IJCSE International Journal of Computer Sciences and Engineering Open Access
Survey Paper Vol.-6, Issue-6, June 2018 E-ISSN: 2347-2693

The Selection of Software Reliability Growth Models in Software Development Life Cycle

J.K. Mantri [*], R.S Bala [2]

[1*]Dept. of Computer Science and Engineering. ABIT, Cuttack, Odisha-753014, India
[2] Dept. of Computer Application, North Orissa University, Baripada, Odisha-57003, India

[*]*Corresponding Author: jkmantri@gmail.com, Tel.: +91-94380-84141*

Available online at: www.ijcseonline.org

Accepted: 14/Jun/2018, Published: 30/Jun/2018

***Abstract*—**The definition of software engineering might blast something like, "An organized, analytical approach to the analysis, design, development, use, reliability and maintenance of software."Software reliability is the probability that a software system will function without failure under a given environment and during a specified period of time. To be cost and time effective, reliability engineering has to be coordinated with quality assurance activities, in agreement with Total Quality Management (TQM) and concurrent engineering efforts. To build in reliability and maintainability into complex equipment or systems, failure rate and failure mode analyses have to be performed early in the software development life cycle (SDLC) and be supported by design guidelines for reliability, maintainability and software quality as well as extensive design reviews. There are different types of software reliability models (SRMs) used for different phases of the software development life-cycle. With the growing demand to deliver quality software, software development organizations need to manage quality achievement and assessment. In this paper, we present the utility of a software reliability growth model is related to its stability and predictive ability. Stability means that the model parameters should not significantly change as new data is added. Predictive ability means that the number of remaining defects predicted by the model should be close to the number found in field use.

***Keywords*—**Software reliability models, model classification, software reliability growth model, Time Between Failure, Fault Count Model.

I. INTRODUCTION

Recently, the rapid advancement of hardware, technology, proper development of software technology has failed miserably to keep pace in all measures, including productivity, quality, cost and performance. Software systems such as operating systems, compiler design, control programs, and application programs have become more complex and larger than ever. Naturally, it is to produce reliable software systems efficiently since the breakdown of the computer system, which is caused by software errors, results in a tremendous loss and damage for social life. Then, software reliability is one of the key issues in modern software product development. Many efforts have been devoted to the study of measuring software reliability quantitatively in the area of software engineering. There is several existing software reliability models, especially applicable to the software testing phase in the software development process, which are of great use to estimate and predict software reliability. During the software testing phase, a software system is tested to detect software errors remaining in the system and correct them. If it is assumed that the correction of errors does not introduce any new errors, the probability that no failure occurs for a fixed time interval, i.e., the reliability, increases with the progress of software testing. A software reliability model describing such an error detection phenomenon is called a software reliability growth model (SRGM) [1].

Rest of the paper is organized as follows: Section II describes the activities and phases of SDLC and also the total quality management (TQM). Section III covers the reliability predictions are used to evaluate design feasibility, compare design alternatives, identify potential failure areas, trade-off system design factors, and track reliability improvement. Section IV describes the definitions of software reliability. Section V presents the various software reliability growth models. Section VI covers Software reliability, as a part of software engineering, software quality, and reliability analysis. Its measurement and management technologies during the software life cycle are essential to produce and maintain reliable software systems. Section VII also presented proposed algorithm. And Section VIII gives the conclusion.

II. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

In this section, it provides a systematic approach to analyzing, designing, using, operating, and maintaining a software system. The standard IEEE computer dictionary has defended the SDLC as "That period of time in which the software is conceived, developed and used." A SDLC consists five successive phases. The phases are Analysis (requirements and functional), Design, Coding, Testing, Operating.

In general, the activities and phases of the SDLC are shown in figure 1. In the early phases of SDLC, a predictive model is needed because no failure data are available. This type of model predicts and the number of initial faults in software reliability improves through perfect functioning and debugging.

In 1970s, the "water fall model" is the first well-known SDLC. It divides the software development processes which was till then consider being an art and one monolithic activity into an engineering process company of several distinct and interactivity tasks. The software development project depends on many factors such as feasibility, cost benefit, availability of resources such as manpower, required technology, development know how to etc. and agreement of customer on cost, time schedule, quality and reliability.

