56
7	26	1	2	63	189	58
8	29	1	5	92	190	63
9	24	0	7	116	190	70
10	9	0	5	125	190	75
11	14	2	1	139	192	76
12	13	0	0	152	192	76
13	12	0	1	164	192	77
14	0	0		164	192	
15	1	11		165	203	
16	3	0		168	203	
17	2	1		170	204	
18	6			176		
Post Release				231	245	83

We have started to apply models after the interval 10 in Release 1 and 2 and after the interval 7 in Release 3, since the proposed algorithm suggests applying models after 60 per cent of the testing completed.

4.2.1. Release 1

We've identified the below assumptions as valid for this data set for F.C. data format.

- I.1, I.2, I.3 and I.4 of the first layer assumptions.

- II.2.1.1, II.2.1.2, II.2.1.3 of the second layer assumptions.
- V.2 of the data format category.

First four assumptions are general assumptions that are applicable to all of the models. Assumption V.2 is applicable since the failure format is F.C type. We have analyzed the data after the week 10 to determine whether it has Poisson or Binomial distribution. We found that, the data does not fit to the Binomial distribution and fits to the Poisson distribution with λ=6.3281. Therefore, assumptions II.2.1.1, II.2.1.2 and II.2.1.3 are applicable for this data set. Figure 4.3 shows the Cumulative distribution of the actual data and estimated Poisson distribution.

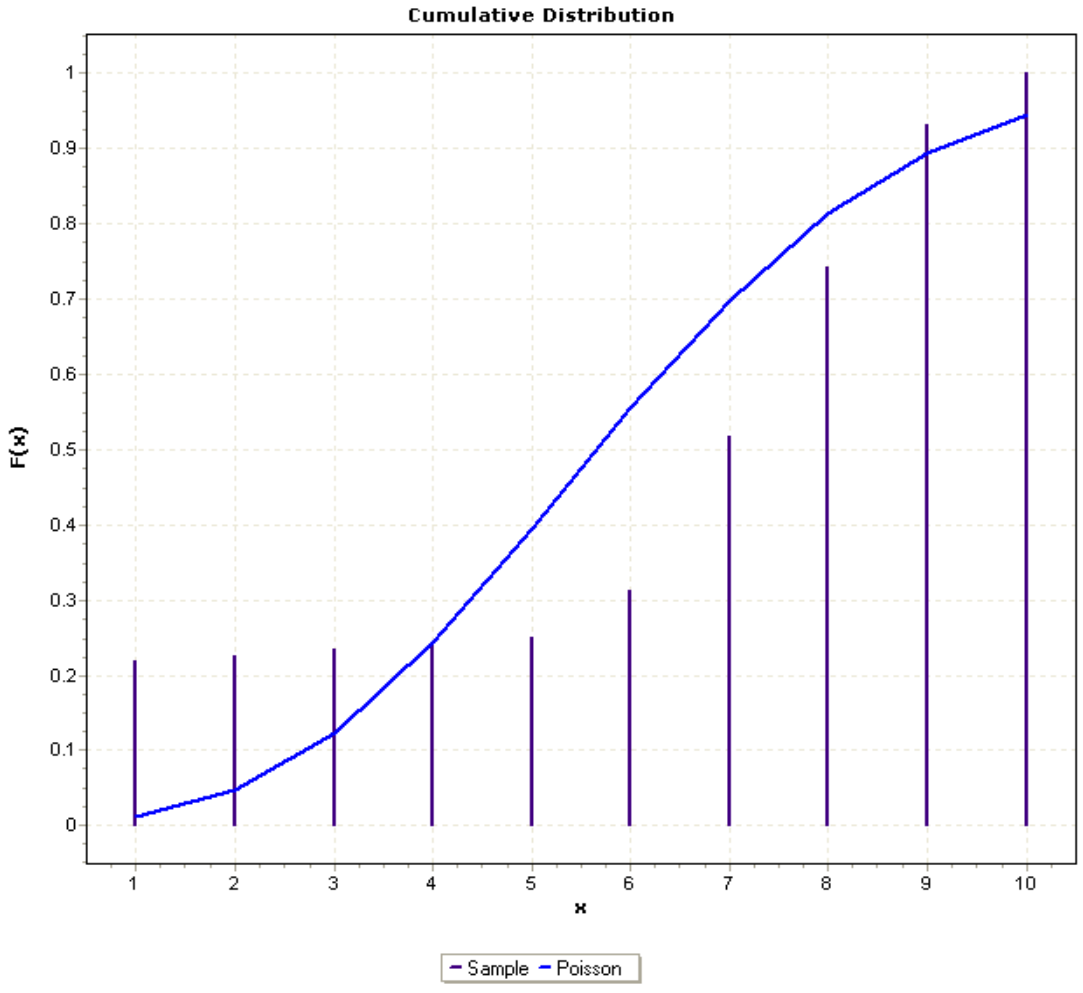

Figure 4.3. Cumulative distribution of the actual data and estimated Poisson distribution

Due to above assumptions; Goel-Okumoto and Yamada S-Shaped models are applicable according to Table 3.1. We have also included the Generalized Poison model, which is a Binomial type model, to test the validity of our assumptions.

Table 4.4 presents the results of Release1 test. In this release, Yamada S-Shaped model is always ranked as the first model. Generalized Poisson and Goel-Okumoto models are started to give estimations in week 14 and 15 respectively. However, they couldn't succeed on calculating the Prequential likelihood function and couldn't be take part in ranking. In week 11, Yamada S-Shaped model predicts 14 failures as the next week prediction and total number of failures as 329. The estimated number of total failures is 140 per cent more than the actual number, which is 139. Due to this huge difference it might be reasonable to continue testing. In week 14, Yamada S-Shaped model predicts the number of failures for the next week as 8 and total number of failures as 235. Note that for this week, the difference between the actual weekly failures and estimated weekly failure is 12. Since the model estimation on weekly failure number is not reasonable, it might be convenient to continue testing. In the following weeks for the Yamada S-Shaped model, differences in the weekly failure numbers are 7, 3, 3 and predicted numbers of remaining failures are 44, 32 and 24. This clearly showed a reliability growth as the system test continued. In week 18 first ranked models' error in weekly failures is only –2, which shows the first ranked model gives reasonable local estimations. However, first ranked models' estimation on the total number of failures is 199, which was 13 per cent higher than the actual number. Since the algorithm suggests that both local and release thresholds have to be satisfied, it might be reasonable to continue testing. Despite the big difference between the estimated total number of failures and the actual failures found, tests were stopped after week 18.

Table 4.4. Release 1

Test Week	Weekly Failure	Total Failures	Generalized Poisson				Goel-Okumoto NHPP				Yamada S-Shaped			
			Estimate	Rank	Next Step Prediction	Actual - Next Step Prediction	Estimate	Rank	Next Step Prediction	Actual - Next Step Prediction	Estimate	Rank	Next Step Prediction	Actual - Next Step Prediction
11	14	139	-	-	-	-	-	-	-	-	329	1	14	-
12	13	152	-	-	-	-	-	-	-	-	313	1	13	1
13	12	164	-	-	-	-	-	-	-	-	303	1	12	1
14	0	164	1054	-	11	-	-	-	-	-	235	1	8	12
15	1	165	392	-	8	10	911	-	10	-	209	1	6	7
16	3	168	308	-	7	5	460	-	8	7	200	1	5	3
17	2	170	264	-	5	5	334	-	7	6	194	1	4	3
18	6	176	268	-	5	-1	325	-	6	1	199	1	3	-2

Figure 4.4 displays the cumulative failure plot of actual data and Yamada S-Shaped model. Because visual inspection of the cumulative failure curve in Release 1 indicated that it was more S-shaped than concave [26], it was expected the S-shaped models perform better. The proposed algorithm selected the Yamada S-Shaped model as the best model, which verified the visual inspection also. Moreover, we have expected that Generalized Poisson model, should behave worse that other models since it is a Binomial type model. The results justified our assumption and Generalized Poisson didn't converge in first couple of weeks and still could not be included in ranking because other ranking criteria are not satisfied.

