International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-8, June 2020

Software Reliability Prediction and Estimation

Sampa ChauPattnaik, Mitrabinda Ray

Abstract*: *The rapid growth of the software products tends to increase the software application complexity. The complexity affects the software quality which is achieved by means of software reliability. It is desirable to perform reliability analysis at the early phase of Software Development Life Cycle. The paper conducts a thorough analysis on Bayesian model and Markov model which are common for both reliability prediction and estimation. We evaluate the state based model and path based model for reliability assessment and results obtained in both are same.

Keywords: Software reliability, Reliability Metrics, Markov Model, Reliability Models, Reliability Prediction, Reliability Estimation

I. INTRODUCTION

The IEEE 982.1 1988 defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." Software reliability modelling is frequently concerned with measuring the activities of software and to evaluate current software reliability status using failure data. Thus Reliability is defined as the failure-free operation of a system which is calculated by the probability over a naive time within a specified situation for a particular reason. Software reliability is the important features of the quality. For any system, it is very difficult to achieve certain level of reliability due to high complexity [45]. Reliability improvement let by examining the correct-incorrect software service. Reliability can also be deliberate as a part of Dependability. (The responsibility of a computer system such that assurance can be made on service at delivers is known as Dependability) In other words, a software product is reliable; meaning represents its trustworthiness.Fig.1 shows the various properties of dependability.

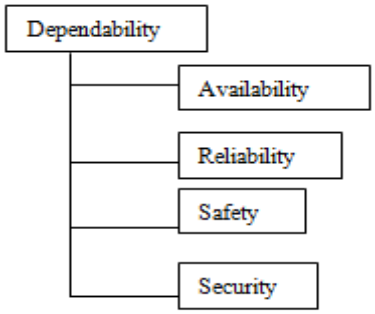

Fig.1.Principal dependability properties [50]

Revised Manuscript Received on June 30, 2020.
***** Correspondence Author
Sampa Chau Pattnaik*, Department of Computer Science and Engineering, SOA University, Bhubaneswar, India.
Mitrabinda Ray, Associate Professor, Department of Computer Science and Engineering, SOA University, Bhubaneswar, India.

As shown in Fig.1, Availability is associated with deliver system services as and when required, similarly safety is associated with the effect of failure of the system and security is associated with unwanted assessments. Software reliability of a system increases by means of minimizing the errors and fault detection. Reliability analysis is used for given software component can be represented as block diagrams, Markov Model[52].(The transfer of control among the components follows Markov property, that is, the next component to be executed depends only on the present component.) Reliability analysis explains with the diagram given below, i.e.

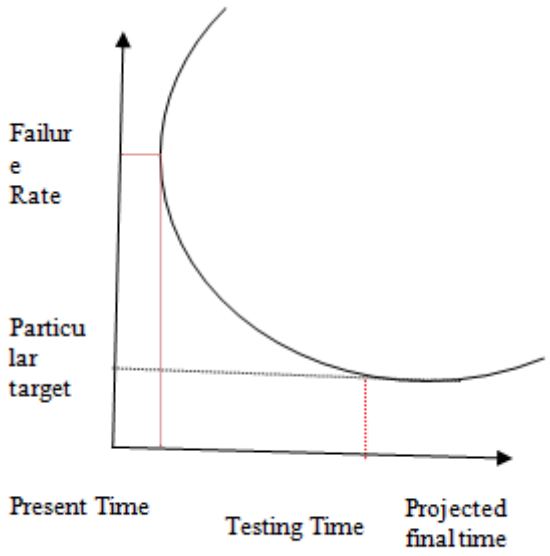

Fig. 2. Testing Vs Failure Rate [32]

The software system's failure rate normally decreases which is shown in Fig.2 where the x-axis represents finishing time and y-axis as failure rate between the testing time of a given project (i.e. present time and final time). The curve of the failure rate (using statisti cal data) calculates the software reliability. Thus we understand that, if software failure decreases then reliability increases. Thus the software reliability defined as the following features: i.e. failure, time and specified operational environment. A failure in terms of software is an erroneous result with unpredicted software performance perceived by the user. The defect can be used as a common term to refer to either a fault (cause) or a failure (effect). Reliability calculation can be done with respect to *time*, program runs or number of transactions. Program runs have several operations. Thus the operational profile [21], is the set of independent functions (similar runs) that a software system executes with their associated probabilities. The steps involved in operational profile are dividing the execution into: i) test runs, ii) categorize the input space, iii) and the input space divided into functions. Software reliability has been widely studied for the past two decades.

The purpose of this paper is to provide an overview of the existing research in the area of reliability analysis at both code level as well as architectural level, discuss its limitation. Various existing methods used for reliability analysis are discussed. The primary objective is to provide a guideline to estimate the system reliability based on component reliabilities. There are broadly two approaches exist for reliability estimation: path based approach and state based approach. In path based approach, a control flow graph (CFG) is generated and transition probability from one component to another component is considered. Using CFG, several execution paths, starting from initial component to the end component, are enumerated. In state based approach, the probabilistic control flow graph of the system is mapped to a state based model.

Reliability assessment consists of test case selection, test case execution, test result collection and updating the estimated reliability if required. Nikora and Lyu [32] propose four mechanisms during the Software Reliability Engineering process. These are: Purpose of the reliability, Testing based on Operational profile, Reliability modeling and measurement, and Reliability validation. To measure the reliability of a system, they propose the following steps:

Step-1: Define **reliability objective**. It is defined either in the early phase (architecture level reliability analysis) of software development life cycle or testing phase (code level reliability analysis.

Step-2: **Testing method is applied** on the system's functions. For reliability analysis, code based testing and usage based testing are applied. In code based testing, each and every statement in the program is executed at least once during the test, whereas in usage based testing, faults that are responsible for frequent failures of the system are detected. In this step, test cases are generated for a specific application. Test case selection is carried out by operational Profile. The following assumptions are taken during the testing conducted for the reliability assessment:

a. The code is frozen.

b. A test case either succeed or failure.

c. The failure of the software at the current time t is dependent on the input provided at that time and is independent of past executed inputs.

d. The operational profile for the software is $\{D_i;P_i;\ i=1;\ 2;.;m;\ \}$, where P_i represents the execution probability of the input sub-domain D_i, and

$$\sum_{i=1}^{m} P_i = 1$$

i.e. Total n number of test cases is allowed to run

Step-3: Collect the **failure data.** Thus, Reliability measure by calculating, Probability of failure = Number of failure cases/Total number of cases under consideration.

Step-4: Collected failure data are used with the help of **software reliability tools [**52,14,**56,27] which are specified by the software reliability model to measure the reliability**. When the targeted reliability value is not obtained, SRGM (Software Reliability Growth Models) are applied for reliability improvement [11,4] (The number of faults (as well as the failure rate) of the software system reduces when the testing progresses, resulting in growth of reliability, these models are often called SRGMs.)

Step-5: Check the validity of the specified **reliability**. In this step, a comparison is done on estimated and the actual reliability of the system.

The estimated parameters for reliability analysis are probability of failure, operational profile, etc. These parameters are explained below:

1) Failure-free behavior including the key factors like critical or minor failure severity, impact of failure , complex vs partial failure extent , probability of failure

2) Operational profile key factors like frequency of execution of different system services and operations, user inputs etc [19].The basic difficulties on reliability prediction arise due to the distribution effect of testing process by not applying an actual operational profile for the specific system .To overcome this problem, appropriate test case selection process is required in which operational profile plays a major role and helps to improve the reliability of the system under test [41].

The outline of the paper is arranged as follows: Section II provides an overview of the theory, metrics used in reliability, existing models and techniques. Section III describes the reliability prediction and estimation with theorems and corresponding models used for both. It also illustrated in brief how the identified restrictions could be conquered. Section IV summarizes the different tools used to predict or estimates the software reliability. Section V explains an example for reliability assessment and Section VI discuss overall summary of the paper.

II. REALIABILITY THEORY

The performance of reliability for both hardware and software are very different. Failures in hardware happen due to component wear and tear [18]. The hardware faults fixed, either replacing or repairing the failed part. But in case of failures in software error is tracked moreover at design or the code is changed. Thus, for hardware its reliability is repaired and maintained before the failure took place; whereas when a software failure is unchanged, the reliability may either increase or decrease. [10**,** 50] Different metrics are used to evaluate the reliability of a system. Those are: Probability of Failure on Demand (POFOD), Rate of Fault Occurrence (ROCOF), and Mean Time to Failure (MTTF), Availability = MTBF / (MTBF+MTTR), Where MTBF = Mean Time Between Failure and MTTR = Mean Time to Repair and Reliability = MTBF / (1+MTBF). Hence, some mathematical models are used to measure both software and hardware reliability.

There are several reasons, that the system will fail by time a) Failure rate - a failure will occur in the given time interval, not earlier than the given time. This can also be written as: b) Hazard rate - boundary of the failure rate within the time interval that approaches to zero. As the software system's failure is a random variable of time, which associated with probability density function, (pdf) $f_T(t)$, and cumulative distribution function(cdf), $F_T(t)$, Thus reliability of a system can be

$$\mathrm{Re}l_T = \Pr(T \le t) = \int_0^\infty f_T(t)d(t) \qquad (1)$$

Let T =the failure time of a physical system. Equation 1 describes the reliability definition in terms of probability distribution function whereas Equation 2 describes the reliability definition in terms of cumulative distribution function.