Now, the total quality management (TQM) is considered to be one of the key technologies needed to produce more high quality software products. Also the total quality management (TQM) used for software development and all phases of SDLC.

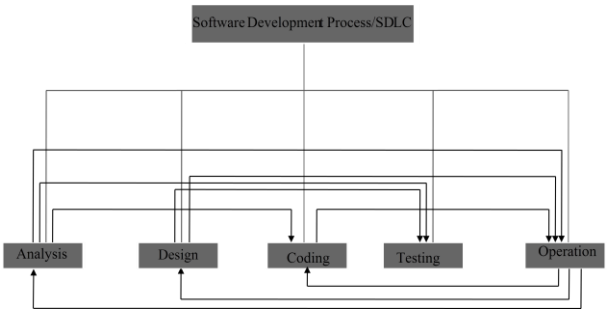

Figure 1. The Software Development Life Cycle (SDLC).

Basically, the concept of total quality management (TQM) means assuring the quality of products in each phase to next phase. Particularly, the quality control carried out at the testing phase which is the last stage of the SDLC. The testing phase is very important for SDLC. During the testing phase, the product quality and the software performance during operation phase are evaluated and assured. A lot of software faults introduced in the software system through the first three phases of SDLC by user or end-user activities are detected, corrected and removed. The fig.2 shows a SDLC called a "Waterfall paradigm".

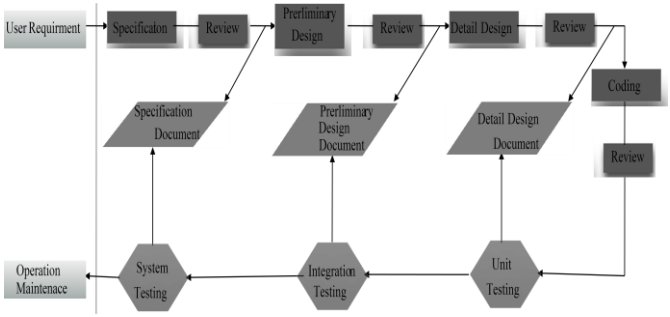

Figure. 2. The Software Development Process (Waterfall paradigm).

Therefore, TQM for software development, that is, software TQM has been emphasized. The software TQM aims to manage the SDLC comprehensively, considering productivity, quality, cost and delivery simultaneously and assure software quality shown in fig 3.

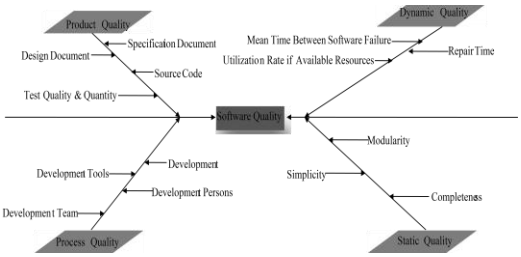

Figure 3. The Elements of Software Quality based on a cause and effect design.

In particular, the management technologies for improving software reliability are very important. The quality characteristic of software reliability is that computer system can continue to operate regularly without the occurrence of failure in software system [2].

III. RELIABILITY PREDICTION

Reliability predictions predict the failure rate of components and overall system reliability. These predictions are used to evaluate design feasibility, compare design alternatives, identify potential failure areas, trade-off system design factors, and track reliability improvement.

A. Reliability and Unreliability:

Definition of Reliability: The Reliability of a system or component is defined as the probability that the component or system remains operating from time zero to time t1, given that it was operating at time zero. It is denoted by R(t).

Definition of Unreliability: The Unreliability of a system or component is defined as the probability that the

component or system experiences the first failure or has failed one or more times during the time interval zero to time t, given that it was operating or repaired to a like new condition at time zero. It is denoted by F(t).

The relationship Reliability (R(t)) and Unreliability (F(t)) hold true since a component or system must either experience its first failure in the time interval zero to t or remain operating over this period. R(t) + F(t) = 1 or Unreliability (F(t)) = 1 – Reliability(R(t)).

B. Availability and Unavailability

- Definition of Availability: The Availability of a component or system is defined as the probability that the component or system is operating at time t, given that it was operating at time zero. It is denoted by A(t).
- Definition of Unavailability: The Unavailability of a system or component is defined as the probability that the system or component is not operating at time t, given that is was operating at time zero. It is denoted by Q(t).

