Synopsis of Software Reliability Growth Models

[1]Harminder Pal Singh Dhami, [2]Dr. Vaibhav Bansal
[1,2]Dept. Computer Science & Engg., OPJS University, Churu, Rajasthan, India

Abstract
In this paper we have assessed the present literature on Software Reliability Growth Models related to software engineering. The paper discusses all the available models related to prediction and estimation of software reliability models of software engineering process. Most of the software reliability growth models reviewed involve hypothesis, parameters and a mathematical functions that relates the software reliability with the parameters.

Keywords
SRE, SRGM, SRM, KLOC, ESLOC, SLIM, MTBF, EFTM, ETM.

I. Introduction

In severe contrast with the rapid advancement of hardware technology, proper development of software technology is not able to keep pace in all measures including quality, efficiency, performance and cost. The market demand for complex software/ hardware systems has increased swiftly than the ability to design, implement, test, deliver and maintain them. As the requirements for the dependencies on computers increase, the chances of failure from computer faults also increase. The effect of these failures ranges from inconvenience (e.g., malfunctions of domestic appliances) to monetary losses (e.g., interruptions of financial systems) to loss of life (e.g., failures of medical software or air traffic system). It is needless to mention, the reliability of computer systems has become a main concern for our society.

Many software companies see a main share of project costs identified with the design, implementation and guarantee of reliable software, and they recognize a great need for systematic approaches using software reliability engineering techniques.

The basic aim of software engineering is to produce high quality efficient software at low cost. With the increase in size and complexity of software the management issues began to dominate. An optimal design strategy without any compromises e.g. cost & time, for the system does not deliver an optimal design. The reason for this is the changes in requirements that may occur in later development phases. Such changes may cause design decisions taken earlier to be less optimal. Design erosion is inevitable with the current way of developing software. Refined methods only contribute by delaying the moment that a system needs to be withdrawn or retire. These approaches do not address the fundamental problems that cause design erosion and makes system unreliable [22].

Component based design is expected to have a strong impact on the quality of software development- due to simplicity, the software development speeds up. The shorter development time results in reduced costs. The extensibility and resolvability of software systems is improved, because components can easily be replaced/ substituted by another component that satisfies the requirements. Software components enhance the reliability of the software, as they are tested, debugged and improved over years [23].

Reliability is the most important quality attribute. With ever increasing use of computers in present time software reliability engineering has emerged as a discipline of its own. Reliability of software is basically defined as the probability of expected outcome over specified time interval. Software Reliability is a significant feature of software quality, together with functionality, usability, performance, serviceability, capability, installs ability, maintainability, and documentation [1].

It is considered as the property of referring 'how well software meets its requirements' & also 'the probability of failure free operation', for the specific period of time in a particular environment. Software reliability defines as the failure free operation of computer software in a specified environment for a specified period of time. Software Reliability is tough to achieve, because the complexity of software is likely to be high. The complexity of software is inversely related to software reliability and is directly related to other vital factors in software quality, especially capability, functionality, etc. Emphasizing these features will likely add more complexity to software. The three major components in the definition of software reliability are Time, Failure and Operational Environment.

II. Software Reliability Engineering

Software Reliability Engineering (SRE) as defined by Musa: SRE is a practice that helps one develop software that is more reliable, and develop it faster and much cheaper. It is a proven standard, well-known, best practice that is generally applicable to systems that include software. SRE is low in cost and its implementation has practically no schedule impact.

Software Reliability Engineering works by quantitatively characterizing and applying two things about the product: (i) the projected relative use of its functions and (ii) its required major quality characteristics. The key quality characteristics are reliability, delivery date, availability and life-cycle cost. In applying software reliability engineering, one can vary the relative emphasis on these factors.

It also maximizes test efficiency by making test highly representative of use in the field. Software Reliability is usually accepted as an essential feature of software quality since it measures software failures-the most unwanted event. Software fails due to the presence of latent faults. IEEE has suggested the following standard definition of a fault/defect [14]. A fault occurs when an individual makes a mistake, called an error, while performing some software activity. Many designers do not distinguish between faults, errors or bugs, as their effects are the same, the dreaded failures. A failure is a departure from the system's required behavior and can occur any time pre or post system deliver, during testing, or operation or maintenance. Some faults may never turn into failures, if faulty code is never executed or a particular state is never entered. Nevertheless software cannot be made fault free and these faults certainly lead to failures. This demands application of Software Engineering tools, techniques and procedures. The concern about failure free operation, has forced developers to lay special emphasis on minimizing the number of defects in the software. Defects/faults can sneak into the software during any stage of its development process. These faults/defects need to be identified and rectified. Software Reliability Engineering is a discipline of Software Engineering that intends a failure free operation of software, at the user end employs scientific tools

and techniques during testing to remove the maximum number of faults. A well-designed metrics with documented purposes can help an organization obtain the information it needs to continue to improve its software product's reliability, processes, and services while maintaining a focus on what is important. A practical, logical and systematic, start-to-end method of selecting, designing, and implementing software metrics is a valuable aid [24].

