DOI: 10.21917/ijsc.2019.0283

PERSPECTIVE VIEW ON SOFTWARE QUALITY PREDICTION SYSTEMS

R. Janarthanan and A. Hema
Department of Computer Applications, Kongunadu Arts and Science College, India

Abstract

This study surveys the effectiveness of Prediction Systems that predominantly assess the quality of software. The quality of the software systems depends on functional and non-functional attributes. In this technological era, software quality prediction is one of the challenging tasks. It became an essential ingredient in many systems that produce reliable, cost effective and less complex software. Hence various researchers are deploying many faults-prone model systems with various testing traits as parameters to develop quality software. This paper surveys the pros and cons of software quality prediction systems that are based on fault- prone models with Software metrics of the quality systems and describes the different kind of measures in the field of software engineering.

Keywords:

Software Quality, Software Metrics, Models, Prediction Features, Fault-Prone Systems

1. INTRODUCTION

In this technological era, the software systems that we use are expected to be infallible. Software intensive systems play a vital role in the life of the people involved in social life, business promotions, marketing etc. Therefore, it is imperative to boost the effectiveness and the accuracy quotient of the software [1]-[3] [6] [9] [18]-[20].

A software product with numerous defects definitely lacks quality, so it is a vital thing to include methodologies and strategies for anticipating effort of testing, beholding the cost and results that can help in increasing the productivity and efficiency of the software [5]. The persuasiveness of the whole process is just sketched out by predicting the fault of the systems. The software community is now enthralling in program testing domain since there is a high demand for complex-free and reliable software [8] [24]-[34].

The quality assurance in the software is the preliminary factor that is to be inculcated by each and every tester and developer before the product release. In the field of computer program, a contemporary research tells that cost becomes high when the bugs are not detected earlier [12] [38] [40].

Considering the life cycle of software testing process early detection of fault prone modules plays a great role in the successful implementation and the quality prediction turns the system highly productive. Software metrics is determined as a measure of software characteristics that are definite and discrete. It plays a critical role in emphasizing the prediction of quality software. There is an unmediated relationship between some changes attributed to faults and complexity metrics that could be sorted out later in verification and validation [15] [42]-[49].

Many researchers casted about the development of relationship between complexity metrics and faults. It is suggested that there is a vital necessity of multiple variable modules in order to add the program size, subsequently examining the relationship between the metrics, faults and programs captivates the interest of the researchers [4]. There are several techniques to enhance the development of predictive software techniques for the categorization of software program modules into fault-prone and non-fault prone categories has been proposed hence a metric based byway can be surveyed for the prediction of software quality by identifying the fault–prone modules [14].

The organization of the remaining sections is as follows. Section 2 illustrates the related works. Section 3 describes a framework of software quality prediction and their factors. Section 4 summaries the sample datasets of software quality prediction. Finally, section 5 concludes the paper.

2. RELATED WORKS

Artificial neural network, one of the recent trends that are encompassed to sixth generation computing, is applied to the model software reliability. In this case there are two types of input given for deriving valid output (i.e.) mistake reports are given to input of the software quality prediction model and software quality metrics is fabricated as input of the artificial neural networks with this effective application of giving software metrics as the input of neural networks, it is comparatively proven that it produces an accurate quality prediction [7].

Considering the severity of fault a software fault- prone prediction model can be fabricated by a support vector machine and the object oriented metrics. Conventionally the quality prediction is rated and qualified by various criteria like accuracy percentage, probability of detection and probability of false alarms [22].

Saida et al. [21] predict the faulty and non-faulty modules based on the mathematical model to analyzing factors are taken as an object oriented metrics [21].

Bellini et al. [23] find the estimation rate of the fault-proneness and size of the objects using estimation model based K-NN algorithm were implemented [23].

Salak Bouktif et al. [39] presented how the general problem of combining quality experts, modeled as Bayesian classifiers, can be tackled via a simulated annealing algorithm customization. The general approach was applied to build an expert predicting object-oriented software stability, a facet of software quality. The findings demonstrate that, on available data, composed expert predictive algorithms and outperform the best available expert and it compares favorably with the expert build via a customized genetic algorithm [39].

3. SOFTWARE QUALITY PREDICTION PROCESS AND ITS FACTORS

Defect prediction is an important fact for analyzing the quality of the software. The Fig.1 shows the process of the software quality prediction, including the following steps:

Step 1: Extract program modules/files/classes based required database.