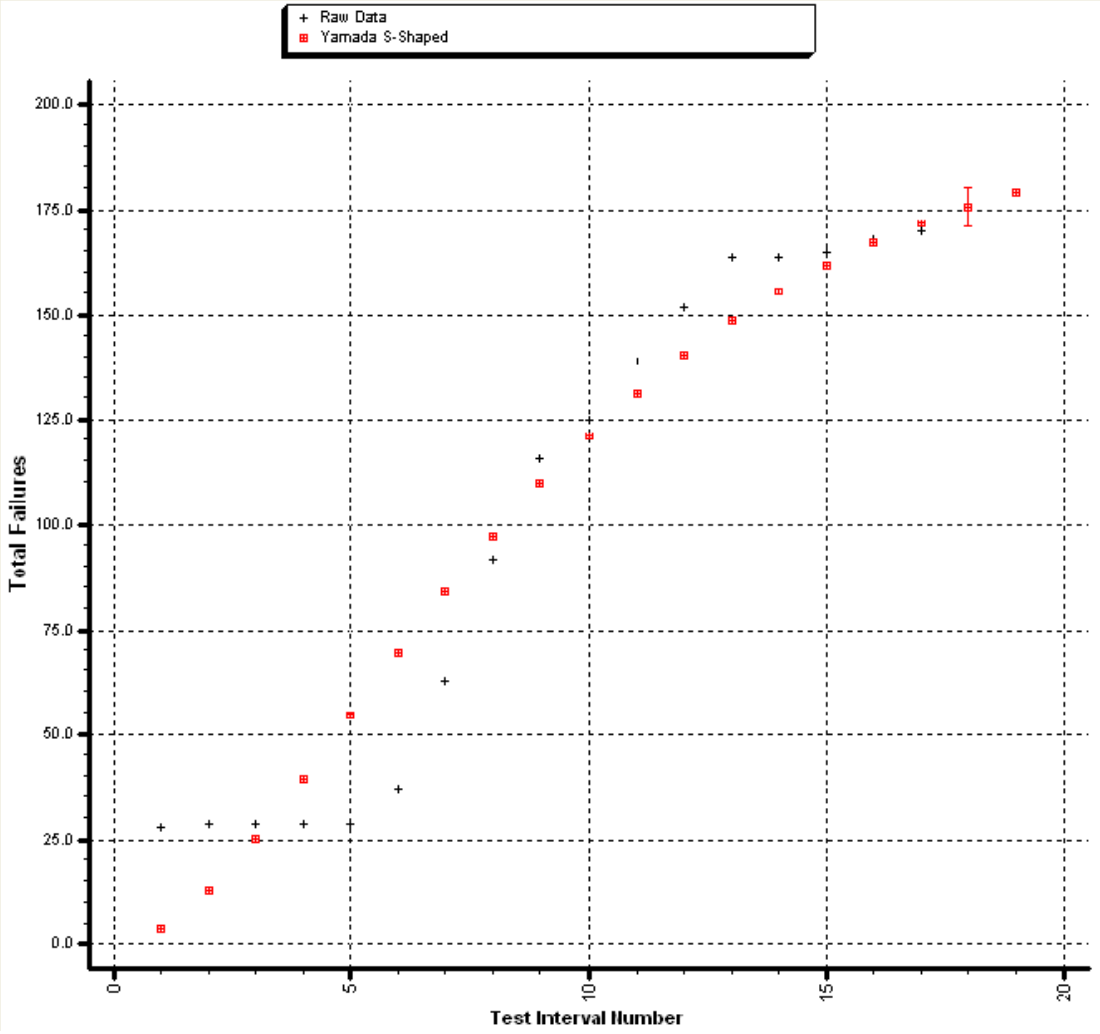

Figure 4.4. Cumulative failures of Release 1 data and Yamada S-Shaped model

From Table 4.3, we can see that total number of failures found in the post-release is 231, which is more than the predicted number, 199. This means, Yamada S-Shaped model has underestimated the total number of failures. However, note that in the release week, prediction of the Yamada S-Shaped model was 13 per cent higher than the actual found failures number, which was 176. Due to this excessive difference between the predicted and the actual number of failures, testing should probably be continued. According to [26], this opinion is supported by one of the testers who believed that too many defects were found after release 1.

4.2.2. Release 2

We've identified the below assumptions as valid for this data set for F.C. data format.

- I.1, I.2, I.3 and I.4 of the first layer assumptions.
- II.2.1.1, II.2.1.2, II.2.1.3 of the second layer assumptions.
- V.2 of the data format category.

First four assumptions are general assumptions that are applicable to all of the models. Assumption V.2 is applicable since the failure format is F.C type. We have analyzed the data after the week 10 to determine whether it has Poisson or Binomial distribution. We found that, the data does not fit to a Binomial distribution and fits to the Poisson distribution with λ=2.724. Therefore, assumptions II.2.1.1, II.2.1.2 and II.2.1.3 are applicable for this data set. Figure 4.5 shows the Cumulative distribution of the actual data and estimated Poisson distribution.

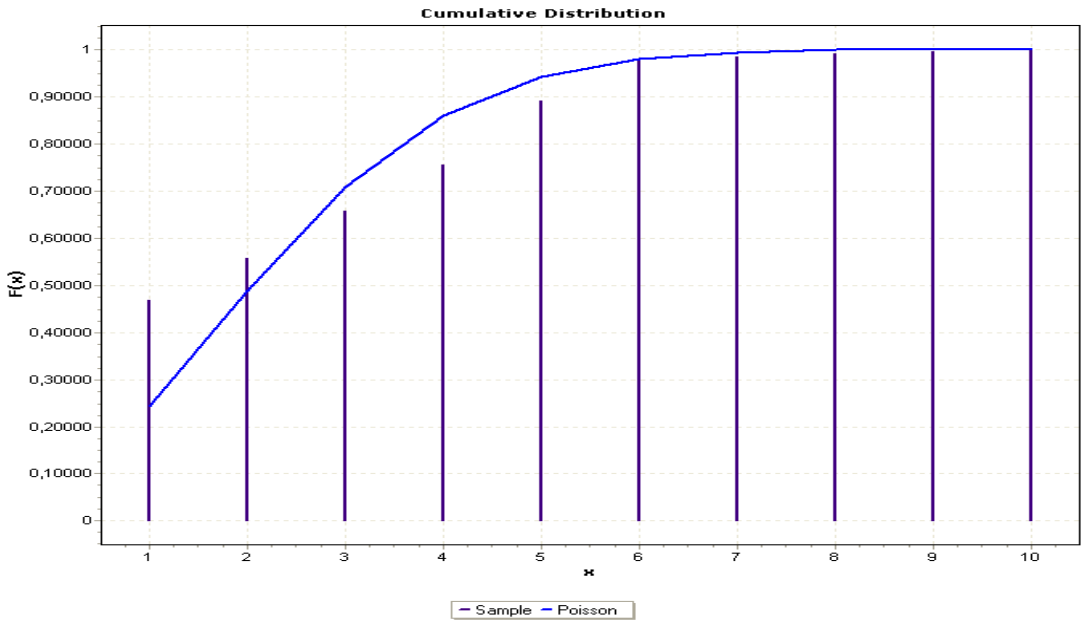

Figure 4.5. Cumulative distribution of the actual data and estimated Poisson distribution

Due to above assumptions; Goel-Okumoto and Yamada S-Shaped models are applicable according to Table 3.1. We have also included the Generalized Poisson model, which is a Binomial type model, to test the validity of our assumptions.

Table 4.5 displays the results of the Release 2 tests. In week 11, Generalized Poisson model is ranked as first. This model predicts the number of remaining failures and the number of failures for the next week as zero. But, after the week 11 this model does not converge again. Between the weeks 12 and 14 Yamada S-Shaped model is ranked as first and NHPP model ranked as second. In these test weeks, no actual failures were recorded. Also, model estimations on weekly number of failures are reasonable. After the week 14, NHPP model is always ranked first and Yamada S-Shaped model is ranked second. In week 15, errors in weekly failure estimations are –11 for both models. So it is not a good idea to stop testing in week 15 since the local threshold might not be satisfied. Release decision is given in week 17. At that time, first ranked model predicts the number of remaining failures as just 1 and the error in the weekly failures is 1.

Table 4.5. Release 2

Test Week	Weekly Failure	Total Failures	Generalized Poisson				Goel-Okumoto NHPP				Yamada S-Shaped			
			Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual
11	0	192	192	1	0	-	193	2	1	-	192	3	0	-
12	0	192	-	-	-	-	192	2	0	1	192	1	0	0
13	0	192	-	-	-	-	192	2	0	0	192	1	0	0
14	0	192	-	-	-	-	192	2	0	0	192	1	0	0
15	11	203	-	-	-	-	204	1	0	-11	203	2	0	-11
16	0	203	-	-	-	-	204	1	0	0	203	2	0	0
17	1	204	-	-	-	-	205	1	0	-1	204	2	0	-1

Figure 4.6 displays the cumulative failure plot of actual data and Goel-Okumoto (NHPP intervals) model. Because visual inspection of the cumulative failure curve in Release 1 indicated that it was more concave than S-Shaped [26], it was expected the Goel-Okumoto model to perform better. Proposed algorithm selected the Goel-Okumoto model as the best model, which verified the visual inspection also. Also, we have expected that Generalized Poisson model, should behave worse than other models since it is a Binomial type model. The results are justified our assumption and Generalized Poisson didn't converge after the week 11.

