Journal of
Computer Science and Software Development

Research Article Open Access

Software Reliability Trends and Mechanism: A Conceptual Framework

Arpita Tiwari*

Department of Information Technology, Manipal University, Jaipur, India

***Corresponding Author:** Arpita Tewari, Department of Information Technology, Manipal University, Jaipur, India, Tel: 8299609348, E-mail: aritiwari10@gmail.com

Received Date: January 16, 2024 **Accepted Date:** February 16, 2024 **Published Date:** February 19, 2024

Citation: Arpita Tiwari (2024) Software Reliability Trends and Mechanism: A Conceptual Framework. J Comput Sci Software Dev 3: 1-13

Abstract

Software Reliability is an intra-disciplinary research area that paves the way for finding customer oriented view of software quality. This paper attempts to establish practical and useful capabilities of software reliability in boisterous paradigms, with a focus on reliability improvement that has a direct bearing on the quality of software.

This paper observes the denotation of software reliability, reviews reliability analysis techniques, identifies the design process as the main source of software faults. This has led to the development of number of design methodologies of reliability models. In this study, the inputs to software reliability are considered. Problems that may arise have also received considerable attention. A small number of models are also shown with their virtues to monitor the reliability data as they progress through the various phases of the software life cycle.

Keywords: Software Quality; Software Reliability; Reliability Prediction

Introduction

In the realm of software quality, software reliability stands as a crucial attribute alongside functionality, usability, performance, serviceability, capability, instalability, maintainability and documentation. Nevertheless, attaining software reliability poses a formidable challenge due to the intricate nature of software which often exhibits high levels of complexity.

For any system with a high degree of complexity, including software, it is difficult to reach a certain level of reliability. But then System developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software.

The reliability of software exhibits an inverse relationship with its complexity, whereas other crucial aspects of software quality, such as functionality and capability, demonstrate a direct correlation with complexity.Emphasizing these features tend to add more complexity to software. Review of literature introduces various prior research work done in the domain of software reliability.

Software Reliability Spectrum

Software reliability growth can be monitored using software reliability growth models. Many software reliability models have been developed till date which focus on error counts and do not directly model the software development environment. Exponential error rate was assumed for these models. There are few models that attempt to model software reliability based on certain design factors that exist during the software development process. Fault introduction, fault removal, and the environment were considered to model software reliability. Some of the model are estimators because they used primarily during test phase and are continually updated as data are collected. A few models are predictive in nature, which actually predict reliability before the coding begins.

In order for a model to be considered effective, it is essential that the assumptions it is based on are applicable to the specific development process it is intended to be utilized for.

Before a model is implemented, the assumptions must be verified as consistent with what is expected to be experienced on the project. If it is not known which of the models most closely fits the current projects, it may be wise to implement more than one model and the examination of the outcomes or the execution of models utilizing past data from a comparable item is essential for a comprehensive analysis.

It is good idea to implement more than one model on a given project even if information is available on the product.

Miscellaneous of work have been done in the area of software fault analysis, prediction and evaluation. This analysis provides a more complete method to apply software reliability modeling techniques to existing software systems and proposed design changes [12].

A system must function sufficiently reliably for its application, but it must also reach the market at the same time before its competitors and with a competitive cost. Some systems may be less market-driven than others, but balancing reliability, time of delivery, and cost is always important. Quantitative planning and tracking can be employed as a highly efficient approach to engineer the test process. By utilizing this method, one can effectively enhance the overall effectiveness of the testing process.

Software reliability engineering combines the use of quantitative reliability objectives and operational profiles. The work done by Musa was experiment by 70 project managers in specification of their practice. But still very less standard practices are explored till now [15].

Virtues and Flaws

One of the most notable limitations in many software reliability models is the strict requirement of independence between consecutive software failures. This assumption assumes that each failure event is completely unrelated to any previous or subsequent failures, which may not always hold true in practice. This research work has studied several significant models and explained the software reliability modeling framework that can consider the phenomena of failure correlation and to study its effects on the software reliability measures, it also explicates the precautions

in using reliability growth models. It allows construction of the software reliability model in both discrete time and continuous time, and (depending on the goals) to base the analysis either on Markov chain theory or on renewal process theory. Thus, modeling approach is an important step toward more consistent and realistic modeling of software reliability.

