Chapter 2

Literature Review

2.1 Introduction

In this chapter, we have provided a brief literature review of our work. The literature review includes Software Bug Prediction, Clone Evolution Prediction, and Software Reliability Prediction. These are well-known research area in software engineering. There are number of approaches for prediction of above mention attributes. However, it was noticed that not much research have been conducted on temporal analysis of above attributes in software engineering.

2.2 Literature Review (Software Bug Prediction)

Software Bug Prediction is one of the challenging aspects of Software Engineering. The techniques used include Rule-Based methods, Artificial Neural Network, Support Vector Machines, Nearest Neighbors, Decision Trees, and also other advanced machine learning techniques. There are number of papers [101, 100, 103, 102, 105] in which the authors applied machine learning for software fault prediction. They use NASA MDP dataset for experiments. Some authors also applied clustering techniques to improve the performance of software fault prediction models. Another group of authors have applied a combination of one or two methods, i.e., ensemble methods for software fault prediction.

Previous studies on temporal bug pattern predictions are based on traditional time series modeling like ARIMA[227] that is used to predict a stationary time series data. In another paper by Hongyu et al.,[76] Polynomial Regression is used to predict the bug growth patterns in Eclipse.

A systematic literature review of different approaches applied in the area of software bug prediction is presented in Table 2.1.

TABLE 2.1: Literature Review(Software Bug Prediction)

Author (Year)	Objectives of Study	Methodology /Approaches/Tool-s/Techniques	Remarks
Khoshgoftaar, Allen, and Busboom (2000)[97]	Predicted Software Quality by using Eight Method Level Metrics	By using Case-Based Reasoning	Type-I and Type-II error were used as performance evaluation metrics, and prediction model was reasonably successful.

Xu, Khoshgoftaar, and Allen (2000)[233]	Predict faults on large telecommunications the system developed with Protel language used 24 method level metrics and four execution metrics.	Used principal component analysis for feature selection and then applied fuzzy nonlinear regression (FNR).	They reported that fuzzy nonlinear regression method is an encouraging technology for early fault prediction.
Guo and Lyu (2000)[71]	Predict software quality on a medical imaging system developed with Pascal and Fortran languages.	Finite mixture model analysis with Expectation–Maximization (EM) algorithm.	Type-I, Type-II errors are used as performance Evaluation metrics. Best value for Type-II error was 13%.
Khoshgoftaar, Gao, and Szabo (2001)[98]	Predicted faults on two large-scale software applications.	Zero-inflated Poisson regression model (ZIP) and file level metrics.	Performance evaluation metrics were average absolute error (AAE) and average relative error (ARE). ZIP provided better results compared to PRM. Also, AAE and are values of ZIP was smaller than PRM's (Poisson Regression Model) error results.

Schneidewind (2001)[189]	Software Quality Prediction by using six method level metrics on a spacecraft software dataset.	Boolean discriminant functions (BDF) and logistic regression functions (LRF).	Type-I error, Type-II error, overall misclassification rate, and the rate of correctly classified non-faulty modules (LQC) parameters were used as performance evaluation metrics. BDF's performance was better than LRF's performance.
Emam, Melo, and Machado (2001)[56]	Predicted the fault-prone Classes on a commercial Java application.	Logistic regression and class level metrics.	They reported that inheritance depth and export coupling are the most useful metrics to identify the fault-prone classes.
Khoshgoftaar, Geleyn, and Gao (2002)[99].	Predicted the fault-prone Classes on two Applications which configure wireless products.	PRM, ZIP, and module-Order modeling techniques with five file level metric.	The performance of module-order model was good on These datasets and other approaches did not provide good results.

Mahaweerawat, et al.(2002)[14]	Predict software faults.	fuzzy clustering and then, they applied radial basis function (RBF)	Type-I error, Type-II error, overall misclassification rate, inspection, and completeness were used as performance Evaluation metrics. RBF's accuracy was 83% and MLP's accuracy was 60%. Therefore, RBF method was better than MLP for software fault prediction in this study.
Khoshgoftaar and Seliya (2002a)[102]	Software Quality Classification.	SPRINT and CART methods with 28 method level metrics (24 product metrics and four execution metrics) SPRINT is a classification tree Algorithm and CART is decision tree algorithm.	Performance evaluation metrics were Type-I error, Type-II error, and overall Misclassification Rate. They reported that SPRINT algorithm had lower Type- I error, and the model based on SPRINT was robust.
Pizzi, Summers, and Pedrycz (2002)[175]	Predicted software quality on a research prototype.	Multi-layer perceptron and method/class level metrics.	Accuracy parameter was used as performance Evaluation metric. They stated that using median-adjusted class labels is an effective pre-processing technique before multi layer perceptron is applied.

Khoshgoftaar and Seliya (2002b)[100]	Software Quality Prediction models for a large telecommunications system.	Used tree based software Quality Prediction Models Applied design metrics. CART-LS (least squares), S-PLUS, and CART-LAD (least absolute deviations) were investigated in this study.	CART-LAD was proposed for software Quality prediction.
Reformat (2003)[183]	Predict software faults on a commercial medical imaging system.	Fuzzy rule-based models for reasoning about the number of software faults and 11 method level metrics used	Classification rates change between 62.82% and 79.49%. Classification rate was 85.90% when Meta-model prediction system was used.
Koru and Tian (2003)[115]	Investigated the relationship between high defect and high complexity modules	Tree-based models and method level metrics (15 method level metrics for IBM products and 49 method level metrics for Nortel Networks) on Six large scale products of IBM and Nortel Networks.	Mann–Whitney U-test was applied for performance evaluation. They showed that high defect-prone modules are complex modules, But they are not the most complex ones.

Denaro, Lavazza, and Pezzè (2003)[52]	Predict Software Faults on an industrial telecommunica-tions system.	Applied logistic regression by using class level metrics.	Expected to see a correlation Between fault-proneness and at least one metric. They found that none of these metrics are correlated with fault-proneness, and multivariate models do not provide any advantage compared to lines of code metric.
Thwin and Quah (2003)[206]	Predicted Software Quality .	General Regression Neural Network (GRNN) and Ward Neural Networks.	Evaluation parameters were R2, r, average square error, average absolute error, Minimum absolute error, and maximum absolute error. reported that GRNN provided Much better results than Ward networks.
Khoshgoftaar and Seliya (2003)[101]	Predicted Software Faults.	CART-LS, CART-LAD, S-PLUS, multiple linear regression, neural networks, case based Reasoning on a large telecommunications system. Twenty-four product and four execution metrics were independent variables For this analysis.	They used two-way ANOVA randomized complete block design model as experimental design approach and multiple-pairwise comparison for performance ranking. Best performance was achieved with the CART-LAD algorithm, and the worst one was S-PLUS.