From Table 4.3, total number of failures found in the post-release is 245, which is more than the predicted number, 205. This means, Goel-Okumoto NHPP model is underestimated the total number of failures and failed to give accurate predictions. However, if we inspect the Table 4.5, we can easily conclude that first ranked model gives successful weekly predictions. Also note that, between the weeks 11 and 17, only 12 failures were found, where 11 of them were found week 15. Number of failures found in week 15 and after the release suggested that software testing might be influenced from something. This might be a change in testing environment, a change in test team members or developed code might be changed substantially.

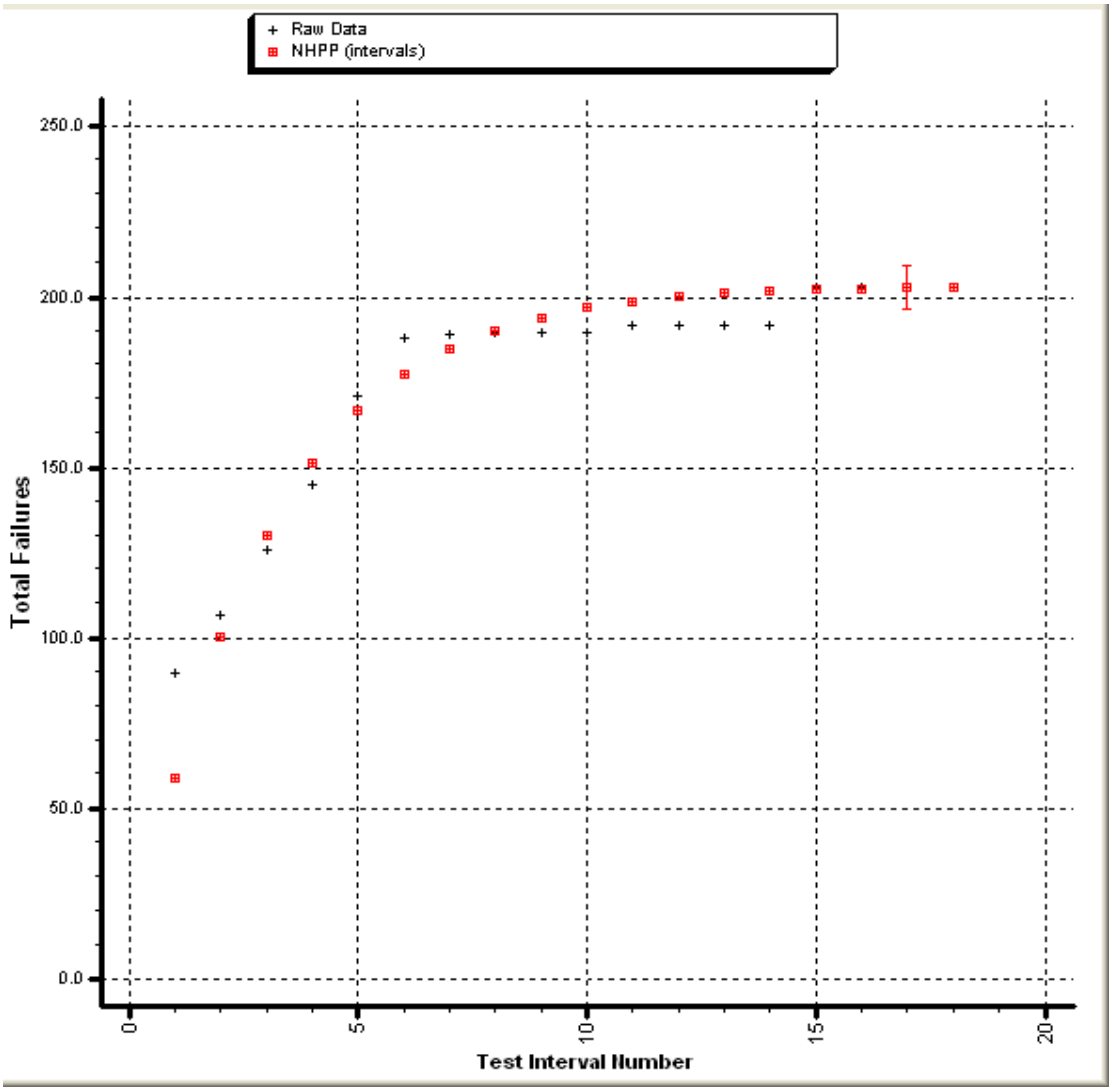

Figure 4.6. Cumulative failures of Release 2 data and NHPP model

4.2.3. Release 3

We've identified the below assumptions as valid for this data set for F.C. data format.

- I.1, I.2, I.3 and I.4 of the first layer assumptions.
- II.2.1.1, II.2.1.2, II.2.1.3 of the second layer assumptions.
- V.2 of the data format category.

First four assumptions are general assumptions that are applicable to all of the models. Assumption V.2 is applicable since the failure format is F.C type. We have analyzed the data after the week 10 to determine whether it has Poisson or Binomial distribution. We found that, the test data fit to the Binomial distribution with n=13 and p=0,27931. Also the test data fit to the Poisson distribution with λ=3.87930. We have identified that Poisson distribution fits the data better by using Kolmogrov-Smirnov goodness of fit test. Therefore, assumptions II.2.1.1, II.2.1.2 and II.2.1.3 are applicable for this data set. Figure 4.7 and 4.8 show the Cumulative distribution of the actual data and estimated Poisson and Binomial distributions respectively.

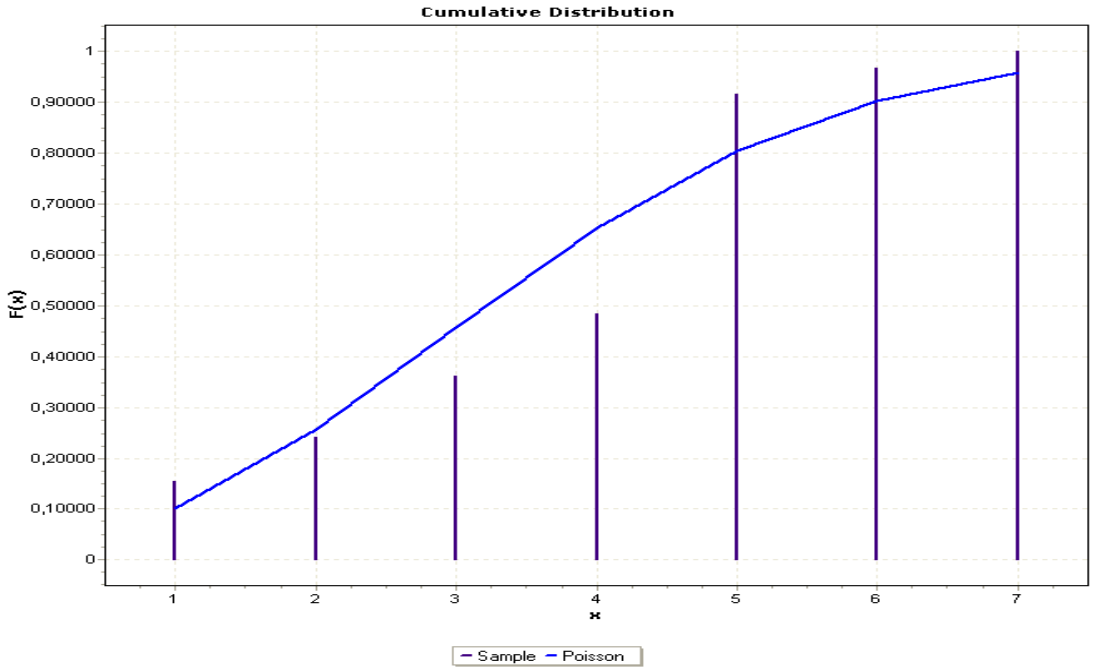

Figure 4.7. Cumulative distribution of the actual data and estimated Poisson distribution

