Overview of Software Reliability Models

Manjula.C.M.Prasad¹, Dr. Lilly Flowrence², Dr. C.V. Srikrishna³

¹Associate Professor, PES Institute of Technology, Bangalore South Campus, Bangalore, INDIA
²Professor, Adhiyamman College of Engineering Hosur, Tamilnadu, INDIA
³Professor, Department of MCA, PESIT, Bangalore, INDIA

ABSTRACT

This paper reviews the available literature on reliability models related to software engineering. The paper lists all the models related to prediction and estimation of reliability of software engineering process. Most of the software reliability models reviewed involve assumptions, parameters, and a mathematical function that relates the reliability with the parameters.

I. INTRODUCTION

The basic goal of software engineering is to produce high quality software at low cost. With growth in size and complexity of software, management issues began dominating. Reliability is one of the representative qualities of software development process. Reliability of software is basically defined as the probability of expected operation over specified time interval.

Software Reliability [1] is an important attribute of software quality, together with functionality, usability, performance, serviceability, capability, install ability, maintainability, and documentation. Reliability is the property of referring ‘how well software meets its requirements’ & also ‘the probability of failure free operation for the specified period of time in a specified environment. Software reliability defines as the failure free operation of computer program in a specified environment for a specified time. Software Reliability is hard to achieve, because the complexity of software tends to be high.

II. SOFTWARE RELIABILITY MODELS

A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. Software Reliability Modelling techniques can be divided into two subcategories: Prediction modelling and Estimation modelling. Both kinds of modelling techniques are based on observing and accumulating failure data and analyzing with statistical inference.

Representative prediction models include Musa's Execution Time Model, Putnam's Model, and Rome Laboratory models TR-92-51 and TR-92-52, etc. Using prediction models, software reliability can be predicted early in the development phase and enhancements can be initiated to improve the reliability. Representative estimation models include exponential distribution models, Weibull distribution model, Thompson and Chelson's model, etc. Exponential models and Weibull distribution model are usually named as classical fault count/fault rate estimation models, while Thompson and Chelson's model belong to Bayesian fault rate estimation models.

III. PREDICTION MODELS

This model uses historical data. They analyze previous data and some observations. They are usually made prior to development and regular test phases. The model follow the concept phase and the predication from the future time.

A. Musa Model:

This prediction technique is used to predict, prior to system testing, what the failure rate will be at the start of system testing. This prediction can then later be used in the reliability growth modelling. For this prediction method, it is assumed that the only thing known about the hypothetical program is the program size and the processor speed.

This model [2] assumes that failure rate of the software is a function of the number of faults it contains and the operational profile. The number of faults is determined by multiplying the number of developed executable source instructions by the fault density. Developed excludes re-used code that is already debugged. Executable excludes data declarations and compiler directives. For fault density at the start of system test, a value for faults per KSLOC needs to be determined. For most projects the value ranges between 1 and 10 faults per KSLOC. Some developments which use rigorous methods and highly advanced software development
processes may be able to reduce this value to 0.1 fault/KSLOC.

**B. Putnam Model:**

Created by Lawrence Putnam, Sr. the Putnam model[3] describes the time and effort required to finish a software project of specified size. SLIM (Software Lifecycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM, Inc. has developed based on his model. It is one of the earliest of these types of models developed, and is among the most widely used. Closely related software parametric models are Constructive Cost Model (COCOMO), Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), and Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM).

Putnam’s observations about productivity levels to derive the software equation:

\[
\frac{B^{1/3} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} \cdot \text{Time}^{4/3}
\]

(1)

where:

- Size is the product size (whatever size estimate is used by your organization is appropriate). Putnam uses ESLOC (Effective Source Lines of Code) throughout his books.
- B is a scaling factor and is a function of the project size. Productivity is the Process Productivity, the ability of a particular software organization to produce software of a given size at a particular defect rate.
- Effort is the total effort applied to the project in person-years.
- Time is the total schedule of the project in years.