Step 2: Extract the features that are related to software defects by analyzing software code or the development process. The features are:

- **a.** Halstead features
- **b.** McCabe features
- **c.** Size related metrics
- **d.** Quality metrics
- **e.** Object oriented metrics

Step 3: Construct defect prediction model by training the instances with the required features based on the below models:

- **a.** Mathematical models.
- **b.** Architecture based models
- **c.** Soft computing based models
- **d.** Machine learning based models
- **e.** Outperform design metrics models
- **f.** Object oriented models

Step 4: Predict the unlabeled program modules and classify them either defective or not.

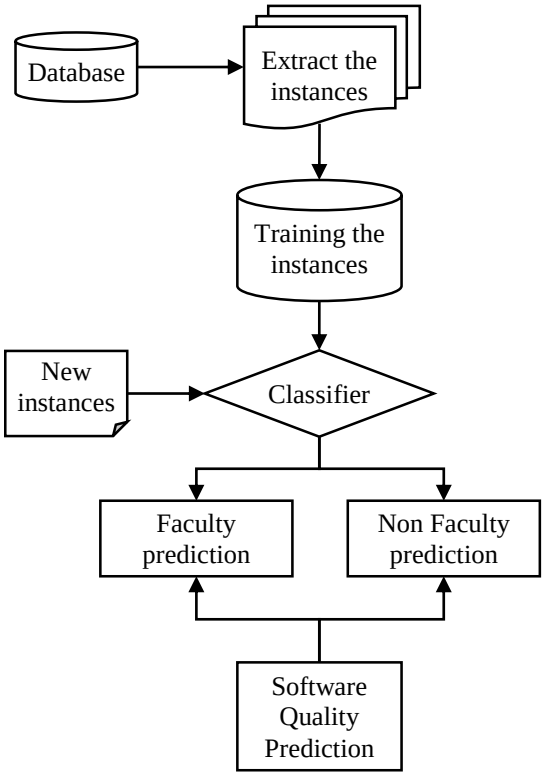

Fig.1. Framework of Software Quality prediction

3.1 HALSTEAD FEATURES

These features are used to predict the effectiveness of the software quality. The features contain the following properties such as program vocabulary, program length, calculated program length, volume, difficulty and efforts [10].

η_1 = the number of distinct operators

η_2 = the number of distinct operands

N_1 = the total number of operators

N_2 = the total number of operands

From these numbers, several measures can be calculated:

$$\text{Program vocabulary: } \eta = \eta_1 + \eta_2 \quad (1)$$

$$\text{Program length: } N = N_1 + N_2 \quad (2)$$

$$\text{Calculated program length: } N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \quad (3)$$

$$\text{Volume: } V = N^* \log_2 \eta \quad (4)$$

$$\text{Difficulty: } D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2} \quad (5)$$

$$\text{Effort: } E = D * V \quad (6)$$

3.2 MCCABE FEATURES

McCabe features are used to predict the quality prediction based on Cyclomatic Complexity metrics the general form

$$V(G) = E\text{-}N+2p \quad (7)$$

where N-Nodes, E-Edges, P-connected procedures

Extended Cyclomatic complexity (ECC): McCabe measures the program complexity based on conditional statement. Extended Cyclomatic complexity that may be defined as:

$$ECC = eV(G)=P_e+1 \quad (8)$$

where, P_e is the number of predicate nodes in flow graph G weighted by number of compound statements.

Information flow metrics may be finding by count the number of local information flows input (fan-in) and flows output (fan-out). The procedure may be defined as:

$$C = [\textit{procedure length}]^*[(\textit{fan-in})^*(\textit{fan-out})]^2 \quad (9)$$

3.3 SIZE RELATED METRICS

Size related metrics are the metrics which can help to enumerate the software size. There are three types of software metrics which are used to measure the software size and predict the software quality the measures are Line of code (LOC), Function point Metrics and bang [11].

3.4 QUALITY METRICS

Software Quality prediction based on the Quality measures carried out the defects, product quality and maintainability of the software package [16]. The quality metrics is classified into three cases such as Defect metrics, Reliability Metrics and Maintainability Index. The defect measures perform the following parameters:

- Counting the defects in the program
- Number of design change

Reliability Metrics is focus on internal product quality which is measured the number of bugs in the software. Maintainability

Index is defined as a number of functions that predict software maintainability [41]. The maintainability index may be measured as follows

$$MI = 171\text{-}5.2*\ln(aveV)\text{-}0.23*aveV(g)\text{-}16.2*\ln(aveLOC) \quad (10)$$

where,

AveV = average halstead volume per module

AveV(*g*) = average extended cyclomatic complexity per module.

aveLOC = average line of code per module.