As reliability prediction activity finds future software reliability based upon existing software metrics and measures. It analyzes previous data and findings. Depending on the software development level or stage, forecast involves different techniques. Using reliability prediction models, software reliability can be estimated early in the software development cycle and enhancements can be initiated to improve it [25].

III. Software Reliability Measurement

The issue of designing reliable software has acquired its importance due to the following reasons.

- Systems are becoming software intensive.
- Many software systems are safety critical.
- Software users are demanding reliable, warranted software systems.
- The cost of software development is increasing.

The above mentioned reasons share one implied factor, the risk of facing a failure when the software is delivered to the end users and its consequent high cost. Hence, there is a rise in demand of software reliability models as people inquire to know the reasons software fails, and try to measure software reliability. Two types of activities in Software reliability measurement includes-

1. Reliability Prediction, and
2. Reliability Estimation.

Both above kinds of modeling techniques are based on observing and gathering failure data and analyzing with statistical inference.

1. Reliability Prediction

This activity finds future software reliability based upon existing software metrics and measures i.e. using historic data. They analyze previous data and findings. Depending on the software development stage, forecast involves different techniques. These models include Putnam's Model, Musa's Execution Time Model and Rome Laboratory models (TR-92-51 & TR-92-52), etc. Using reliability prediction models, software reliability can be estimated early in development cycle and improvements can be started to improve the reliability.

2. Reliability Estimation

It finds existing software reliability by using statistical inference techniques to failure data obtained during system test or operation. This is an assessment regarding the accomplished reliability from the past until the present point. Its main purpose is to measure the present reliability and compute whether a reliability model is a good fit in retrospect. Such models include Exponential distribution models, Thompson and Chelson's model, Weibull distribution model, etc. Exponential models and Weibull distribution model are generally named as classical fault count/fault rate estimation models, while Thompson and Chelson's model belong to Bayesian fault rate estimation models.

Most current Software Reliability Models (SRMs) fall in the estimation category to do reliability prediction. An SRM specified the general form of the dependence of the failure process on the primary factors that affect it: fault introduction, fault removal and the operational environment [13][15].

In prediction and estimation of software reliability a typical method is the use of statistical models [16-17]. These models make use of either past/historical data of similar projects or direct software measures such as defect density, fault density, and defect detection rate of the software under investigation.

IV. Prediction Models

Software reliability prediction model uses past or historical data. They analyze previous/old data and some observations. They are usually made before development and regular test phases. Software reliability prediction methods aree specially useful when knowledge of approximate reliability level of the software to be developed is preferred at early phases of development cycle [19]. When that information is of critical importance, the performance of prediction process in determination of a preliminary estimate can be improved by the use of more than one prediction model over the identical data [18][21].

A. Musa Model

This reliability prediction method is used to predict, what the failure rate will be at the start of system testing. This prediction can then be used in the reliability growth modeling. For this reliability prediction process, it is supposed that the only thing known about the hypothetical program is a prediction of the processor speed and its size.

This model [2] presumes that failure rate of the software is a function of the number of defects it contains and the operational profile. The number of defects or faults is determined by multiplying the number of developed executable source codes by the fault density. Developed code does not include reused code that is already debugged. Executable code excludes data declarations and compiler directives. For fault density at the start of system test, a value for faults per KLOC needs to be determined. For most projects the value ranges between 1 and 10 faults per KLOC. Some developers who use rigorous methods and highly advanced software development processes may be able to reduce this value to 0.1 Fault/KLOC.

The hazard function for this model is given by

$$z(\tau) = Øf (N- nc)$$

where τ is the execution time utilized in executing the instructions/ program up to the present, f is the linear execution frequency i.e. average instruction execution rate divided by the number of instructions in the program, Ø is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, and n is the number of faults corrected during (0, τ). One of the main features of this model is that it explicitly put emphasis on the dependence of the hazard function on execution time. Musa also presented a systematic approach for changing the model so that it can be applicable for calendar time as well. The Musa software reliability growth model can be used to find out the optimum release time for minimizing overall cost. Each failure during development entails a cost c1.