A lot has been said and published about the merits and limitations of reliability predictions as contrasted to reliability testing and assurance techniques from a product development standpoint. It also endeavors to address inquiries such as: are reliability prediction methods based on MIL (Meaningful Information Level) beneficial? During which phases of the product development proceeds? Which aspects of the prediction can be feasibly utilized, and which ones should be disregarded? How can the precision of reliability predictions be enhanced? Each method presents a specific advantage at a particular expense and is constrained by a temporal factor. There is no solitary solution in accurately predicting and showcasing reliability.

Balancing \cost, benefit and time, the essential elements of a new product reliability and quality assurance program, provide a framework for selecting the methods. Specific, theoretical and practical concepts are employed to exemplify the principles and elucidate the techniques that have been effectively utilized with promising outcomes. In addition, useful interpretations of reliability predictions are presented, as it appears many popular misconceptions exist in the electronics industry.

Many input-domain and time-domain models can be derived as special cases under the assumption of failure - independence. This paper aims at showing that the classical software reliability theory features and extensions to consider a sequence of possibly – dependent software runs, *viz,* failure correlation. It does not deal with inference or with predictions, *per se.e This gives the detailed* assumptions about the nature of the overall reliability growth.

Way modeling-parameters change as a result of the fault-removal attempts [19].

Assortment of Software Reliability Models

Software reliability models are classified according to Software Development Life Cycle (SDLC) phases. This work figures out and defined a number of criteria on the basis of its importance level; for software reliability model selection. Different phases of SDLC. Software reliability assessment methods for concurrent distributed system development can be done by using the Analytic Hierarchy Process. Also, in this work a comparison has been made between the inflection S-shaped software reliability growth model and the alternative models utilizing a heterogeneous Poisson process have been employed to evaluate the dependability of the complete system comprising multiple software elements.

Moreover, this work analyzes actual software fault count data to show numerical examples of software reliability assessment.