In practical use, when making an estimate for a software task the software equation is solved for effort:

\[
\text{Effort} = \left[ \frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}} \right]^{3} \cdot B
\]

(2)

An estimated software size at project completion and organizational process productivity is used. Plotting effort as a function of time yields the Time-Effort Curve. The points along the curve represent the estimated total effort to complete the project at some time. One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.

**C. Rome Lab TR-92-52 Model:**

Software Reliability Measurement and -Test Integration Techniques Method RL-TR-92-52 contain empirical data that was collected from a variety of sources, including the Software Engineering Laboratory. The model [4] consists of 9 factors that are used to predict the fault density of the software application. The nine factors are:

<table>
<thead>
<tr>
<th>Factor</th>
<th>Measure</th>
<th>Range of values</th>
<th>Applicable phase</th>
<th>Tradeoff range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A – Application</td>
<td>Difficulty in developing to test various application types</td>
<td>1 to 4 (ESLOC)</td>
<td>A.T</td>
<td>None – Slight</td>
</tr>
<tr>
<td>D – Development organization</td>
<td>Development organization methods, tools, techniques, documentation</td>
<td>2 to 6</td>
<td>Known at A, D, E, F</td>
<td>1 to 3</td>
</tr>
<tr>
<td>SA – Software manageability</td>
<td>Indication of fail tolerant design</td>
<td>0 to 1.1</td>
<td>Normally, CT</td>
<td>Small</td>
</tr>
<tr>
<td>ST – Software reliability</td>
<td>Reliability of design and code to requirements</td>
<td>0 to 2.0</td>
<td>Normally, CT</td>
<td>Large</td>
</tr>
<tr>
<td>SQ – Software adherence to coding standards</td>
<td>Quality of coding standards</td>
<td>0 to 1.1</td>
<td>Normally, CT</td>
<td>Small</td>
</tr>
<tr>
<td>SL – Software</td>
<td>Normalized fault density by language</td>
<td>Not applicable</td>
<td>CT</td>
<td>NA</td>
</tr>
<tr>
<td>SX – Software</td>
<td>Size of complexity</td>
<td>0 to 1.5</td>
<td>CT</td>
<td>Large</td>
</tr>
<tr>
<td>SM – Software</td>
<td>Size of units</td>
<td>0 to 2.0</td>
<td>CT</td>
<td>Large</td>
</tr>
<tr>
<td>SR – Software</td>
<td>Completeness of design review</td>
<td>0 to 1.5</td>
<td>CT</td>
<td>Large</td>
</tr>
</tbody>
</table>

There are certain parameters in this prediction model that have tradeoff capability. This means that there is a large difference between the maximum and minimum predicted values for that particular factor. Performing a tradeoff means that the analyst determines where some changes can be made in the software engineering process or product to experience an improved fault density.
prediction. A tradeoff is valuable only if the analyst has the capability to impact the software development process. The output of this model is a fault density in terms of faults per KSLOC. This can be used to compute the total estimated number of inherent defects by simply multiplying by the total predicted number of KSLOC.

**D. Raleigh Model:**

This model [5] predicts fault detection over the life of the software development effort and can be used in conjunction with the other prediction techniques. Software management may use this profile to gauge the defect status. This model assumes that over the life of the project that the faults detected per month will resemble a Raleigh curve.

The Rayleigh model [6] is the best suited model for estimating the number of defects logged throughout the testing process, depending on the stage when it was found (unit testing, integration testing, system testing, functional testing etc.).

Defect prediction function is the following:

\[ F(t)f(K_{cum},t,m) = (1) \]

Parameter K is the cumulated defect density, t is the actual time unit and tm is the time at the peak of the curve.

For example, the estimated number of defects impacts the height of the curve while the schedule impacts the length of the curve. If the actual defect curve is significantly different from the predicted curve then one or both of these parameters may have been estimated incorrectly and should be brought to the attention of management.

**IV. ESTIMATION MODEL**

This model uses the current data from the current software development effort and doesn’t use the conceptual development phases and can estimate at any time.

