Journal of Theoretical and Applied Information Technology
15th April 2018. Vol.96. No 7
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE RELIABILITY PREDICTION IN VARIOUS SOFTWARE DEVELOPMENT STAGES

MOHAMMAD IBRAIGHEETH (ABU-AYYASH)[1], SYED ABDULLAH FADZLI[2]

Faculty of Informatics and Computing, Universiti Sultan ZainalAbidin, 21300 Kuala Terengganu, Malaysia[1,2]

E-mail: [1]mayyash2010@gmail.com;[2]fadzlihasan@unisza.edu.my

ABSTRACT

Software reliability prediction is performed at different stages during software development process to assess if the software meets reliability requirement and avoid any potential software failures. Reliability prediction methods play an important role in guiding software project decision makers to recognizing strategies that can transform project outcomes from failure to success. This paper presents a summary of several recently published reliability prediction methods. The presented approaches are classified into either earlier or later stage. Various techniques for reliability prediction, such as probability, metric based, fuzzy logic, and neural networks are discussed. The theoretical bases of these approaches are explained whereas many of their limitations are identified. General points related to software reliability prediction topics are concluded based on this review.

Keywords *Software Reliability, Software development stages, Reliability Models, failure prediction, software metrics*

1. INTRODUCTION

The development of reliable and high-quality software projects is crucial given the extensive effect of software on the business and public sectors. Every individual is affected in certain ways by the presence of software [1]. All domains, such as medical, education, defense, transportation, and entertainment, are directly or indirectly affected by software. Reliability is an attribute of quality that must be considered in most safety-critical systems [2]. Software reliability is defined as the probability of failure-free operation for a specified period in a specific environment [3]. Reliability has become the main factor of quality that establishes a successful software project. Consequently, various techniques have been proposed for software reliability prediction. This prediction is important because it can facilitate decision makers in extracting management decisions to avoid software failures.

According to the Standish Group (2016) which is a projects database that consists of 50,000 projects, 71% of projects in 2015 failed or were challenged. One of the main reasons for software project failure is the increase in complexity because project developers continuously integrate different subsystems into project completion [4]. Poor consideration and estimation of design variables, such as reliability, quality, and user satisfaction, can result in inconsistent, incomplete, and complex systems [5]. Software reliability prediction helps decision makers avoid software failure by applying early modifications that can improve project outcomes.

Software reliability prediction is performed through various stages of the software development life cycle (SDLC). In this study, software reliability models are classified into two categories, namely, earlier and later stages. This paper aims to review several recent reliability prediction models in the literature.

This paper presents a review of the various techniques that are used for reliability prediction in different SDLC stages. We summarize eight of the most recent reliability approaches that have been published in the last 10 years. The main contribution of this study is to classify the reliability prediction approaches based on the development stages that take place on: earlier and later SDLC stage models.

The scope of this paper is limited to describe and compare several reliability prediction models which have taken place during the last 10 years. Also, this work is tightly restricted to the available material which we believe will serve our purpose. The drawn results of each model are illustrated after each description. An assessment and limitation description for each model is presented.

The included models in this paper are limited to the following criteria:

- The included models are published in the last 10 years.
- The included models focus on reliability prediction of software systems
- The prediction in the included models is performed through the software development process to predict the future reliability based on a number of expected failures/ defects.

The rest of this paper is organized as follows. Section 2 introduces the earlier stage reliability models. Moreover, this section describes the following four models in this category: component-based reliability prediction [10], component-based software reliability prediction with fault tolerance mechanisms [11], defect prediction using fuzzy logic [12], and reliability prediction using requirement and object-oriented design metrics [15], along with their assumptions, capabilities, and outputs. Section 3 presents the later stage of reliability models and describes the following four models in this category that recently appeared in the literature: software reliability prediction with test coverage [16], neural network ensemble approach [17], software reliability prediction based on the Rayleigh function [19], and neuro-fuzzy approach [21]. Section 4 discusses the prediction techniques and their capabilities used in the presented approaches. Section 5 concludes the study with a general observation related to software reliability prediction.