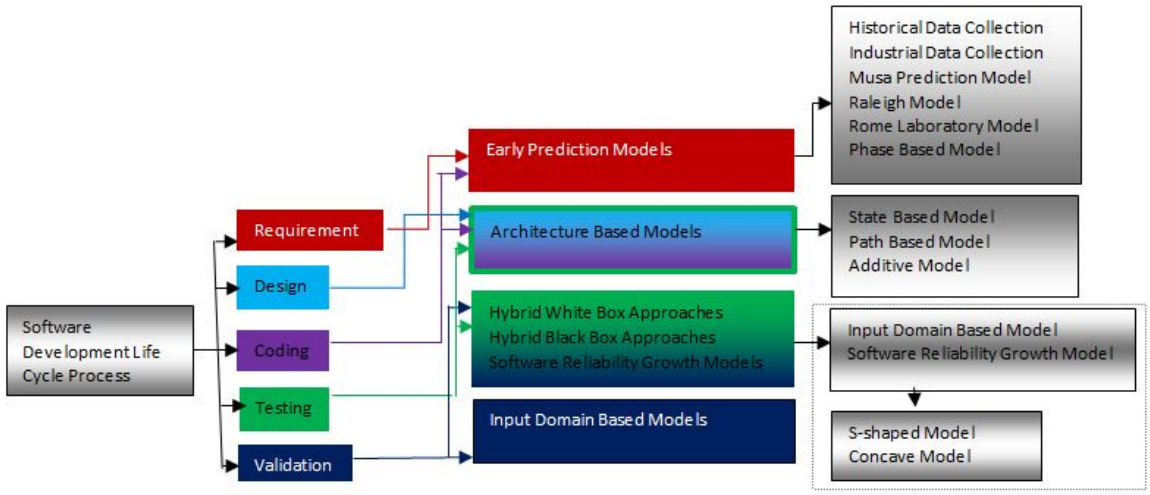


Figure 1: Software Development Life Cycle Process

Early Life Cycle Activities that Affect Reliability

The reliability of software can be improved by reducing the number of faults introduced through human-error, increasing the rate of discovery of these faults, or both. This can be accomplished through four activities linked with the software life cycle: fault avoidance, fault elimination, fault tolerance, and structured maintenance (Lyu, 1996:19). Fault Avoidance Fault avoidance consists of applying sound software engineering practices, including comprehensive standards such as documentation, design, and programming; rigorous quality assurance techniques like formal reviews, inspections, and audits; and independent verification and validation (Lyu, 1996:20). Inspections, reviews, audits and independent verification and validation can each be applied to any well-defined work product such as requirements and design documents, test plans, hardware logic, and code (Russell, 1991:26). Another engineering technique, the Cleanroom Software 2-7 Development Process, also known simply as Cleanroom, also provides a mathematical correctness verification approach to software fault avoidance. Detection of faults in the transformation of requirements to specification is an early step in fault avoidance. By inspecting random samples of the formal specification as it is being written, ambiguities and misunderstandings in the transformation process can be identified. These ambiguities are brought to the attention of the developer and the defects are pointed out. According to Gilb, (1996:26) 62% of defects occur during the design process, and 38% are created during coding. Inspections and Cleanroom processes are proactive approaches to ensuring that defects are not allowed to reside in a software program and that they are removed prior to coding or testing. The use of inspections and Cleanroom processes result in a software product that is less prone to defects, thus increasing software quality and reliability. Fault Elimination Fault elimination is accomplished through design and code inspections, mathematical correctness verification, and effective testing. According to Lyu (1996:20), eliminating all faults in the code through exhaustive testing or mathematical-proof-of-correctness is theoretically possible. However, in practical terms, it becomes impractical to achieve this in systems of significant size and complexity.

Inspections are a more practical approach than exhaustive testing to eliminating software faults and they are conducted earlier in the development life cycle when the cost-savings are the greatest. Test coverage models are available to assess the effectiveness of the test strategies employed by the software 2-8 developer later in the life cycle. Common testing methods will identify many, but certainly not all, faults (Lyu, 1996:20). Fault Tolerance Fault tolerance is achieved through special programming techniques that enable the software to detect and adequately recover from error conditions. One method of programming fault tolerant software is the development of redundant software elements that provide alternative means of fulfilling the same function. The different versions must be programmed

such that they will not both fail in response to the same input state. A more common, but less effective, example of fault tolerance is the use of exception handling in Ada (Lyu, 1996:21). Structured Maintenance Each software maintenance action should be performed as a microcosm of the full development life cycle. As such, the techniques of fault avoidance, elimination, and tolerance can be applied to modifications made during the maintenance of software as well. This is necessary to avoid introducing new faults as a result of code modifications made to correct known faults, add enhancements, or adapt the software to changes in the computing environment (Lyu, 1996:21).