Therefore, the relationship between Availability (A(t)) and Unavailability (Q(t)) holds true since a component or system must be either operating or not operating at any time: A(t) + Q(t) = 1.

- Definitions of Reliability Prediction:
- Failure Rates: Reliability predictions are based on failure rates. It is denoted by (λ).
- Conditional Failure Rate or Failure Intensity: It can be defined as the anticipated number of times an item will fail in a specified time period, given that it was as good as new at time zero and is functioning at time t. It is a calculated value that provides a measure of reliability for a product. It is denoted by λ(t).

The Failure rate calculations are based on complex models which include factors using specific component or system data. In the prediction model, assembled components are structured serially. Thus, calculated failure rates for components within the assembly. There are three common basic categories of failure rates:

- Mean Time to Failure (MTTR): It is defined as the total amount of time spent performing all corrective or preventative maintenance repairs divided by the total number of those repairs. It is the expected span of time from a failure (or shut down) to the repair or maintenance completion.
- Mean Time To Repair (MTTF): It is a measure of reliability for non-repairable systems. It is the mean time expected until the first failure of a piece of equipment. MTTF is a statistical value and is intended to be the mean over a long period of time and with a large number of units. Mathematically,

$$MTTF = \frac{1}{\lambda / t}$$

Where, t is the period of time.

- Mean Time Between Failures (MTBF): It can be calculated as the inverse of the failure rate, λ, for constant failure rate. Mathematically,

$$MTTF = \frac{1}{\lambda}$$

C. Failure Frequencies

There are four failure frequencies, which are commonly used in reliability.

- Failure Density f(t) :The failure density of a component or system is defined as the probability per unit time that the component or system experiences its first failure at time t, given that the component or system was operating at time zero.
- Failure Rate r(t): It is defined as the probability per unit time that the component or system experiences a failure at time t, given that the component or system was operating at time zero and has survived to time t.
- Conditional Failure Intensity (or Conditional Failure Rate) λ(t) : It is defined as the probability per unit time that the component or system experiences a failure at time t, given that the component or system was operating, or was repaired to be as good as new, at time zero and is operating at time t.
- Unconditional Failure Intensity or Failure Frequency ω(t): It is defined as the probability per unit time that the component or system experiences a failure at time t, given that the component or system was operating at time zero.

The relationships between failure frequencies parameters are as follows:

$$R(t) + F(t) = 1$$

$$f(t) = \frac{dF(t)}{dt}$$

$$F(t) = \int_0^1 f(u)du$$

$$r(t) = \frac{f(t)}{1 - F(t)}$$

$$R(t) = \ell^{-\int_0^1 r(u)du}$$

$$F(t) = 1 - \ell^{-\int_0^1 r(u)du}$$

$$f(t) = r(t)\ell^{-\int_0^1 r(u)du}$$

D. Repairable and Non-repairable Items

It is important to distinguish between repairable and non-repairable items when predicting or measuring reliability.

- Non-repairable items: It is defined as Conditional Failure Intensity or Conditional Failure Rate (λ(t)) is equal to the hazard rate or failure rate (r(t)), that is, λ(t)=r(t).
- Repairable Items: There is also the concern for availability, A(t), of repairable items since repair takes time. Availability, A(t), is affected by the rate of occurrence of failures (failure rate, λ) or MTBF plus maintenance time; where maintenance can be corrective (repair) or preventative (to reduce the likelihood of failure). Availability, A(t), is the probability that an item is in an operable state at any time.

$$Availability A(t) = \frac{MTBF}{MTBF + MTTR}$$

IV. SOFTWARE RELIABILITY

- The definitions of software reliability are as follows: "Software reliability is defined as the probability of failure free operation of a computer program in a specified environment for specified time." Or "Reliability of a software product essentially denotes its trust worthiness or dependability." Or "Reliability of a software product can also be defined as the probability of product working correctly over a given period of time."

The important points for software may be retired only if it becomes obsolete. Some of the contributing factors are change in environment, change in infrastructure or technology, major change in requirement, increase complexity, extremely difficult to maintain, deterioration in structure of the cost, slow execution speed and poor graphical user interface.