$Rel_T(t_0)$ = reliability function where, the probability of a software not failed by time t_0 .Then

$$Rel_T(t_0) = \Pr(T \geq t_0) = 1 - F_T(t_0) \quad (2)$$

The hazard rate function, $H_T(t)$ is important in reliability modelling. As, the software will fail during the interval (t, t+Δt) with conditional probability before time *t* then failure occurs at T time as

$$H_T(t)\Delta t = \Pr(t < T < t + \Delta t \mid T > t) \quad (3)$$

Equation 3 describes hazard function for the failure of the system, is a conditional probability density function.

Therefore, hazard rate can be represented as,

$$H_T = \frac{f_T(t)}{1 - F_T(t)} = \frac{f_T(t)}{Rel_T(t)} \quad (4)$$

Cumulative distribution function (cdf) is used for the system's reliability, conditional reliability in terms of time and reliability computation of components, etc. Similarly, the probability distribution function (Pdf) is used for the failure distribution of each component's for a given system [as mentioned in probability theory].

III. REALIABILITY ANALYSIS

This section deals with various types of reliability prediction and estimation models used in architectural level and code level. Section A. describes the overview of reliability analysis, black box and white box reliability analysis and architectural analysis. It also gives introduction to software reliability models: prediction and estimation model. Section A gives details about the classification of software reliability prediction and estimation models. It also discusses various metrics such as time between failures metric, fault count metric, fault seeding metric and input domain metric used in the reliability models. Section B and Section C describe parameters used for the reliability prediction models in code level and architectural level. Section D and Section E describe parameters used for the reliability estimation models in code level and architectural level.

This paper explains the reliability estimation method at code level in terms of both time period & coverage of code [35] in section A1. Next, we discuss the reliability estimation at architectural level. From the literature survey, we find that the researchers focus on the reliability estimation techniques using Bayesians Model as well as Markovian Model [55]. We observe that Markov Model is generally used for single failure system where as it is also extended into multiple failure system one at a time [6,41]. Reliability analysis of a software application depends on its architecture and individual component reliabilities, explain the sensitivity of the application reliability to the component reliabilities. Several authors [40,38,48] describe architecture based models for estimation of system reliability. These estimation techniques must be applicable to large-scale, complex systems, often lack of the failure behavior. In architectural level, system architects focus primarily on modeling the system, and ignore the details of how to gather information to estimate the model's parameters. Even if, software requirements are specified clearly and functionalities are defined accurately, it does not necessarily guarantee the software will be free of errors. For example, the architecture process also needs to be properly defined by means of documenting business goals, architecture drivers, quality attributes, and tactics for addressing each scenario.

A. Different Types of Reliability analysis: It is conducted based on source code or the behavior of the system or the architect tural level data. Basically, the reliability analysis is classified into three types. They are given below:

- Black box reliability analysis: In this loom [26,28] internal information of the software are not considered. This type of testing is non-functional requirements based on the output results in early phases for the reliability estimation. For this type of analysis, *Petri net* method [26,18] is used, where the total defects found and defects repaired are considered as random processes. The attributes are: Failure rate = total no of defects found per total time spent on review /testing, Repair rate = total no of defects repair/total time spent for rework and MTTF = 1/Failure rate; MTTR = 1/ Repair rate can be calculated.
- Software metric based reliability analysis: Reliability estimation is based on the static analysis of the software 13, 42] (e.g., lines of code, number of statements, complexity) or its development process and conditions (e.g., developer experience, applied testing methods). The above attributes are used in different reliability models i.e. McCall's, Boehm's, FURPS and ISO 25010 that replaced the ISO 9126 [13,42].
- Architecture-based reliability analysis: It is also called component-based reliability estimation (CBRE), or grey box approaches [36, 40,43, 20]. The assessment of the system reliability can take place from software component's architecture. Fig. 3 shows the classification of architecture based software reliability model.

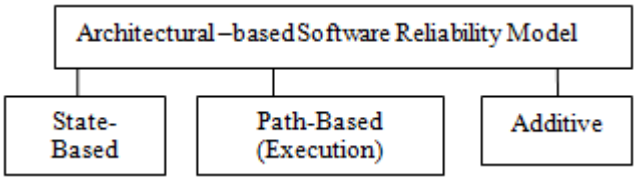

Fig. 3.Classification of architectural-based software reliability model [54]

During the literature survey, the component-based reliability model developed by Cheung [44] is considered for system, where the transition diagrams are used. In this model, a state corresponds to the execution of a single component and the transition probability which is in the form of Markovian from one state to another is obtained from the application of the system. Assuming that a component, c, represents the architecture, there will be c states in the Markov model. The transition matrix T is obtained as follows:

$$\begin{cases} T(i,j) = Rel_i * Pij, if\ Si\ can\ reach\ Sj \\ T(i,j) = 0, if\ Si\ can't reach\ Sj \end{cases} \quad (5)$$

T (i,j)= transition probability from state S_i to state S_j
Then, the overall system reliability can be computed as follows:

$$Rel = (-1)^{c+1} \frac{|E|}{ID - T} * Rel_c \quad (6)$$

Where ID = identity matrix of size c x c. |ID-T| is the determinant of matrix and |E| = the determinant of the remaining matrix excluding the n^{th} row and first column of the matrix ID-T).For this analysis, several software reliability models are needed. Once the model is setup within the given data, then used to estimate current permanence of the software and formulate predictions about the program for future reliability.
Software Reliability Models (SRM**):** A software reliability model indicates the failure process on the major issues that influence it: fault introduction, fault removal and the operational environment. A software reliability growth model [15,17] is used after the prediction period, as the result of testing and fault correction improved the reliability. In black box reliability is estimated from failure during testing or operation. Software metrics based reliability is estimated from software analysis done before testing process. The objective of software metric analysis is to calculate the residual fault frequencies or failure frequencies which have to be expected when executing the software. Thus, black box reliability and software metrics based reliability analysis occur during the code level. Reliability analysis occurs also during the architectural level or design phase. Software reliability measurement includes two activities, they are:

- Prediction Model (PM**):** Reliability prediction conducted before program execution. It is determined from the characteristics of software product and the process of development. The characteristic of a software product includes size of the program, branchiness, loopiness and processing speed. During development process, the following factors are consider for reliability prediction:
  a) The number of time specification changes.
  b) Thoroughness of designing document.
  c) Average programmer skill level
  d) Percentage of review accomplished
  e) Percentage of code read

The objective is to get the fault exposure ratio[23].It is represented as the proportion of time that a hypothetical encounter of a fault based on processing speed and program size that would lead to failure. It measures software metrics including the early failure rate λ_{e0} to make a decision for the future software reliability based on, ultimate failure rate, error per executable lines of code, fault profile, as well as the parameters of a software reliability growth model. Prediction involves availability of failure data. Similarly, the metrics obtained from the software development process and the characteristics of the resulting product can be used to determine reliability of the software upon testing, when failure data are not available.

- Estimation Model (EM**)**: Estimation is conducted to estimate the number of remaining faults before implementation of the product. The objective is to estimate failure intensity during test. It determines up to date or present software reliability information using statistical techniques to failure data acquired in system test. Estimation is a measure concerning on reliability from the earlier period of testing to the present point.

The measurement is to predict the additional time required for testing the software and the estimated reliability of the software when the testing is completed [36]. The above models are based on monitoring and gathering failure data, analysing with statistical inference. Table -1 talk about the characteristics of the prediction model and estimation model.Table-1 shows the essential issues like types of data (statistical), when the software is used during the development cycle and regarding the time frame of the prediction models as well as estimation models.

Table-I: The difference between PM and EM

Subject concern with Models	Prediction Model	Estimation Model
Design of data	historical data like number of failures, failure impact , failure origin etc used	make use of data from the up to date software advance products
When to use in development cycle	used during the early phase of the testing	Usually made later in life cycle; not typically used in concept or development phases
Outline of the time	can be done at a little future time	can be done moreover at present or a little future time

A1. Classification of Software reliability models: Several software reliability models are there. Fig.4 describes the different models used in different stages of software development. Fig -4 shows the models use for reliability testing either at code level or at the design/architectural level. Thus, researchers are doing reliability testing for given system using time-domain models, which is also called SRGM (Software Reliability Growth Models)[11,4]. To predict the performance of the program, these models are applied on the failure data at the earlier period during testing with a given operational profile [36)].In software development life cycle phases, several models are used for reliability prediction as well as estimation for getting better results and advises whether that software product can be further used or not. In Fig-4, it shows that in requirement phase Raleigh Model, Musa's Prediction Model, Rome Laboratory Model etc are used. Similarly at design phase and the implementation phase several architecture-based models (i.e. Gokhale et al model, Shooman model, Yacoub, Cukic and Ammar model). In testing phase several software reliability growth models like Goel Okumoto NHPP Model, Musa Okumoto NHPP Model, Musa Poisson Execution Time Model,. Jelinski Moranda Model, Littlewood Verall Models are used. For validation the input domain based models are used. According to the model assumptions, software reliability models are classified as 4 metrics [32]. They are:

- Fault Count: The number of faults is counted in precise time intervals more willingly than time between failures. The failure rate parameters are estimated from the experimental value of failure count or from failure times. With this metrics several models are employed.