3.5 OBJECT ORIENTED METRICS

Object-oriented metrics is used for predicting the software quality based on object-oriented software development. Some of the object oriented metrics are Chen Metrics, Morris's Metrics, Lorenz and Kidd Metrics, MOOSE Metrics, EMOOSE, MOOD Metrics, QMOOD Metrics, LI Metrics and SATC metrics [17].

Chen Metrics is deals with behavior of the object oriented design. The behaviors of the object oriented design are

- CCM (Class Coupling Metric)
- OXM (Operating Complexity Metric)
- OACM (Operating Argument Complexity Metric)
- ACM (Attribute Complexity Metric)
- OCM (Operating Coupling Metric)
- CM (Cohesion Metric)
- CHM (Class Hierarchy of Method)
- RM (Reuse Metric)

Morris metrics deals with the cohesion metrics and complexity of the program which based on the depth of the tree [5].

Lorenz and Kidd Metrics is a set of metrics that can be grouped in four categories are size, inheritance, internal and external these metrics are evaluate to the predict of software Quality the metrics are

- Class Size (CS)
- Number of Operations overridden by a Subclass (NOO)
- Number of Operations added by a Subclass (NOA) Specialization Index (SI)
- Average Operation Size (OS)
- Operation Complexity (OC)
- Average number of Parameters per Operation (NP).

Metrics for Object-Oriented Software Engineering (MOOSE) is a set of metrics which is based on cohesion and coupling [35]. The following parameters are evaluating the quality factors

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of children (NOC)
- Coupling between Objects (CBO)
- Response for class (RFC)
- Lack of Cohesion in Methods (LCOM)

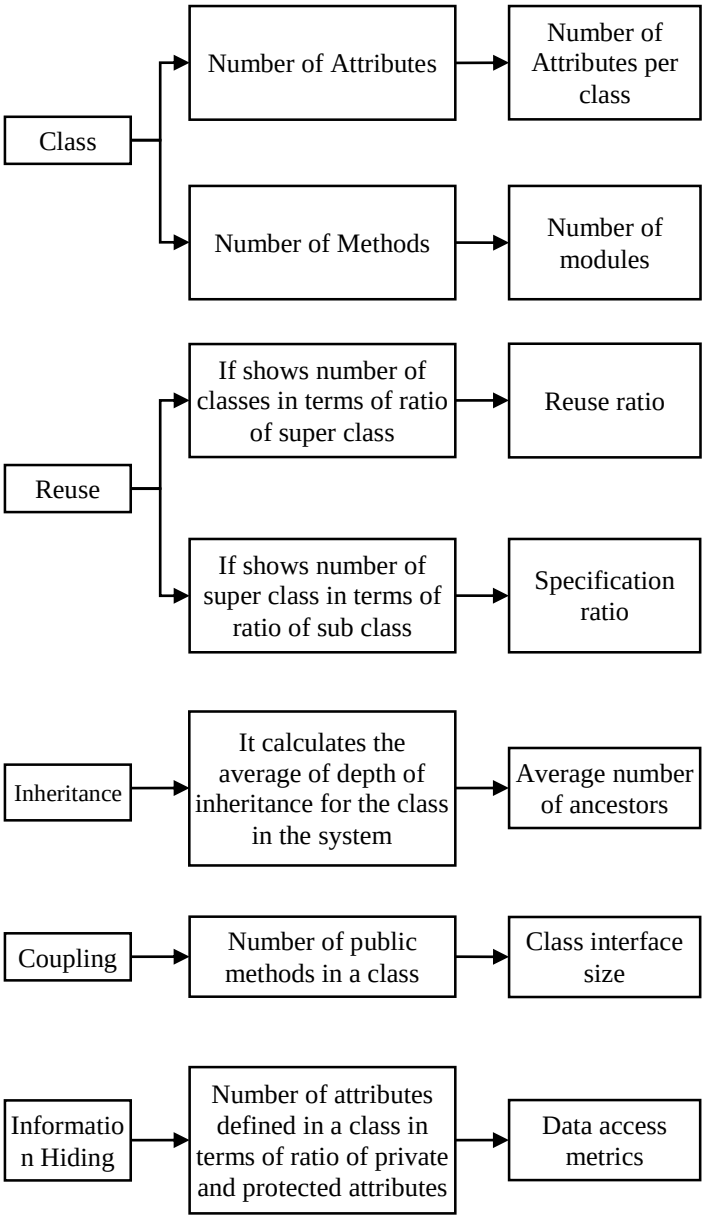

Fig.2. QMOOD Metrics

Extended Metrics for Object-Oriented Software Engineering EMOOSE is an object oriented metrics which is used to estimate the quality of the software. They may be described as

- Message Pass Coupling (MPC)
- Data Abstraction Coupling (DAC)
- Number of Methods (NOM)

The number of methods depend on the size of the program. NOM consist of two sizes such as size1 and size2.