Figuration of Software Reliability Growth Models

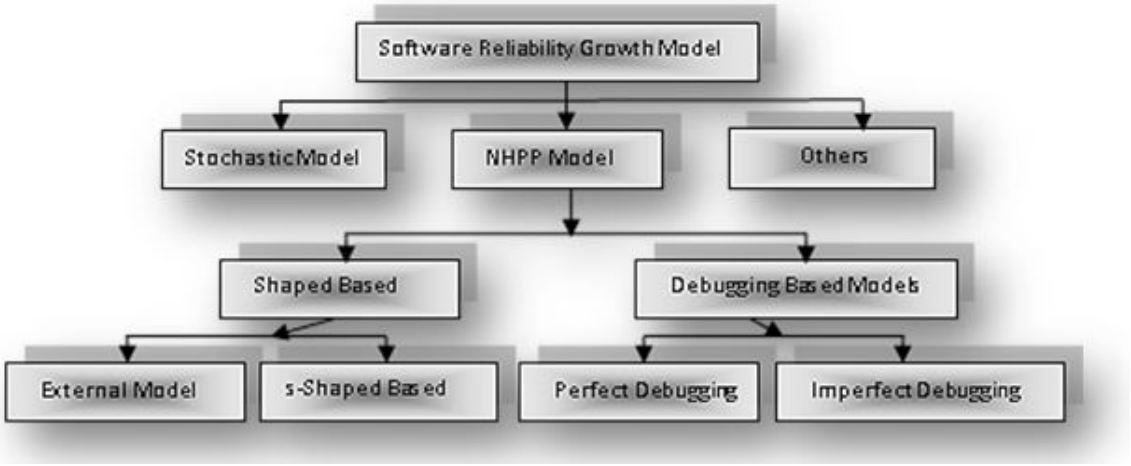


Figure 2: Types of Software Reliability Growth Model

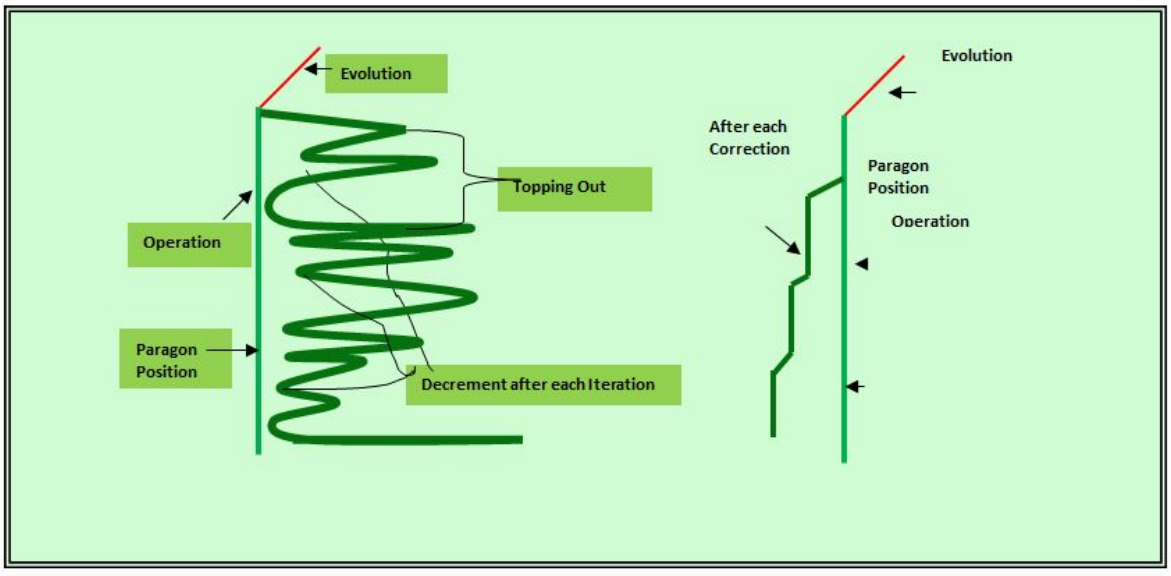


Figure 3: Process comparison between Normal development and Software Growth model

Intensity of Failure: Normal Development and Deployment Vs Reliability Growth Model

Taxonomy of Software Reliability Models

Research Question

Software reliability models determine the amount of testing required to say with confidence that the software

is fault-free. These models are used late in the development life cycle when the costs to make those corrections are 100 times more expensive (Fagan, 1976:37). Therefore, what is needed are software reliability models that can be used early in the software development life cycle to determine the reliability of the software. This need led to the question that the Air Force Operational Test and Evaluation Center (AFOTEC) wanted answered: "What are the current early life cycle software reliability prediction models and which should be recommended for AFOTEC to use in support of operational assessment?" delineates the patterns of software malfunctions over time.

Time variable is regarded as a random variable characterized by a certain probability density function, (pdf). The reliability models in this class vary with respect to the assumptions made with regard to the form of the probability density function.

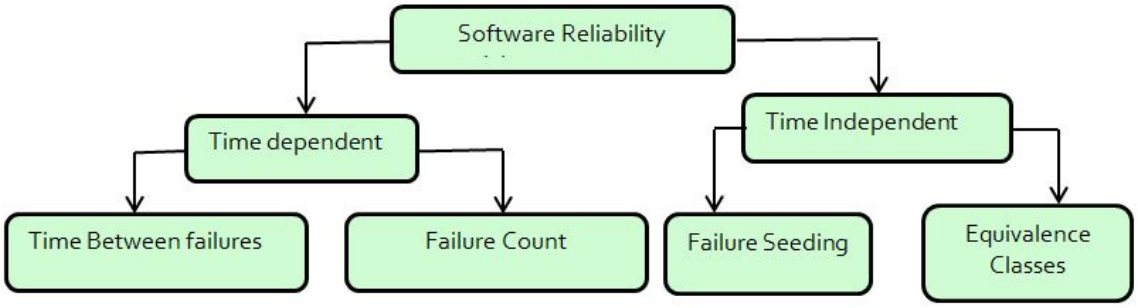


Figure 4: Pattern of Software Reliability Model

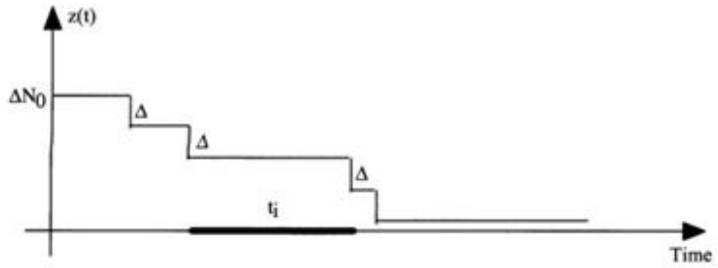
The Software Reliability Model characterizes the structure of a stochastic process that delineates the patterns of software malfunctions over time.