Those are Shooman Model [37], Goel-Okumoto NHPP Model [1,2,4], Goel Generalized NHPP Model, and Musa Execution Time Model [21].

- Time between Failures: It is the most basic model for software reliability estimation. The time between i^{th} and $(i-1)^{th}$ failures data consider for analysis. JM De-Eutrophication Model [57], Goel & Okumoto Imperfect Debugging Model[2,4], Littlewood-Verrall Bayesian Models [8] are used for time between failure.
- Fault seeding: The known faults are identified during the testing phase. Using different models (Mills Seeding Model [4], Basin Model [49], Lipow Model [34]), the reliability of the software is estimated.
- Input Domain: The test cases are randomly created from an input domain. It is not so easy to divide the input domain into equivalence classes. The reliability is considered from the number of failures inputted or execution of the test cases successfully. Nelson model [16], Ramamoorthy and Bastani models [12] are used for input domain.

Table-2, Table-3, Table-4, and Table-5show the metrics use for models, different reliability model's name prediction or estimation for system, features of the different models with mathematical representation and the objective of the reliability models.

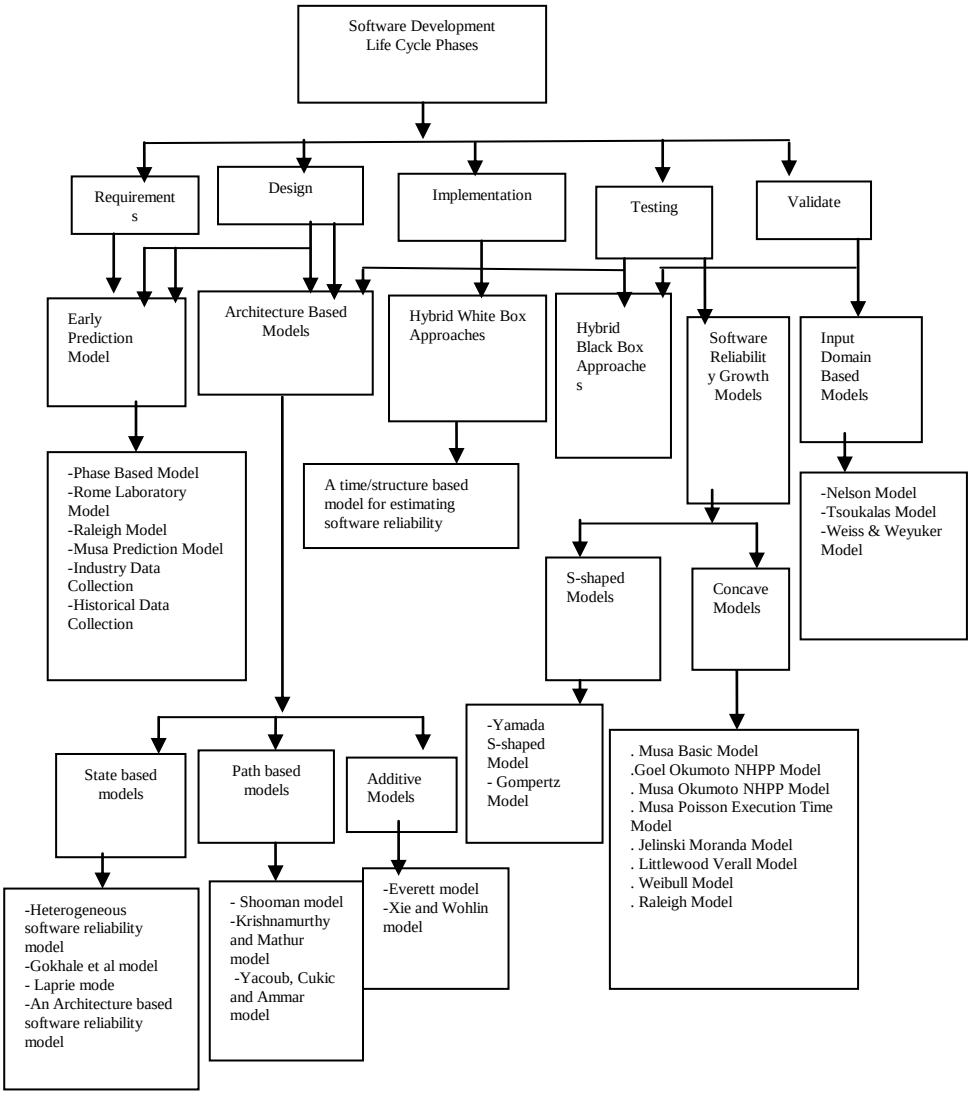

Fig.4.Overall Classification of software reliability models [36]

Table 2 contains the time between failure model like Musa's model, Goel model and Little wood models are described by the model assumptions, and features. The mathematical model for each model described as follows:

i) Musa's Model: It is the most primitive model for evaluating software reliability. During testing, the number of software faults independent of each other and equally likely reason for failure. Thus, a detected fault is removed within a trivial time and no extra faults are computed. It is a black box model. Mathematically express below:

$H(t_i) = \phi[N-(i-1)]$,Where t_i = time between $(i-1)^{th}$ and i^{th} failure, H(t) = hazard function, Assumes N =no. of assuming the ratio of expected number of faults corrected to expected number of failures experienced is a constant.

ii) Goel Model: Practically the detected faults removal with confidence not always happened, due to the uncertainty in the correctness of a program. Goel proposed an imperfect debugging model. Mathematically as follows:

$$H(t_i) = [N - pf(i-1)]\lambda$$

N=no. of error at the starting of testing phase
pf= the probability of fault debugging

λ = the failure rate per fault
This model can be applied for software reliability estimation during the development process.
iii) Little wood model: The number of errors in the program should not be considered as the software reliability. In this model, failure data is unavailable. This model is based on Bayesian Software Reliability Growth Model[7], where the time failures (time between $(i-1)^{th}$ to i^{th} failure) are independent exponential random variables. Mathematically given as:

$$f(t_i \mid \lambda_i) = \lambda_i e^{-\lambda_i t_i}$$

$$\lambda(t) = (\alpha - 1)(N^2 + 2B\varphi(\alpha - 1))^{-1/2}.$$

The probability density function is denoted by $f(t_i \mid \lambda_i)$.

This model comes under black box testing. The main reason is here that, for better reliability the programmer carries out a repair process to make the program better than before the failure occurs. The exponential distributions consider for time between failures as being a software interaction (a programme may work perfectly with data from a sample space) is a random process (Poisson process).
Where: λ(t)= failure intensity ,α = programmer quality
B = fault reduction factor, φ = fault per hazard rate
iv) Goel Okumoto(GO) NHPP model: The expected number of failures in GO model treated as Poisson distribution with a non decreasing mean function. The no. of inter failure intervals are not correlated. Mathematically as follows:
f= the fault detection rate per fault:

$$\Pr\{N(t) = y\} = \frac{(o(t))^y}{y!} e^{-o(t)} \qquad \text{m(t)}$$

estimated no. of failures occurred by means of time t

$$o(t) = p(1 - e^{-dt}) \text{ And } \lambda(t) = o'(t) = pde^{-dt}$$

p= the predictable no. of failures chosen experimentally
d= the fault detection rate per f

Table-II: Times between Failure Models

Models	Features	Objective of the Model
JM De-Eutrophication Model [57]	1. Each fault is sovereign of others. 2. Identify the removal faults with neglecting time 3. No new faults are considered during the process.	1. The fault removal is same 2. Data requirement for this model is the elapsed time between failures or actual time the software failed. 3. The elapsed time is proportional to the remaining faults.
Goel & Okumoto Imperfect Debugging Model [4, 2]	1. It is the annex of JM model. 2.The various faults considered as a Markov process	1. The no. of faults is treated as the transition probability of faulty debugging. 2. The transition time is exponentially distributed.
Littlewood-Verrall Bayesian Model [7,8,9]	1. Number of errors in the program should not considered accurate for testing the reliability and	1. The time between failure is considered as exponential distribution 2. It is used to obtain the posterior distribution from the prior distribution
	failure parameter is a gamma distribution	

Table-III: Fault Count Model

Models	Features	Analysis Objective of the Model
Shooman Model [37]	1. Hazard rate is used for fault constant 2. Faults are corrected during execution	Same as Musa Model
Goel-Okumoto NHPP Model [2] GO Model Goel Generalized NHPP Model[3]	1. Failure with unknown constant value. 2.Fundamental Poisson process is applied. Practically ,the failure rate observed by increasing or decreasing of time	In this model the no. of faults indentified as random variable depends on test set. Sa Same as GO Model me as GO model
Musa Execution Time Model [23]	1. During the execution time intervals the no. of failures are specified 2. The model applicable for calendar time.	Same as Moranda Model
Musa-Okumoto Logarthimic Poisson Execution Time Model [22]	The failures are implicit to a NHPP It is based to the GO model.	Same as JM model Faults/ errors can be considered as hyper geometric distribution