Guo, Cukic, and Singh (2003)[70]	Predicted fault-prone modules on NASA's KC2 project.	Dempster–Shafer Belief Networks and 21 method level metrics.	Performance evaluation metrics were the probability of detection, effort, and accuracy.
Denaro, Pezzè, and Morasca (2003)[52]	Predict the fault-prone modules on antenna configuration system Apache 1.3, and Apache 2.0.	Logistic regression with Method level metrics.	Performance Evaluation metrics used: R2, completeness, completeness of faulty modules, and correctness of faulty modules. They showed that logistic regression with cross-validation is an effective approach.
Menzies, DiStefano, Orrego, and Chapman (2004)[141]	Predicted fault-prone modules on public datasets locating in PROMISE repository.	Naïve Bayes algorithm. Method level metrics were used.	Probability of detection (PD) and the probability of false alarm (PF) were performance evaluation Metrics. Naive Bayes provides better performance than J48 algorithm. Furthermore, they reported that PD on KC1 dataset was 55% and PD for Fagan inspections was between 33% and 65%. For industrial inspections, PD for Fagan inspections was between 13% and 30%.

Khoshgoftaar and Seliya (2004)[103]	Software Quality Classification Techniques on large telecommunications System.	Logistic regression, case based reasoning, classification and regression trees (CART), tree based classification with S-PLUS, Spring-Sliq, C4.5, and Treedisc by using 24 product metrics and four execution metrics.	Expected cost of misclassification metric was chosen as performance evaluation parameter. They stated that data and system characteristics affect the performance of prediction models in software Engineering.
Wang, Yu, and Zhu (2004)[220]	Quality prediction on a large telecommunications system developed With C language.	Artificial neural networks. Seven product metrics calculated with MATRIX analyzer. To improve the understandability of neural networks, they applied Clustering Genetic Algorithm (CGA).	Accuracy parameter was used for performance evaluation. the accuracy of the prediction which was performed with rule set of CGA is lower than the accuracy of the neural networks based prediction; results were more understandable When rule set of CGA was used.

Mahaweerawat, Sophat-sathit, Lursinsap, and Musilek (2004)[135]	Identity the fault-prone modules Of 3000 C++ classes collected from different web pages.	First used Multi-layer perceptron. Later they applied radial basis functions (RBF).	Type-I error, Type- II error, inspection, and achieved quality parameters. Accuracy, achieved quality, inspection, Type-I error, Type-II error were 90%, 91.53%, 59.55%, 5.32%, and 2.09% respectively. Also, they stated that they could not identify only 2.09% of faulty classes.
Kanmani, Uthariaraj, Sankara-narayanan, and Tham-bidurai (2004)[90]	Software quality prediction. on student projects developed in Pondicherry Engineering College.	General Regression Neural Networks (GRNN) technique By using 64 class level metrics. Principal component analysis was used for feature selection.	Evaluation parameters were correlation coefficient (r), R2, average square error, average absolute error, maximum absolute error, and minimum absolute error parameters. They reported that GRNN The technique provided good results for fault prediction.

Zhong, Khoshgof-taar, and Seliya (2004)[245]	Cluster as fault-prone or not fault-prone by examining not only the representative of each cluster, but also some statistical data such as global mean, minimum, maximum, median, 75%, and 90% of each metric.	K-means and Neural-Gas clustering methods to cluster modules, also supported by an expert who is 15 years experienced engineer.	Mean squared error (MSE), average purity, and time parameters were Used. False positive rate (FPR), false negative rate (FNR), and overall misclassification rate parameters were used. Neural-Gas performed much better than K-means according to the MSE parameter, and its average purity was slightly better than K-means clustering's Purity value.
Xing, Guo, and Lyu (2005)[232]	Predicted software quality on a medical imagining software.	Support Vector Machines (SVM) and 11 method level metrics (Halstead, McCabe, Jensen's program length, and Belady's bandwidth)	Type-I error and Type-II error were used to evaluate the performance of the model. They reported that SVM performed better than quadratic discriminant analysis and classification tree.

Koru and Liu (2005)[113]	Investigated the effect of module size on fault prediction on public NASA datasets.	J48 and KStar algorithms.	F-measure was used for performance evaluation. Both method level and class level metrics were investigated. The best performance was achieved with the J48 algorithm and Bayesian Networks.
Khoshgoftaar, Seliya, and Gao (2005)[104]	A new three group Software quality classification technique on two embedded software which configures wireless products.	C4.5 decision tree, discriminant analysis, case based reasoning, and logistic Regression. They applied five file level metrics.	Performance evaluation metrics was expected the cost of misclassification. They reported that three groups such as high, medium, and low labels provided encouraging performance for fault prediction.
Koru and Liu (2005)[112]	Built fault prediction models on public NASA datasets.	J48, K-Star, and Random Forests.	F-measure was selected as performance evaluation metric. They stated that large modules had higher F-measure values for J48, K-Star, and Random Forests algorithms.

Challagulla, Bastani, Yen, and Paul (2005)[41]	Software fault prediction on public NASA datasets.	Linear regression, pace regression, support vector regression, neural network For continuous goal field, support vector logistic regression, a neural network for discrete goal field, Naive Bayes, instance based learning (IBL), J48, and 1-R techniques by using method level metrics.	Performance Evaluation metric was an average absolute error. IBL and 1-R was better than the other algorithms according to the accuracy parameter, and they stated That principal component analysis did not provide an advantage.
Gyimothy, Ferenc, and Siket (2005)[72]	Validate object oriented metrics for fault prediction on Mozilla Open source project.	Logistic regression, Linear regression, decision trees, and neural networks. Class level metrics were used.	Performance evaluation metrics were completeness, correctness, and Precision. They reported that coupling between object classes (CBO) metric is very useful for fault prediction.
Ostrand, Weyuker, and Bell (2005)[156]	Predicted the location and number of faults on two industrial systems.	Negative binomial regression model. some metrics they used were programming Language, the age of the file, and file change status.	Performance Evaluation metric was accuracy. They reported that the accuracy of general performance was 84% and the simplified model's accuracy Was 73%.