- Size1 describes the number of lines of code.
- Size2 is used to count the number of local attributes and the number of operation defined in the class [13].

Metrics for Object-Oriented Design (MOOD) is a quality metrics which is based on methods and attribute features [37]. MOOD is a structural model which contain the following parameters

- Encapsulation as (MHF, AHF)

- Inheritance (MIF, AIF)
- Polymorphism (POF)
- Message passing (COF)

Quality Model for Object-Oriented Design (QMOOD) is an object oriented measure based on OOPS principles the Fig.2 describes the QMOOD Metrics.

LI metrics is used to analyze the software quality in terms of tree complexity of the coupling. The metrics are

- Number of Ancestor Classes (NecauseAC)
- Number of Local Methods (NLM)
- Class Method Complexity (CMC)
- Number of Descendent Classes (NDC)
- Coupling Through Abstract data type (CTA)
- Coupling through Message Passing (CTM)

SATC's Metrics is a based on methods, coupling, inheritance and internal and external psychological complexity factors. The metrics are

- Cyclomatic Complexity (CC)
- Line of Code
- Comment percentage

The new object oriented measures are shown in Table.1.

Table.1. New Object Oriented Metrics

Source Construct	Metrics	Object Oriented Structure
Traditional Metrics	Cyclomatic Complexity (CC)	Methods
	Line of Codes	Methods
	Comment Percentage (COM)	Methods
New Object Oriented Structure	Weight Method per Class (WMC)	Methods/Class
	Response for a Class (RFC)	Class/Message
	Lack of Cohesion of Methods (LCOM)	Class/Cohesion
	Coupling between Object (CBO)	Coupling
	Depth of Inheritance Tree (DIT)	Inheritance
	Number of Children (NOC)	Inheritance

4. SOFTWARE QUALITY PREDICTION FACTORS

In software quality process the following factors are evaluated to predict the fault and quality of the required systems. The important software predictions factors are

- *Cohesion*: It is refers to the degree to which the elements inside a module belong together. In case of high cohesion which is an ordinal type of measurement. Modules with high cohesion is inferable, because high cohesion is associated with the reusability and reliability of the software with high cohesion it is easy to maintain, test, reuse and even understand which are the desirable traits of software.
- *Coupling*: Coupling is usually contrasted with cohesion. High cohesion infers desirable traits of software which low coupling correlates with it to give desirable software traits and determines the strength of the relationship between modules. It increases the understandability and maintainability of the software.
- *Complexity*: It is the term that will affect the internal interactions of the modules in the software it makes the product to be less effectuate in terms of time and cost. High level of complexity will reduce the desirable traits of the software [36].
- *Feasibility*: Feasibility study is done as part of systems developments life cycles that incurs and reveal the strength and weakness of technical, operational and economical aspects of the product.
- Customizability: The major perspective of customizability is user interface that will definitely improve the software quality in order to satisfy the requirement of the end user.

5. DATASETS DESCRIPTION

In order to evaluate the software quality we consider the following dataset for empirical study. In this paper we are discuss four open datasets in NASA, Promise, AEEEM and Relink which are used for defect prediction based quality assessment.

NASA Dataset- this dataset was collected by NASA metrics data program. This dataset contain 40 features may contain both Hallstead features and McCabe features. The features are given in Table.2.

Promise Dataset- it is open source java projects which contain different metrics such as lines of code, Response for class, Average method complexity, coupling between object classes etc. this metrics is used for evaluation of software quality effectiveness. The Fig.3 describes the metrics and attributes of the Promise dataset.

AEEEM Dataset- this dataset which comes from Eclipse and Apache. This type of dataset may contain classic metrics, object oriented metrics and source code entropy metrics which is used for predicting the quality of the software. The Table.3 describes the attributes and description about the AEEEM Dataset.

Relink Dataset- it is object oriented software metrics which was developed by Wu et al. it consists of 26 complexity metrics. The data set range will be 56 to 7782. The Table.4 describes the attributes of Relink dataset description.