Table 3 contains the fault count model like Shooman model, Goel –Okumoto NHPP (Non Homogeneous Poisson process model) Musa Execution Time Models are described by the model assumptions, and features. The mathematical model for each model described as follows:
a) Shooman Model: This model is a white box model in which the reliability estimation occurs in modular programs. Here, the reliability is obtained by computing the possible execution paths. Thus this model is also referred as path based model. Mathematically as follows

$$H(t) = \Omega\left[\frac{N}{I} - n_c(\tau)\right]$$

Ω = proportionality constant
t = time required to operate the system
τ = debugging time as start of the system integration.
of the programes
b) Musa's Execution time model [20]: It is a SRGM. Here, a time variant is used when testing take place. This model is used in the code level reliability estimation.
Mathematically represented as follows:
Musa's Execution time model: It is a SRGM. Here, a time variant is used when testing take place. This model is used in the code level reliability estimation [21]. Mathematically represented as follows:

$$\mu(\tau) = \frac{1}{\theta}\ln(\lambda_0\theta_\tau + 1),$$

Where λ = is the initial failure intensity and θ =is the rate of reduction in the normalized failure intensity per failure

c) Musa's Logarithmic model [22] represented as follows;

$$\lambda(t) = \varphi(N - \mu(t)) = Ef(N - \mu(t))$$

Where λ (t) = failure intensity during the early time t,

φ = Product of linear frequency f,

E= fault exposure ratio constant over time i.e. the average no. of failures occurring per fault remaining in the code during one liner execution of the program.

Table 4 contains the fault seeding models like Mills seeding model, Basin Model, Lipow Model are describe features and objective of the each models.

Table-IV: Fault Seeding Model

Proposed Models	Features	Analysis Objective of the Model
Mills Seeding Model [18]	The no. of identified faults in the program seeded randomly seeded which is to be tested.	The original faults perhaps expected from the no. of seeded faults discovered during the test.(i.e.Tagging Model)
Basin Model [49]	Estimate original faults in the program.	Two-stage testing method has suggested, in first detects, and keeps track of n_i faults from N unknown native faults. In second, assigned the testing program independently, and collects the records r from N possible faults. These faults are then compared with Mill's equation.
Lipow Model[34]	Finding any kind of fault by testing of the software and determines the probability of novel fault in the program	This model is a combination of binomial and hyper geometric test distribution and can

The mathematical model for each model described as follows:

Mills Seeding Model: In this model, a number of known faults are randomly seeded in the program. Mathematically expressed as:

$P(k;N,n_i,r) = \binom{n}{k}\binom{N}{r-k}/\binom{N+n_i}{r}$ such that $N \geq r - k \geq 0$, where N= total no. of in built error, n_i = no. of induced errors,

r = total no. of error removed during debugging, i = sum of seeded error in r removed error, r-k = sum of natural error in r removed error

Pr= {Probability of correct program of every inputs in [a, a+V] } it is exact test cases including successive distances

Mill's model uses statistical procedure where, seeding technique evaluates the number of indigenous / unknown errors in a program. This model avoids problem of time.

In Lipow model is modified of Mill's model considering the probability of finding an error in each of the tests. The probability is same for both actual and seeded errors.

Table 5 contains the input domain models like Nelson model, Ramamurthy and Bastani Model, are describe features and objective of the each models. Nelson model, the reliability of software can be calculated by functioning n sample inputs as randomly. The inputs are chosen according to probability distribution, which is either from operational profile or from user distribution.

The mathematical represented as follows;

$$x_j, j=1,2,\ldots n-1\} = e^{-\lambda V}\prod_{j=1}^{n-1}\left[\frac{2}{1+e^{-\lambda x_j}}\right]$$

λ =some measure of the complexity of the source code

In Ramamurthy and Bastani model no system failures detected during the reliability estimation phase. So this model provides an estimation of the conditional probability that the program is correct for all possible inputs given that it is correct for a specified set of inputs.

Table-V: Input Domain Model

Proposed Models	Features	Analysis Objective of the Model
Nelson Model[16]	Testing is performed randomly. For specific, input can be divided into equivalence classes	Operational usage distribution knowledge earlier should be known
Ramamurthy and Bastani Model [12]	- In reliability estimation, failures are not detected and the conditional probability for all the inputs is estimated. -The result of each test case gives some stochastic information.	By selecting appropriate test cases the testing can be minimized which use error prone constructs

Table 4 and 5 represent various fault seeding models and input domain models respectively, based on data domain model. Table 2 and 3 represent time between failure models and fault count models, based on time domain model. The time domain model depends on the failure process while the data domain models focus on the failure content of the software product under consideration. The time-domain models provide analytical expressions whereas, data domain models require lengthy testing with a representative and statistically meaningful set of inputs for their validation.

N.B.: In the above table discussion, the assumptions are evaluated one at a time. The software development process is environment dependent. (i.e. what holds true in one environment(testing one system) may not be true in another.)[30]

B. Code Level Reliability Prediction: There is no clear explanation to software reliability since we don't get the exact nature of software. Software reliability prediction is calculated on the failure rate of a system at the start of or any point throughout the system test. The software developers consider reliability is equivalent to correctness, i.e. after testing, the number of bugs found and should be fixed through all of the stages of the software lifecycle. The quality of the products of software development is related to the reliability by means of the code, test plans, and testing. The performance of software reliability is measured in terms of either continuous time or discrete time**.** Software reliability can be measured using discrete time [25]. In discrete-time domain specified, how several test cases are executed rather than what amount of time is required prior to a software failure occurs. It deals with two types of models. i.e.:

- Type I model – demonstrates by the number of runs between two successive failures.
- Type II model – demonstrates by the number of failures in number of test cases.

The basics parameters used to evaluate in discrete time domain software reliability are given below:

1. Run: It is a minimum execution unit of software application context which can be expressed as test cases of a software path. A run cannot be subdivided further. [25]

2. Run Life time:

$$RL_k = \begin{cases} 1, \text{ if s/w passes } k^{th} \text{ run} \\ 0, \text{ if s/w fails } k^{th} \text{ run} \end{cases} \quad (7)$$

$$S = \begin{cases} 1, \text{ if } RL_1 = 0 \text{ (i.e. s/w fails first run)} \\ k, \text{ if } RL_1 = R_2 = ... = R_{k-1} = 1 \\ RL_k = 0 \end{cases} \quad (8)$$

3. Run Life time distribution: In the probability perspective RL_k and S are random variables represented as discrete time. Let

$$P_r\{RL_k = 0\} = r_k,$$
$$P_r\{RL_k = 1\} = \bar{r}_k = 1 - r_k \quad (9)$$
$$k = 0,1,2,.....$$
$$\text{with } r_0 = 0$$

Where r_k=P {software fails the k^{th} run}; and 1-r_k= P {software passes the k^{th} run}.Hence, the run life time distribution is defined as

$$p(k) = P_r\{S = k\} \quad (10)$$

Assume $\{RL_k\}$ = series of independent random variables

$$p(k) = r_k \prod_{i=0}^{k-1} \bar{r}_i \quad (11)$$

$$k = 1,2,....$$

4. Run Reliability Function: It is defined as

$$R_k = P_r\{X > k\} = \prod_{i=0}^{k} \bar{r}_i = \sum_{i=k}^{\infty} \prod p(i) \quad (12)$$

Assume $\{RL_k\}$ = sequence of random variables which are independent to each other

R (k) = no failure probability occurs in the first k runs.

R (1) = run reliability

5. Run Failure Function:

$$Fail(k) = \sum_{i=1}^{k} P\{x = i\} = \sum_{i=1}^{k} P(i) \quad (13)$$

6. Mean Run To Failure:

$$E\{S\} = \sum_{i=1}^{\infty} kp(k) = \sum_{k=0}^{\infty} R(k) \quad (14)$$

7. Hazard Rate Function:

$$h_k = P\{S = k \mid X \geq k\}$$
$$= P\left\{RL_0 = RL_1 = ... = RL_{k-1} = 1, RL_k = 0\right\} \quad (15)$$

Since RL_1, RL_2, RL_3..... RL_K are independent.

$$\text{So } P(k) = R(k-1) = h_k \quad (16)$$

8. Failure Intensity Function: Let M (n) = no. Of failure that occurs with first n runs and (17)

Failure intensity of software at the n^{th} run is

$$\lambda(n) = I(n) - I(n-1) = r_n; \quad (18)$$
$$n = 1,2,... \text{with } I(0) = 0$$

$\lambda(n)$ =mean no. of software failure measured within a given time interval n to n-1.

C. Architectural Level Reliability Prediction: The system architecture is based on its software application en route for optimization of different attributes such as performance, reliability, and cost. The real aim of architecture based reliability approach includes the following: how components depends on the overall system reliability, for a given architecture how the reliability application validates the reliabilities of components and interfaces selecting a proper architecture that is suitable for the system. This software architecture is related with the failure behavior which can be an execution period of any component or during the control transfer between two components. In general, a component with more the execution time, the probability of failure is high and it uses a constant failure rate for reliability prediction. The accurate reliability predictions can occur by time-dependent failure models for testing and the measurement of code coverage for the components. Different approaches of architectural level reliability are:

State-based models: apply the probabilistic control flow graph (CFG) to represent software architecture and estimate software reliability analytically. This model uses software architecture with a discrete time Markov chain (DTMC), Continuous time Markov chain (CTMC), or Semi Markov process (SMP). Little Wood Models [7,9], Cheung models [44], Kubat model [43] etc are use for prediction.