Tomaszewski, Lundberg, and Grahn (2005)[208]	Accuracy of early fault prediction in modified code on a large Telecom-munications system.	Regression techniques and method/class level metrics.	Performance evaluation parameter R2 (determination coefficient). Performance of models improves when this metric is used.
Hassan and Holt (2005[75])	Identify the top ten fault-prone Components of six open source projects.	They proposed some techniques such as most frequently modified (MFM), most recently modified (MRM), most frequently fixed (MFF), and most recently fixed (MRF). Metrics such as change Frequency and size metrics were used.	Performance evaluation Metrics were hit rate and average prediction age (APA). MFM and MFF were more successful than the other methods.
Challagulla, Bastani, and Yen (2006)[40]	Predicted software faults on public NASA datasets by using 21 method level metrics.	Memory Based Reasoning (MBR).	Performance evaluation metrics used: Probability of detection, probability of false alarm, and Accuracy. They proposed a framework and users can choose the MBR Configuration which can provide the best performance from this framework.

Khoshgoftaar, Seliya, and Sundaresh (2006)[105]	Predict software faults on a Large telecommunications system to predict software faults.	Case-based Reasoning by using 24 product and four execution metrics.	Performance evaluation metrics were average absolute error and average Relative error. They reported that case based reasoning works better than multivariate linear regression and correlation based feature selection and stepwise regression model selection did not improve the performance of models.
Nikora and Munson (2006)[152]	Built high-quality software fault Predictors on mission data system of Jet Propulsion Laboratory.	Method Level Metrics.	In this study, they built a framework which includes the rules for fault definition, and they proved that fault predictors which look at the token differences between two versions are more effective.
Zhou and Leung (2006)[248]	Predicted high and low severity faults on NASA's KC1 dataset.	Logistic Regression, Naive Bayes, Random Forests, The Nearest neighbor with generalization techniques. Object-oriented software metrics.	Performance evaluation metrics were correctness, completeness, and precision. They reported that low severity faults could be predicted with a better performance compared to high severity faults.

Mertik, Lenic, Stiglic, and Kokol (2006)[143]	Estimated software quality on NASA datasets.	Prunned C4.5, unpruned C4.5, multimethod, SVM using RBF kernel, SVM using linear kernel techniques by using method level metrics.	Multi-method includes several methods and its performance with respect to performance was very high.
Boetticher (2006)[33]	Investigated the effects of datasets on software engineering.	Applied J48 and Naive Bayes techniques for the analysis.	Datasets were divided into three parts: a training set, nice neighbors test set, and nasty neighbors test set. Nice neighbors are neighbors who are close to the same class, and nasty neighbors are neighbors who locate in different classes. He showed that the accuracy was 94% for nice neighbors test set and the accuracy was 20% for nasty neighbors test set.
Bibi, Tsoumakas, Stamelos, and Vlahvas (2008)[30]	Software Fault Prediction on Pekka dataset which was Collected in one of Finland's banks.	Regression via Classification (RvC) Technique. Used different metrics such as disk usage, processor usage, number of users, and document Quality.	Performance evaluation metrics were average absolute error and accuracy. They reported that RvC could be used to enhance the understandability of regression models.

Gao and Khoshgoftaar (2007)[64]	Software Fault Prediction on two embedded software applications which configure the wireless telecommunications Products.	Poisson regression, zero-inflated Poisson regression, negative binomial regression model, Zero-Inflated negative binomial, and Hurdle Regression (HP1, HP2, HNB1, HNB2) techniques. Used using five file level metrics.	Performance evaluation metrics were Pearson's chi The square measure, information criteria, average absolute error (AAE), and average relative error (ARE). They reported that model based on Zero-Inflated negative binomial technique performs better than the other algorithms according to the information criteria and chi-square measures.
Li and Reformat (2007)[124]	Predicted software faults on the JM1 dataset .	SimBoost method was proposed in this study and fuzzy labels for classification were suggested. Method level metrics used.	Accuracy: was used for performance evaluation. This method with fuzzy labels worked well on the dataset.
Mahaweerawat, Sophatsathit, and Lursinsap (2007)[134]	Software Fault prediction on a dataset which is not explained in the paper.	Self-organizing map clustering and then applied RBF. Method level metrics were used.	Performance evaluation metric was mean of absolute residual (MAR). They reported that accuracy was 93% in this study, but accuracy is not an appropriate Parameter for imbalanced datasets.

Menzies et al. (2007a)[86]	Software fault prediction on public NASA datasets.	Investigated several data mining algorithms.	Performance evaluation metrics were PD, PF, and balance. They achieved the best performance Naive Bayes algorithm, and before this algorithm is applied, they used a logNum filter for software metrics. They reported that Naive Bayes outperformed J48.
Zhang and Zhang (2007)[243]	Software fault prediction on public NASA datasets.	Investigated several data mining algorithms.	Criticized Menzies et al.'s study (2007a) and they stated that PD and PF parameters are not enough to evaluate the performance of models. They reported that precision was very low in Menzies et al.'s study (2007a) and this model would not be very useful in practice.

Menzies, Dekhtyar, Distefano, and Greenwald (2007b)[86]	Software fault prediction on public NASA datasets.	Investigated several data mining algorithms.	Responded to comments of Zhang and Zhang (2007) and they stated that precision is not a useful parameter for software engineering problems, models which have low precision provided remarkable results for different problems. Precision is supposed to be high too, but in practice, this is not a real case.
Ostrand, Weyuker, and Bell (2007)[157]	Predicted fault-prone modules.	Negative binomial regression model by using several Metrics such as file size, file status, the number of changes on file, and programming language.	They reported that negative binomial regression model is very useful according to the accuracy parameter.

Yang, Yao and Huang (2007)[235]	Software Fault Prediction on an artificial dataset.	Fuzzy Self-Adaptation Learning Control Network-(FALCON)	Inputs for FALCON were software metrics, and outputs were reliability and effectiveness. They stated that this model could measure several quality features such as reliability, performance, and maintainability, but they did not try this approach with real datasets. Therefore, it is clear that their study is at early stages.
Pai and Dugan (2007)[158]	Calculate the fault-proneness of modules On NASA Datasets.	Linear regression, Poisson regression, and logic regression to calculate conditional probability densities of Bayes Networks' nodes and then, they used these networks to calculate the fault-proneness of modules	Evaluation parameters were sensitivity, specificity, precision, false positive and false negative parameters. They reported that weighted methods count (WMC), coupling between object classes (CBO), the response for a class (RFC), and lines of code metrics are very useful to predict the fault-proneness of modules.