2. EARLIER STAGE RELIABILITY MODELS

The earlier stage models aim to predict reliability early during the requirement or design phase of SDLC. The models for the requirement phase utilize requirement metrics as an input to the model to forecast the expected number of failures. Various requirement phase metrics [12,15,22,23], such as requirement stability, specification regularity, function complexities, requirement team experience, requirement defect density, and many other metrics, are used.

The design stage models typically investigate the effect of the architectural design components and their usage profile on the overall system reliability [9,10,11,24]. Certain design level approaches use object-oriented design metrics [15,25,26], such as the complexities of coupling, encapsulation, inheritance, and cohesion.

The following discussions are summaries of selected reliability prediction models performed in the earlier stages of the software development process. Four approaches with their corresponding techniques are presented:

2.1 Component- Based Reliability Prediction

This approach [10] aims to predict software reliability early during the design stage by proposing a framework for component reliability prediction at the architectural level.

This approach assumes that the reliabilities of individual components, which significantly impact the overall system reliability, are unknown during the architectural design level because the component has not been implemented. Therefore, the component operational profile is unavailable. The approach used other available information, such as expert intuitions, component simulation, and other similar component logs, to produce the component profiles and compensate the lack of component information used for reliability prediction.

This approach uses a discrete time Markov chain process that consists of the following techniques:

- Set of states S = {S1, S2,..., SN}. This approach involves two types of component states, namely, normal and faulty behavior.
- Transition matrix P = {pij}, where pij is the probability of transition from state Si to state Sj.

The proposed component reliability prediction framework comprises three phases as follows.

- **Phase 1: Determining States**

In this phase, two types of states, namely, normal and faulty behavior, are determined. The normal behavior states B = {B1, B2,…,Bn} were obtained directly from existing documentation on the model under consideration. The faulty behavior states F = {F1, F2,…,Fm} were determined by applying a model defect analysis technique [8], which finds and classifies component inconsistencies in the architectural models. The analysis result can be used to identify these defects.

- **Phase 2: Determining Transitions**

This approach classifies transitions into three types, namely, behavioral (transition from behavioral to behavioral state), failure (transition from behavioral to failure state), and recovery (transition from failure to behavioral state). In this phase, the probabilities for each transition are determined using different processes depending on available information. Possible information sources include domain experts, requirement document, simulation, and existing similar components.

The authors suggest using hidden Markov models (HMMs) to obtain the probabilities of behavioral transition; HMMs are defined as follows:

- Set of states S = {S1, S2,..., SN} is the set obtained from phase 1.

- Transition matrix A = $\{A_{ij}\}$ represents the state transition probabilities and can be initialized with random values.

- Observation set O = {O1, O2,…, OM} represents an event/action pair of the model.
- E = $\{E_{ik}\}$ is the observation probability matrix, which denotes the probability of event O_k that occurs in state S_i. This matrix is also initialized with random values.

The training data that are used to train the HMMs are generated using available information sources, namely, expert knowledge, similar components, and available simulation traces.

f_{ij} and r_{kl} were defined to obtain failure and recovery probabilities. $f_{ij}$is the probability of evolving to failure state from F_j, which originates from behavior state B_i, whereas r_{kl} is the probability of shifting from defect K to behavioral state B_l after recovery. The assignment of values for f_{ij} and r_{kl} can be used by varying the failure and recovery probabilities, observing their effects on reliability, exploiting available information sources, and then suggesting a range of values for f_{ij} and r_{kl}.

- **Phase 3: Computing Reliability**

A component reliability obtained from this model can be expressed as follows:

$$\boldsymbol{R = 1 - \sum_{i=1}^{M} \pi(F_i)},$$

where$\pi(F_i)$ is the probability that the component is in failure state *i* and its value can be obtained from domain experts.

The proposed approach addresses the problem of predicting reliability in the component design level by leveraging available information sources and focuses only on reliability prediction for individual components. Error propagation is disregarded in the study presented in [9], which can lead to inaccurate prediction results. The proposed approach also assumes that each component independently fails, thereby indicating that the component will recover from one failure before encountering another failure. Therefore, this approach is unsuitable for multi-threaded components. Furthermore, this approach is validated by only one example, namely, the SCRover controller component, which may pose a limitation to the model.