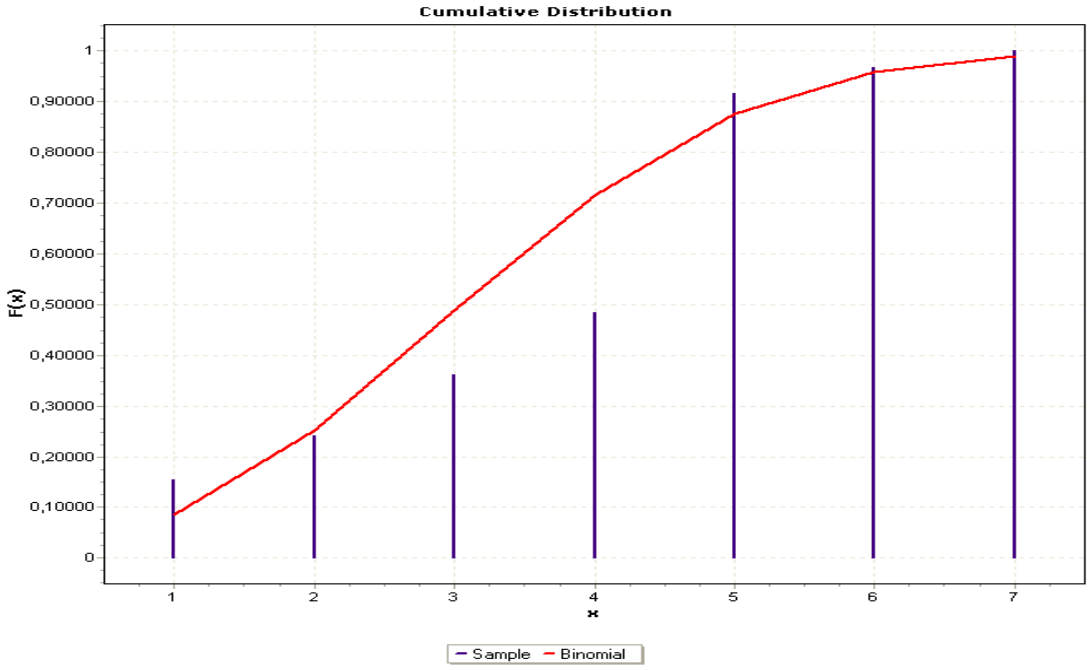

Figure 4.8. Cumulative distribution of the actual data and estimated Binomial distribution

Due to above assumptions; Goel-Okumoto and Yamada S-Shaped models are applicable according to Table 3.1. We have also included the Generalized Poison model, which is a Binomial type model, to test the validity of our assumptions.

Table 4.9 displays the results of the Release 3 tests. Yamada S-Shaped, Generalized Poisson and NHPP models are ranked as first, second and third in all the test weeks, respectively except the weeks 9 and 10. In week 9 and 10; NHPP, Generalized Poisson and Yamada S-Shaped models are ranked in the given order. In week 8 Yamada S-Shaped model predicts the number of failures for the next week as three and total number of failures as 75. The estimated number of total failures is 18 per cent more than the actual number, which is 63. In weeks 9 and 10 NHPP model predicts the total number of failures as 201 and 174 respectively. The estimated numbers of total failures are 187 and 132 per cent more than the actual numbers, which are 70 and 75. In weeks 11 and 12 Yamada S-Shaped model predicts the total number of failures as 83 and 80 respectively. The estimated numbers of total failures are 9 and 5 per cent more than the actual numbers, which are both 76. Release decision is given in week 13. Weekly error in week 13 of the first ranked model is zero, which clearly satisfies the local threshold. First ranked model, which is Yamada S-Shaped model, predicts the number of remaining failures as just three.

Table 4.6. Release 3

Test Week	Weekly Failure	Total Failures	Yamada S-Shaped				Generalized Poisson				Goel-Okumoto NHPP			
			Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual	Estimate	Rank	Next Step Prediction	Next Step Prediction - Actual
8	5	63	75	1	3	-	93	2	4	-	174	3	6	-
9	7	70	84	3	3	-4	117	2	4	-3	201	1	6	-1
10	5	75	87	3	3	-2	122	2	4	-1	174	1	5	1
11	1	76	83	1	2	2	101	2	3	3	121	3	4	4
12	0	76	80	1	1	2	89	2	2	3	99	3	3	4
13	1	77	80	1	1	0	87	2	1	1	93	3	2	2

Figure 4.9 displays the cumulative failure plot of actual data and Yamada S-Shaped model. Visual inspection of the cumulative failures curve shows that the selected model fits to the actual data well. Also note that, from Table 4.3, we can see that total number of failures found in the post-release is 83, which is slightly more than the predicted number.

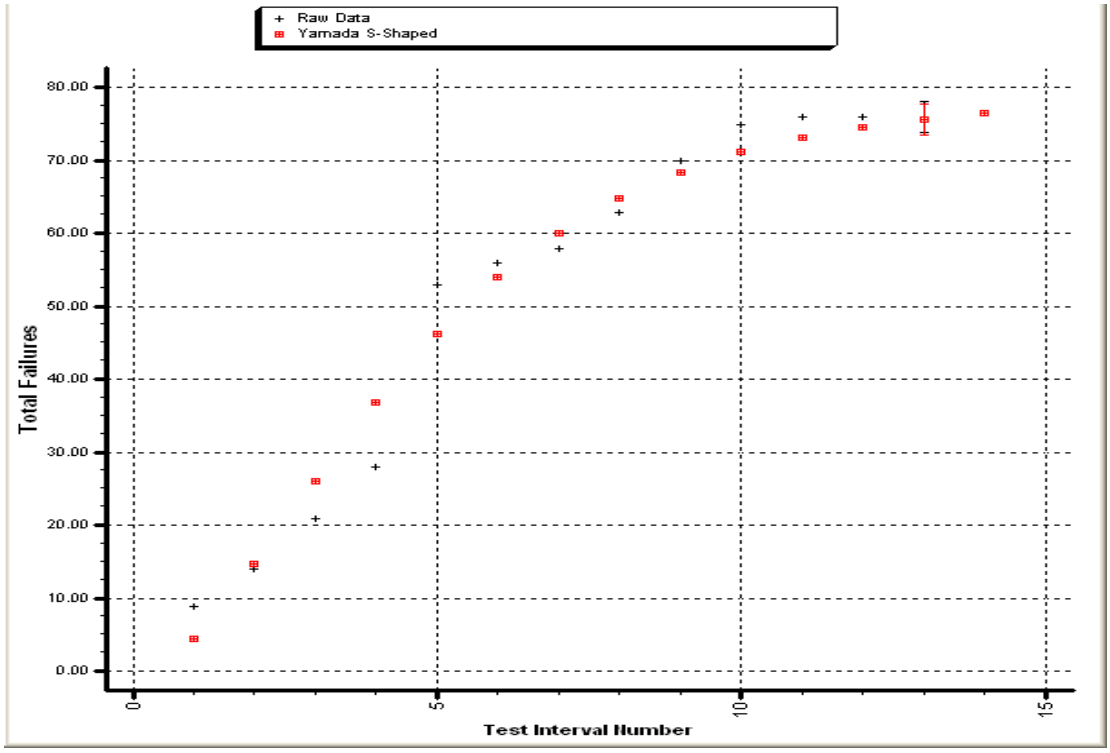

Figure 4.9. Cumulative failures of Release 3 data and Yamada S-Shaped model

5. CONCLUSION

Although there have been more than hundred models proposed in the literature, there does not exist a software reliability model that can be used in all software environments. Also, a reliable guideline does not exist for the potential users to select the best model for their environments. Therefore, there is a need for a software reliability model selection algorithm that can be used in all environments. Moreover, the required selection algorithm might be used not only by the software reliability professionals but also software practitioners who have not detailed technical background on software reliability theory.

In this study, we have proposed an algorithm for software reliability growth model selection, which is easily applicable. The proposed algorithm takes advantages of the algorithms, namely "The Method Of Software Reliability Growth Models Choice Using Assumptions Matrix" [25] and "An Empirical Method For Selecting Software Reliability Growth Models" [26]. We have eliminated the weaknesses of these methods and combined the strengths of these algorithms to introduce a new, but powerful algorithm. We have also added some novel approaches in the proposed algorithm. The proposed algorithm is tested with publicly available data sets, which were also used in [25] and [26].

Software environments may have different reliability objectives, such as estimating number of remaining failures, achieving a specified failure intensity level or estimating time to next failure. The method proposed in [25] does not discuss how this method could be used to achieve a reliability objective, where the method discussed in [26] proposes to use the number of remaining failures for determining the release decision. In the proposed algorithm we've supported all of the three software reliability objectives discussed above to have a more general algorithm.