Wang, Zhu, and Yu (2007)[221]	Predicted software quality on two large telecommunica-tions system.	Genetic algorithm by using 18 method level and 14 file level metrics.	Performance evaluation Metrics were a Type-I error, Type-II error, and overall misclassification rate. They reported that the proposed model is better than S-PLUS and TreeDisc.
Seliya and Khoshgof-taar (2007a)[190]	Predicted software faults with Limited fault data. The JM1 dataset was used as training dataset and KC1, KC2, and KC3 were used as test datasets.	Expectation–Maximization (EM) technique.	They used Type-I error, Type-II error, and overall misclassification rate parameters for performance evaluation. EM technique is first used to give labels to unlabeled data and then all the modules are used to build the prediction model. EM based quality model provided acceptable results for semi-supervised fault prediction problem.
Koru and Liu (2007)[114]	Identified change-prone classes on K-Office and Mozilla open-source projects.	Tree-based models and Class label metrics used.	20% of classes changed and tree based models were very useful to identify these change-prone classes.

Cukic and Ma (2007)[48]	Predicted fault-proneness of modules and investigated 16 learning algorithms on JM1 dataset	Method level metrics used.	Evaluation parameters were PD and PF. Only four algorithms provided PD value which is higher than 50% and PF value which is lower than 50%.
Tomaszewski, Håkansson, Grahn, and Lundberg (2007)[207]	Predict software faults on two software systems developed by Ericsson.	Expert opinion and univariate linear regression analysis.	Accuracy parameter was used for performance evaluation. The accuracy of the prediction model based on class level metrics was better than the model based on component level metrics. They Reported that statistical approaches are more successful than expert opinion and experts did not predict faults easily in large datasets.

Seliya and Khoshgoftaar (2007b)[190]	Predict the fault-proneness of program modules when the defect labels for modules are unavailable.	A constraint-based Semi-supervised clustering scheme that uses K-means clustering method. However, their approach uses an expert's domain knowledge to iteratively label clusters as fault-prone or not.	Performance evaluation metrics were Type-I error, Type-II error, and overall misclassification rate. They reported that semi-supervised clustering scheme provided better performance than traditional clustering methods and half of the Modules which could not be labeled were noisy.
Olague, Gholston, and Quattlebaum (2007)[155]	To predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes (on open-source Rhino project's six versions)	Univariate binary logistic regression (UBLR) and multivariate binary logistic regression (MBLR) techniques were used for the analysis. Chidamber–Kemerer (CK) metrics suite, Abreu's object-oriented metrics (MOOD), and Bansiya and Davis's quality metrics (QMOOD) were investigated.	Accuracy parameter was used for model validation, and Spearman correlation was applied to examine the metrics' effects. They reported that CK and QMOOD metrics are very useful for fault prediction, but MOOD metrics are useless. Furthermore, they stated that MBLR models are useful for iterative and agile software development processes.

Binkley, Feild, and Lawrie (2007)[31]	Predicted Software Fault.	Linear mixed-effects Regression model. QALP score, total lines of code, and lines of code except for comments and blank lines used.	Determination coefficient was used as performance evaluation parameter. They reported that neither QALP nor size measure is a good predictor for Mozilla project.
Menzies T, Greenwald J, Frank A (2007)[86]	Mining Static Code Attributes to Learn Defect Predictors.	Rule-based or decision-tree learning methods and naive Bayes data miner with a log-filtering preprocessor on the numeric data.	Bayesian method performs best.
Jiang, Cukic, and Menzies (2007)[86]	Predicted software faults using early life cycle data.	Investigated 1-R, Naive Bayes, Voted Perceptron, Logistic Regression, J48, VFI, IBk, and Random Forests for fault prediction by using metrics extracted from textual requirements and code metrics.	PD and PF were selected as performance Evaluation metrics. They reported that the performances of algorithms except Voted Perceptron algorithm improved when code metrics were combined with requirement metrics.

Bibi et al. (2008)[30]	Software fault prediction problem (to estimate the number of software faults with a confidence interval) Pekka dataset from a big commercial bank in Finland and ISBSG dataset were used for The analysis.	Regression via Classification (RvC).	Performance evaluation metric was Mean Absolute Error (MAE). RvC provided better regression error than the standard regression methods.
Hongyu Zhang(2008) [242]	Study of the growth of eclipse defects	Growth of the number of defects modeled by polynomial functions.	For most of the components, the MRE values are below 25%, falling within the acceptable levels.

Bingbing, Qian, Shengyong, and Ping (2008)[236]	Software fault prediction on two datasets. (A medical imaging system and Celestial Spectrum Analysis System datasets were used for the analysis.)	Affinity Propagation Clustering algorithm compared the performance of it with the performance of K-means clustering Method.	Performance evaluation metrics were Type-I error, Type-II error, and entirely correct Classification rate (CCR). Affinity Propagation clustering algorithm was better than K-means clustering on two datasets according to the Type-II error
Marcus, Poshy-vanyk, and Ferenc (2008)[137]	Software fault prediction Modeling on WinMerge and Mozilla projects.	Proposed a new cohesion metric named Conceptual Cohesion of Classes (C3).	Performance evaluation metrics used: Precision, Correctness, and Completeness. Univariate logistic regression analysis showed that C3 ranks better than many of the cohesion metrics.
Shafi, Hassan, Arshaq, Khan, and Shamail (2008)[191]	Compared the performance of 30 techniques on two datasets for software quality Prediction. (JEdit and AR3 data from PROMISE repository.)	Classification via regression and LWL.	Performance evaluation parameters used: Precision, Recall, Specificity, and Accuracy. Classification via regression and LWL performed better than the other techniques.

Riquelme, Ruiz, Rodríguez, and Moreno (2008)[184]	Investigated Naive Bayes and C4.5, on five public datasets from PROMISE repository for Software Fault Prediction.	Two balancing techniques.	Performance evaluation metrics used: AUC and Percentage of correctly classified instances They reported that balancing techniques improve the AUC measure, but did not improve the percentage of correctly classified Instances.
Catal and Diri (2009a)[18]	Objective is to find high-performance fault predictors on Public NASA datasets.	Machine learning such as Random Forestsand algorithms based on a new computational intelligence approach called Artificial Immune Systems.	Random Forests provides the best prediction performance for large datasets, and Naive Bayes is the Best prediction algorithm for small datasets in terms of the Area under Receiver Operating Characteristics Curve (AUC) evaluation parameter.
Chang, Chu, and Yeh (2009) [42]	Fault prediction approach to discover fault patterns.	Association rule mining.	They reported that prediction results were excellent.
Mende and Koschke (2009) [140]	Fault Prediction on thirteen NASA datasets.	Evaluated lines of code metric based prediction.	AUC is used as Performance Evaluator. The model performed well in terms of Area under ROC curve (AUC) parameter, and they could not show statistically significant differences to some data mining Algorithms.