2.2 Component-based Software Reliability Prediction with Fault Tolerance Mechanisms (FTMs)

This approach [11] aims to predict reliability in the architectural-design level with FTMs to improve component reliability and obtain accurate reliability prediction. This approach also aims to predict and improve reliability using the following six steps:

Step1: Creating Specification for Component Reliability

In this step, the component developer creates modeling elements, such as components, services, and service implementation. Each component provides multiple service implementations, whereas the model supports four types of service implementation structures, namely, sequential, branching, looping, and parallel. Each implementation provides various activities, such as internal or calling from other components.

The developer then introduces failure models and fault tolerance structures (FTS). Failure models refer to failure types and the probabilities of their occurrence. FTS consists of activities that handle specific failure types. This approach introduces different FTSs, such as RetryStructure and MultiTryCatchStructures. RetryStructure uses service re-execution to handle failures, whereas MultiTryCatchStructures is similar to exception handling in object-oriented programming and is composed of two or more parts. For example, MultiTryCatchStructures uses one part to model the normal execution and another part to handle specific failures.

Step2: Creating System Reliability Model

This step includes system architecture and usage profile modeling. System architecture modeling includes creating instances for each component and connecting them to obtain the required functionality. Usage profile comprises component services, their sequences, probabilities of transitions between them, and collection of different use cases probabilities.

Step3: Transforming Model

The created component reliability specification and reliability model are transferred into the Markov models.

The success probability sp(ia) for internal activity (ia) can be calculated as follows:

$$\mathbf{sp(ia)} = \mathbf{1} - \sum_{j=1}^{m} \mathbf{fp}_j\,(\boldsymbol{ia}),$$

where m is the number of failure types, and $fp_j(ia)$ is the probability of different failure types to occur. sp and fp_j are calculated differently and varies based on the type of service structure (sequential, branching, looping, or parallel)

Step4: **Reliability Conclusion**

In this step, reliability prediction is deduced, whereas sensitivity analysis is performed by analyzing the Markov models.

This approach defines reliability as follows:

$$\boldsymbol{R = 1 - POFOD},$$

where POFOD is the probability of failure on demand, which denotes that reliability is equivalent to the probability of service success. Therefore, service is provided for users as indicated in the usage profile.

Step5: Applying Possible Modifications

If the prediction does not meet the reliability requirement, then the developer may apply different modifications, such as revising the components and usage profiles and reconfiguring the FTSs. Otherwise, perform Step 6.

Step6: Component Implementation

This step includes implementing real system components, following the applied architectural model.

This approach is validated using only two case studies. Similar to the previously described approach of Cheung et al. [10], the proposed approach assumes that failure for each component independently occurs. Furthermore, this approach neglects the effect of component propagation error. One of the main limitations of this approach is that it assumes that the transition of control among the components follows the Markov property. This assumption indicates that this approach is inapplicable to different domains. This approach has not defined the specific methodology for estimating failure probability, usage profile, and FTS metrics. This approach assumes that these estimations could be obtained from available system document and specifications.

2.3 Defect Prediction using Fuzzy Logic

This approach [12] presents a fuzzy logic-based model for reliability prediction using software size (KLOC) and three requirement phase metrics to predict residual defects that could be found during the testing phase. The authors selected the three requirement metrics based on the study presented in [13], which specifies twelve metrics that are available during the requirement phase. The three selected metrics are as follows:

- **ERT: Experience of Requirement Team.** This metric measures the project team relevant skills and experiences through the SDLC requirement phase analysis.

- **RDD: Requirement Defect Density.** This metric measures defective requirement rates, which are obtained by reviewing requirement specifications.

- **RS: Requirement Stability.** This metric is a measure of client requests for changing during the requirement phase. A high level of changing requests leads to low requirement stability.

The abovementioned metrics are used as inputs to the implemented fuzzy model. The length of program measure (KLOC) is used as an indication of software complexity that directly affects the software residual defects and is used in model development. The model output is the predicted number of residual defects before the testing phase that could help in selecting the strategy for testing during the SDLC testing phase.

Fuzzy rules are designed with the following general form after assigning membership functions (to represent linguistic states: low (L), medium (M), and high (H)) for the three inputs and outputs.

IF ERT is L/M/H, **RDD** is L/M/H, **AND RS** is L/M/H, **THEN the number of defects** is L/M/H.