Path-based models: software reliability calculated by in view of the possible execution paths of the program either experimentally by testing or algorithmically. The reliability of each path is computed by multiplying the reliabilities of the components along that path. Then, the system reliability is estimated by averaging path reliabilities over all paths [5]. Shooman model [37], Yacoub, Cukic and Ammar model[47] are used in the path based models. Additive models: The use of software components by the component's failure data can be done in this model. Here, the component reliability can be represented by non- homogeneous Poisson process (NHPP). Xie and Wohlin model [39] explains additive models. Table-6, Table-7 and Table-8 elaborate the different types of architectural model as said above with model assumptions and solution methods.

Table -6 describe the path based model with features and objective of models. The Shooman model does not provide the solution of the different execution path. The Solution for Shooman model is given by below equation

$$n_f = \sum_{i=1}^{m} Nf_i r_i \quad (19)$$

Where, n_f = The total number of failures
N=no. of test runs. This model assumes the failure rate is directly proportional to the number of remaining errors.
In Shooman model there is no component dependency.
In Krishnamurthy model [48]: This model is based on white-box model. Here, the reliability of components is known. This model focuses on the intra component dependency (looping) problem. This model does not consider errors in between the components. To estimate software

Table-VI: Path Based Model

Reliability Model Name	Architectural Assumption	Failure Behaviour
Shooman model[M.Shooman[38]	The model considers the different paths(run) i and the frequencies f_i	r_i =the probability of failure path i on each run
Krishnamurthy and Mathur model [48]	For each test run different paths is considered with a sequence of components -Component reliabilities are known	R_m=the sequence of each component is described by its reliability
Yacoub, Cukic and Ammar model (Scenario based)[47]	-Component dependency graph (CDG), a probabilistic model can be drawn through the scenario based. -A node n_i with an average execution time t_i	R_c = component reliabilities with failure process $(1-v_{ij})$ = transition reliabilities w.r.t. a node n_i to n_j

reliability this model can be expressed as follows:

$$R_P = \prod_{c \in C(P,tc)} R_i \quad (20)$$

R_i = Reliability of component i
Hence, Reliability estimation for program with respect to a test set TS = $R_{sys} = \frac{\sum R_p}{|TS|}$ (21)

Rp = Path Reliability, C(P, tc) is a trace for a program P having components c executed alongside a test case tc belonging to a test suite TS. Where as In Yacub model, a CDG a tree traversal algorithm (And OR operation are used) [43] to estimate the reliability.
Everett's model for reliability estimation[55] uses Additive Model in CBS where, component's reliability is analyzed using Extended Execution Time (EET) model Software properties like fault content from prior releases, code size, processing time distribution are determined from. the parameters of the model. This model considers software reliability growth model. Additive models do not explicitly consider the software architecture.

Table-VII: Additive Model

Reliability Model Name	Architectural Assumption	Failure Behavior
Xie and Wohlin model[39]	The System components are arranged parallel and tested independently,	The component or system failure reliabilities are represented by NHPP
Everett model[55]	The each component' reliability is calculated by extended execution time (EET) model.	components are NHPP

The solution method for Kubat Model is given by the equation (23):

$$\lambda_{Sf} = \sum_{k=1}^{K} r_k [1 - \mathrm{Re}\, l(k)] \quad (22)$$

$\lambda_{Sf} = \sum_{i=1}^{n} \pi_i \lambda_i$ the system failure rate tends to be by the assumption λi, << μi, *asymptotic* behaviour relative to the execution process
The times between successive program failures

$$\lambda_s = \sum_i a_i \lambda_i + \sum_{i,j} b_{ij} v_{ij}$$

Where

$$a_i = \frac{\pi_i \sum_j p_{ij} m_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}} \quad (23)$$

Proportion of time spent in module i,
and

$$b_{ij} = \frac{\pi_i p_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}}$$

The transfer of control frequency between i and j
π_i = steady-state probabilities of the embedded

Limitations:

We discuss the limitations into three categories- modeling, analysis and parameter estimation. These limitations are broadly discussed in architectural level. Modeling limitations include the following:

a) Execution of the state space models are synchronised (at any given time only one component execute not for the multiple components)

b) The transform of control of the components (state models) are free from the past history and dep ends on the Markov property of the previous component.

c) Most of the states in models are exponential time distribution. Generally to model, each state time should be deterministic.

The Parameter estimation [54], is applied for reliability analysis during the real software application in architectural and component failure model from different software artifacts like UML, SDL, ROOM etc. The limitation is that the profile data generated may be different for different environment. Thus data collection must be coarsely studied and for failure components estimated using statistical testing.

D. Code Level Reliability Estimation: To certify the reliability level for the product which meets the requirements set, we estimate the reliability. When the reliability is estimated, the future reliability can be predicted using the modeled data. Reliability estimation can be done at the different levels for the software product.(i.e. code level or design level).During the code level estimation though we get the reliability accurately but in case of architectural level the

Table-VIII: Sate Based Model

Model Name	Architectural Assumption	Failure Behavior
Little Wood Model[9]	1.software architecture can be explained by an irreducible SMP 2.a finite number of modules with transition probabilities p_{ij} 3. finite mean m_{ij}	failures occur by means of Poisson method by means of λ_{ij} -The failure transfer control done between interfaces with probability v_{ij}
Cheung model [44]	1. The components transition can be described by an absorbing DTMC 2. The reliability estimation can be applied for heterogeneous software.	Each module fails independently to produce the correct reliability
Laprie model[20]	1. The component's transfer control is a CTMC. 2. $1/\mu_i$ = mean execution time of a component i	λi = constant component's failure rate
Kubat model[43]	- Modules transitions follow a DTMC task .Pd_i (k, t) = Probability density function	λ_i =failures occur with a c onstant failure intensity

reliability estimation is accurate and complexity is less as we can estimate the product quality during the design level for which we decided whether to further develop the software or not[48]. The probability estimation of software failure within a specified time is a key metric usually considered as software reliability. This type of reliability is an important for both the software developer and its user. Therefore the software reliability estimation can be done on time/structure based models. Several models are used for estimate software reliability (Goel-Okumoto model and Musa's execution model is based on time based). [22]

Let P = denote a program under test where reliability is to be estimated

d = P is executed on a test case d selected from the input domain D

P (d) = the output of P obtained by executing it on d each.

Let TB_k = denote the time at which the k^{th} failure occurs and NB_k to no. of test cases

E_K =effort spent in testing as follows:

$$E_k \triangleq \begin{cases} TB_k - TB_{k-1} & ,for\ time\ basedmodel \\ NB_k - NB_{k-1} & ,for\ testcase\ based\ model \end{cases} \quad (24)$$

Let $e_{i:}$ denote the effort spent during i^{th} execution of P then

$$E_K = \sum_{i=l_1}^{l_2} e_i \quad (25)$$

Where e_{l1} and e_{l2} denote the effort spent in the first and last execution of P during the k^{th} failure interval.

The reliability R of P is defined as the probability of no failure over the entire input domain

i.e. R = P { P(d) is correct for any d∈ D} (26)

S_k = be the cumulative effort S_k be defined as

$$S_k = \sum_{i=1}^{k} E_i \quad (27)$$

Thus,

$$\mathrm{Rel}(x|t) \equiv P\{E_k > x \,|S_{k-1} = t\} \quad (28)$$

Where x=exposure period and Rel (x|t) = reliability during the next failure interval of x units specified the failure history. In code level reliability estimation, sequential model [40] is used. In this model the estimation can be done at the end of a software system. A given software system has consist of error s or faults in a number of sequential way that are independent to each other. For this number of sequential test and correction reviews can be accomplished. These test reviews are sequential means that, when a fault detected for the duration of one cycle is approved before the next cycle starts, so that the same fault can't be detected further one cycle. This effort can be obtained by all faults probabilities detected at the end of the last analysis and evaluate the software reliability. The Bayesian method is used for probability evaluation.

Let r = the r^{th} review in sequential which are arranged in such a way that review r is completed before review r + 1 starts. The following theorems are obtained:

Theorem 1: In a sequential model, the error estimation review both for prior probability p (s) with mean λ_0 and the posterior probability P_r (s|T_r) is Poisson distribution. i.e.

$$P_r(s \mid T_r) = \frac{(\lambda_r)^s}{s!} e^{-\lambda_r}$$, with mean

$$\lambda_r = \lambda_0 \prod_{j=1}^{r} (1 - F_r)$$

Where F_r= probability of identifying a given fault during the r^{th} evaluation

N_r = number of faults identified within the r^{th} evaluation

T_r = no. of faults detected in the first r evaluation

in the form of a set {N_1, $N_{2,.}$ . ., N_r}

p(s) = system's s number of faults before detecting any data which is calculated as prior probability

P_r (s|T_r) = Given data D_r, the conditional probability in which s faults remain undetected after completion of r reviews in the system

$P_r(s)$ = after completion of r reviews the unconditional probability i.e. faults not detected in the system s

Corollary 1**:** After r independent reviews with no faults remain and observations D_r can be represented as $P_r(0 \mid T_r) = e^{-\lambda r}$.

