Software Reliability Models and Procedure for Fitting a Software Reliability Model

Syed Khasim, Dr. R. Satya Prasad

1Dept. of Computer Science, Rayalaseema University, Kurnool, AP, India
2Dept. of Computer Science & Engg, Acharya Nagarjuna University, AP, India

Abstract

Software reliability is a useful measure in planning and controlling resources during the Software development process, so that high quality software can be developed. It is also a useful measure for giving the user confidence about software corrections. A number of analytical models have been proposed during the past twenty years for evaluating the reliability of a software system. In this paper we present Software reliability models and procedure for fitting a software reliability modeling.

Keywords

Modeling, Estimation, Software Failures, Reliability, Fitted Model

I. Introduction

Computers are finding an ever increasing number of applications. The expenditure on computers software is increasing faster than on associated hardware. An important value attribute of a computer system is the degree to which it can be relied upon to perform its intended function. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers from the early days of their evolution. Until the late 1960’s, attention was almost solely on the hardware related performance of the system. In the early 1970’s, software also became a matter of concern, primarily due to a continuing increase in the cost of software relative to hardware, in both the development and the operational phases of the system.

Software is effectively an instrument for transforming a distinct set of inputs into a distinct set of outputs. It comprises of a set of coded statements whose function may be to evaluate an expression and store the result in a temporary or permanent location, decide which statement to execute next, or to perform input or output operations. There are two approaches available for indicating the existence of software faults. They are program proving, and program testing. Program proving is formal and mathematical. The program testing is more practical and heuristic. Program proving by using inference rules is known as the inductive assertion method. Program testing is the symbolic or physical execution of a set of test cases with the intent of exposing embedded faults in the program. Program proving, program testing remains an imperfect tool for assuring program correctness. The perfect tool is one such quantifiable metric of quality that is commonly used in software engineering practice is software reliability [4].

II. Meaning and Measurement of Software Reliability

Software reliability is a probabilistic measure and can be defined as the probability that software faults do not cause a failure during a specific exposure period in a specific user environment. Software reliability, suppose that a user executes a software product several times according to its usage profile and finds that the results are acceptable 95 percent of the time. Then the software is said to be 95 percent reliable for that user. Planning and controlling the testing resources through the software reliability measure can be done by balancing the additional cost of testing and the corresponding improvement in software reliability. Current approaches for measuring software reliability basically parallel those used for hardware reliability assessment with suitable modifications to account for the inherent differences between software and hardware. Two users exercising two different sets of paths in the same software are likely to have different values of software reliability. Software reliability models can be used to predict the reliability when the software is put into operational use. The software reliability model use the information of the number of errors debugged during the development of a software program. A number of analytical models have been proposed to address the problem of software reliability measurement. These approaches are based mainly on the failure history of software and can be classified according to the nature of the failure process [Amrit L. Goel]. The software reliability measurement models are three types. They are i). Times between failure models ii). Failure count models iii). Fault seeding & Input domain based models.

Fig. 1: Software Reliability Measurement Models

A. Times Between Failures Models

In this group of models the process under study is the time between failures. The most common approach is to assume that the time between, say, the (i - 1)st and the ith failures, follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. Estimates of the parameters are obtained from the observed values of times between failures and...
estimates of software reliability, mean time to next failure, etc., are then obtained from the fitted model.

1. Jelinski and Moranda Model
This is one of the earliest and probably the most commonly used model [14] for assessing software reliability from 1972. \( \lambda_{JMM}(x_i) = K_{JM}[E_0 - (i-1)] \)

Where \( K_{JM} \) = constant of proportionality
\( x_i \) = time between the \( i \)th and \( (i-1) \)st errors discovered

The reliability function and the mean time to failure can be obtained
\( R(t_i) = \exp[-K_{JM}(E_0 - i + 1)i] \) and
\( MTTF = \int_{0}^{\infty} R(t_i) dt_i = \frac{1}{K_{JM}[E_0 - i + 1]} \)

2. Schick and Wolverton (SW) Model
The Schick Wolverton model [35] assumes the hazard rate proportional to the number of remaining errors and the debugging time
\( \lambda_{SW}(t_i) = K_{SW}[E_0 - (i - 1)]x_i \)

Where \( x_i \) = time interval between the \( (i-1) \)st and the \( i \)th error.
\( R(t_i) = \exp[\int_{0}^{t_i} \lambda_{SW}(x) dx ] \)
\( = \exp[-K_{SW}(E_0 - i + 1)i^2] \)
\( MTTF = \int_{0}^{\infty} R(t_i) dt_i = \exp[-K_{SW}(E_0 - i + 1)i^2/2] \)
(1/2)

It could possibly be argued both for and against having the hazard rate proportional to debugging time. Probably the only way to judge the suitability of this model is by fitting it to the experimental data.