Table.2. NASA Dataset metrics

Dataset	G		H		I		J		K		L		M	
	Identical cases		**Inconsistent cases**		**Cases with missing values**		**Cases with conflicting feature values**		**Cases with implausible values**		**Total problem cases, DS′**		**Total problem cases, DS″**	
	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom
CM1	26	94	0	2	161	0	2	3	0	1	161	3	178	61
JM1	2628	2628	889	889	0	5	1287	1294	0	1	1287	1294	3158	3165
KC1	1070	1070	253	253	0	0	12	14	0	1	12	14	945	947
KC2	n.a.	182	n.a.	118	n.a.	0	n.a.	38	n.a.	1	n.a.	38	n.a.	197
KC3	12	170	0	2	258	0	0	0	29	29	258	29	264	142
KC4	10	n.a.	9	n.a.	0	n.a.	125	n.a.	0	n.a.	125	n.a.	125	n.a.
MC1	7972	7972	106	106	0	0	189	189	4841	4841	4841	4841	7619	7619
MC2	4	6	0	2	34	0	0	0	0	0	34	0	36	5
MW1	15	36	5	7	139	0	0	0	0	0	139	0	152	27
PC1	85	240	13	13	348	0	3	26	48	49	355	74	411	196
PC2	984	4621	0	100	4004	0	129	129	1084	1084	4055	1163	4855	4297
PC3	79	189	6	9	438	0	2	2	52	52	444	54	490	138
PC4	166	166	3	3	0	0	60	60	111	111	112	112	182	182
PC5	15370	15370	1725	1725	0	0	185	185	1772	1772	1782	1782	15507	15507

```
% 1. Title/Topic: KC2/software defect prediction
% 2. Sources:
%    -- Creators:  NASA, then the NASA Metrics Data Program,
%    {....omitted...}
% 5. Number of instances: 522
% 6. Number of attributes: 22 (5 different lines of code measure,
%         3 McCabe metrics, 4 base Halstead measures, 8 derived
%          Halstead measures, a branch-count, and 1 goal field)
% 7. Attribute Information:
%      1. loc            : numeric % McCabe's line count of code
%      2. v(g)           : numeric % McCabe "cyclomatic complexity"
%      3. ev(g)          : numeric % McCabe "essential complexity"
%      4. iv(g)          : numeric % McCabe "design complexity"
%      5. n              : numeric % Halstead total operators + operands
%      6. v              : numeric % Halstead "volume"
%      7. l              : numeric % Halstead "program length"
%      8. d              : numeric % Halstead "difficulty"
%      9. i              : numeric % Halstead "intelligence"
%     10. e              : numeric % Halstead "effort"
%     11. b              : numeric % Halstead
%     12. t              : numeric % Halstead's time estimator
%     13. lOCode         : numeric % Halstead's line count
%     14. lOComment      : numeric % Halstead's count of lines of comments
%     15. lOBlank        : numeric % Halstead's count of blank lines
%     16. LOCodeAndComment: numeric
%    {....omitted....}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@relation KC2

@attribute loc                  numeric
@attribute v(g)                 numeric
@attribute ev(g)                numeric
@attribute iv(g)                numeric
@attribute n                    numeric
@attribute v                    numeric
@attribute l                    numeric
@attribute d                    numeric
@attribute i                    numeric
@attribute e                    numeric
@attribute b                    numeric
@attribute t                    numeric
@attribute lOCode               numeric
@attribute lOComment            numeric
@attribute lOBlank              numeric
@attribute lOCodeAndComment numeric
@attribute uniq_Op              numeric
@attribute uniq_Opnd            numeric
@attribute total_Op             numeric
@attribute total_Opnd           numeric
@attribute branchCount          numeric
@attribute problems             {no,yes}

@data

1.1,1.4,1.4,1.4,1.3,1.3,1.3,1.3,1.3,1.3,1.3,1.3,2,2,2,2,1.2,1.2,1.2,1.2,1.4,no
1,1.0,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,yes
{...............omitted...............}
5,1.0,1,1,7,19.65,0.4,2.5,7.86,49.13,0.01,2.73,2,1,0,0,5,2,5,2,1,no
9,2.0,1,1,21,87.57,0.25,4,21.89,350.27,0.03,19.46,8,0,0,0,8,10,11,10,3,no
9,2.0,1,1,21,87.57,0.25,4,21.89,350.27,0.03,19.46,8,0,0,0,8,10,11,10,3,no
6,1.0,1,1,16,57.36,0.35,2.86,20.08,163.88,0.02,9.1,5,0,0,0,5,7,8,8,1,no
6,1.0,1,1,16,57.36,0.35,2.86,20.08,163.88,0.02,9.1,5,0,0,0,5,7,8,8,1,no
```