Time variable is regarded as a random variable characterized by a certain probability density function, (pdf). The reliability models in this class vary with respect to the assumptions made with regard to the form of the probability density function.

Time between Failure Reliability Models

According to Jelinsky&Moranda, 1972; Failures occur at some discrete time $F_i(t_i)=z(t_i)\exp(-z(t_i)t_i)$ where moments $t_1, t_2, \ldots t_i$ are independent exponential distributed random variables

$z(t_i) = [\boxtimes N_0 - (I - 1)]$ where N_0 - number of initial faults is unknown Hazard rate (the probability of failure in interval t_i)



After n failures the mean Time to Failure (MTTF) is computed as follows:

$$MTTF = \frac{1}{(N-n)\Delta}$$

Inference procedure: maximum likelihood estima- tion

$$L(t_1, t_2, ..., t_n; N_0) = \Pi_{i=1}^{n} \Delta P(Ti < ti) = \Pi_{i=1}^{n} Z(ti) \exp(-z(t_i)t_i)$$

Objective

$Max_{\Delta, N_0} L$

Resolve numerically the following two equations with respect to the parameters of the model using any method of non-linear optimization:

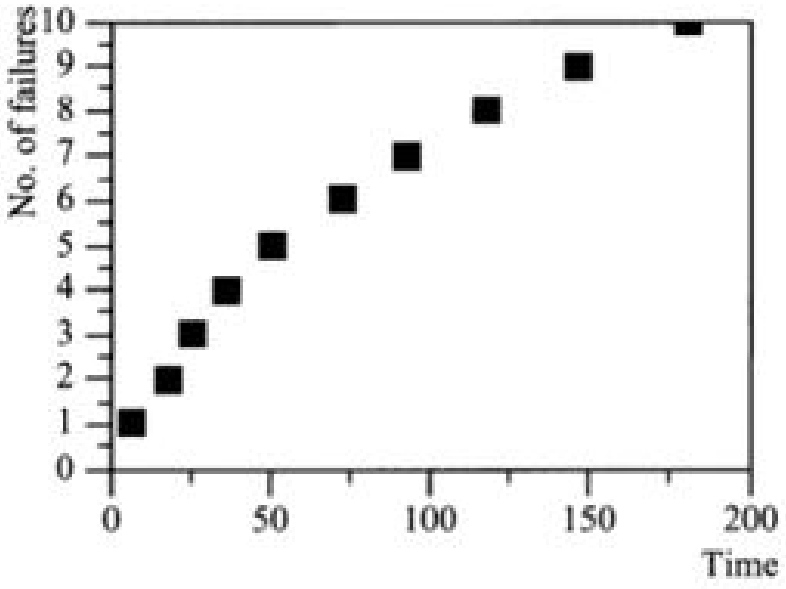
$$\Delta = \frac{1}{\sum_{i=1}^{n} (N0 - i + 1)\, ti}$$

$$\Sigma_{(i=1)}^{n}(N0 - i + 1)ti = \frac{(n\Sigma_{(i=1)}^{n} ti)}{(\Sigma_{(i=1)}^{n}(N0 - i + 1)}$$

Illustration of Jelinsky & Moranda Model

Sample software reliability data:

t1=7, t2=11, t3=8, t4=10, t5 =15, t6 =22, t7 =20, t8 =25, t9 =28, t10=35



Model Parameters Values

$\Delta = 0.0096$ and $N_0 = 11.6$

Estimated MTTF:

$$MTTF = \frac{1}{0.0096(11.6 - 10)} = 65.1$$

Jelinski-Moranda Model

Assumptions:

The software has N_0 faults at the beginning of the test.

Each fault is autonomous, and every single fault will result in a failure when subjected to testing.