Tosun, Turhan, and Bener (2009) [209]	Conducted experiments on public datasets to validate Zimmermann and Nagappan's paper Published in ICSE'08. Three embedded software projects were used for the analysis.	Complexity and network metrics from five additional systems.	Performance Evaluation metrics were PD, PF, and precision. Reported that network measures are significant indicators of fault-prone modules for large systems.
Turhan, Kocak, and Bener (2009) [212]	Investigated 25 projects of a Telecommunication system and trained models on NASA MDP data.	They used static call graph-based ranking (CBGR) and nearest neighbor sampling to build defect predictors.	They reported that at least 70% of faults could be identified by inspecting only 6% of code with Naive Bayes model and 3% of code with CBGR model.
Bacchelli A, D'Ambros M, Lanza M (2010)[55]	Investigate whether the information contained in e-mail archives is correlated to the defects found in the system.	Introduce metrics that measure the "popularity" of source code artifacts.	They reported that developers discuss problematic entities more than unproblematic ones.

D'Ambros M, Lanza M, Robbes R (2010)[150]	Present a benchmark for defect prediction, in the form of a publicly available dataset consisting of several software systems.	WCHU (Weighted Churn of source code metrics) and LDHH (Linearly Decayed The entropy of source code metrics), two novel approaches that they proposed.	They gave consistently good results –often in the top 90% of the approaches– across all five systems.
Mende T, Koschke R (2010)[140]	Defect prediction model to determine Quality assurance activities.	Compare two different strategies to include treatment effort into the prediction process, and evaluate the predictive power of such models.	Both strategies improve the cost effectiveness of defect prediction models significantly, in the statistical and practical sense.
Menzies T, Milton Z, Turhan B, Cukic B, Bener YJA (2010)[142]	Defect prediction Static code features.	Binary classification scheme (Defective ∈true, false) and not, say, number of defects or severity of defects.	Learners must be chosen and customized to the goal at hand

Turhan B, Bener AB, Menzies T (2010) [213]	Fault Prediction three embedded controller software, two versions of an open-source anti-virus software (Clam AV) and a subset of bugs in two versions of GNU gcc compiler.	Proposed that using imported data from different sites can make it suitable for predicting defects at the local site.	Software construction is a surprisingly uniform endeavor with simple and repeated patterns that can be discovered in local or imported data using just a handful of examples.
Wolf T, Schröter A, Damian D, Nguyen THD (2009)[225]	Predicting build failures using social network analysis.	The combination of communica-tion.structure.measures into a predictive model.	Our predictive model yielded recall values between 55% and 75% and precision values between 50% to 76%.
Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009)[249]	Cross-project defect prediction models on a large-scale. For 12 real-world applications, 622 cross-project predictions.	Derived decision trees that can provide early estimates for precision, recall, and accuracy.	Simply using models from projects in the same domain or with the same process does not lead to accurate predictions. Identified factors that do influence the success of cross-project predictions.

Timea Illes-Seifert, Barbara Paech(2010) [81]	Exploring the relationship of a file's history and its fault-proneness.	Propose an empirical approach that uses statistical procedures and visual representations of the data in order to determine indicators for a file's defect count.	Results show that software's history is a good indicator of its quality.
Wenjin Wu; Wen Zhang; Ye Yang; Qing Wang (2010)[227]	Debian Bug Number Prediction.	ARIMA Model for Modelling Debian Bug Numbers.	Moreover, both ARIMA and X12 enhanced ARIMA outperform the baseline as polynomial regression.
Mahmoud O. Elish, Ali H. Al-Yafei, Muhammed Al-Mulhem (2011) [57]	Fault prediction in packages of Eclipse system.	Three suites of package-level metrics (Martin, MOOD, and CK) are evaluated and compared empirically in predicting the number of pre-release faults and the number of post-release faults in packages.	The results indicate that the prediction models that are based on Martin suite are more accurate than those that are based on MOOD and CK suites across releases of Eclipse.

Peng Zhang; Yu-tong Chang (2012)[244]	Software fault prediction	Applied Grey neural network based on grey theory (exponential growth) and artificial neural network.	The proposed model reduced the prediction relative error effectively.
Partha S. Bishnu; Vandana Bhattacher-jee(2012) [32]	Software Fault Prediction	Proposed a Quad Tree-based K-Means algorithm	Reduced Error than other techniques.
Yue Jiang; Jie Lin; Bojan Cukic; Shuye Lin; Zhijian Hu (2013)[86]	Early Fault Prediction Using Design Metrics: Condition Count, Multiple Condition Count, Decision Count, Branch Count, Node Count, Edge Count	11 code metrics replaced by 6 design metrics using Canonical Correlation Analysis (CCA), a multivariate statistical analysis Method.	This would make it Possible to identify faults earlier before code implementation in the software lifecycle.

Karel Dejaeger; Thomas Verbraken; Bart Baesens (2013)[51]	Software Fault Prediction on NASA Dataset of NASA Metrics Data Program (MDP) repository.	Proposed 15 different Bayesian Network (BN) classifiers and comparing them to other popular machine learning techniques.	Augmented Naive Bayes classifiers can yield similar or better Performance than the commonly used Naive Bayes classifier.
A. Shanthini; R M Chandrasekaran (2014)[192]	Software Fault Prediction on Eclipse Package level dataset and NASA KC1 dataset.	Ensemble approach of Support Vector Machine (SVM) for fault prediction.	Ensemble of Support Vector Machine is superior to individual approach for software fault prediction in terms of classification rate through Root Mean Square Error Rate (RMSE), AUC-ROC, ROC curves.
Jiaqiang Chen; Shulong Liu; Wangshu Liu; Xiang Chen; Qing Gu; Daoxu Chen(2014) [43]	Software Fault Prediction on Eclipse and NASA datasets.	Proposed a novel two-stage data are preprocessing approach, which incorporates both feature selection and instance reduction.	The two-stage data preprocessing approach can greatly reduce both the number of features and the number of instances of the original dataset.