B. Putnam Model

Lawrence H. Putnam created this model in 1978. The Putnam model [3] describes the time and effort required to complete a system project of specified size. SLIM (Software LIfecycle Management) is the name given to the proprietary suite of tools

by Putnam developed by his company QSM, Inc. based on this model. It is one of the earliest types of models developed, and is among the most commonly used. Closely related software parametric models are Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), Constructive Cost Model (COCOMO), and Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM). The Putnam model is an empirical software effort estimation model. Putnam's observation about productivity levels to derive the software equation is as given below:

$$\frac{B^{1/3} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} \cdot \text{Time}^{4/3} \qquad (1)$$

Where: B is scaling factor and is a function of the project size.

- Size is the product size. Putnam uses ESLOC (Effective Source Lines of Code) throughout this model and his books.
- Productivity is the Process Productivity; it is the ability of a specific software organization to produce software of a given size at a particular defect rate.
- Effort is the total effort applied to the project in man-years.
- Time is the total schedule of the project in years.

In practical use, when making an approximate for a software task the software equation is solved for Effort, a given below:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B \qquad (2)$$

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

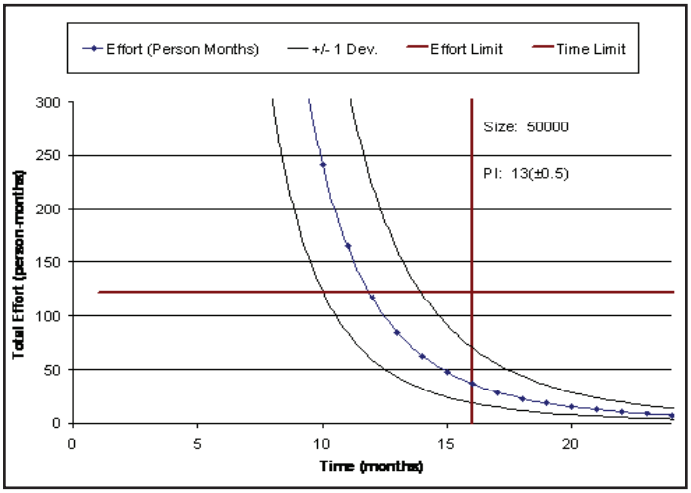

Fig. 1:

As a result, estimating method is fairly sensitive to uncertainty in both size and process productivity. Putnam promotes that obtaining process productivity by calibration.

C. Rome Lab TR-92-52Model

Software Reliability Measurement and Test Integration Techniques Method RL-TR-92-52 contain empirical data that was collected from a variety of sources. The model [4] consists of nine factors that are used to forecast the fault density of the software. The nine factors are shown in table 1 given below:

Table 1: Reliability factors

Factor	Measure	Range of values	Applicable Phase*	Tradeoff Range
A – Application	Difficulty in developing various application types	2 to 14 (defects/KSLOC)	A-T	None – fixed
D - Development organization	Development organization, methods, tools, techniques, documentation	.5 to 2.0	If known at A, D-T	The largest range
SA - Software anomaly management	Indication of fault tolerant design	.9 to 1.1	Normally, C-T	Small
ST - Software	Traceability of design and code to requirements	.9 to 1.0	Normally, C-T	Large
SQ - Software quality	Adherence to coding standards	1.0 to 1.1	Normally, C-T	Small
SL - Software	Normalizes fault density by language type	Not applicable	C-T	N/A
SX - Software	Unit complexity	.8 to 1.5	C-T	Large
SM - Software	Unit size	.9 to 2.0	C-T	Large
SR - Software standards review	Compliance with design rules	.75 to 1.5	C-T	Large

Key: A- Concept or Analysis Phase
D- Detailed and Top level Design
C - Coding
T - Testing

If there are development policies which are defined and it is known what these items will be, even if the code does not yet exist, then these items can be used earlier in the prediction phase. However, the analyst needs to be sure that the prediction reveals what software engineering practices will actually be performed. There are certain attributes in this reliability prediction model that have tradeoff capability. This means that there is a huge difference between the maximum and minimum predicted values for that specific attribute. Working a tradeoff means that the analyst find out where some changes can be made in the software engineering process or product to experience an improved fault density prediction. A tradeoff is significant only if the analyst has the capability to influence the software development procedure. The tradeoff analysis can also be carried out to perform a cost analysis. A cost analysis can also be done by multiplying the difference in expected total number of faults by either a fixed or relative cost parameter. The outcome of this model is a fault density in terms of faults per KLOC. This can be used to work out the total estimated number of inherent faults by simply multiplying by the total predicted number of KLOC.