The objective of estimating the number of remaining failures can be obtained from the Failure Counts models and the objective of the estimating the next failure times can be obtained from Time Between Failures models. If the reliability objective is to estimate the next failure time after the release then one can use the proposed algorithm with time between failures data, whereas if the reliability objective is to estimate number of

remaining failures then one can use the proposed algorithm with failure counts data. In the case of achieving a specified failure intensity level for the reliability objective, the proposed algorithm can be used either or both with T.B.F. and F.C. data.

Tests demonstrated that the proposed algorithm is more successful on selecting the best model than using the assumptions matrix alone, which is proposed in [25]. Also, tests showed that it is not reasonable to use some of the assumptions. For example, one cannot easily assume the number of failures in the system is finite or infinite. Therefore, we recommend to the practitioners to use only those assumptions, which can be shown as valid. We've categorized the assumptions as static and dynamic assumptions. Static assumptions can be decided before the tests, whereas dynamic assumptions can be decided after collecting and analyzing some part of the failure data. We showed that the dynamic assumptions resulted in more accurate software reliability models selections than the model proposed in [25]. Assumptions matrix method alone is useful if the reliability model selection time is limited, software statistical failure information is poor and the reliability objective is not so high. But to achieve a high reliability and find out a model, which fits to the test data best, the proposed algorithm should be considered.

The initial set of reliability models is selected intuitively in [26]. However, intuitive model selection approach may not always select the correct set of models and may cause best models not to be evaluated. Therefore, we've proposed a more quantitative method to select initial set of models by using the assumptions matrix. We've tested the proposed algorithm with same data sets used in [26] to compare the results of the algorithms. Although, we've expected that the proposed algorithm would produce better estimations, we observed that both algorithms' made close predictions. In the tests, we've used a CASE tool, named CASRE, which includes a few but widely known models for parameter estimation and model evaluation. The method discussed in [26] has also used same widely known models, which is the reason for close predictions. We found out that, testing the proposed algorithm in CASRE limits the prediction performance of the proposed model. We believe that starting with a large number of models would be resulted in better estimations. Therefore as a future work, we aimed to extend the assumptions matrix and develop a new Software Reliability Tool, which includes all the models in the extended assumptions matrix, in order to improve the proposed algorithm selection performance.

The method proposed in [26] decides to keep or discard the models according to results of the GOF tests. Then from the remaining models it selects the most pessimistic one, which predicts the maximum number of remaining failures. However, as discussed in [28], the GOF test is not sensitive enough to make fine distinctions among models. Therefore, we've changed the approach of selecting the best model and proposed the usage of the ranking algorithm discussed in [28]. Also, the method proposed in [26] eliminates the models in case they do not converge or pass the GOF tests. In our proposed algorithm such models are not discarded immediately but kept and evaluated instead. Because, in our tests we've observed that although some models may diverge or not pass GOF test in early test intervals, they might converge or pass the GOF tests later and give successful estimations.

Also, we propose a novel approach, which is called local threshold, to help the release decision. Local threshold is used before applying the release threshold and ensures that the best model gives meaningful local estimations. It suggests that, if the previously estimated value (number of failures for F.C. data, next time of the failure for T.B.F. data) for the current interval/instance is not satisfactory then one should not test for the release threshold and continue testing. In some cases we found that, although the release threshold is good enough, the most recent estimation is much different than the actual results. For example in case of F.C. data, the difference between the estimated number of failures per interval and actual failures in that interval can be too high. The difference may be due to several reasons. For example, test team might be changed, new code might be deployed or environment might be changed, etc. Whatever the reason is, it may be a good idea to continue testing until the model results are satisfactory for both local and release thresholds. Test results showed that checking the local threshold before the release threshold helps to make more accurate release decisions.

For the end user, using the proposed algorithm helps to analyze, manage and improve the reliability of the software products. With the correctly decided reliability objectives, one can determine when the software is good enough to release and also minimizes the risks of deploying the software with serious failures. Moreover, by using the proposed algorithm one can avoid excessive testing time and release the product at the correct time with the required reliability.

In conclusion, we have proposed an algorithm for software reliability growth model selection, which takes advantages of the two software reliability model selection algorithms, proposed in [25] and [26]. The main contributions of the proposed algorithm are that, it eliminates the weaknesses and combines the strengths of those algorithms to introduce a new, but powerful algorithm. Moreover, the proposed algorithm contains novel approaches for software reliability model selection.

APPENDIX A: RANDOM VARIABLES

A.1. Random Variables

Let S be a sample space for a random experiment. Then X(ξ) is a random variable, which is a single-valued real function that assigns a real number called the value of X(ξ) to each sample point ξ of S. Generally a single letter X is used in place of X(ξ) and r.v. is used to denote the random variable. Informally, a random variable can be defined as a quantity whose values are random and which has assigned probability distributions.

The domain of the r.v. X is the sample space S and the range of the r.v. X is the collection of all numbers, that is values of X(ξ). Thus the range of X is a certain subset of the set of all real numbers (Fig. A.1) [20].

Although more than one different sample points could give the same value of X(ξ), no two different numbers in the range can be assigned to the same sample point.

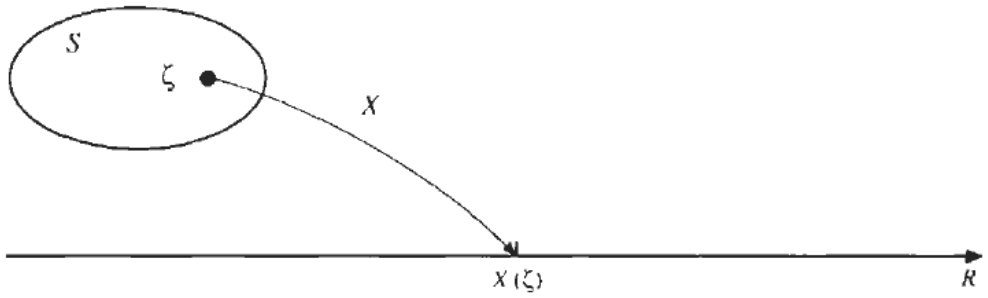

Figure A.1. Random variable X as a function

Let X be a r.v. and x be a fixed real number, then the event (X = x) can be defined as

$$(X = x) = \{\xi: X(\xi) = x) \tag{A.1}$$

Also, for fixed numbers x, $x_1$and x_2, the following events can be defined:

$$(X \le x) = \{\xi: X(\xi) \le x)$$
$$(X > x) = \{\xi: X(\xi) > x) \qquad (A.2)$$
$$(x_1 < X \le x_2) = \{\xi: x_1 < X(\xi) \le x_2)$$

These events, defined in A.2, have probabilities that are expressed by

$$P(X = x) = P\{\xi: X(\xi) = x\}$$
$$P(X \le x) = P(\xi: X(\xi) \le x\} \qquad (A.3)$$
$$P(X > x) = P\{\xi: X(\xi) > x)$$
$$P(x_1 < X \le x_2) = P\{\xi: x_1 < X(\xi) \le x_2)$$

A.2. Distribution Functions

The distribution function, or cumulative distribution function (cdf), is a function giving the probability that the random variable X is less than or equal to x. It is defined as;

$$F_x(x) = P(X \le x) \qquad (A.4)$$

$F_x(x)$ is an important function, since it is used in discovering lots of information about the random experiments defined by the random variables. Properties of the cumulative distribution function, $F_x(x)$, are;

1. $0 \le F_x(x) \le 1$ (A.5)
2. $F_x(x_1) \le F_x(x_2)$ if $x_1 \le x_2$ (A.6)
3. $\lim_{x \to \infty} F_x(x) = F_x(\infty) = 1$ (A.7)
4. $\lim_{x \to -\infty} F_x(x) = F_x(-\infty) = 0$ (A.8)
5. $\lim_{x \to a+} F_x(x) = F_x(a+) = F_x(a)$ $\qquad a+ = \lim_{0<\epsilon \to 0} a + \epsilon$ (A.9)

A.3. Discrete Random Variables And Probability Mass Functions

Let X be a r.v. with cdf $F_x(x)$. If $F_x(x)$ changes values only in jumps (at most a countable number of them) and is constant between jumps-that is, $F_x(x)$ is a staircase function then X is called a discrete random variable[20]. Alternatively, X is a discrete r.v. only if its range contains a finite or countably infinite number of points [20].