Here, the prior probability p(s) has the Poisson distribution with mean λ_0

$P_r(0)$ = the unconditional probability which can be inferred as the probability with no unobserved fault for performing r independent reviews in the future

Theorem 2: On behalf of some prior distribution Pd(s), the unconditional probability can be projected as,

$$P_r(0) = \sum_{s=0}^{\infty} Pd(s)\left(\sum_{j=1}^{r} p_j\right)^s \quad , \text{ where}$$

$$p_j = F_j \prod_{m=1}^{j-1}(1-F_m)$$

N.B. p_j = probability that a fault is discovered early time during the j^{th} review

Theorem 3: The unconditional probability with no fault after conducting r independent reviews can be evaluated as $P_r(0) = e^{-\lambda_r}$ considering p(s) as the prior probability Poisson distribution with mean λ_0.

E. Architectural Level Reliability Estimation: At the architecture level the reliability estimation can be done using a model known as Dependability Model. This model can be of two types:

a) Non-state-space models: In this model, it does not require the entire state space of the underlying model, not required the possible interactions between the components such as cascading failures. Solution methods for these models include reliability block diagrams (RBD), fault trees (FT)[1], and reliability graphs (RG).This is specifically for System reliability calculation.

b) State space models: But in case of State-space models, such as CTMCs (Continuous Time Markov Chains), require details of all possible combinations of states of components, which grow exponentially in the number of components in the system. The system operating in an environment randomly subject to probabilistic cascading failures (calculated as a rooted tree). The cascading failure is not a finite process. Thus, the cascading failure mechanism grows exponentially in size. The intensity matrix is calculated by the Markov chain, creates significant challenges. [31] For cascading produces with a single transition component failure model can be described as following steps: (Considering the rooted tree as Breadth First Search)

1) Collection of system's each components
2) Environments used for the model (i.e. load on system which can be change randomly)
3) Failure and repair rates (exponential distribution)
4) Cascading Failures (probabilistic, instantaneously failing components which are statically independent)
5) Repair Discipline (components repairing use as Markov Chain instead of First come first served)

To measure the Dependability [31] several metrics are used such as: Steady-State Unavailability(SSU) , Mean Time To Failure (MTTF) ,Steady-State Distribution of Cascade Size(SSDC), Distribution of Cascade Size Until Failure(DCSUF) etc. The above said cascading failure can be further applied to the multiple transition component failure.In this section, we discuss some model and the parameter estimation limitation in favour of software reliability estimation. The limitation of reliability prediction model is also applicable for software reliability estimation. Apart from these, some other limitations are- component replication and heterogeneous architectural styles which are determined during the application of reliability. As sensitivity analysis is conducted during the reliability estimation, the failure behavior of the components is expected to be uncertain in design phase. So, it is significant to determine the confidence level for risk associated with reliability estimation.

IV. RELIABILITY TOOLS FOR PREDICTION AND ESTIMATION

Reliability decides the overall quality of software; Researchers developed various tools for estimating Software Reliability [33].These tools give the detail analysis of reliability in such a way that it can be further utilized to improve the reliability criteria for real-time application. The various steps that are used in software reliability tools are: identify necessary reliability, build up operational profile, set up for test, perform test, and pertain failure data to perform decision. The software reliability tools use two basic input data: a) time-domain data b) failure-count data various tools are defined as follows:

CASRE (COMPUTER-AIDED SOFTWARE RELIABILITY ESTIMATION TOOL) [29] is a software reliability modeling tool that was developed in 1993 by the Jet Propulsion. SMERFS (STATISTICAL MODELING AND ESTIMATION OF RELIABILITY FUNCTIONS FOR SOFTWARE) [56] is a software application tool to evaluate the test data for failure rate. SOFTREL software reliability process simulator or SoftRel captures the properties of two types of failure events, specifically, defects in specification documents and faults in code. This is specifically used in early development, testing and operational management phase (SDLC).The parameter consider for this tool is defects in documents faults in code. It supports 2 models (refer table 2).The basic function is integration and test procedures, management plans, and other ancillary documentation can be done. This tool is used both for reliability estimation and prediction. SOREL (SOFTWARE RELIABILITY ANALYSIS AND PREDICTION) SoRel[24] is a tool for Software Reliability analysis and prediction. It is composed of two parts: reliability growth tests and application of growth models. After completion of test cycle, is used to measure the reliability of code. It support 4 models (refer table 2,3,4,5). The parameters are the total failure rate remaining failure performed interval domain data and time domain data. The function of SoRel is Maximum Likelihood Estimation uses reliability for mean time to failure, cummulative no. of failures etc. This tool is used both Prediction and estimation can be done. MEADEP (MEASURE AND DEPENDABILITY)[14]It is a failure data based for dependability analysis. In this tool the basics functions are obtained such as no. total failure and remaining failure considered Mean failure rate, recovery rate,

Mean Time between Events (MTBE) Time To Recovery (TTR) etc are obtained from data and Mean Time Between Failures (MTBF) for repairable and non-repairable systems, steady-state availability (SSA) etc are obtained from model. SRMP (STATISTICAL MODELING AND RELIABILITY PROGRAM) is developed by the Reliability and Statistical Consultants, Limited of UK in 1988. It uses the maximum likelihood parameter estimation technique to compute reliability function, failure rate, mean time to failure, median time to failure and the model parameters for each model. In this tool reliability analysis and the inter failure times of all the failures etc are calculated. SOFTWARE RELIABILITY ESTIMATION AND PREDICTION TOOL (SREPT [51] offers several techniques that can be used at various stages in the software development life-cycle, basically at functional testing and non-functional testing techniques. It is developed during SDLC for software reliability engineers. It supports 1 model and the basic parameters are remaining number of faults, test to be covered. This tool is an architecture-based approach which is essential to assess the reliability and performance of a system. It calculates the factors like failure intensity, conditional reliability etc. Both Prediction and estimation can be done here. SRTPRO (Software Reliability Tool professional) was developed as a software reliability measurement tool during software development life cycle for software reliability engineers. It is developed during SDLC (Software Development Life Cycle) for software reliability engineers supports 14 models, Reliability or failure rate, total failure etc. It used to iteratively manage software reliability.

Table -9 describes the objective of Software reliability tools as well as the different parameters used for different tools and specifies the uses of the above tools. (i.e. whether the tool is used for prediction or estimation)

V. RESULT FOR RELIABILITY ASSESSMENT

We illustrate a simple component-based application for ATM (Automatic Teller Machine) application system. The architecture [26] consists of the following components Deposit, Check balance, Withdraw, Transaction, Login, Bad Pin, and Print Receipt. The analysis specify with a set of scenarios in the execution of the application. Such as enterPassword(), verifyAccount(), withDrawcash() , checkBalance(), printReceipt() , requestPassword() etc. With the help of scenarios a probabilistic model is used for the reliability analysis in component-based systems. To deduce the probabilistic model we use a graph called as Component Dependency graph (CDG) which is similar to Control flow Graph (CFG). The CDG shows the relationship between components and determine the probable execution paths.

The above scenarios can be represented as an intermediate graph called CDG [47] to estimate the whole system reliability. An ATM bank system [46] is employed to validate the architecture-based reliability model. This system consists of 11 components. Fig.5 explains the sequence diagram of the ATM system with different scenarios. The corresponding component dependency graph for the sequence diagram is described in Fig.6. The transition probability

Table-IX: Tools Used for Reliability Prediction and Estimation

Tools Name	Objective	Parameters	Function
1.CASRE	-After completion of test cycle is used to measure the reliability of code - It is performed reliability of individual component	- failure rate -total failure -remaining failure -inter failure times or failure frequencies -supported 14 models (refer table 2,3,4,5)	-evaluates the inter failure time Cumulative failures, actual and estimated reliability growth etc. -Use for estimation not prediction
2. SMERFS	1.The data collection provides multiple time domain 2. After completion of test cycle is used to measure the reliability of code	failure count -failure intensity -defect discovery rate prediction -same as CARSE tools -supported 11models (refer table 2,3,4,5)	-evaluates the number of errors, mean time-to-failure predictable from number of faults for the further testing period etc. -Uses both prediction and estimation
3. SRMP	It used during testing phase and does not execute on the interval domain data	supports 9 models (refer table 2,3,4,5) -Median time to failure(MTTF	The data input are like file contains the name of projects, the number of failures.(Prediction and Estimation can be done)
4. MEADEP	dependability model for failure data analysis tool used in SDLC phase	failure rate - location, type, impact and other failure characteristics and support 1 model	In this tool the results obtained from either data or model.(Prediction and estimation can be done)

between the components is represented in the Table 10. Each component's reliability is given by the Table 11.For state based model, the reliability of the program is evaluated as the following method. Let {C1 , C2, - - , Cn} be the set of nodes in the program graph with C1 the starting node and Cn the end node.