The repair process is instantaneous and perfect, i.e., the time to remove the fault is negligible, new faults will not be introduced during fault removal.

Goel-Okumoto Imperfect Debugging Reliability Model

This model extends the basic JM model by adding an assumption:

A fault is removed with probability p whenever a failure occurs.

The failure rate function of the base JM model with imperfect debugging at the ith failure interval becomes

$\lambda (ti) = \phi [N- p(i – 1)], i =1, 2,…,N$

The reliability function is

$R(t_i) = e -\phi (N-p(i-1))t_i$

Failure Counting Reliability Models

Concerned with counting the number of faults detected in a certain time interval.

A representative model: Goel-Okumoto NHPP reliability model

Non-homogeneous Poisson process (NHPP)

This group of models provides an analytical framework for describing the software failure phenomenon during testing.

The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain time point.

Goel-Okumoto NHPP Reliability Model

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

N(t): Cumulative Number of Failures at time *t*

- N(t)is as a Poisson process with a time-dependent failure rate
- File dependent rate follows an exponential distribution

Where $\mathbf{m(t) = a(1-e^{-bt})}$ **and** $\mathbf{\lambda(t) == m'(t) = abe^{-bt}}$

In this equation:

- *m(t)* is expected # of failures over time
- (a.k.a. the cdf F(t)) is the failure density (a.k.a. probability density function f(t))
- **a** is the expected number of failures to be observed eventually
- **b** is the fault detection rate per fault

The Proposed Framework

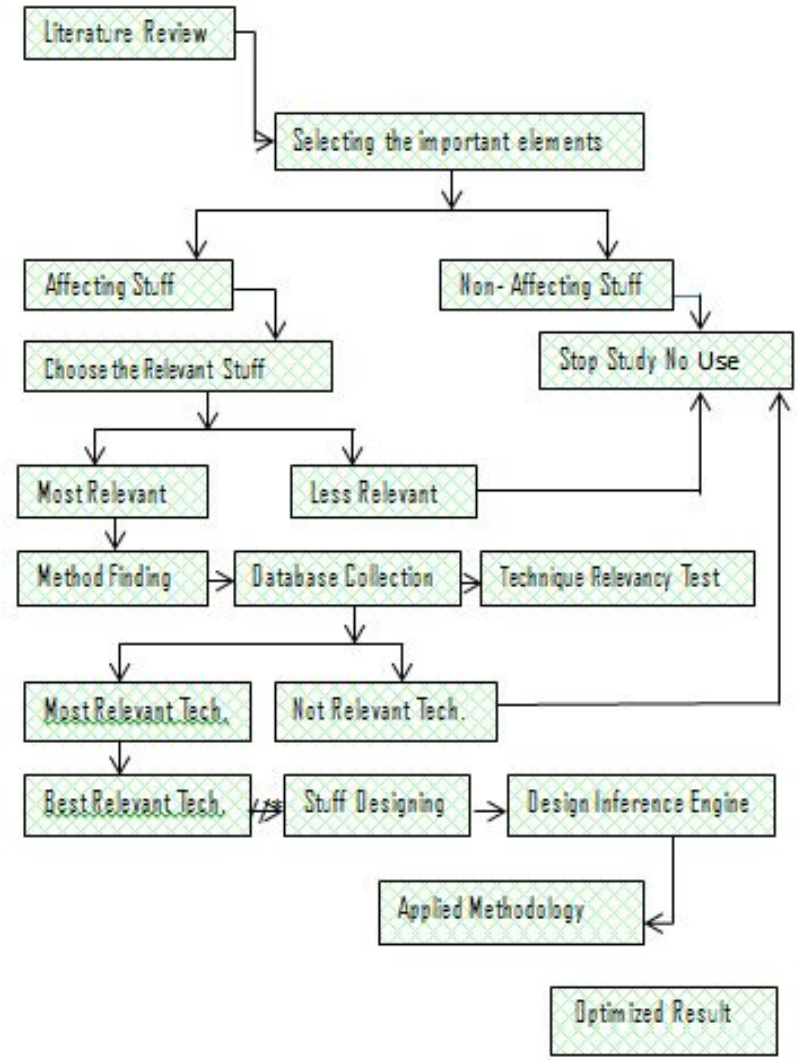


Figure 5: Structure of Proposed Methodology

The proposed methodology contains the following steps

S1. Holding a review of literature

S2. Making a selection of significant elements as arousing affect stuff as well as non-affecting stuff

S3. Choosing the relevant stuff

S4. Selection of most relevant process between most and less relevant elements

S5. Determining the properties of method

S6. Collection of Databases

S7. Testing the relevancy of technique

S8. From databases choose the most relevant technique

S9. Among multiple techniques like most relevant, less relevant; choose the best.