A probability mass function (pmf) is a function, which gives the probability that a random variable is exactly equal to some point. It is related with the cumulative distribution function $F_x(x)$ as follows;

Let the points $x_1, x_2, \ldots$ be a sequence, which is finite or countably infinite, and jumps of $F_x(x)$ occurs in this sequence. Then we have;

$$F_x(x_i) - F_x(x_{i-1}) = P(X \le x_i) - P(X \le x_{i-1}) = P(X = x_i) \quad \text{where } x_i < x_j \text{ if } i < j \qquad (A.10)$$

$$\text{Let} \qquad p_x(x) = P(X = x) \qquad (A.11)$$

The function $p_x(x)$ is called the probability mass function (pmf) of the discrete r.v. X. Properties of the probability mass function are;

$$1.\ 0 \le p_x(x_k) \le 1 \qquad k = 1,2,\ldots \qquad (A.12)$$

$$2.\ p_x(x) = 0 \qquad \text{if } x \ne x_k \quad k = 1,2,\ldots \qquad (A.13)$$

$$3.\ \sum_k p_x(x_k) = 1 \qquad (A.14)$$

The cdf $F_X(x)$ of a discrete r.v. X is related with the pmf as following;

$$F_x(x) = P(X \le x) = \sum_{x_k \le x} p_x(x_k) \qquad (A.15)$$

A.4. Continuous Random Variables And Probability Density Functions

Let X be a r.v. with cdf $F_X(x)$. If $F_X(x)$ is continuous and also has a derivative $dF_X(x)/dx$ which exists everywhere except at possibly a finite number of points and is piecewise continuous, then X is called a continuous random variable[20]. Alternatively, X is a continuous r.v. only if its range contains an interval (either finite or infinite) of real numbers[20]. Thus, if X is a. continuous r.v., then

$$P(X = x) = 0 \tag{A.16}$$

The function $f_x(x)$ is called the probability density function (pdf) of the continuous r.v. X and defined as;

$$fx(x) = dFX(x)/dx \tag{A.17}$$

Properties of the probability density function are;

1. $f_x(x) \geq 0$ (A.18)

2. $\int_{-\infty}^{\infty} f_x(x)\, dx = 1$ (A.19)

3. $f_x(x)$ is piecewise continuous.

4. $P(a < X \leq b) = \int_{a}^{b} f_x(x)\, dx$ (A.20)

Cumulative distribution function, $F_x(x)$, of a continuous r.v. X can be obtained from the probability density function, $f_x(x)$, as following;

$$FX(x) = P(X \leq x) = \int_{-\infty}^{x} fx(\xi)\, d\xi \tag{A.21}$$

If *X* is a continuous r.v., then from (A.16) we have;

$$P(a < X \le b) = P(a \le X \le b) = P(a \le X < b) = P(a < X < b)$$

$$= P(a < X < b) = \int_a^b f_x(x)\,dx = F_X(b) - F_X(a) \qquad \text{(A.22)}$$

A.5. Mean, Moment and Variance

Mean is the expected value a random variable, which is also called the population mean. It is denoted by μ_x or E(X) and defined by

$$\mu_x = E(X) = \begin{cases} \sum_k x_k p_x(x_k) & \text{X: discrete} \\ \int_{-\infty}^{\infty} x f_x(x)dx & \text{X: continuous} \end{cases} \qquad \text{(A.23)}$$

The *n*th moment of a r.v. *X* is defined by;

$$E(X^n) = \begin{cases} \sum_k x^n{}_k p_x(x_k) & \text{X: discrete} \\ \int_{-\infty}^{\infty} x^n f_x(x)dx & \text{X: continuous} \end{cases} \qquad \text{(A.24)}$$

The variance of a r.v. X is a measure indicating how its possible values are spread around the expected value. It is denoted by σ_x^2 or *Var(X),* is defined by;

$$\sigma_x^2 = \mathrm{Var}(X) = E\{[X - E(X)]^2\} \qquad \text{(A.25)}$$

$$\sigma_x^2 = \begin{cases} \sum_k (x_k - \mu_x)^2\, p_x(x_k) & \text{X: discrete} \\ \int_{-\infty}^{\infty} (x_k - \mu_x)^2\, f_x(x)dx & \text{X: continuous} \end{cases} \qquad \text{(A.26)}$$

The standard deviation of a r.v. X, denoted by σ_x, is the positive square root of Var(X). From (A.25), one can obtain the following relation:

$$\mathrm{Var}(X) = E(X)^2 - [E(X)]^2 \tag{A.27}$$

A.6. Some Special Distributions

A.6.1. Binomial Distribution

A random variable is called a binomial random variable with parameters (n, p) if its probability mass function is given by [20];

$$p_x(k) = P(X=k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{A.28}$$

where $k = 0,1,\ldots, n$ and $0 \le p \le 1$ and $\binom{n}{k} = n! / (k! (n-k)!)$. Then the corresponding cumulative distribution function of X is given as;

$$F_x(x) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \qquad n \le x < n+1 \tag{A.29}$$

Figure A.2 and A.3 illustrate the examples of probability mass and cumulative distribution functions, where n=6 and p = 0.6

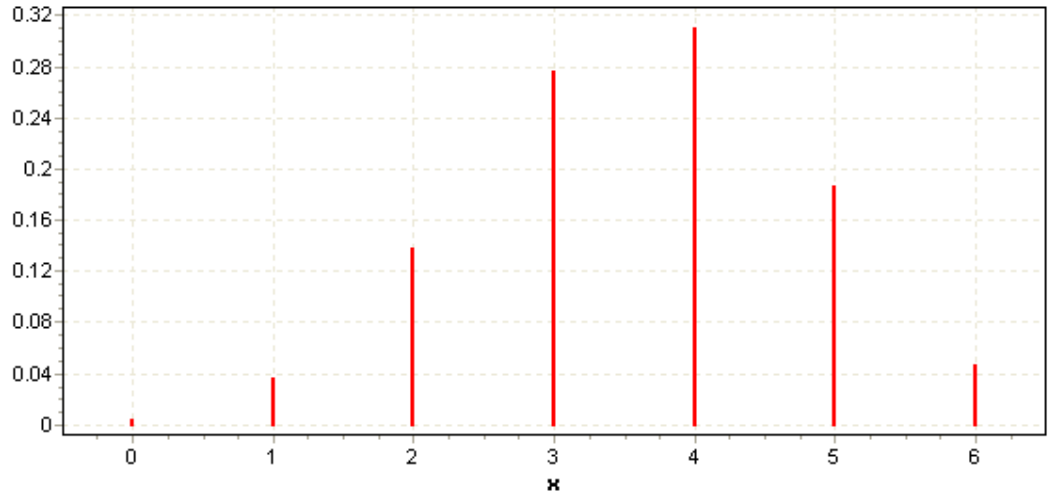

Figure A.2. Probability mass function

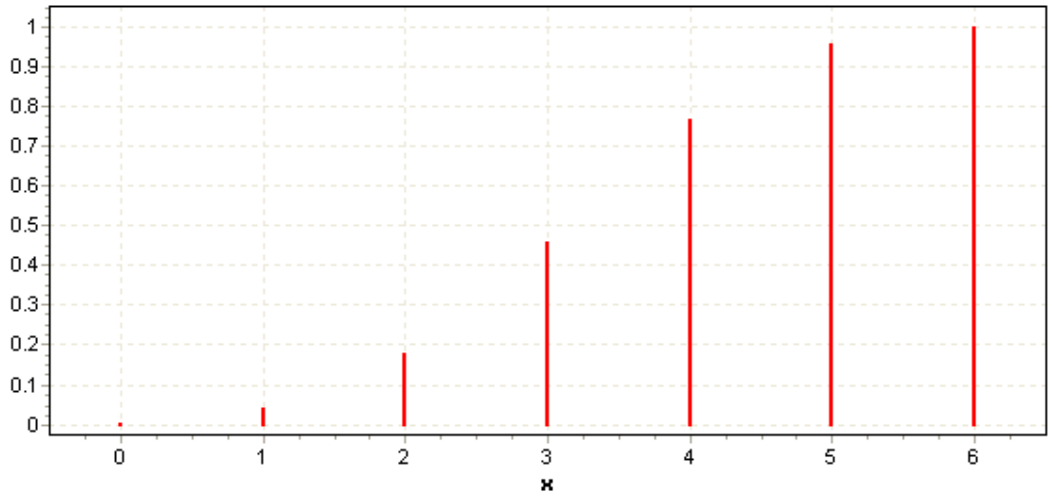

Figure A.3. Cumulative distribution function

A.6.2. Poisson Distribution

A random variable X is called a Poisson random variable with parameter λ if its probability mass function is given by [20];

$$p_x(k) = P(X=k) = e^{-\lambda} \lambda^k / k! \quad k = 0, 1, \ldots \tag{A.30}$$

The corresponding cumulative distribution function of X is given by;

$$F_x(x) = \sum_{k=0}^{n} e^{-\lambda} \lambda^k / k! \quad n \le x < n+1 \qquad (A.31)$$