3. Goel and Okumoto Imperfect Debugging Model
Goel and Okumoto proposed an imperfect debugging model which is basically an extension of the JM model. In this model, the number of faults in the system at time \( t \), \( X(t) \), is treated as a Markov process whose transition probabilities are governed by the probability of imperfect debugging [10-11]. Times between the transitions of \( X(t) \) are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the \( (i - 1) \)st and the \( i \)th failures is given by
\( Z(t_i) = [N - p(i - i)] \)

Where \( N \) is the initial fault content of the system, \( p \) is the probability of imperfect debugging, and \( X \) is the failure rate per fault.

4. Littewood-Verrall Bayesian Model
The Littewood-Verrall model disposes of this assumption [21-22], based on the observation that a program with defects in rarely exercised sections of the code will be more reliable than the same program with the same number of defects in frequently exercised portions of the code. This model also assumes that the failure rate, instead of being constant, is a random variable.

In the above, \( \psi(t) \) describes the quality of the programmer and the difficulty of the programming task. It is claimed that the failure phenomena in different environments can be explained by this model by taking different forms for the parameter \( \psi(t) \).

B. Failure Count Models
These types of models are in the number of faults or failures in specified time intervals rather than in times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. Parameters of the failure rate can be estimated from the observed values of failure counts or from failure times. Estimates of software reliability, mean time to next failure, etc.

5. Goel-Okumoto Nonhomogeneous Poisson Process Model:
This model Goel and Okumoto assumed that a software system is subject to failures at random times caused by faults present in the system. Letting \( N(t) \) be the cumulative number of failures observed by time \( t \), they proposed that \( N(t) \) can be modeled as a nonhomogeneous Poisson process, i.e., as a Poisson process with a time dependent failure rate. Based on their study of actual failure data from many systems, they proposed the following form of the model [31].

\[ P\{N(t) = y\} = \frac{(m(t))^y}{y!} e^{-m(t)}, y = 0,1,2, \ldots \]

Where \( m(t) = a(1 - e^{-bt}) \), and \( \lambda(t) = m(t) = abe^{-bt} \)

Here \( m(t) \) is the expected number of failures observed by time \( t \) and \( \lambda(t) \) is the failure rate. Then the reliability function given by
\( R_{x_i}/S_{x_i} (x/s) = e^{-[m(x+x) - m(x)]} \)

Failure count models assume that a software system exhibits a decreasing failure rate pattern during testing. Goel proposed the following generalization of the Goel-Okumoto NHPP model.
P[N(t) = y] = \frac{(m(t))^y}{y!} e^{-m(t)}, y = 0, 1, 2, ...

m(t) = a(1 - e^{-bt}),

Where a is an expected number of faults to be eventually detected, and b and c are constants that reflect the quality of testing. The failure rate for the model is given by

$$\lambda(t) = m'(t) = abc \ e^{-bt} t^{c-1}$$

Where $\gamma$ is the execution time utilized in executing the program up to the present, $f$ is the linear execution frequency $\Phi$ is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, and $n_c$ is the number of faults corrected during $(0, \gamma)$.

7. Musa Execution Time Model
In this model Musa [26] makes assumptions that are similar to those of the JM model except that the process modeled is the number of failures in specified execution time intervals. The hazard function for this model is given by

$$z(\gamma) = \delta f(N - n_e)$$

Where $\delta$ is the execution time utilized in executing the program up to the present, $f$ is the linear execution frequency $\Phi$ is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, and $n_c$ is the number of faults corrected during $(0, \gamma)$.

8. Shooman Exponential Model
This model is [37] essentially similar to the JM model. For this model the hazard function is of the following form

$$z(t) = k \left[ \frac{N}{I} - n_e (\gamma) \right]$$

9. Generalized Poisson Model
This is a variation of the NHPP [1] model of Goel and Okumoto and assumes a mean value function of the following form.

$$m(t) = \delta (N - M_{i-1})/t_i$$

Where $M_{i-1}$ is the total number of faults removed up to the end of the $(i-1)$st debugging interval, $\delta$ is a constant of proportionality, and $a$ is a constant used to rescale time $t_i$.