S10. Designing of stuff is done after best relevant technique

S11. Designing of Inference Engine

S12. Applying the proposed methodology

S13. Making the result optimal

The keys out features of the proposed model says that

Step 1: Holding a review of literature

A lot of research has been carried out reliability estimation, Reliability prediction; sometimes at early stage or later stages of software development life cycle. The objective of literature review is to look back upon a period of time or sequence of events. It shows the major research contribution in the field of software reliability and identifies the future research areas in software reliability estimation and prediction.

Step 2: Making a selection of significant elements as arousing affect stuff as well as non-affecting stuff

After reviewing the literature it is very important to select the elements that are touching on and also separate the elements i.e. non-affecting stuff.

Step 3: Choosing the relevant stuff

Selecting stuff to go through towards research requires thought and care. Not all data are relevant. Some data may be misleading. Generally different subsets of the stuff address different research questions among all finding the relevant stuff is based on the multiple factors.

Step 4: Selection of most relevant process between most and less relevant elements

Focusing on the most relevant process in a potentially overwhelming environment; repeatedly selecting examples and then labeling them according to the requirement In this process, feature selection, alternatively referred to as variable selection, attribute selection, or variable subset selection, encompasses the procedure of choosing a subset of pertinent features (variables, predictors) to be employed in the construction of a model.

Step 5: Determining the properties of method

In this step, Methods for determining the properties of heterogeneous methods are analyzed. Determination of the physical properties, methods and means of determining other properties, elemental analysis of method properties, finding the properties through experimental method are concerned.

Step 6: Collection of Databases

Depending on what is being referenced, databases - can be grouped by: “package”, “compilation” · “aggregate”, “corpus” “cluster” anything that makes sense. This step determines that databases hold one or more collections of documents. Among collection, select a database to use only.

Step 7: Testing the relevancy of technique

The choice of the right techniques is critical to test but is essential to achieve a good return on the suitable investment. This step tests the technique for finding the most relevant technique towards its process.

Step 8: From databases choose the most relevant technique

In addition, one of the most important factors is the choice of most relevant technique from database. Query for optimized technique in Database Design and Architecture often seem that prioritizing good design up-front will cost more while natural data types that fit the data being stored indicates that it is no longer relevant and can be ignored having no use.

Step 9: Among multiple techniques like most relevant, less relevant; choose the best.

One of the selections of method depends on the most relevant. To handle the increasing variety and complexity of predicting problems, many techniques have been developed in recent years. Each has its special use, and care must be taken to select the best technique for a particular application.

Step 10: Designing of stuff is done after best relevant technique

This step is concerned about the designing of stuff i.e done after the choice of most relevant technique from database. Hence, good Technique needs a good beginning in the design process. Appropriate actions or operations used in making something or bringing about a desired result: a ... No matter what is doing, it must have proper meth-

ods about the design purpose and processes after the relevant technique.

Step 11: Designing of Inference Engine

This step involved in drawing a conclusion or making a logical judgment on the basis of circumstantial evidence and prior conclusions rather than on the basis of direct observation. This inference engine interprets and evaluates the facts in the knowledge base in order to provide an answer. Typical tasks for expert systems involve classification, diagnosis, monitoring, design, scheduling etc. The expert system is empowered by the inference engine to make logical conclusions based on the rules stored in the knowledge base (KB).

Step 12: Applying the proposed methodology

All the steps will be arranged according to proposed methodology and the necessary execution will be performed on this.

Step 13: Making the result optimal

It determines the final result.

Benefits of Proposed Model

The proposed model is well organized and each step has taken with conscious thought.

- It will well efficient because only important elements have been included.
- It increases the productivity
- It Saves time
- It Saves money
- High-quality visuals increase viewer interaction
- Graphic communication: it's more than a trend.
- Data helps us analyze Decisions. Important business decisions have to be made on almost a daily basis.
- Transparent Information
- It utilizes the Analysis to make Improvements.
- Use Data to for Advantage.
- Reliability and validity are intricately connected, although they encompass distinct concepts.