Figure A.4 and A.5 illustrates the examples of probability mass and cumulative distribution functions, where $\lambda = 3$

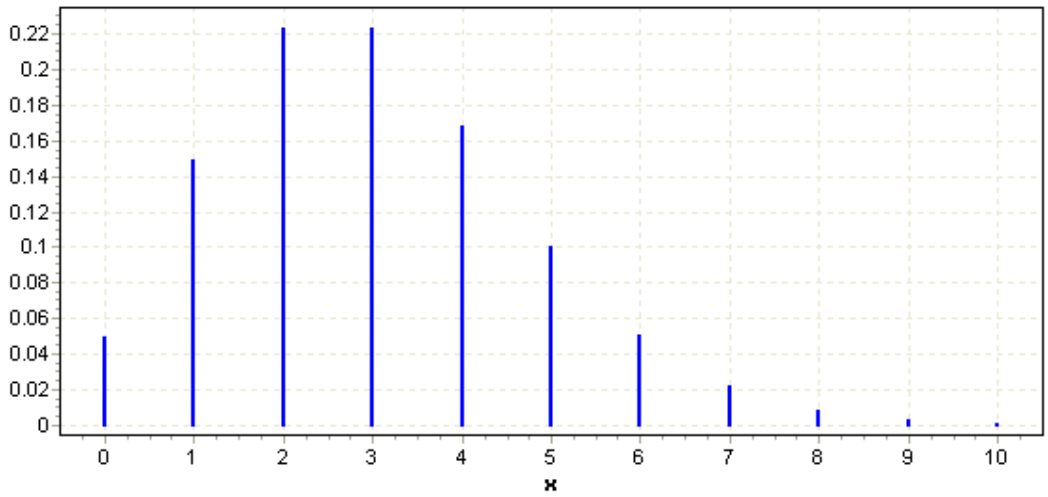

Figure A.4. Probability mass function

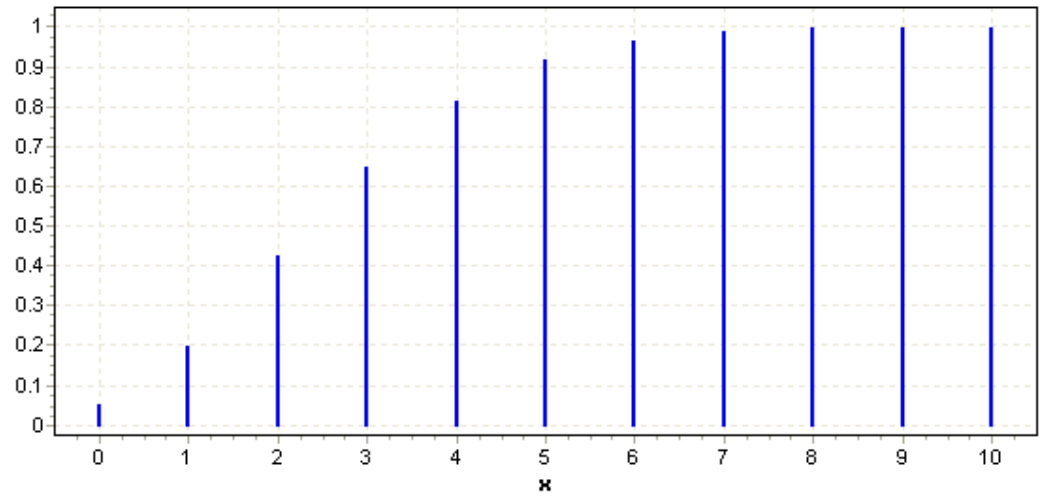

Figure A.5. Cumulative distribution function

APPENDIX B: Computer Aided Software Reliability Estimation Tool (CASRE)

CASRE is a case CASE tool for a systematic and automatic application of software reliability modeling for real-world projects [33]. It is an extension SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software), and is designed to execute both in a DOS Windows environment and a UNIX X-windows environment.

Figure B.1 [33] demonstrates the high level structure of CASRE. There are four major functional areas;

- Data Modification: CASRE supports data modification, including; creating new failure data files, modifying existing files, and performing global operations on files such as editing, smoothing and data transformation.
- Failure Data Analysis: CASRE allow users to analyze the failure data set to see if the data set exhibits reliability growth.
- Modeling And Measurement: CASRE allows users to execute single models or multiple models and combine the results of the models. The following models are implemented in the CASRE:
    - Geometric
    - Jelinski-Moranda
    - Littlewood-Verrall Linear
    - Littlewood-Verrall Quadratic
    - Musa Basic
    - Musa-Okumoto
    - Nonhomogenous Poisson(NHPP)
    - Generalized Poisson
    - Schnedewind: all
    - Schnedewind: Cumulative 1 st
    - Schnedewind: Last n
    - Schick-Wolverton
    - Yamada S-Shaped

- Modeling/Measurement Results Display: Following forms are supported graphically by the CASRE[33]:
    - Interfailure times/failures frequencies, actual and estimated
    - Cumulative failures, actual and estimated
    - Reliability growth, actual and estimated

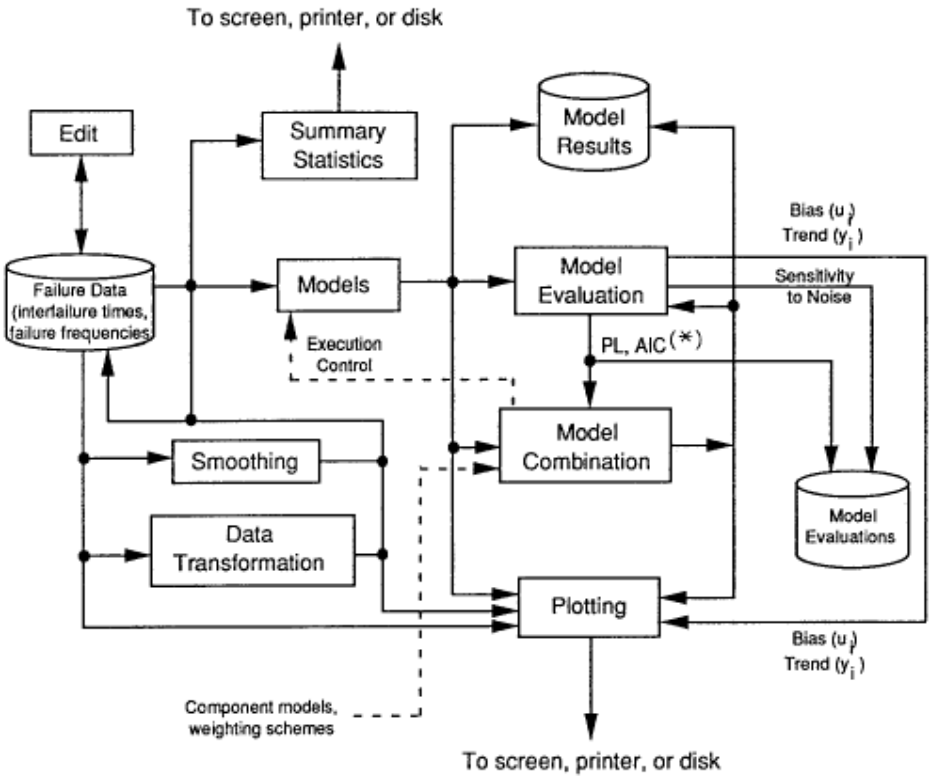

Figure B.1. High level architecture of CASRE

Figure B.2 [21] shows a flowchart of a typical CASRE session. Typically, users will select a set of failure data, choose how far into the future they want to predict reliability, select and run models, look at model results, and determine which model is most appropriate to the data [21]. Detailed explanations of the typical CASRE session can be found in [21].