The maximum value for all input metrics is 1 whereas the maximum value for output metric is evaluated based on historical information of similar software projects or using the following equation based on the study presented in [14].

$$\boldsymbol{D = 4.2 + 0.0015(L)^{\frac{4}{3}}},$$

where D is the total number of defects, and L is the number of line of code (LOC).

All rule consequent parts are aggregated to one fuzzy set, and this set is defuzzified to obtain a final crisp value, which represents the predicted number of defects before testing. This predicted number is considered an indicator of software reliability.

One of the main limitations of this approach is that only four metrics were considered to predict reliability, where other possible factors could be used to obtain an accurate model.

2.4 Reliability Prediction using Requirement and Object-oriented Design Metrics

This model [15] uses metrics from the requirement and design levels to predict software reliability through the fuzzy interference system. This approach selects the following measures from the requirement stage and input requirement phase fuzzy model.

- Requirement Stability (**RS**): High RS leads to high reliability.

- Regularity of Specification and Documentation Reviews (**RIW**): High RIW leads to a reliable system.

- Requirement Defect Density (**RFD**): High RFD leads to low reliability.

- Complexity of New Functionality (**RC**): High RC leads to low reliability

Moreover, additional four metrics from the "object-oriented" design phase are used as inputs to the design stage fuzzy model.

- Complexity of inheritance metric (**IMc**): This metric is the number of methods inherited by a class. Reliability decreases with the increase in IMc..

- Complexity of coupling metric (**CMc**): coupling metric measures class dependencies. Reliability decreases with the increase in CMc.

- Complexity of encapsulation metric (**EMc**): Encapsulation conceals the internal programming of an entity and can be observed only by its interface. Encapsulation simplifies the program modifications. Therefore, reliability increases with the increase in EMc.

- Complexity of cohesion metric (**CoMc**): Cohesion metric measures the relationship among the elements in each class. Reliability increases with the increase in CoMc.

The abovementioned eight metrics are used as inputs to the model. Two outputs, namely, requirement-level reliability (RLR) and design-level reliability (DLR), are selected. RLR is the output of the requirement stage fuzzy model and is used as an input to the next design stage fuzzy model. DLR is the output of the design stage model. The membership functions of the model are developed based on domain experts and are categorized into very low, low, medium, high, and very high after selecting the inputs and outputs of the model. Then, fuzzy rules are defined for models in each stage in an IF–THEN form. Presently, fuzzification is performed by combining all rules from the THEN parts for each model to obtain the final output sets. Furthermore, defuzzification is performed to obtain crisp values for RLR and DLR.

This approach considers only four requirement- and four design-level object-oriented metrics. Other possible metrics for increasing reliability prediction accuracy may be considered. In this approach, reliability has no specific definition, thus, a specific reliability measure does not exist. Reliability can be increased or decreased depending on the changes in the eight input metrics.

3. LATER STAGE RELIABILITY MODELS

Many of the later stage reliability models predict reliability during the SDLC testing phase or even during system usage [16,17,19,21,27].

The times for software testing and between failures or any parameter that is related to testing data are used as the model input. The expected number of failures is a common model output. Software reliability growth models (SGRMs) are known as major reliability prediction and estimation models. SGRMs consider testing failure data or other metrics if data are unavailable in reliability prediction. SGRMs are mathematical models, which show that reliability is improved while faults are predicted and corrected. SGRMs relate the test (failure) data to popular mathematical functions, such as exponential and logarithmic.

3.1 Software Reliability Prediction with Test Coverage

This approach [16] proposes a software reliability model that integrates two factors to predicting reliability: time between failures (failure intensity) and software testing coverage. Therefore, the number of failures detected during testing is related to execution time and test coverage. The following failure intensity function, which is related to time and coverage, is derived.

$$\boldsymbol{\lambda(t,c) = \alpha_1\gamma_1 e^{-\gamma_1 c}\lambda_1(t) + \alpha_2\gamma_2 e^{-\gamma_2 t}\lambda_2(c),}$$

where$\lambda_1(t)$ is the time failure intensity function, and $\lambda_2(c)$ is the coverage failure intensity function. $\alpha_{1,}\gamma_{1,}\alpha_2, \gamma_2$are constants.

For the time failure intensity function $\lambda_1(t)$, one of the popular SGRMs can be used because SGRMs depend on execution time during testing to predict the number of failures.