**A. Weibull Model (WM):**

This model [7] is used for software/hardware reliability. The model incorporates both increasing/decreasing and failure rate due to high flexibility. This model is a finite failure model.

\[ MTTF = \int (1-F(t)) = \exp (-\alpha t^b) \]  

Where \( MTTF \) = Mean Time To Failure

\( a, b = \) Weibull distribution parameters. \( t = \) Time of failure.

A Weibul-type testing-effort function with multiple change-points [8] can be estimated by using the methods of least squares estimation (LSE) and maximum likelihood estimation (MLE) The LSE minimizes the sum of squares of the derivations between actual data patterns and predicted curve, while the MLE estimates parameters by solving a set of simultaneous equations

**B. Bayesian Models (BM):**

The models [7] are used by several organizations like Motorola, Siemens & Philips for predicting reliability of software. BM incorporates past and current data. Prediction is done on the bases of number of faults that have been found & the amount of failure free operations.

The Bayesian [9] SRGM considers reliability growth in the context of both the number of faults that have been detected and the failure-free operation. Further, in the absence of failure data, bayesian models consider that the model parameters have a prior distribution, which reflects judgement on the unknown data based on history e.g. a prior version and perhaps expert opinion about the software.

The Littlewood – Verrall model is one example of a bayesian SRGM that assumes that times between failures are independent exponential random variables with a parameter \( \theta(i) \), \( i = 1, 2, \ldots, n \) which itself has parameters \( (i) \) and reflecting programmer quality and task difficulty) having a prior gamma distribution. The failure intensity as obtained from the model using a linear form for the \( \theta(i) \) function

\[ \hat{\theta}(i) = (1-i)(\alpha^2 + 2\theta(i)(1-i))^2 \]  

Where \( B \) represents the fault reduction factor, as in Musa’s basic execution time model. This model requires tune between failure occurrences to obtain the posterior distribution from the prior distribution.

**C. J-M Model (JMM):**

This model [7] assumes that failure time is proportional to the remaining faults and taken as an exponential distribution. During testing phase the number of failures at first is finite. Concurrent mitigation of errors is the main strength of the model and error does not affect the remaining errors. Error removal is all human behavior which is irregular so it cannot be avoid by introducing new errors during the process of error removal.
This model \cite{9} assumes that \( N \) initial faults in the code prior to testing are a fixed but known value. Failures are not correlated and the times between failures are independent and exponentially distributed random variables. Fault removal on failure occurrences is instantaneous and does not introduce any new faults into the software under test. The hazard rate \( z(t) \) of each fault is time invariant and a constant. Moreover, each fault is equally likely to cause a failure.

**D. Goel-Okumoto Model (GOM):**

G-O model \cite{7} takes the number of faults per unit time as independent random variables. In this model the number of faults occurred within the time and model estimates the failure time. Delivery of software within cost estimates is also decided by this model.

The Goel-Okumoto (G-O)\cite{9} non-homogeneous Poisson process (NHPP) model has slightly different assumptions from the J-M model. The significant difference between the two is the assumption that the expected number of failures observed by time \( t \) follows a Poisson distribution with a bounded and non-decreasing mean value function \( f(t) \). The expected value of the total number of failures observed in infinite time is a finite value \( N \).

**E. Thompson and Chelson’s Model:**

In this model the number of failures detected in each interval \( (f_i) \) and length of testing time for each interval \( (T_i) \)

\[
\frac{(f_i + f_0 + 1)}{(T_i + T_0)}
\]

Software is corrected at end of testing interval. Software is operational. Software is relatively fault free. The Thompson Chelson model can be used, if the failure intensity is decreasing, has the software been tested or used in an operational environment representative of its end usage, with no failures for a significant period of time.

**V. OTHER RELATED MODELS**

**A. Geometric Model (GM):**

The model \cite{10} is based on the version of J-M. The time between failures is an exponentially distributed and mean time failure decreased geometrically.

\[
(MTBF)_t = \frac{1}{(N^{-0.1})}
\]

Where, \( N = \) Total number of faults.