A. Soleimani; F. Asdaghi (2014)[201]	Software Fault Prediction on NASA's public dataset KC1 available at promise software engineering repository.	Proposed Artificial Immune System (AIS) based feature selection method to make a better prediction.	Selected subset of features increases the accuracy of classifier from 82.44% to 83.72%
Santosh Singh Rathore; Sandeep Kumar (2015)[180]	Fault Prediction on fault datasets collected from the PROMISE data repository.	Neural network and Genetic programming.	ERROR: Neural Network outperformed genetic programming, Recall and Completeness: Genetic programming produced the result better than neural network.
Wangshu Liu; Shulong Liu; Qing Gu; Xiang Chen; Daoxu Chen (2015)[127]	Software Fault Prediction with Noises.	Proposed a novel method FECS (feature Clustering with Selection strategies)	FECS a robust feature selection method with a certain noise tolerate ability for software fault Prediction.

2.3 Literature Review (Clone Detection and Clone Evolution Prediction)

It has been observed that the developers have a tendency to copy the modules completely or partially and modify them. This practice gives rise to identical or very similar code fragments called software clones. Detecting the cloned fragments is an important but challenging task. There has been a large number of studies on software clone detection[187, 28, 181]. There exist many clone detection approaches based on types of cloning. They are Line based technique [54, 87, 136, 121], Metric-based technique[139, 161, 36, 53], Token based techniques[89, 128, 214], Tree-based techniques[237, 218, 85, 110], PDG (Programme dependency graph) based techniques[117, 125] and also Abstract Syntax Tree (AST) based technique[187, 27]. The AST completely captures the whole system information and is a most efficient clone detection approach[27]. A large number of papers identify clones based on the same version of software systems. There is a paper by Antoniol, G et al.[22] in which the author Applied ARIMA for predicting the evolution of cloned component across different version of mSQL.

A systematic literature review of different approaches applied in the area of Software Clone Detection and Clone Evolution Prediction is presented in Table 2.2

TABLE 2.2: Literature Review (Clone Evolution Prediction)

Author (Year)	Objectives of Study	Methodology /Approaches/Tool-s/Techniques	Remarks
S. Ducasse, M. Rieger, and S. Demeyer (1999)[54]	A Language Independent Approach for Detecting Duplicated Code.	The approach is based on 1) Simple line-based string matching. (2) The visual presentation of the duplicated code. (3) Detailed textual reports from which overview data can be synthesized.	Easily identify (1) the duplicated code between several files, (2) within the same file, (3) cloned files and (4) evolution files
J. H. John-son(1994) [87]	Visualizing textual redundancy in the legacy source.	A strategy based on fingerprinting is used to obtain raw matches indicating where repetitions occur. (Line Based Technique)	The approach appears to be a powerful method of providing much information with comparatively little noise.
U. Man-ber(1994) [136]	Finding similar files in a large file system.	Present a tool, called SIF, for finding all similar files in a large File system. (Line Based Technique)	Application of SIF can be found in file management, information collecting (to remove duplicates), program reuse, file synchronization, data compression, and also Plagiarism detection.

S. Lee and I. Jeong (2005)[121]	Code Clone Detection System for Large Scale Source Code.	SDD (Similar Data Detection) algorithm (Line Based Technique).	Detected duplicated parts of source code in huge software with high-performance.
J. Mayrand, C. Leblanc, and E. M. Merlo (1996)[139]	Automatically identify duplicate and near duplicate functions in two telecommunication monitoring systems totaling one million lines of source code.	Technique is based on metrics extracted from the source code using the tool Datrix. (Metrics Based Technique).	The information provided by this study is useful in monitoring the maintainability of large software systems.
J.-F. Patenaude, E. Merlo, M. Dagenais (1999)[161]	Identifying parts of the system which have unusual characteristics.	Extensions to Bell Canada source code quality assessment suite (DATRIX tm) for handling Java language.	Through clone detection, it was found that about 6% of the 512 000 lines of code are clones.

F. Calefato, F. Lanubile, and T. Mallardo(2004) [36]	Identify cloned functions within scripting code of web applications.	The approach is based on the automatic selection of potential function clones and the visual inspection of selected script functions. (Metrics Based Technique)	Semi-automated approach is both effective and efficient at identifying function clones in web applications.
G. A. Di Lucca, M. Di Penta, and A. R. Fasolino (2002)[53]	Identify duplicated web pages.	In this paper, they propose an approach. Based on similarity metrics, to detect duplicated pages in web sites and applications, implemented with HTML language and ASP technology. (Metrics Based Technique)	The methods produced results that are comparable, but with different computational costs. Successfully applied to identify a case of plagiarism too.

T. Kamiya, S. Kusumoto, and K. Inoue (2002)[89]	Code Clone Detection System for large-scale source code.	Developed a tool, named CCFinder (Code Clone Finder), which extracts code clones in C, C++, Java, COBOL and other source files. In addition, metrics for the code clones have been developed. (Token-Based Technique)	CCFinder has effectively found clones, and the metrics have been able to effectively identify the characteristics of the systems.
S. Livieri, Y. Higo, M. Matushita, and K. Inoue (2007)[128]	Code Clone Analysis and Visualization of Open Source Programs.	D-CCFinder has been implemented. 400 million lines in total have been analyzed. (Token-Based Technique)	D-CCFinder illustrates a fairly cheap and practical Method for large-scale code clone analysis.
Y. Ueda, T. Kamiya, S.Kusumoto, and K. Inoue (2002)[214]	Detection of gapped code clones.	Proposed a method to find gapped clones using the gap location information. (Token-Based Technique)	Successfully found the gapped clones which are composed of several short clones.

W. Yang (1991)[237]	To detect syntactic differences between two programs.	Parse Tree-based Technique. Tree-matching algorithm and the synchronous pretty-printing technique are used. (TREE Based Technique)	Two programs are pretty-printed 'synchronously' with the differences highlighted so that the differences are easily identified.
V. Wahler, D. Seipel, J. W. von Gudenberg, and G. Fischer (2004)[218]	Clone detection in the source code.	Frequent item set techniques. (TREE Based Technique).	Approach is very flexible; it can be configured easily to work with multiple programming languages.
L. Jiang, G. Misherghi, Z. Su and S. Glondu (2007)[85]	Software Clone Detection on large code bases written in C and Java including the Linux kernel and JDK.	Implemented our tree similarity algorithm as a clone detection tool called DECKARD	DECKARD is both scalable, accurate and also language independent.