A. Difference between Software reliability and Hardware reliability

- Hardware reliability: Failure rate has a bathtub curve. The burn-in state is similar to the software debugging state. Material deterioration can cause failures even though the system is not used. Failure data are fitted to some distributions. The selection of the underlying distribution is based on the analysis of failure data and experiences. Emphasis is placed on analyzing failure data. Failures are caused by material deterioration, random failures, design errors, misuse, and environment. Hardware reliability can be improved by better design, better material, applying redundancy and accelerated life testing. Hardware repairs restore the original condition. Hardware reliability can be improved by better design, better material, applying redundancy and accelerated life testing. Hardware repairs restore the original condition. Hardware failures are usually preceded by warnings. Hardware components can be standardized. Hardware can usually be tested exhaustively
- Software reliability: Without considering program evolution, failure rate is statistically non-increasing. Failures never occur if the software is not used. Most models are analytically derived from assumptions. Emphasis is on developing the model, the interpretation of the model assumptions, and the physical meaning of the parameters. Failures are caused by incorrect logic, incorrect statements, or incorrect input data. This is similar to design errors of a complex hardware system. Software reliability can be improved by increasing the testing effort and by correcting detected faults. Reliability tends to change continuously during testing due to the addition of problems in new code or to the removal of problems by debugging errors. Software repairs establish a new piece of software. Software failures are rarely preceded by warnings. Software components have rarely been standardized. Software essentially requires infinite testing.

B. Similarity between Software reliability and hardware reliability

Software reliability is similar to hardware reliability. For hardware, components wear out, due to factors such as corrosion, shock, overheating, and aging. It is usually physical in nature and probabilistic. For software, we can use the same basic approach although we do not have the same physical issues. It is probabilistic. The probabilities vary over time, we can graph them and model them, and, for each model, there is a probability distribution function (PDF) [3,4,5].

V. SOFT WARE RELIABILITY MODELS

Software reliability evaluation is playing an important role in software reliability engineering. The role of statistics is also very important in reliability estimation for software [6]. Software Reliability Models are mainly used to measure the quality of the software. In this model, software is tested for a period of time, during which failures may occur. These failures cause a modification in design the new version of design is tested again. This cycle is repeated until design objectives are met. The software reliability classification shown in figure 4. In software reliability models, it is divided into two types deterministic model and probabilistic model In deterministic model, it can be divided into two types Halstead's software metric and McCabe's cyclomatic complexity metric.

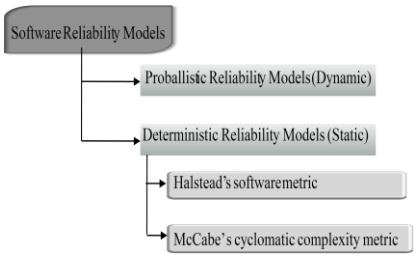

Figure. 4. The classification of software reliability models.

A. Deterministic Model

- Halstead's software metric

Typically, the measures depend on program size, control structure, or the nature of module interfaces. The most widely known measures are those devised by Maurice Howard Halstead in 1977 and his colleagues that are collectively known as software science. The Halstead measures are functions of the number of operators and operands in the program. The major components of software science are n1 the number of unique operators, n2 the number of unique operands, N1, the total number of operators, and N2 the total number of operands. Halstead defined the volume, V, of a program to be

$$V = (N_1 + N_2)\log(n_1 + n_2)$$

and program difficulty, D, to be

$$D = \frac{n_1 \times N_2}{2n_2}$$

Halstead derived a number of other measures. The most extensively studied of these is an estimate of the effort, E, required to implement a program:

$$E = D \times V$$

Halstead's bug prediction:

$$B = \frac{V}{3000}$$

- McCabe's cyclomatic complexity

The complexity is defined by the execution time and storage required to perform the computation. If the interacting system is a programmer, then complexity is defined by the difficulty of performing tasks such as coding, debugging, testing, or modifying the software. The term software complexity is often applied to the interaction between a program and a programmer working on some programming task. McCabe's cyclomatic complexity metric based on cyclomatic number *V(G).* Mathematically, The cyclomatic number *V(G)* of a graph G with n vertices, e edges, and p connected components is $V(G) = e - n + p$

B. Probabilistic Models

The dynamic software reliability models also known as the probabilistic models which include the failure rate model (times between failure models), failure or fault count model (NHPP models), error or fault seeding model and reliability growth model etc. The hierarchy of dynamic Software Reliability Models shown in figure 5.