For the coverage failure intensity function $\lambda_2(c)$, two models, namely, hyper-exponential and beta, are proposed. In the hyper-exponential model, the authors assume that the fault and test coverage follow the G–O reliability growth model. In the Beta model, the authors assume that the fault and test coverage follow the nonhomogeneous Poisson process model.

The failure detection in this approach is related not only to the software testing execution time but also to the volume of code that has been executed during testing. One of the main limitations of this approach is the requirement for collecting coverage measurements during the testing time, which could be unavailable.

3.2 Neural Network Ensemble Approach

This approach [17] uses neural network ensembles (PNNEs) to predict software reliability. The proposed model uses software execution time as an input, whereas the output of this system is the expected number of failures. The ensemble of the neural networks consists of the number of neural network components. Each component contains a three-layer feed-forward neural network, whereas all components have an identical architecture as follows: input, hidden, and output layers. Each component is trained with the initial weight. Then, the weights are adjusted by using training data based on the Levenberg–Marquardt learning algorithm [18]. The neural network component outputs are then combined to produce the final output. The output can be the average, median, or weighted rule of all component outputs.

In this model, a part of failure data is used as training data. The trained model is then used to check the remaining data. The predicted number of failures based on the remaining training data for a certain execution time is compared with the actual number of failures to measure the model performance. Two real-time datasets are used to measure the model performance. Among the three output combinations, the median rule provides the best performance, whereas the average rule is the worst. The performance of this model will improve if the number of neural network components in the PNNE increases.

This approach is non-parametric and does not have the difficulties of parametric models because creating many assumptions and deriving complex mathematical formulas are unnecessary. This model is applied to two datasets, and its performance is compared with the single neural network model and certain traditional SGRMs. The prediction results show that this model has a low prediction error. The main limitation of this model is the requirement to obtain statistical data to train the neural network components for each new software project.

3.3 Software Reliability Prediction based on the Rayleigh Function

This approach [19] is based on the Rayleigh function for predicting the number of defects throughout the testing process. Figure.1 illustrates the Rayleigh curve. In this model, the Rayleigh function represents the number of defects present at a certain period (that is, the Rayleigh functionrepresents the rate of defects present).

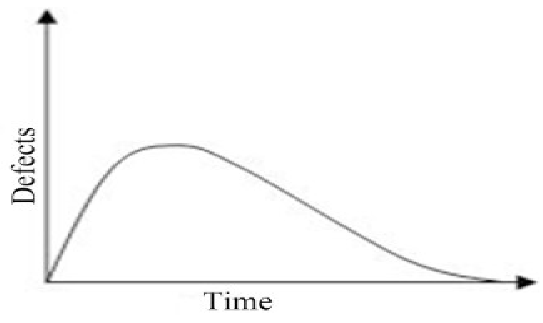

Figure.1. Rayleigh Curve

The proposed prediction model is based on the assumption that the same organization releases of a given product behave similarly to that of the same defect rate. Furthermore, defect density is related to values, such as the number of code lines and number of errors in previous testing stages. Therefore, the number of defects can be predicted for the second release of a given product based on the Rayleigh function and these values.

This study uses the second version of a product as an application, thus indicating that the testing starts from a stable product version. Defect data are collected from previous releases, whereas the expected total number of defects is calculated using linear regression. Defect prediction starts by obtaining the testing starting date and project duration. The maximum point of the Rayleigh curve (t_{max}) is determined given the two points. According to [20], approximately 40% of defects at t_{max}isfound. The expected total number of defects and defect rate is calculated based on this observation.

The total number of obtained defects could be different every release, depending on the features implemented for each release, their complexity, and other product variables that are hard to determine at the beginning of the testing phase. The prediction accuracy of this approach depends on the correctness of estimating the Rayleigh function parameters.

3.4 Neuro-Fuzzy Approach

This approach [21] uses fuzzy logic with neural networks in software reliability prediction. The fuzzy min–max algorithm, which is one of the neuro-fuzzy algorithms is used to optimize the recurrent neural network by selecting the optimal number of nodes in its hidden layer. Failure data are used as the input to the model. Then, the fuzzy min–max algorithm is used to determine the initial number of hidden layer neurons of the neural network. The network is initially framed with the basic number of hidden layer neurons and is dynamically reconfigured to predict the next failure. The cumulative execution time is used as the input to the recurrent neural network, whereas its output is the number of failures.