This model can be used for planning and control as well as for forecast purposes. This model can also be used to obtain absolute as well as relative measures of reliability. For example, the aspect in this model that have the widest possible range of values are development factor, complexity factor, modularity factor and software review factors. These attributes provide relative improvement values. They also allow relationship between projects. Cost comparisons can be achieved by evaluating the progress in fault density of a more aggressive development approach.

D. Raleigh Model

The significant feature of this model is its focus on early detection and fault removal related to the preceding items. Fault detection

over the life cycle of the software development effort is predicted in this model [5]. It can be used in conjunction with the other prediction techniques. It is good overall model for quality management. Software management may use this to measure the defect status. Raleigh model assumes that over the life of the project that the faults detected per month will resemble a Raleigh curve (as shown in figure 2).

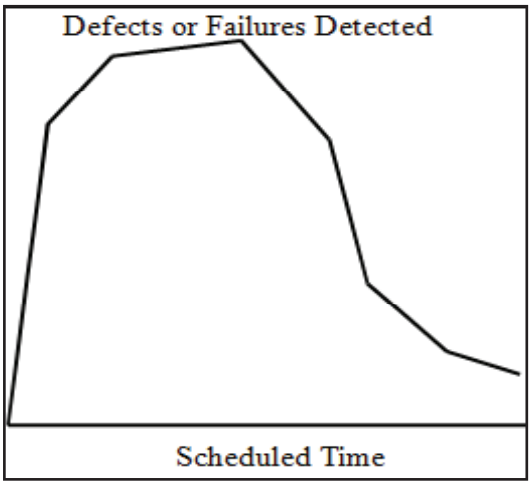

Fig. 2: Raleigh Curve

The Raleigh model [6] is the best suited model for calculating approximately the number of faults or defects logged throughout the testing process, depending on the stage when it was found (i.e. unit testing, integration testing, system testing, functional testing etc.).
Defect prediction function is given below:

$$F(t) f(K, t, t) m = (1) \quad (3)$$

Parameter K is the cumulated defect/fault density, t is the actual time unit and tm is the time at the peak of the curve.
For example, the expected number of defects affects the height of the curve while the schedule impacts the length of the curve. If the real defect curve is significantly dissimilar from the predicted curve then one or both of these parameters may have been predicted incorrectly and should be brought to the notice of management. This method can be used to determine the defect discovery process. The width of this curve is based on the accuracy of the target scheduling or the effectiveness of the committed activities at each stage. The height of this curve depends on the expected number of inherent faults Er. The curve should be updated in the case that either one of these estimates is updated.

V. Estimation Model

This reliability model uses the live data from the existing or 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 or hardware reliability. The model integrates both increasing or decreasing and failure rate due to high flexibility. It is a finite failure model.

$$MTTF = \int (1-F(t)) = \int \exp(-\beta t^{\alpha}) \quad (4)$$

Where, MTTF = Mean Time to Failure
α, β=Weibull Distribution Parameters. t=Time of Failure

Testing effort of Weibull type function with multiple change points [8] and can be estimated by using the methods of Maximum Likelihood Estimation (MLE) and least squares estimation (LSE). The MLE estimates parameters by solving a set of simultaneous equations, while the LSE minimizes the sum of squares of the derivations between actual data patterns and expected curve.

B. Bayesian Models (BM)

The Bayesian analysis techniques is used to estimate the unknown parameters of the models. The models [7] are used by several organizations like Siemens, Motorola & Philips for predicting reliability of software. BM incorporates past and current/live data. Prediction is done on the bases of fault numbers that have been found & the amount of failure free operations. The Bayesian [9] SRGM considers reliability growth in context of both the fault number that have been detected and the failure-free operation.
This technique assists the use of gathered information by developing similar software project. Based on this information the parameter of given model are assumed to pursue some distribution, known as priori distribution. Given the software test data, based on a priori distribution, a subsequent distribution can be acquired, which in turn describes the failure phenomenon.
Littlewood and Verrall, recommended the first software reliability model based on Bayesian Analysis [20]. The Littlewood–Verrall model is an example of a Bayesian SRGM that assumes that times between failures are non dependent exponential random variables with a parameter ξi, i = 1,2, …, n which itself has parameters (i) and reflecting programmer quality and task difficulty) having a priori gamma distribution. As obtained from the model using a linear form for the (i) function, the failure intensity is