- The Failure Rate Model: It based on the assumption that is time between failure models. The time between i-1th and ith failures is a random variable which follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. It estimates of the parameters are obtained from time between failures, mean time to next failure then obtained from the fitted model.
- The Failure Count Models: It is based on the number of failures that occur in each time interval. The random variable of interest is the number of faults (failures) occurring during specified time intervals. It is assumed that failure counts follow a known stochastic process. Usually a Poisson distribution with a time dependent will be discrete or continuous failure rate. The time can be calendar time or CPU time Parameters of the failure rate can be estimated from the observed values of failure counts and then the Software reliability parameters are obtained from the appropriate expression.
- Error or Fault Seeding Model : In this model, a predefined number of artificially generated errors are "incorporated" in the program code. After that, test runs are used to detect the errors and to examine the ratio between actual and artificial errors based on the total number of detected errors. Naturally, the artificially generated errors are not known to the testers.
- Input Domain Based Category: when the test cases are sampled randomly from well known operational distribution of inputs program. By finding all unique paths through the program and then execute each and everyone it is possible to guarantee that everything is tested.

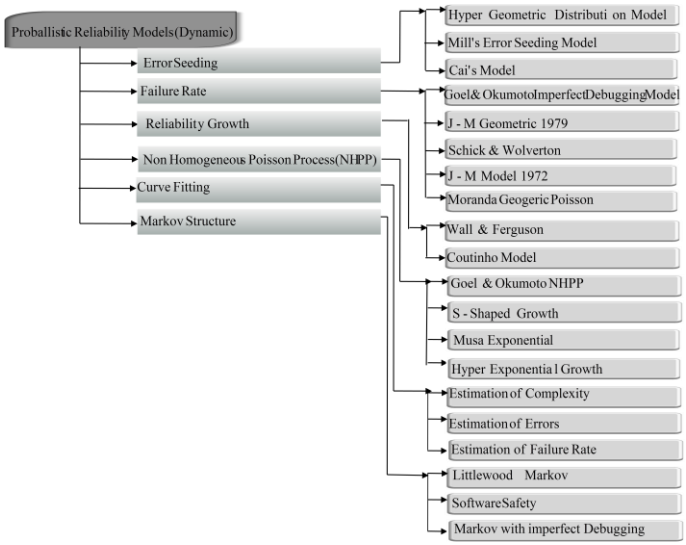

Figure 5. The hierarchy of dynamic Software Reliability Models.

VI. SOFTWARE RELIABILITY MODELS

Software reliability, as a part of software engineering, software quality, and reliability analysis. Its measurement and management technologies during the software life cycle are essential to produce and maintain reliable software systems. IEEE Std. 982.2-1988 states "A software reliability management program requires the establishment of a balanced set of user quality objectives, and identification of intermediate quality objectives that will assist in achieving the user quality objectives." ISO 9000-3 specifies measurement of field failures as the only required metric at a minimum, some metrics should be used which represent reported field failures and/or defects from the customer view point. The supplier of the software products should collect and act on quantitative measures of the quality of these software products".

The software reliability growth model (SRGM) can be estimated to be a mathematical expression which fits the experimental data. It may be obtained simply by observing the general trend of reliability growth. However some of the models can be achieved analytically by making some assumptions about the testing and debugging process. Further, there are essentially two types of software reliability models. The first type of models are usually called "defect density" models and use code characteristics such as lines of code, nesting of loops, external references, input/outputs, and so forth to estimate the number of defects in the software. The second type of models are usually called "software reliability growth" models. These models attempt to statistically correlate defect detection data with known functions such as an exponential function. If the correlation is good, the known function can be used to predict future behavior.