The recurrent neural network is trained using the back-propagation algorithm. The number of failures and cumulative execution time in the failure dataset is used as input to the network to predict the next step failure. The input dataset is structured as pairs of failure sequence number and time and is used to predict the next step failure. For validation of the model output, 80% of the dataset is used as the training data to train the network whereas 20% of the dataset is used as the test data. The average, root mean square, and mean absolute errors are used as parameters for checking the prediction accuracy. These parameters obtain the difference between the predicted and the actual values.

In this approach, a combination of fuzzy logic and neural network is used to handle reliability prediction. One limitation of this system is the arbitrary partitioning of the dataset into training and testing data. In addition, the weights are selected as random values in the neural network training process, whereas the result values are different, even for the same dataset and learning rule.

4. Software Reliability Prediction Techniques and Their Capabilities

Table 1 summarizes the eight reliability prediction techniques that are presented in this study. These techniques are classified based on their SDLC stage. The table describes the prediction techniques used in each approach and their capabilities.

5. ANALYSIS AND DISCUSSION

In the previous sections, a description of various software reliability prediction approaches has been presented. Each of described approaches could be accurate in certain cases and their performance cannot be exactly compared. Therefore, we cannot determine which approach is better than the other. Each approach is categorized based on development stage in which it takes place. Table 1 presents the techniques used and the capabilities of each approach. Some techniques use design stage metrics and hence can be applied in this stage. For example the approach used by [10] addresses the problem of reliability prediction in the component design level by using available information sources, and it focuses only on the reliability prediction for individual components. Also it assumes that each component independently fails, therefore, this approach cannot be applied for multi-threaded components. Other models depend on statistical metrics such as LOC for reliability estimation. These models can be applied once the metrics are available (LOC metric is available after implementation stage).

Fuzzy Logic and neural network techniques have the ability to predict reliability early in SDLC, even if the software metrics are not available. For example, in requirement and design phases, LOC metric is not available. These approaches also need similar or historical data from similar software projects to train the predictive model [17].

In this paper, many limitations are identified for the presented approaches. For example, the proposed approach in [11] which is component based reliability prediction model, neglects the effect of component propagation error, also it assumes that the components follow the Markov property which make this approach not applicable to different domains. In [12 and 13], a limited number of metrics were considered to predict the reliability. However, other possible metrics could be used to obtain more accurate model. The approach presented in [16] depends on collecting measurement about the volume of executed code during testing time which could be unavailable. Another example of performance limitations identified in [19], is that the prediction accuracy depends on the correctness of estimating the Rayleigh function parameters. Also the Neuro-Fuzzy approach proposed in [21] has limitations such as: an arbitrary partitioning of the dataset into training and testing data is performed, and the weights for neural network training process are selected randomly.

6. CHALLENGES FOR RELIABILITY PREDICTION AND RESEARCH DIRECTIONS

Even though there are many reliability prediction models are proposed in the literature, the applying of those models in practice has many challenges:

• Most models are verified using certain cases. However, the prediction model could not be applicable for other software projects. Finding a general reliability prediction approaches are required.

• The metrics used in reliability prediction models could not ensure an accurate prediction. As a new software projects appear, many other metrics can be extracted and investigated.

• Many software project datasets are not shared by the software companies because of privacy issues [28]. Thus, the prediction models may be very limited to some of publically available datasets and open source software projects. Increasing the available datasets will improve the prediction model evaluation process.

• Usually, the publically available datasets are extracted from different domains and contain different metrics types. Usingthe cross-analysis to identify a common reliability metrics (factors) will be helpful for producing general prediction approaches.

7. CONCLUSION

Software reliability prediction is important topic in software system engineering. Reliability prediction can improve the quality of software systems and convert their overcome from failure to success. This paper describes eight reliability prediction models which have taken place during the last 10 years, and it is limited to describe and compare the techniques used in those models.