In this proposed model, Reliability refers to how consistently a method measures something.

Precautions in Using Reliability Growth Models

- A fixed number of software faults will be removed within a limited period of time.
- In particular condition of in the observed process the number of faults is not fixed e.g. new faults are inserted due to imperfect fault removal, or new code is added),then one should adopt a model that does not suffer from this assumption.
- Software should not be operated in a manner different from the way it is tested the failure history of the past will not reflect these changes, and poor predictions may result.
- Most reliability growth models are primarily applicable from testing onwards. The software is believed to have reached a level of maturity where significant modifications are no longer being implemented.

Predictions about Future Work

Software reliability models are employed to estimate and forecast the reliability of software. The selection of an appropriate software reliability model for a specific scenario has garnered significant attention from researchers in the realm of software reliability. The limitation and implementation issue of the model concerns future predictions.

Conclusion

An evaluative description specified in this paper is

intended to provide a foundation for future estimation or prediction of software reliability using fit reliability growth model. In this context, current description will extend with an incorporate notation and mechanism.

Although, measurement models have been used in many implemented systems, they seem to have been used in complicating ways, possibly because a clear reliability measurement has never been constructed. Here a very brief description of the specification has been given that will carry forward in relation to the framework in future work.

References

1. G Sri Krishna, Rajib Mall, "Model Based Software Reliability

2. Bhagat Singh Rajput, Vaishali Chourey (2015) "UML Based Approach for System Reliability Assessment", International Journal of Computer Applications (0975 – 8887) 131: 2.

3. Prediction", (2010) ICISTM 2010,CCIS 54, pp. 145–55, Springer-Verlag Berlin Heidelberg 2010

4. Awad Ali et al. (2016) "Technique of Early Reliability Prediction of Software Components Using Behaviour Models".

5. Boby John, Rajeshwar S Kadadevaramath, Immanuel A Edinbarough. (2017) "Review of Software Reliability Prediction Model", "International Journal for Research in Applied Science & Engineering Technology" (IJRASET) 5: 4.

6. Martin Jedlicka et al. (2011) " UML Support for Reliability Evaluation", International Symposium on Computing, Communication, and Control (ISCCC 2009) , IACSIT Press, Singapore, Proc .of CSIT 1.

7. Durga Patel, Pallavi, (2016) Software Reliability: Models, International Journal of Computer Applications (0975 – 8887) 152: 9.

8. James J. Cusick, PMP, the First 50 Years of Software Reliability Engineering: A History of SRE with First Person Accounts

9. Recent Review and current issues in Software Reliability Growth Models under fuzzy environment, International, journal of latest trends in engineering and Technology, 6.

10. Pham H, Pham M (1991) Software reliability models for critical applications (No. EGG-2663). EG and G Idaho, Inc., Idaho Falls, ID (United States).

11. Mahmood A, Hameed K, Zameer A, Abdullah A, Sajid S (2022) Review of Software Reliability through Prediction Models. In 2022 International Conference on Recent Advances in Electrical Engineering & Computer Sciences (RAEE & CS) 1-5.

12. Mahapatra GS, Roy P (2012) Modified Jelinski-Moranda software reliability model with imperfect debugging phenomenon. International Journal of Computer Applications, 48: 38-46.

13. Dwivedi YK, Kshetri N, Hughes L, Slade EL, Jeyaraj A, Kar AK et al. (2023) "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. International Journal of Information Management, 71: 102642.

14. Di Bucchianico A (2018) Analyzing various estimation techniques for software reliability growth models.

15. Hanagal DD, Bhalerao NN (2021) Software reliability growth models. Springer Singapore.

16. Al Turk LI, Alsolami EG (2016) Jelinski-moranda software reliability growth model: a brief literature and modification. International journal of software engineering & applications (ijsea) 7.

17. Leonard JG, Nordgren RK (1997) An analysis of early software reliability improvement techniques.

18. Sahu K, Srivastava RK (2019) Revisiting software reliability. Data Management, Analytics and Innovation: Proceedings of ICDMAI 1: 221-35.

19. Kumar A (2016) Software reliability growth models, tools and data sets-a review. In Proceedings of the 9th India Software Engineering Conference 80-8.