A. Software Reliability Growth Models (SRGMs) in SDLC

The Software reliability models have been classified according to Software Development Life Cycle phases, based principally on the phases of software development life cycle(SDLC) during which the model is applicable. The SRGMs are further classified based on the software failure phenomenon, as given in Figure 6. Most of the existing models can be used during the design, implementation and testing phase. The Software Reliability Growth Models (SRGMs) is one of the best way to measure software reliability. This type of model captures failure behavior of software during testing, and extrapolates it to determine its behavior during operation. Hence, this category of models uses failure data information and trends observed in the failure data to derive reliability predictions Wood in 1996. These models are also called black-box models. The SRGMs encounter major challenges. First, the software testers seldom follow the operational profile to test the software, so what is observed during software testing may not be directly extensible for operational use. Secondly, when the number of failures collected in a project is limited, it is hard to make statistically meaningful reliability predictions. Thirdly, some of the assumptions of SRGM are not realistic, that is, the assumptions that the faults are independent of each other; that each fault has the same chance to be detected in one class and that correction of a fault never introduces new faults. Further, the SRGMs are classified as Failure Rate Model (times between failure models), Failure or Fault Count Model (NHPP models), Error or Fault Seeding Model and Reliability Growth Model [7,8].

B. Classification of Software Reliability Models

In [9], the software reliability models are classified as Early prediction Models, Architecture Based Models, Hybrid Black Box Models, Hybrid White Box Models, Software Reliability Growth Model and Input Domain Based Models.

- Early prediction Models: In this model, it uses the characteristics of the software as well as the characteristics of the software development process. The development process characteristics are used to extrapolate the software operational behavior. This model uses information collected during the reviews performed in the requirements, design, and implementation development phases. This information includes the fault statistics that are used to predict the reliability of the system.(Smidts et al., 1996) proposes a reliability model based on a systematic identification of software process failure modes and their likelihoods. This model uses a Bayesian approach where the prior knowledge is provided by failure process identification.
- Architecture Based Models: In this model, it puts emphasis on the architecture of the software, and derives reliability estimates by combining estimates obtained for the different modules of the software Gokhale et al. in 1998. Different approaches for the architecture-based reliability estimation of the software are based on the

Module identification, Architecture of the software, Failure behaviour and Combining the architecture with the failure behaviour. Further, the architecture based software reliability models are classified into state based models, path based models, and additive models.

- Hybrid Black Box Models: All SRGMs are of the black box type since they only consider failure data, or metrics that are gathered if testing data are not available. Black box models do not consider the internal structure of the software in reliability estimation and are called as such because they consider software as a monolithic entity, a black box. The hybrid black box models combine the features of input domain based models, and SRGMs.
- Hybrid White Box Models: White box software reliability models consider the internal structure of the software in the reliability estimation as opposed to black box models which only model the interactions of software with the system within which it operates. In this model, it uses selected features from both white box models and black box models. However, since these models consider the architecture of the system for reliability prediction, therefore these models are considered in hybrid white box models.
- Software Reliability Growth Model :This type of model captures failure behavior of software during testing, and extrapolates it to determine its behavior during operation. Hence, this category of models uses failure data information and trends observed in the failure data to derive reliability predictions Wood (1996).Further, the SRGMs are classified as failure rate models and NHPP models. To incline practitioners to use a SRGM, it has to be simple in concept and inexpensive to collect the required input data. The analytical research on SRGMs is extensive as a number of models are proposed with individual assumptions. Ehrlich et al. (1990), Wood (1996), Wood (1997), String fellow and Andrews (2002), and Jeske & Zhang (2005) have given practical experiences of use of SRGMs in a variety of contexts.
- Input Domain Based Models:The basic approach taken here is to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program. Because of the difficulty in obtaining this distribution, the input domain is partitioned into a set of equivalence classes, each of which is usually associated with a program path. 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. This category include that assess the reliability of a program when the test cases are sampled randomly from well-known operational distribution of inputs program.

VII. ALGORITHM

We have proposed Software Reliability growth Model Selection algorithm that helps in the selection of models by applying the SDLC.

A. Algorithm for Software Reliability Growth Model Selection

Step: 01 The software reliability models are used in different phases of the Software Development Life Cycle, it is required to determine which of these models can be used in a particular life cycle phase.

Step: 02 In a particular phase, it is deciding criterion which will reduce the number of candidate models in that phase.