Software reliability prediction and analysis are performed during different SDLC stages. The prediction results can be used as indicators of quality and provide feedback to the software project decision makers in improving project performance and avoiding expected failures.
Most models were verified using certain case studies, and were implemented based on specific failure metrics. Consequently, those models could not be applicable for other projects. Therefore, it is a necessity to develop prediction models to be generally applied on any software project. Cross-analysis approach could be used to figure out the common failure factors of the different projects in order to develop new prediction models. Also, most of those models are implemented to predict the reliability during specific phase of SDLC. There is a need to develop predictive tool that can be used during any phase of SDLC.

The following points comprise the conclusion of our study:

a. Software reliability is a function of the expected number of failures in the considered software.

b. In general, reliability prediction depends on available failure data or/and software metrics collected during various stages of the software implementation process.

c.The selection of the model structure is one of the main issues in the reliability prediction process. Several models are probability-based, whereas other models use additional techniques, such as soft computing techniques.

d. The Markov model is an example of probability-based models, whereas fuzzy logic and neural network approaches are examples of soft computing reliability models.

e. Soft computing approaches are useful in cases, where the relationship between model input and predictive output is nonlinear or deviates from a regular form.

f. Model parameter estimation is typically performed using available failure data or by analyzing software requirement and specifications.

g. The validation of prediction approaches is typically performed using software project case studies and by comparison with other model implementations.

h. Several models are suitable for certain cases. However, they are not the best option for other cases.

i. The evaluation of prediction model accuracy is determined based on measures, such as average and root mean square errors that find the difference between the actual and predicted values.

REFERENCES

[1]Dalal SR, Lyu MR, Mallows CL. Software Reliability. Encyclopedia of Biostatistics. 2005. (book)

[2]Pandey AK, Goyal NK. Early Software Reliability Prediction.Springer, India; 2015.

[3]Radatz J, Geraci A, Katki F. IEEE standard glossary of software engineering terminology. IEEE Std. 1990 Sep;610121990(121990):3.

[4]Ryan J, Sarkani S, Mazzuchi T. Leveraging variability modeling techniques for architecture trade studies and analysis. Systems Engineering. 2014 Mar 1;17(1):10-25.

[5]Eisner H. Managing complex systems: thinking outside the box. John Wiley & Sons; 2011 Jan 6.

[6]Cai X, Lyu MR. Software reliability modeling with test coverage: Experimentation and measurement with a fault-tolerant software project. InSoftware Reliability, 2007.ISSRE'07. The 18th IEEE International Symposium on 2007 Nov 5 (pp. 17-26). IEEE.

[7] Pham, H, System Software Reliability: Reliability Engineering Series, Springer; 2006.

[8] Roshandel R, Schmerl B, Medvidovic N, Garlan D, Zhang D. Understanding tradeoffs among different architectural modeling approaches. InSoftware Architecture, 2004. WICSA 2004.Proceedings. Fourth Working IEEE/IFIP Conference on 2004 Jun 12 (pp. 47-56). IEEE.

[9] Cortellessa V, Grassi V. A modeling approach to analyze the impact of error propagation on reliability of component-based systems.Component-Based Software Engineering. 2007:140-56.

[10] Cheung L, Roshandel R, Medvidovic N, Golubchik L. Early prediction of software component reliability.InProceedings of the 30th international conference on Software engineering 2008 May 15 (pp. 111-120).ACM.

[11] T.-T. Pham and X. Defago, "Reliability prediction for component-based software systems with architecturallevel fault tolerance mechanisms," in Proc. of the 8th International Conference on Availability, Reliability and Security (ARES'13), Regensburg, Germany. IEEE, September 2013, pp. 11–20.

[12] Yadav DK, Charurvedi SK, Mishra RB. Early software defects prediction using fuzzy logic. International Journal of Performability Engineering. 2012 Jul 1;8(4):399-408.

[13] Li M, Smidts CS. A ranking of software engineering measures based on expert opinion. IEEE Transactions on Software Engineering. 2003 Sep;29(9):811-24.

[14] Gaffney JE. Estimating the number of faults in code.IEEE Transactions on Software Engineering. 1984 Jul(4):459-64.

[15] Rizvi SW, Khan RA, Singh VK. Early Stage Software Reliability Modeling using Requirements and Object-Oriented Design Metrics: Fuzzy Logic Perspective. International Journal of Computer Applications. 2017;162(2).