Let Re_1 be the reliability of node Ci and PR_{ij} the transition probability from node (Ci, Cj). Let $PR_{ii} = 0$ if the transition from (Ci,Cj) does not exist. As we discuss above that the state models can be of the Markov model, so it is represented by {S,E, C1, C2,…. , Cn}. The system reliability can be calculated by the following procedure. [44,53]

$s=I+q+q^2 +q^3+\ldots\ldots= \sum_{i=0}^{\infty} q^i$,I= Identity Matrix and $q=Re_i*PR_{ij}$. Hence SR= =s(1, n) Re_n ,n=no.of nodes/states.

Similarly the Path based model can be considered by the components reliability with transition probability along the execution path. i.e. from starting component to end. We follow the Yacub model for path based method for reliability assessment. So, we get the result in both the methods i.e. state based method using (chunge's user model) system reliability estimation is 0.560 and in path based method system reliability estimation is 0.577.The result obtained by both the models give possibly accurate estimations compared to the actual reliability.

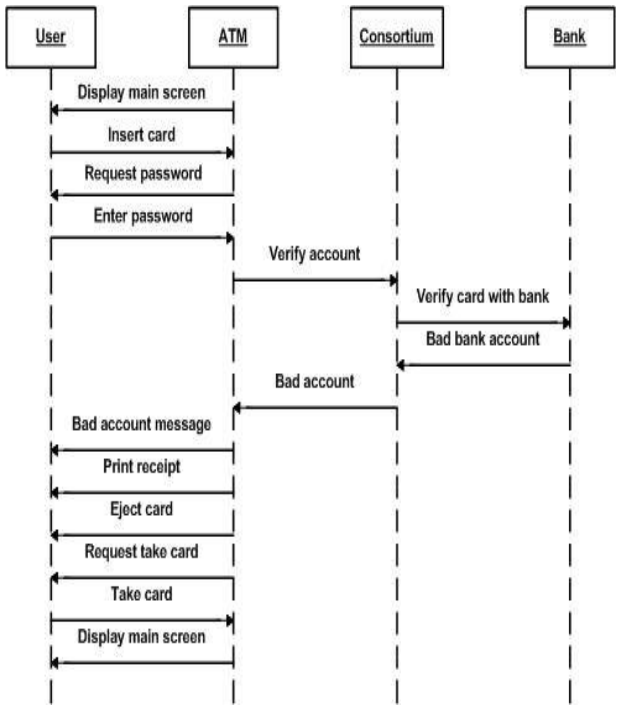

Fig. 5.ATM system Sequence diagram

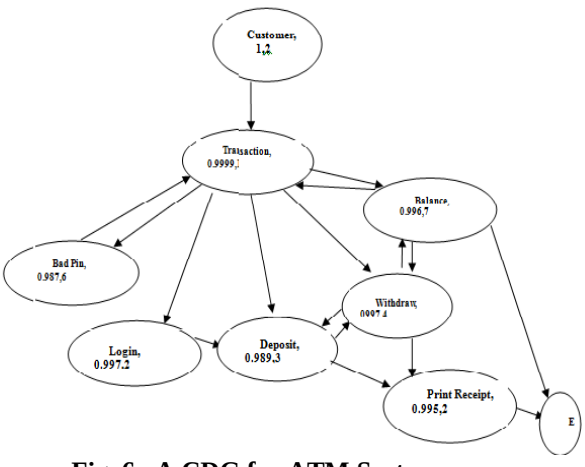

Fig. 6. A CDG for ATM System

Table-X: Transition Probability (tp) between components

$tp_{1,2}$ = 1.0				
$tp_{2,3}$ = 1.0	$tp_{2,4}$= 0.999	$tp_{2,11}$ = 0.001		
$tp_{3,5}$ = 0.227	$tp_{3,6}$ = 0.669	$tp_{3,8}$ = 0.104		
$tp_{4,5}$ = 0.227	$tp_{4,6}$ = 0.669	$tp_{4,8}$ = 0.104		
$tp_{5,2}$ = 0.048	$tp_{5,6}$ = 0.951	$tp_{5,11}$ = 0.001		
$tp_{6,3}$ = 0.4239	$tp_{6,4}$ = 0.4239	$tp_{6,7}$ = 0.1	$tp_{6,9}$=0.4149	$tp_{6,11}$=0.0612
$tp_{7,6}$= 1.0				
$tp_{8,6}$= 1.0				
$tp_{9,6}$= 0.01	$tp_{9,10}$ =0.99			
$tp_{10,6}$= 1.0				

Table-XI: Component Reliability of the ATM System

Component #	Reliability
CR1	1
CR2	0.982
CR3	0.97
CR4	0.96
CR5	1
CR6	0.996
CR7	0.99
CR8	1
CR9	1
CR10	0.8999
CR11	1

VI. CONCLUSION

Achieving software reliability is the major task for software industry. Therefore, the software quality can be achieved by means of software reliability. To get better reliability of a product, we require three components i.e. modeling, measurement and improvement. We highlight the basic definition of software reliability, the reliability analysis, prediction and estimation.

We discuss the existing software reliability prediction and estimation models at different phases in software development process and the metrics used for software reliability at different levels (i.e. code level and architectural level). Here, we represent various models of reliability analysis. Most of them are analytically derived from assumptions. Further, we discuss the limitation of the prediction models as well as architectural models. We represent the different parameters considered for reliability prediction and estimation. Thus, software reliability can be improved by increasing the testing effort and by correcting detected faults. From the existing experimental studies, we observe that during the early stage of design in software development process, the failure data affects the software reliability prediction, where as for the developed process (coding level), the factors integrated into software reliability estimation are operational profile and logical errors. From the above software reliability models, we analytically observe that exponential distribution plays an important role in reliability since it has a constant failure rate. Finally, we discuss some familiar tools to measure the software reliability prediction and estimation.

For the reliability prediction we focus the architecture based software reliability (used in design phase), that can be represented either state based method or path based methods. We discuss several models for both methods and represented an ATM system for compare the both method for reliability assessment. We conclude that the state based method is better than path based method. In state based methods the failure behavior assists the impact of individual component reliability on the overall system reliability. For path based approaches, this is not applicable.