$$\text{is } (t) = (-1)(N^2+2B\square\square(-1))\lambda^{1/2}(5)\alpha \qquad \alpha$$

where B stand for 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] supposed that failure time is relative to the remaining/ hidden faults and taken as an exponential distribution. During testing phase the number of failures at first is finite. Concurrent reduction/lessening of errors is the main strength of the model and error does not affect the remaining errors. Error removal is all human an activity which is irregular so it cannot be evaded by introducing new errors during the error removal process.

$$(MTBF)_t = 1/(N-(i-1)) \quad (6)$$

Where, N= Total number of faults.

i= Number of fault occurrences. MTBF=Mean Time between failure.

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

JM model [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. Defects or Fault removal on failure occurrences is immediate 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 expected to cause a failure.

D. Goel-Okumoto Model (GOM)

G-O model [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 determines the failure time. Delivery of software within cost estimates is also decided by this model. The Goel-Okumoto (G-O)[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 (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 Thompson and Chelson's model the Number of failures detected in each interval (fi) and Length of testing time for each interval i (Ti)

$$(fi + f0 + 1)/(Ti+T0) \quad (7)$$

Software is debugged or modified at end of testing interval. Software is operational and it is relatively fault free. Thompson and Chelson's model belong to Bayesian fault rate estimation models and 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.

VI. Other Related Models

A. Geometric Model (GM)

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

$$E(Xi)=1/Z(ti-1) \quad (8)$$

Where, E(Xi)= Expected time between failure.

Z(ti-1) =Fault detection rate.

B. S-Shaped Model (SSM)

This model [11] considers that the number of failure within time period is a Poisson type model. In this model time between failures depends on the time to failure. Alleviation of fault occurs immediately as failure occurred.

$$\mu(t)=\alpha[1-(1+\beta t)e^{-\beta t}] \quad (1) \quad (9)$$

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

α, β=Distribution parameters. $e^{-\beta t}$= Exponential distribution.

C. L-V Reliability Growth Model

This model [12] tries to describe fault generation in the fault correction process by permitting for the probability that the software program could become less reliable than before.

$$D(i) = \mu(1/fi(i)) \quad (10)$$

Where, fi (i) =Sequence of independent variable.

D (i) =Distribution for the i^{th} failure.

For each correction of fault, a separate program has to write. Fault correction is achieved by fixing fault.

D. Exponential Failure Time Model (EFTM)

Exponential models [12] consist of all finite failure models. Two categorization of EFTM are Binomial and Poisson. These types are based on per fault constant hazard rate. Hazards rate function is defined as the function of the remaining or leftover number of faults and the failure function is exponential.

$$H(Z) = f(RNF) + f((\exp(FF)) \quad (11)$$

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 [2]. The intensity function is directly relative to the number of defects remaining in the program and defects correction is relative to the number of failure occurrence rate.

$$\mu(t) = \beta 0\ (1-\exp(-\beta 1 t)) \quad (12)$$

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

$\beta 0$=Total number of faults.

VII. Comparison between Software Reliability Prediction and Estimation Models

Both modeling techniques are based on observing and gathering failure data and analyzing with statistical inference. The major differences are shown in Table 2.

Table 2: Difference between Software Reliability Prediction Estimation Models

Issues	Reliability Prediction Models	Reliability Estimation Models
Data Reference	Uses past/historical data	Uses data from the live/current software development effort
When Used In Development Cycle	Usually made prior to development or test phases; can be used as early as concept and design phase	Usually made later in development life cycle (after some data have been collected); not typically used in concept or development phases
Time Frame	Predict reliability at some future time	Estimate reliability at either current/present or some future time

One of the main problems of software reliability prediction models is that they fail to guess the reliability precisely [9]. The reason is that they assume limited past/historical data of special kind of organizations or of specific type of projects. That creates the problem of loss of control over specification of model's criteria to fit it to a particular organization [9]. Reliability estimation models can overcome this problem up to a certain extent [10].

The problem with the estimation approach is that it cannot be used in early software development stage/phase and can only be used at later stages/phases, which makes organizations to use of reliability prediction techniques[6].