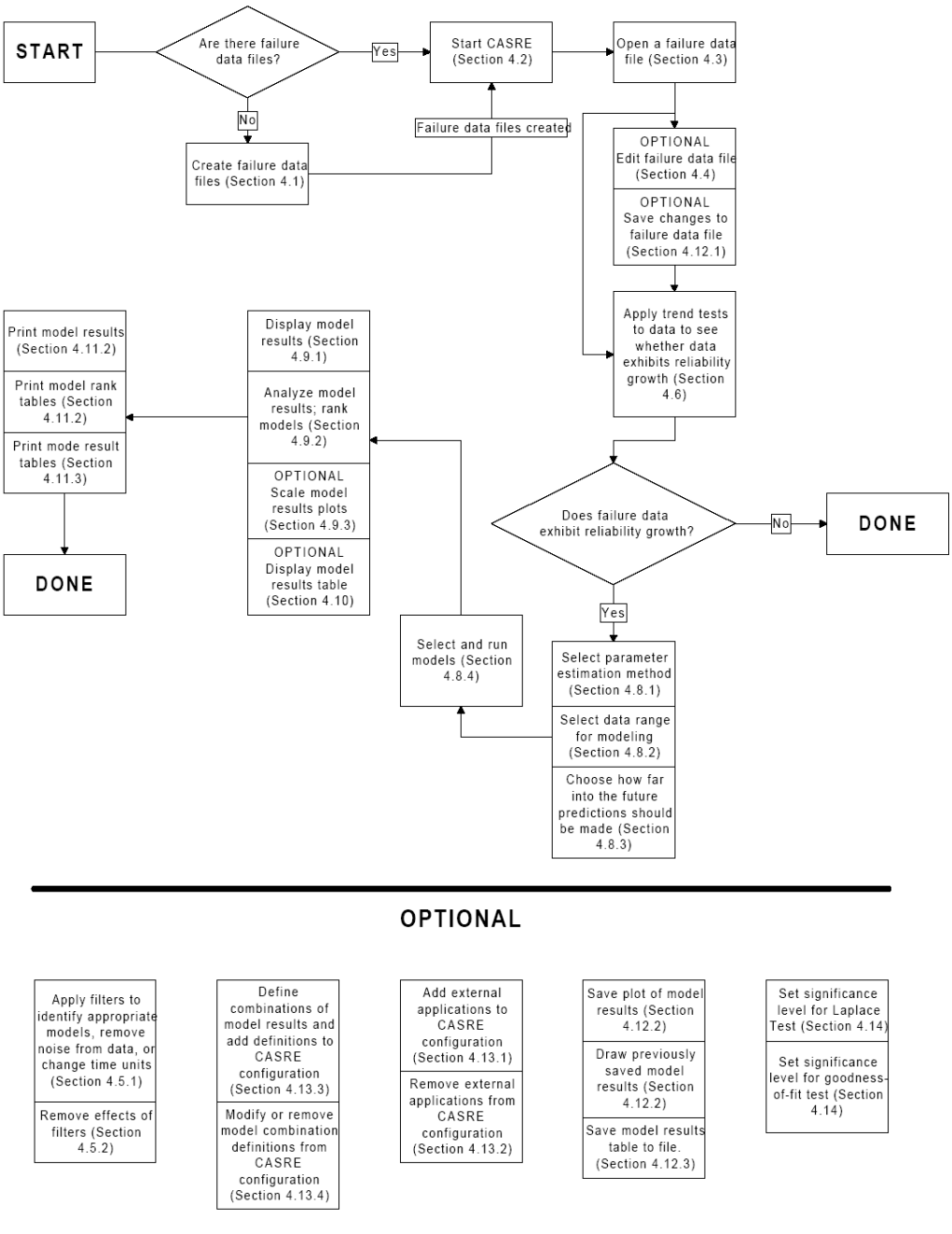

Figure B.2. Typical work session for CASRE

REFERENCES

1. Schneidewind, N.F., "*A Quantitative Approach to Software Development Using IEEE 982.1*", Software, IEEE Volume 24, Issue 1, Page(s): 65 – 72, 2007.

2. Lyu, M.R., "*Software Reliability Engineering: A Roadmap*", Future of Software Engineering (FOSE'07) Page(s):153 – 170, 2007.

3. Cavano, J.P. and J.A. McCall, "*A Framework for the Measurement of Software Quality*", Proceedings of the ACM Software Quality Assurance Workshop, pp. 133-139, 1978.

4. ANSI/IEEE, "*Standard Glossary of Software Engineering Terminology*", STD-729-1991, ANSI /IEEE, 1991.

5. Musa, J.D., "*Software Reliability Engineering: More Reliable Software Faster And Cheaper*", AuthorHouse, 2004.

6. Lyu, M.R., "*Software Reliability Theory*", Encyclopedia of Software Engineering, Article Online Posting, January 15, 2002.

7. Lyu, M.R., "*Handbook of Software Reliability Engineering*", IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

8. Shooman, M.L., "*Probabilistic Reliability: An Engineering Approach*", 2nd ed., Krieger, New York, 1990.

9. Lloyd, D.K. and M. Lipow, "*Reliability: Management, Methods, and Mathematics*", 2nd ed., ASQC, Milwaukee, WI, 1984.

10. Musa, J.D. and K. Okumoto, "*Software Reliability Models: Concepts, Classification, Comparisons, and Practice*", NATO ASI Series F3, 395–424, 1983.

11. Jelinski, Z. and P. Moranda, “Software Reliability Research”, in: Freiberger, W. (ed.): Statistical Computer Performance Evaluation, New York, pp. 465 – 484, 1972

12. Moranda, P.B., “*Prediction of software reliability software during debugging*”. Proceedings on the 1975 Annual Reliability and Maintainability Symposium, 327-32, 1975.

13. Goel, A. L. and K. Okumoto , “*Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures*”, IEEE Trans. Reliability 28, pp. 206 – 211, 1979.

14. Musa, J.D., “*A Theory of Software Reliability and Its Application*”, IEEE Trans. Software Eng. 1, pp. 312 – 327, 1975.

15. Farr, W.H., “*A survey of software Reliability Modeling and Estimation*”, NavSWC technical Report TR 82-171, 1983.

16. Musa, J.D. and K. Okumoto, “*A Logarithmic Poisson Execution Time Model for Software Reliability Measurement*”, Proceedings Seventh International Conference on Software Engineering, Orlando, Florida, pp. 230-238, 1984.

17. Schneidewind, N.F., "*Analysis of Error Processes in Computer Software*", *Sigplan Note,* vol. 10, no. 6, pp. 337-346, 1975.

18. Yamada, S. and M. Ohba and S. Osaki, "*S-Shaped Reliability Growth Modeling for Software Error Detection*", *IEEE Transactions on Reliability,* vol. R-32, no. 5, pp. 475-478, December, 1983.

19. Ohba, M., "*Software Reliability Analysis Models*", IBM Journal of Research and Development, vol. 21, no. 4, pp. 428-443, July, 1984.

20. Hsu, H., "*Probability, Random Variables, & Random Processes*", McGraw-Hill Book Company, 1997.

21. Nikora, A., "*Computer Aided Software Reliability Estimation Tool User's Guide*", March, 2000.

22. Brocklehurst, S. and P.Y. Chan and B. Littlewood and J. Snell, "*Recalibrating Software Reliability Models*", IEEE Transactions on Software Engineering, Vol.16, April, 1990.

23. Chakravarti, I.M. and R.G. Laha and J. Roy, "*Handbook of Methods of Applied Statistics, Volume I"*, John Wiley and Sons, pp. 392-394, 1967.

24. Denton, A.D., "*Accurate Software Reliability Estimation*", Master of Science Thesis, Colorado State University, Fort Collins, Colorado, ,1999.

25. Kharchenko, V.S. and V.V. Tarasyuk and V.Yu. Dubnitsky, "*The Method of Software Reliability Growth Models Choice Using Assumptions Matrix*", Proceedings of the 26 th Annual International Computer Software and Applications Conference(COMPSAC'02), 2002.

26. Stringfellow, C. and A.A. Amschler, "*An Emprical Method for Selecting Software Reliability Growth Models*", Emprical Software Engineering, 7, 319-343, 2002.

27. Asad, C.A. and M.I. Ullah and M.J. Rehman, "*An Approach for Software Reliability Model Selection*", Proceedings of the 26 th Annual International Computer Software and Applications Conference(COMPSAC'02), 2004.

28. Lyu, M.R. and A. Nikora, "*An Experiment in Determining Software Reliability Model Applicability*", Proceedings of the Sixth International Symposium on Software Reliability Engineering, Tolouse, France, pp. 304-313, October, 1995.

29. Schneidewind, N.F. ”*Reliability Modeling for Safety Critical Software*”, IEEE Transactions on Reliability, Vol. 46, No.1, March, 1997.

30. Musa, J.D. and A. Iannino and K. Okumoto, “*Software Reliability - Measurement, Prediction, Application*”, New York, St. Louis, et al., 1987.

31. Wood, A., “*Predicting software reliability*”, IEEE Computer 29(11): 69–78, 1996.

32. Musa, J.D., "*Software Reliability Data*", Technical Report, Data and Analysis Center for Software, Rome Air Development Center, Griffins AFB, New York, 1979.

33. Lyu, M.R. and A. Nikora, “*CASRE: a computer-aided software reliability estimation tool*”, Computer-Aided Software Engineering, Proceedings, Fifth International Workshop on, 1992.