REFERENCES

1. Ahmed Ali Baig, Risza Ruzli, and Azizul B. Buang(2013),"Reliability Analysis Using Fault Tree Analysis: A Review ", International Journal of Chemical Engineering and Applications, Vol. 4, No. 3, June 2013
2. A.L.Goel and Kazu Okumoto(1979) , "A Markovian model for reliability and other performance measures of software system", in Proc. Nat. Computer Conf., New York .vol.48,1979, pp.769-774.
3. L. Goel,(1983), "A guidebook for software reliability assessment",Rep. RADCTR-83-176, Aug. 1983.
4. Amrit L. Goel(1985)," Software Reliability Models: Assumptions, Limitations and Applicability ",Member IEEE, IEEE Transactions on Software Engineering , Volume: SE-11, Issue: 12, Dec, 1411 – 1423
5. AP Singh, Pradeep Tomar (2013), "A new model for Reliability Estimation of Component-Based Software", Published in: Advance Computing Conference (IACC), 2013 IEEE 3rd International, Date of Conference: 22-23 Feb. 2013, Date Added to IEEE Xplore: 13 May, 1431-1436
6. Wesslen, P. Runeson, and B. Regnell(2000), "Assessing the sensitivity to usage profile changes in test planning," in Proc. 11th Int'l. Symp. SoftwareReliability Engineering, 2000, pp. 317–326
7. B.Littlewood and J.L. Verrall (1973),"A Bayesian reliability growth model for computer software," APPl. Statist., vol.22, pp.332-346
8. B.Littlewood (1980),"Theories of software reliability: How good are they and how can they be improved?" IEEE Trans. Software Engg., vol.SE-6.pp.489-500
9. Littlewood(1979), "Software reliability model for modular program structure, IEEE Trans. Reliability 28 (3) (1979), pp-241–246.
10. Chu, T.L., Yue, M., Martinez-Guridi, M., and Lehner(2010), J."Review of Quantitative Software Reliability Methods". United States: N. p., 2010. Web. doi:10.2172/1013511
11. Stringfellow and A. Amschler Andrews(2002) , "An Empirical Method for Selecting Software Reliability Growth Models", Empirical Software Engineering, December Volume 7,Issue-4,pp 319–343
12. C.V. Ramamurthy and F.B. Bastani(1982), "Software reliability: status and perspectives," IEEE Trans. On Software Engineering, vol. Se-8, , pp. 359-371,july 1982.
13. Dalila Amara, Latifa Ben Arfa Rabai (2017) ,"Towards a New Framework of Software Reliability Measurement Based on Software Metrics ", The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017), Science Direct Available online at www.sciencedirect.com Procedia Computer Science 109C,pp: 725–730 , Elsevier
14. Dong Tang, Myron Hecht, Jeffrey Miller and JadyHandal(1998), "MEADEP — A Dependability Evaluation Tool for Engineers", IEEE Transactions on Reliability, vol-47,issue 4, Dec 1998
15. D.Swamydoss and Dr. Kadhar Nawaz(2011), "Enhanced Version of Growth Model in Web Based Software Reliability Engineering", Journal of Global Research in Computer Science ,Volume 2, No. 12, December,44-46
16. E. Nelson(1978), "Estimating software reliability from test data," Microelectron. Rel., vol. 17, pp. 67-74
17. G.J. Pai(2002). "A survey of software reliability models". Technical report, DCE, University of Virginia, 2002.
18. H.D. Mills(1972),"On the statistical validation of computer programs," IBM Federal Syst. Div., Gaithersburg, MD, Rep.72-6015,1972.
19. I.vo Krka,Leslie Cheung, George Edwards, Leana Golubchik, and Nenad Medvidovic ,(2008)"Architecture-Based Software Reliability Estimation:Problem Space, Challenges, and Strategies", In: DSN 2008 Companion: Proceedings of DSN 2008 Workshop on Architecting Dependable Systems
20. J.C. Laprie(1984), "Dependability evaluation of software systems in operation", IEEE Trans. Software Eng. 10 (6) ,pp: 701–714.
21. J.D. Musa (1971),"A theory of software reliability and its application ",IEEE Tans. Software Eng.,vol. SE-1,pp.312-327,1971
22. J.D. Musa and K. Okumoto(1983) ,"A logarithmic Poisson execution time model for software reliability measurement."in Proc. 7th int.Conf. Software Eng., Orlando , FL. Mar.1983,pp.230-237
23. J.D.Musa , ; W.W. Everett (1990), Software-reliability engineering: technology for the 1990s, IEEE Software (Volume: 7 , Issue: 6 , Nov. 1990),Page(s): 36 - 43
24. Jiantao Pan (1999),"Software Reliability", Carnegie Mellon University, 18-849b Dependable Embedded Systems, Spring 1999
25. Kai-Yuan Cai (2000), "Towards a conceptual framework of software run reliability modeling ",Information Sciences 126 (2000) ,137-163
26. Katerina GoSeva- Popstojanoval , Aditya P. Mathur, Kishor S. Trivedi,(2002)"Comparison of Architecture-Based Software Reliability Models" , Proceedings 12th International Symposium on Software Reliability Engineering, IEEE Xplore: 07 August ,22-31
27. KaramaKanoun, Mohamed Kaˆaniche, Jean-Claude Laprie, Sylvain Metge(2002),"SoRel: A tool for reliability growth analysis and prediction from statistical failure data", FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing,Date of Conference: 22-24 June 1993,Date Added to IEEE Xplore: 06 August 2002,pp-654-659
28. L. Peterson, (1997),"Petri Nets," ACM Computing Surveys, Vol 9 No.3, pp. 223- 252
29. Lyu, M. and A. Nikora (1992a, July)., "CASRE – a computer-aided software reliability estimation tool", In Proc.of the Fifth Internation al Workshop on Computer-Aided Software Engineering, Montreal, Que., pp. 264–275. IEEE Computer Society
30. 30.M. H. Chen, J. R. Horgan, A. P. Mathur, V. J. Rego(1992),"*A time/structure based model for estimating software reliability*, Dec. 1992.
31. Mihir Sanghavi, Sashank Tadepalli, Timothy J. Boyle, Jr., Matthew Downey, and Marvin K. Nakayama (2017),"Efficient Algorithms for Analyzing Cascading Failures in a Markovian Dependability Model",, IEEE Transaction on Reliability, vol. 66, no. 2, June 2017
32. Michael Rung-Tsong Lyu (2002) , "Software Reliability Theory ", The Chinese University of Hong Kong, Encyclopedia of Software Engineering Copyright © 2002 by John Wiley & Sons, Inc. All rights reserved. DOI: 10.1002/0471028959.sof329 Article Online Posting Date: January 15, 2002
33. M. R. Lyu, (2007, May). Software reliability engineering: A roadmap. In Future of Software Engineering (FOSE'07) (pp. 153-170). IEEE.
34. M.Lipow(1972), "Estimation of software package residual errors." TRW, Redonodo Beach,CA, Software Series Rep. TRW-ss-72-09.1972
35. Mei-Hwa Chen, Michael R. Lyu, and W. Eric Wong (2001),"Effect of Code Coverage on Software Reliability Measurement ",IEEE Transactions on Reliability, vol.50,no.2,June 2001, pp:165-170
36. Michael R. Lyu(1995), "Handbook of Software Reliability Engineering", McGraw-Hill publishing,1995,ISBN0-07-039400-8.http://portal.research.bell-labs.com/orgs/ssr/book/reliability/introduction.html
37. M.L.Shooman (1972),"Probabilistic models for software reliability prediction," in Statistical Computer Performance Evaluation, W.Freiberger,Ed., New York : Academic,1972.pp.485-502
38. M. Shooman(1976), "Structural models for software reliability prediction", in: Proceedings of the Second International Conference on Software Engineering, 1976, pp. 268–280.
39. M. Xie, C. Wohlin(1995), "An additive reliability model for the analysis of modular software failure data", in: Proceedings of the Sixth International Symposium on Software Reliability Engineering (ISSRE'95), 1995, pp. 188–194
40. Nancy E. Rallis and Zachary F. Lansdowne (2001), "Reliability Estimation for a Software System with Sequential Independent Reviews", IEEE Transactions on Software Engineering,vol. 27, no. 12, pp-1057-1061,DECEMBER 2001
41. N. Langberg and N.D. Singpurwallw(1985),"Unification of some software reliability models via the Bayesian approach," SIAM J. Sci. and Stat. Comput., 6(3), 781–790.
42. Norman E Fenton and Martin Neil (1998), "Software Metrics: Successes, Failures and New Directions" , For Special Issue of Journal of Systems and Software 20 November 1998
43. P. Kubat(1989)," Assessing reliability of modular software",Oper. Res. Lett. 8 ,pp: 35–41

44. R.C. Cheung(1980), "A user-oriented software reliability model", IEEE Trans. Software Eng. 6 (2) , pp-118–125.
45. R.S Pressman, Associates (2016), "Software Engineering Resources" .
46. S.C. Pattnaik, M. Ray (2019) ,"Architecture Based Reliability Analysis", ICICC -2019ICICC-2019: International Conference on Intelligent and Cloud Computing,Institute of Technical Education and Research (ITER), Siksha 'O' Anusandhan Deemed to be University,Bhubaneswar, India, December 16-17, 2019. (Accepted)
47. Sherif Yacoub, Bojan Cukic, and Hany H. Ammar, Member, IEEE (2004), "A Scenario-Based Reliability Analysis Approach for Component-Based Software", IEEE Transactions on Reliability,vol: 53,Issue: 4, Dec. 2004, pp-465- 480
48. S.Krishnamurthy,A.P. Mathur(1997), "On the estimation of reliability of a software system using reliabilities of its components", in: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE'97), 1997, pp. 146–155.
49. S.L. Basin(1974), "Estimation of software error rate via capture-recapture sampling," Science Applications, Inc., Palo Alto, CA, 1974.
50. Sommerville.I.ansommerville.com/software-engineering-book/Chapter 1
51. Srinivasan Ramani, Swapna S. Gokhale, and Kishor S. Trivedi (1998), "SREPT: Software Reliability Estimation and Prediction Tool", International Conference on Modelling Techniques and Tools for Computer Performance Evaluation Tools:Computer Performance Evaluation,pp 27-36
52. Srinivasan M. Iyer, Marvin K. Nakayama, and Alexandros V. Gerbessiotis(2009),"A Markovian Dependability Model with Cascading Failures ", IEEE Transactions on Computers,vol. 58, no. 9,September- 2009
53. S. Gokhale, W.E. Wong, K. Trivedi, J.R. Horgan(1998), "An analytical approach to architecture based software reliability prediction, in: Proceedings of the Third International Computer Performance and Dependability Symposium (IPDS'98), 1998, pp. 13–22.
54. Swapna S. Gokhale (2007), "Architecture-Based Software Reliability Analysis: Overview and Limitations", Senior Member, IEEE, IEEE Transactions on Dependable and Secure Computing , vol. 4, no. 1, January-March, 32-40
55. W. Everett(1999), "Software component reliability analysis", in Proceedings of the Symposium on Application-specific Systems and Software Engineering Technology (ASSET'99), 1999, pp. 204–211
56. William Farr, Oliver Smith(1996), "SMERFS – Statistical Modeling and Estimation of Reliability Functions forvSystems", http://www.slingcode.com/smerfs/drfarr.php, 1996.
57. Z.Jelinski and P. Moranda (1972),"Software reliability research", in statistical Computer Performance Evaluation,W.Freiberger,Ed., New York : Academic,1972, pp.465-484

AUTHORS PROFILE

Sampa Chau Pattnaik has completed her M.E. in Computer Science and Engineering from UKAL UNIVERSITY, Odisha. Now she is continuing her research (Ph.D.) in SOA (DEEMEDD TO BE), University, Bhubaneswar. She has attended many conferences like IEEE, Springer, Elsevier, etc. Her area of interest is software testing.

Mitrabinda Ray has completed her Ph.D. degree from NIT, Rourkela. Now she is working as an associate professor in SOA, University, Bhubaneswar, in the department of Computer Science and Engineering. Her area of interest is software testing, software reliability & estimation. She has published a number of papers in different journals and conferences.