VIII. Conclusion

Software reliability is a major part in software quality. Software reliability cannot be directly measured, so other associated factors

are measured to estimate software reliability and compare it among products. Development process, defects/faults and failures created are all factors associated to software reliability. There exist many models, but no single model can capture a required amount of the software characteristics. Hypothesis and abstractions must be made to simplify the problem. There is no single model that is general to all the situations. It is the capability of software to maintain a specified level of performance within the time period. Software reliability is a measuring technique for faults/defects that causes software failures in which software behaviour is different from the expected behaviour in a defined environment with fixed time. On the basis of the analysis the classification of software reliability models has been presented as a foremost contribution. This Classification is based on the various dimensions of reliability models. The main finding of the study is that the models under analysis reflect based on the failure data model and the data requirements model.

References

[1] Jiantao Pan,"Software Reliability, Carnegie Mellon University", 18-849 Dependable Embedded Systems Spring, 1999.

[2] J. D. Musa, K. Okumoto,"A Logarithmic Poisson Execution Time Model for Software Reliability Measurement", Bell Lab, IEEE, pp. 230-238, 1984.

[3] Putnam, Lawrence H.,"A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions On Software Engineering, Vol. SE-4, No. 4, pp. 345-361, 1978.

[4] Larry Putnam, Ware Myers,"Measures for Excellence, Quantitative Software Management", Yourdon Press, Englewood Cliffs, NJ, 1992.

[5] Lawrence H Putnam, Ware Myers,"Measures For Excellence: Reliable Software On Time", Within Budget 1st Edition, Prentice Hall, 1991.

[6] Ana Maria Vladu,"Software Reliability Prediction Model using Rayleigh Function", 1999.

[7] A.Yadav, R.A.Khan,"Critical Review on Software Reliability Models", International Journal of Recent Trends in Engineering, Vol. 2, No. 3, 2009.

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

[9] Ganesh Pai,"A survey of software reliability models". Dept of ECE, University of Virginia, 2002.

[10] A. A. Ghaly, P. Chan, B. Littlewood,"Evaluation of competing software reliability predictions", IEEE, 1986.

[11] S.Yamada, M.Obha, S.Osaki,"S.Shaped Reliability Growth modeling for Error Detection", IEEE Transaction Reliability.

[12] A. Wood,"Software Reliability Growth Models", Technical report, Tandem computer, 1996.

[13] Lyu MR(editor),"Handbook of software reliability engineering", New York: McGraw Hill, 1996.

[14] Pfleeger SL,"Software Engineering- theory and practice", New Jersey: Prentice Hall, 1998.

[15] Pham H,"Software Reliability", Springer-Verlag Pte. Singapore. 2000.

[16] Kapur PK, Garg RB,"Cost reliability optimal release policies for a software system with testing effort", OPSEARCH (India). 27(2), pp. 109-116, 1990b.

[17] Kapur PK, Garg RB,"Optimal release policies for software systems with testing effort", Int. Journal System Science. 22(9), pp. 1563-1571, 1990a.

[18] Cai KY,"A critical review on software reliability modeling", Reliability Engineering and System Safety. 32: pp. 357-371, 1991

[19] Bass FM,"A new product growth model for consumer durables", Management science. 15(5): pp. 215-224, 1969.

[20] Littlewood B, Verrall JL,"A Bayesian reliability growth model for computer software. Applied Statistics. 22: pp. 332-346, 1997.

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

[22] H. P. S. Dhami, Dr Anuj Kumar (July 2013),"Analysis of Software Design Erosion Issues, International Journal of Advanced Research in Computer Science and Software Engineering 3 (7), pp. 1393-1398. ISSN: 2277 128X, Impact Factor: 2.080.

[23] A. Kaur, H. P. S. Dhami, J. Kaur (2009): Component Based Software Engineering, IEEE-SOFT-3140, IEEE International Advance Computing Conference IACC-2009.

[24] H. P. S. Dhami (Sept.2016): Comparative Study and Analysis of Software Process Models on Various Merits, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 6(9), pp. 234-243. ISSN(online): 2277 128X, ISSN(print): 2277 6451 Impact Factor: 2.5.

[25] Harminder Pal Singh Dhami (Dec.2016): Improving Software Reliability, Productivity and Quality using software metrics, International Journal of Computer Science & Technology (IJCST) Vol.7 Issue: 4(1), ISSN 0976 – 8491 (online), ISSN 2229 – 4333 (print,), Impact factor 1.012.