\( f = \) Number of fault occurrences.

\( MTBF = \) Mean Time between failure.

\( t = \) Time between the occurrence of the \((i-1)\)th and ith fault occurrences.

\[
E(X_i) = \frac{1}{Z(t_i - 1)}
\]

Where, \( E(X_i) = \) Expected time between failure.

\( Z(t_i - 1) = \) Fault detection rate.

**B. S-Shaped Model (SSM):**

This model \cite{11} considers that the number of failure with in time period is a Poisson type model. In this model time between failures depends on the time to failure. Mitigation of fault occurs immediately as failure happened.

\[
\mu(t) = a[1 - (1 + p)e^{bt}]
\]

Where, \( \mu(t) = \) mean value function at time \( t \).

\( a, b, p = \) Distribution parameters. \( e^{bt} = \) Exponential distribution.

**C. L-V Reliability Growth Model:**

This model \cite{12} 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. For every correction of fault, a separate program has to write. Fault correction is obtained by fixing fault.

\[
D(i) = \mu(1 = f_i(1))
\]

Where, \( f_i(1) = \) Sequence of independent variable.

\( D(i) = \) Distribution for the ith failure.

**D. Exponential Failure Time Model (EFTM):**

Exponential models \cite{12} comprise of all finite failure models. Poisson and Binomial are two categorization of EFTM. The Binomial and Poisson types are based on per fault constant hazard rate. Hazards rate function is defined as the function of the remaining number of faults and the failure function is exponential.

\[
H(Z) = f(RNF) + f(\exp(FF))
\]

Where, \( H(Z) = \) Hazard rate.

\( RNF = \) Renaming number of faults. \( FF = \) Failure Function.

**E. Execution Time Model (ETM):**

Musa’s Basic model assumes that all faults are equally likely to occur and independent of each other \cite{2}. The intensity function is directly proportional to the number of faults remaining in the program and fault correction is proportional to the number of failure occurrence rate.

\[
\mu(t) = \beta 0 (1 - \exp(-\beta 1 t))
\]

Where, \( \mu(t) = \) mean value function at time \( t \).

\( \beta 0 = \) Total number of faults.
F. Hyper Exponential Model (HEM):

The idea behind this model is that the different parts of the software experience an exponential failure rate. However, the rate varies through these parts to ponder different behaviours. Different failure rate are placed in different sections [7].

$$\lambda(t) = N \sum p_i \exp(\theta_i t)$$  \hspace{1cm} (13)

Where, $\lambda(t) =$ Failure Intensity Function.

$t =$ number of failures.

$N =$ finite number of failures.

$p_i =$ particular $i$th class.

$\theta_i =$ total number of $i$th faults.

VI. LOGARITHMIC POISSON (LM)

The model [10] assumes that code has an infinite number of failures. The model follows NHPP. When failure occur distribution decreases exponentially. The possible number of failures over the time is a logarithmic function therefore it is called Logarithmic Poisson.

VII. GEOMETRIC MODEL (GM)

The model [10] is based on the version of J-M. The time between failures is an exponentially distributed and mean time failure decreased geometrically.

$$E(X_i) = 1/Z(t-1)$$  \hspace{1cm} (14)

Where, $E(X_i) =$ Expected time between failure.

$Z(t-1) =$Fault detection rate.

VIII. CONCLUSION

Reliability is the capability of software to maintain a determined level of performance within the time period. Software reliability is a measuring technique for defects that causes software failures in which software behaviour is different from the specified behaviour in a defined environment with fixed time. On the basis of the review the classification on software reliability models has been presented as a major contribution. This Classification is based on the various dimensions of reliability models. The major finding of the study is that the models under review reflect based on the failure data model and the data requirements model.

REFERENCES

[8] Chu-Ti Lin and Chin-Yu Huang Department of Computer Science National Tsing Hua University Hsinchu, Taiwan Software Reliability Modeling with Weibull-type Testing-Effort and Multiple Change-Points