Step: 03 After step number 2, we will be left with the remaining criteria that are applicable in that phase. Now these criteria will be applied. First of all importance weights will be assigned to these criteria. Suggested weights are assigned to these criteria according to the phases of SDLC.As per criterion weights, two possibilities can exist.

Step: 03a. If all the remaining criteria apply on all models in that phase.

Step: 03b. If for any model in that phase, the number of criteria applied is less.

Step: 04 In this step, after applying the criteria, a weight will be assigned to each model for every criterion according to the level by which it fulfills the criterion. These weights are called applicability weights. If weight = 1 then satisfied otherwise 0(unsatisfied).

Step: 05 The respective applicability weights and criterion weights are multiplied for each criterion and this number is summed for every model to obtain the total points for that model.

Step: 06 Select the applicable model to the assessment of software reliability are called SRGM. SRGM are useful for estimating how software reliability improves as faults are detected and repaired.

Step: 07 Software Reliability Models can be classified based on Failure History and the other one is Data Requirements.

Step: 08 Based on Failure History for probabilistic models include Failure Rate Model (times between failure models), Failure or Fault Count Model (NHPP models), Error or Fault Seeding Model, Reliability Growth Model. And the other one is Data Requirements for empirical model, analytical mode.

Step: 09 Selected Models for Parameter Estimation and Comparison Criterion which are popular and frequently used for comparison of SRGM.

VIII. SCONCLUSION

In this paper, Software reliability is a measuring technique for defects that causes software failures and we have classified software reliability growth models according to Software Development Life Cycle (SDLC) phases. We have identified and defined a number of criteria for software reliability model selection. We have proposed an algorithm based on these criteria for the selection of software reliability growth models.

REFERENCES

[1] Yamada, S. and Osaki, S., "Software Reliability Growth Modeling: Models and Applications, IEEE Trans. On Software Engineering, vol. SE-11, no. 12, pp. 1431-1437, December1985.

[2] Hoang Pham, "Software Reliability", Springer-Verlag Singapore,Pte. Ltd 2000.

[3] A. Birolini, "Reliability Engineering Theory and Practice", @ Springer-Verlag Berline, 1999.

[4] http://www.reliabilityeducation.com/ReliabilityPredictionBasics.pdf.

[5] Linda M. Laird and M. Carol Brennan, "Software Measurement and Estimation", Copyright John Wiley & Sons, Inc., 2006.

[6] A. Loganathan, R. Jeromia Muthuraj, "Importance Of Environmental Factors Affecting Software Reliability", Global And Stochastic Analysis, Vol. 4 No. 1, 119-125, January 2017.

[7] Alan Wood,"Software Reliability Growth Models",© Tandem Computers, 1996.

[8] A. L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability", IEEE Transactions on Software Engineering, Volume SE-11. Number 12, pages 1411–1423, December 1985.

[9] Ch. Ali Asad, Muhammad Irfan Ullah, Muhammad Jaffar-Ur Rehman, "An Approach for Software Reliability Model Selection", Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04), IEEE, 2004.

Authors Profile

Mr J. K. Mantri pursed Master of Technology in Computer Sc. from Utkal University in year 2002 and Ph.D degree in Computer Application from Sambalpur Univeresity in year 2010.. He is currently working as Reader in Department of Computer Application, North Orissa University,India,. He is a life member of ISTE and ISCA. He has published 5 books and more than 58 research papers in reputed international journals including Thomson Reuters (SCI & Web of Science) and conferences including IEEE, & Springer. His main research work focuses on Cryptography, Software Engineering &Computational Intelligence. He has 28 years of teaching experience.

Mr. R. S. Bal pursed Master of Technology in Computer Science and Engineering degree from Biju Patnaik University of Technology in year 2009 and Maseter in Computer Application degree from Fakir Mohan Univeresity in year 2002. He is currently working as Assistance Professor in Department of Computer Science and Engineering ,Cuttack 75301, BPUT,India. He is a life member of ISTE. He has 11 research papers in reputed international journals. Served in guest faculty in different premier institutes namely Institue of Management & Information Technology, Cuttack, Ravenshaw University, Cuttack and IGNOU study center, Cuttack, Odisha, India. Having research interests include Wireless Sensor Network, Software Engineering &Computational Intelligence. He has more than 15 years of teaching experience and currently pursuing Ph.D.