R. Komondoor and S. Horwitz (2001)[110]	Identify Duplication in the source code.	Program Dependence graphs (PDGs) and program slicing to find isomorphic PDG Subgraphs that represent clones. (Tree-Based Technique)	Can find non-contiguous clones (Clones whose components do not occur as contiguous text in the program).
J. Krinke (2001)[117]	To identify similar code in programs.	Used of program dependence graphs.	Approach is feasible and gives very good results despite the non-polynomial complexity of the problem.
Antoniol, G.; Casazza, G.; Di Penta, M.; Merlo (2001)[22]	Cone Evolution Prediction on 27 subsequent Versions of mSQL.	ARIMA	Preliminary results are encouraging.
C. Liu, C. Chen, J. Han, and P. S. Yu (2006)[125]	Detection of software plagiarism	Proposed a new plagiarism detection tool, called GPLAG, which detects plagiarism by mining program dependence graphs (PDGs).	GPLAG is both effective and efficient: It detects plagiarism that easily slips over existing tools, also it takes a few seconds to find plagiarism in Large Codes.
R. Koschke, R. Falke, and P. Frenzel (2006)[116]	Clone Detection in Source Code.	Suffix trees to find clones in abstract syntax trees. (AST Based Technique).	Better precision for type-2 clones than Token-Based Technique.

I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier (1998)[27]	Detecting exact and near miss clones over arbitrary program fragments in program source Code.	Abstract Syntax Tree-Based Approach.	Proposed a Technique to remove detected clones.
C. K. Roy, J. R. Cordy, and R. Koschke (2009)[187]	Comparison and Evaluation of software clone detection tools and techniques.	1: Proposed a scheme for classifying clone detection techniques and tools and 2: Proposed a taxonomy of editing scenarios that produce different clone types and also evaluation of current clone detectors based on this taxonomy.	One might use the results of this study to choose the most appropriate clone detection tool or technique in the context of a particular set of goals and constraints.
S. Bellon, R.Koschke, G.Antoniol, J. Krinke, and E. Merlo (2007)[28]	Compared and evaluated clone detection tools.	Presents an experiment that evaluates six clone detectors based on eight large C and Java programs (altogether almost 850 KLOC).	The techniques work on the text, lexical and AST, software metrics, and program dependency graphs.

S. K. Abd-El-Hafiz (2012)[16]	Detection of function clones in software systems.	Proposed an efficient metrics-based data mining clone detection approach.	The approach is very space efficient and linear in the size of the dataset.
D. Rattan, R. Bhatia, and M. Singh (2013)[181]	Extensive systematic literature on software clone detection.	Thirteen intermediate representations and 24 match detection techniques are reported.	Empirical evaluation of clone detection tools/techniques is presented.
K. Kaur and R. Maini(2015)[94]	Comparative analysis of various code clone detection techniques.	Classify Clone Detection Techniques on the basis of Clone Types.	The AST completely Captures the whole system information and is a most efficient clone detection approach.

2.4 Literature Review (Software Reliability Prediction)

Reliability is an important factor of software quality. The accurate prediction of software reliability is a challenging task. There exist many reliability models to predict the reliability based on software testing activities. There are many software reliability growth models (SRGMs) developed to predict the reliability but they have many unrealistic assumptions, and they are also environment dependent. The accuracy of the models is also questionable.

There have been a large number of studies in the prediction of software reliability. A number of software reliability growth models (SRGMs) have been proposed[67, 68] which describe the software behavior with respect to failures that occur in software applications for estimating and predicting software reliability. These models have the drawback that they use unrealistic assumptions, independence of time between failures and fault correction without the introduction of new faults[179].

There are also some models proposed based on nonparametric statistics[61] and Bayesian networks [185] to predict the software reliability without any specific assumptions. Although they solve the problem of unrealistic assumptions as considered by SRGMs but they suffer a lot from the issue of application and accurate prediction [67]. Some of the authors also use neural networks and machine learning approach[93] to predict software reliability. The main drawback is that these methods require a large number of data for learning and it is a time-consuming process. Some authors have also applied time series ARIMA model[230] as an alternative way to predict software reliability. The limitations of these papers are that they have not checked the underlying assumptions of ARIMA for a correct prediction. In another paper[20] the authors have considered all assumptions and specification for predicting software reliability. They have also compared the performance of their model with the existing models. The limitation is that they have taken a lot of statistical analysis for choosing the best fitting model which is a computationally expensive and time-consuming process.

A systematic literature review of different approaches applied in the area of Software Reliability Prediction is presented in Table 2.3.

TABLE 2.3: Literature Review(Software Reliability Prediction)

Author (Year)	Objectives of Study	Methodology /Approaches/Tool-s/Techniques	Remarks
Jelinski, Z., Moranda, P. (1972)[146]	Software reliability prediction using the software failure-occurrence time data.	NHPP (nonhomogeneous Poisson process).	Performance Measures: U-plot, Y-plot, and AIC. Logarithmic Poisson execution time model fits the data set best.
Goel, A. (1985)[67]	Software reliability prediction based on failure data from a medium-sized real-time command and control software system.	Four Types of Model Analyzed: 1: Times Between Failures Models, 2: Failure Count Models, 3: Fault Seeding Models, 4: Input Domain Based Models.	Models require Underlying assumptions to be applied.
Lyu, M., Nikora, A. (1992)[133]	Software Reliability Prediction Based on Failure Patterns.	Applied Combination of Existing Models.	Combining the results of individual Models have a substantial improvement than using single component models.

Lyu, M (1996)[132]	Data, analysis and case studies on Software Reliability Prediction.	Emerging research methods including software metrics, testing schemes, fault-tolerant software, fault-tree analysis, process simulation, and neural networks.	Presents a statistical study of how well software systems satisfy user requirements on user premises, and for how long.
Zeitler, D. (1991)[240]	Prediction Of Software Reliability Growth.	Auto-regressive integrated moving average (ARIMA) Models.	Model used Realistic assumptions for software reliability models.
Xie, M., Ho, S (1995)[231]	Software Reliability Prediction.	ARIMA	Time series models have outperformed the traditional Duane model in terms of predictive performance.
Wood, A (1997)[226]	Discussed on Reliability Model Assumptions against Reality on Tandem's software development and test environment.	Proposed Technique for Compensating Loss of Accuracy due to Violation of Assumptions.	Suggestions: - Simple models are good. Realistic assumptions are good.
Xie, M., Hong, G., Wohlin, C (1997)[230]	Predict Software Reliability-based on Software Failures.	Double exponential smoothing techniques.	The method is very easy to use and requires a very limited amount of data storage and computational effort.

Xie, M., Ho, S.(1999) [231]	Reliability Analysis on Repairable Systems.	Time series models for analyzing failure data.	The time series method gives satisfactory results in terms of its predictive performance.
Robinson, D., Dietrich, D. (1987)[185]	Reliability Analysis of Failure Rate of a System.	Non-Parametric Reliability Growth Model.	Model has improved performance in terms of relative error and mean square error.
Barghout, M., Little-wood, B., Abdel-Ghaly, A.(1998) [26]	Prediction of Software Reliability.	Non-Parametric Reliability Growth Model (allows the data to speak for themselves)	Better predictions than parametric reliability growth models.
Bai, C., Hu, Q., Xie, M., Ng, S. (2005)[25]	Predicting Reliability From Software Failures.	Markov Bayesian networks.	Improved predictive performance via Bayesian network but increased complexity and Computational effort.