[16] Cai X, Lyu MR. Software reliability modeling with test coverage: Experimentation and measurement with a fault-tolerant software project. InSoftware Reliability, 2007.ISSRE'07. The 18th IEEE International Symposium on 2007 Nov 5 (pp. 17-26). IEEE.

[17] Zheng J. Predicting software reliability with neural network ensembles.Expert systems with applications. 2009 Mar 31;36(2):2116-22.

[18]Haykin SS. Neural networks: a comprehensive foundation. Tsinghua University Press; 2001.

[19] Vladu AM. Software reliability prediction model using rayleigh function. Politehnica University Scientific Bulletin. 2011;73(4).

[20] Laird L. In Praise of Defects Stevens Institute of Technology.Retrieved September 20, 2013.

[21] Bhuyan MK, Mohapatra DP, Sethi S. Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach. International Journal of Electrical and Computer Engineering. 2016 Aug 1;6(4):1929.

[22] Radjenović D, Heričko M, Torkar R, Živkovič A. Software fault prediction metrics: A systematic literature review. Information and Software Technology. 2013 Aug 31;55(8):1397-418.

[23] Li M, Smidts CS. A ranking of software engineering measures based on expert opinion. IEEE Transactions on Software Engineering. 2003 Sep;29(9):811-24.

[24] Gokhale SS, Trivedi KS. Reliability prediction and sensitivity analysis based on software architecture. InSoftware Reliability Engineering, 2002.ISSRE 2003.Proceedings. 13th International Symposium on 2002 (pp. 64-75). IEEE. (conference)

[25] Anil K, Namrata D. Reliability Estimation of Object-oriented Software: Design Phase Perspective. International Journal of Advanced Research in Computer and Communication Engineering. 2015;4(3):573-7.

[26] Yadav A, Khan RA. Reliability Quantification of an OO Design-Complexity Perspective.InAdvances in Computer Science, Engineering & Applications 2012 (pp. 577-585). Springer, Berlin, Heidelberg.

[27] Okamura H, Dohi T. A novel framework of software reliability evaluation with software reliability growth models and software metrics.InHigh-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on 2014 Jan 9 (pp. 97-104).IEEE.

[28] Peters F, Menzies T. Privacy and utility for defect prediction: Experiments with morph. InProceedings of the 34th International Conference on Software Engineering 2012 Jun 2 (pp. 189-199). IEEE Press.

Table 1: Software Reliability Prediction Techniques and Their Capabilities

Section	SDLC	Techniques Used	Capabilities

	stage		
2.1	Earlier	• Uses available information, such as expert knowledge and component simulation to produce the component usage profiles. • Uses the Markov chain process to compute component failure probability.	Reliability prediction for design-level individual components
2.2		• Defines failure types and probabilities of their occurrences. • Provides FTS activities to handle specific failure types. • Uses the Markov process for reliability and sensitivity analyses.	• Component-based architectural design-level reliability prediction • Presents FTM to improve prediction accuracy
2.3		Uses the fuzzy logic system for reliability prediction through KLOC and three requirement phase metrics.	Predicts residual defects that may occur during the testing phase
2.4		Uses four metrics from the requirement phase and four from the design phase to predict software reliability through the fuzzy interference system.	Provides two outputs, namely, RLR and DLR.
3.1	Later	• Relates software execution time and testing coverage to find failure-intensity function • Uses SGRMs to predict the number of failures based on the testing execution time. • Two models for finding coverage failure intensity have been proposed.	Predicts reliability during the testing time based on two factors, namely, the time between failures and software testing coverage.
3.2		• Uses PNNE to predict software reliability. • Uses software execution time as the input. • Trains the network using failure datasets.	The output of this system is the expected number of failures.
3.3		• The prediction of the number of defects throughout the testing process is based on the Rayleigh function. • Assumes that the same organization releases of a given product behave similarly to that with the same defect rate. • Relates to defect density to values, such as the number of code lines and number of errors in the previous testing stages.	Predicts the number of defects for the second and later releases of a given product.
3.4		• Uses fuzzy logic and neural networks. • Uses fuzzy min–max algorithm to optimize the recurrent neural network. • Uses cumulative execution time as the input to the recurrent neural network. • Uses failure data as the input to the model.	Predicts the next-step failure based on pairs of failure sequence number and time.