10. IBM Binomial and Poisson Models
In these models Brooks and Motley [6], consider the fault detection process during software testing to be a discrete process, following a binomial or a Poisson distribution.

11. Musa-Okumoto Logarithmic Poisson Execution Time Model
In this model [27] the observed number of failures by some time $T$ is assumed to be a NHPP, similar to the Goel-Okumoto model, but with a mean value function which is a function of $\gamma$, viz.

$$\mu(\gamma) = \frac{1}{\theta} \ln(\lambda \theta \gamma + 1),$$

C. Fault Seeding & Input Domain Based Models
The basic approach in this class of models is to “seed” a known number of faults in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed numbers of seeded and indigenous faults are counted. The basic approach taken here domain based models are to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program. An estimate of program reliability is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain [Amrit L. Goel].

12. Mills Seeding Model
The most popular and most basic fault seeding model is Mills’ Hypergeometric model [23]. This model requires that a number of known faults be randomly seeded in the program to be tested. The program is then tested for some amount of time. The number of original indigenous faults can be estimated from the numbers of indigenous and seeded faults uncovered during the test by using the hypergeometric distribution.

13. Nelson Model
In this input domain based model [29], the reliability of the software is measured by running the software for a sample of $n$ inputs.

$$R = 1 - \sum_{j=1}^{n} \left( \frac{f_j}{n_j} \right) P(E_j)$$

Where $n_j$ is the number of runs sampled from input sub-domain $E_j$ and $f$ is the number of failures observed out of $n_j$ runs.

14. Ramamoorthy and Bastani Model
In this input domain based model [32], the authors are concerned with the reliability of critical, real-time, process control programs. In such systems no failures should be detected during the reliability estimation phase, so that the reliability estimate is one.

$$P \{ \text{program is correct for all points in } [a, a+V] \}$$

$$= e^{-2N} \prod_{j=1}^{n-1} \left[ \frac{2}{1 + e^{-2N}} \right]$$

III. Procedure for fitting Software Reliability Modelling
The various steps involved for software reliability modeling.

A. Step-1 Collect Failure Data
These models are discussed in this paper require that failure data be available. For most of the models, such data should be in the form of either times between failures or as failure counts. The first step in developing a model is to carefully study such data in order to gain an insight into the nature of the process being modeled.

B. Step-2 Choose an Appropriate Model
The next step is to choose an appropriate model based upon an understanding of the testing process and assumptions. A check about the “goodness” of the choose model for the data.

C. Step-3 Obtain the Fitted Model
The fitted model is obtained by substituting the estimated values of the parameters in the chosen model. At this stage, we have a fitted model based on the available failure data and the chosen model form.

D. Step-4 Obtain Estimates of Model Parameters
Different methods are generally required depending upon the nature of valuable data. The most commonly used one is the method of maximum likelihood because it has very good statistical properties. However, sometimes, the method of least squares or some other method may be preferred.
Collect failure Data

Choose an appropriate Model

Perform Goodness of Fit test

No

Yes

Obtain fitted Model

Estimate Model Parameters

Obtain estimates of performance measures

Software Reliability

Choose another Model

E. Step-5 Obtain Estimates of Performance Measures

At this stage, we can compute various quantitative measures to assess the performance of the software system. Some useful measures are shown below[4].

F. Step-6 Software Reliability

Finally obtain the software reliability. This can be used for planning, scheduling and assumption for software development.

IV. Conclusion

In this paper, we have explained about Software reliability models. The use and applicability of these models during in the software development process. It should be distinguished that the above models are basically useful to predicting and monitoring software reliability.

References


Syed Khasim received the Master’s degree M.C.A from Madurai Kamaraj University in 2002. He received Master of Philosophy M.Phil in Computer Science from Periyar University in 2008. He is pursuing Ph.D. in Computer Science from Rayalaseema University, Kurnool, A.P., India. His research interests are Software Engineering, Database Management Systems, and Computer Networks.

Dr. R. Satya Prasad received Ph.D. degree in Computer Science in the faculty of Engineering in 2007 from Acharya Nagarjuna University, Andhra Pradesh. He received gold medal from Acharya Nagarjuna University for his outstanding performance in Masters Degree. He is currently working as Associate Professor and H.O.D, in the Department of Computer Science & Engineering, Acharya Nagarjuna University. His current research is focused on Software Engineering. He has published several papers in National & International Journals.