Junhong, G., Hongwei, L., Xiaozong, Y(2005) [88]	Software Reliability Prediction	Proposed a Transformation of Goel-Okumoto model into one-order autoregressive stochastic time series model with independent increment.	Proposed model is superior to that of Goel-Okumoto model in terms of estimation and prediction ability.
Pai, P., Hong, W. (2006)[159]	Software reliability forecasting	Support vector machines (SVMs) + Simulated annealing algorithms (SA) [for selection of parameters of an SVM model]	SVM model with simulated annealing algorithms (SVMSA) results in better predictions than the other methods.
Kiran, N., Ravi, V.(2007) [108]	Software Reliability Prediction.	Wavelet neural networks (WNN)	WNN outperformed BPNN, GRNN, and MLR and other techniques.

Lyu, M.R., (2007) [131]	Software Reliability Engineering.	Proposed a new software reliability engineering paradigms that take software architectures, testing techniques, and software failure manifestation mechanisms into consideration.	Present a review of the history of software reliability engineering, the current trends and existing problems, and specific difficulties.
Fenton, N., Neil, M., Marquez, D.(2008) [61]	Predicted software defects and reliability on organizations such as Motorola, Siemens, and Philips.	Bayesian networks (BNs)	Significantly improved accuracy for defects and reliability prediction type models.

Raj Kiran, N., Ravi, V.(2008) [182]	Software Reliability Prediction	Various statistical (multiple linear regression and multivariate adaptive regression splines) and intelligent techniques (Back Propagation trained a neural network, dynamic evolving Neuro-fuzzy inference system, and TreeNet).	Nonlinear ensemble outperformed all the other ensembles and also the constituent statistical and intelligent techniques.
Zaidi, S., Danial, S., Usmani (2008)[239]	Software inter-failure time series analysis	ANN Modelling.	The calculated RMSE of the ANN model is much lesser than the other modelling Techniques.
Lo, J.(2009) [129]	Proposed a General framework of the modeling of the failure detection and fault correction processes.	Applied and Constructed ANN for modelling software failure data.	Eliminated some unrealistic assumptions by SRGMs.

Sharma, K., Garg, R., Nagpal, C., Garg, R(2010) [193]	Selection of An optimal SRGM.	Proposed a deterministic quantitative model based on a distance based approach (DBA) for ranking SRGMs.	This paper addresses the issue of optimal selection of software Reliability growth models.
Yang, B., Li, X., Xie, M., Tan, F.(2010) [234]	Software Reliability Modelling	Proposed a generic data-driven software reliability models (DDSRMs) with multiple-delayed-input single-output (MDISO). A hybrid genetic algorithm (GA)-based algorithm is developed which adopts the model mining technique to discover the correlation of failures and to obtain optimal model parameters.	Proposed model outperforms existing DDSRMs.

Huang, C., Lyu, M.(2011) [78]	Software Reliability Prediction from software failure data.	Incorporate the concept of multiple change-points, i.e., points in time when the software environment Changes into software reliability modeling.	Proposed models can provide good software reliability prediction in the various stages of software development and operation.
Kapur, P., Pham, H., Anand, S., Yadav, K.(2011) [92]	Developing Advanced Software Reliability Growth Models	Proposed two general frameworks for [(GINHPP-1) and (GINHPP-2)] deriving several software reliability growth models based on a Nonhomogeneous Poisson Process (NHPP) in the presence of imperfect Debugging and error generation.	Models discussed in this paper have quite encouraging performance on real dataset.
Moura, M., Zio, E., Didier Lins, I., Droguett, E. (2011)[49]	Failure and Reliability Prediction.	Support Vector Machines Regression.	SVM outperforms other techniques like ARIMA, MLPNN, RNN, etc.

Palviainen, M., Evesti, A., Ovaska, E. (2011) [160]	Software reliability evaluation during the design and implementation phases.	A coherent approach by combining both predicted and measured reliability values with heuristic estimates in order to facilitate a smooth reliability evaluation process. (Component Level Reliability +System Level Reliability).	Helps Software Developers in Early Reliability Prediction (at Design Time).
Wiper, M., Palacios, A., Marí n, J.(2012) [224]	Software Reliability modelling using Software Metrics Information.	Neural network regression to estimate failure rates in models based on inter failure times or numbers of failures. The inference is carried out using a Bayesian approach.	An Efficient Approach for Software Reliability Prediction.
R. Mo-hanthy, et al. (2014)[145]	Predict software reliability based on data collected from the literature.	Ant Colony Optimization Technique (ACOT)	Proposed Method Outperforms BPNN, GRNN, DENFIS, and other techniques in terms of NRMSE.

Kewen Li; Kang Zhao; Wenying Liu (2013) [122]	Software Reliability Prediction.	ANN +K Means Clustering Method.	Improved Accuracy than Back Propagation Algorithm
Amin, A., Grunske, L., Colman (2013)[20]	Software Reliability Prediction	ARIMA	Results Better Than Traditional SRGMs.
Shirin Noekhah; Ali Akbar Hozhabri; Hamideh Salimian Rizi (2013)[153]	Software reliability prediction on the basis of number of faults.	Multi-Layer Perceptron (MLP) neural network + Imperialist Competitive Algorithm (ICA) [as training algorithm].	Proposed predicting model is more efficient than the existing techniques in prediction performance.
Najeeb Ullah; Maurizio Morisio; Antonio Vetrò (2015)[215]	Model to Predict Residual Defects in Open Source Software.	Evaluates eight popular software reliability growth models and selects the one that can best predict the Software's remaining faults.	Proposed Model will Provide Practical Support to Project Managers.

2.5 Conclusion

This chapter presents a systematic literature survey on different models on software defect prediction, clone detection, clone evolution prediction and also software reliability prediction. From the survey, it is observed that though there exist a large number of papers on predicting these software characteristics only a few research papers aim at modelling the temporal patterns of different evolving software characteristics across different versions of the software application. It is also observed that the time series model applied by the research papers are based on simple Polynomial regression and ARIMA. The survey shows there is a need for modelling the evolving software characteristics using advanced time series approach based on both statistical and machine-learning techniques.