Fig.3. Promise dataset description

Table.3. AEEEM Dataset description

Project	**Total number of subclasses (H_{opt})**	**Original metric number of instances (p)**	**Learned feature dimensionality ($\leq R$)**
EQ	10	61	6
JDT	20	61	13
LC	26	61	15
ML	116	61	38
PDE	93	61	23

Table.4. Relink Dataset Description

Project	**Number of defective instances**	**Number of total instances**	**Number of metrics**	**Percentage of defective instances**
Apache	98	194	26	50.52%
Safe	22	56	26	39.29%
Zxing	118	399	26	29.57%

6. CONCLUSIONS

Based on the survey, an assortment of software fault predictions techniques has been reviewed, but none has proven to be consistently accurate. So hybrid algorithms were implemented to obtain the better results when compared to the traditional techniques like statistical methods, machine learning methods, parametric and non-parametric models. In this paper we summarize different kinds of Metrics, factors and open source dataset that are known to deliver the desired results in the

prediction of Software Quality. In future mathematical features are to be implemented to accurately detect software faults.

REFERENCES

[1] M.R. Lyu, "*Handbook of software Reliability Engineering*", IEEE Computer Society Press, 1996.

[2] B.W. Boehm and P.N. Papaccio, "Understanding and Controlling Software Costs", *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp. 1462-1477, 1988.

[3] F.G. Sayward, A.J. Perlis and M. Shaw, "*Software Metrics: An Analysis and Evaluation*", MIT Press, 1981.

[4] V.Y. Shen, T. Yu, S.M. Thebaut and L.R. Paulsen, "Identifying Errorprone Software-An Empirical Study", *IEEE Transactions on Software Engineering*, Vol. 11, No. 7, pp. 317-323, 1985.

[5] S.G. Crawford, A.A. McIntosh and D. Pregibon, "An Analysis of Static Metrics and Faults in C Software", *Journal of Systems and Software*, Vol. 5, No. 1, pp. 27-48, 1985.

[6] Liang Tian and Afzel Noore, "On-Line Prediction of Software Reliability using An Evolutionary Connectionist Model", *Journal of System and Software*, Vol. 77, No. 2, pp. 173-180, 2005.

[7] Liang Tian and Afzel Noore, "Evolutionary Neural Network Modeling for Software Cumulative Failure Time Prediction", *Reliability Engineering and System Safety*, Vol. 87, No. 1, pp. 45-51, 2005.

[8] Q.P. Hu, M. Xie, S.H. Ng and G. Levitin, "Robust Recurrent Neural Network Modeling for Software Fault Detection and Correction Prediction", *Reliability Engineering and System Safety*, Vol. 92, No. 3, pp. 332-340, 2007.

[9] T.M. Khoshgoftaar, E.B. Allen and Zhiwei Xu, "Predicting Testability of Program Modules using a Neural Network", *Proceedings of 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pp. 57-62, 2000.

[10] Zhiwei Xu and T.M. Khoshgoftaar, "Software Quality Prediction for High Assurance Network Telecommunications Systems", *Computer Journal*, Vol. 44, No. 6, pp. 557-568, 2001.

[11] Donald E. Neumann, "An Enhanced Neural Network Technique for Software Risk Analysis", *IEEE Transactions on Software Engineering*, Vol. 28, No. 9, pp. 904-912, 2002.

[12] S. Kanmani, V. Rhymend Uthariaraj, V. Sankaranarayanan and P. Thambidurai, "Object Oriented Software Fault Prediction using Neural Networks", *Information and Software Technology*, Vol. 49, No. 5, pp. 483-492, 2007.

[13] Jon T.S. Quah and Mie Mie Thet Thwin, "Prediction of Software Readiness using Neural Network", *Proceedings of 1st International Conference on Information Technology and Applications*, pp. 2312-2316, 2002.

[14] Mie Thet Thwin and Tong Seng Quah, "Application of Neural Networks for Software Quality Prediction using Object Oriented Metrics", *Journal of Systems and Software*, Vol. 76, No. 2, pp. 147-156, 2005.

[15] S. Kanmani, V. Rhymend Uthariaraj, V. Sankaranarayanan, P. Thambidurai, "Object Oriented Software Quality Prediction using General Regression Neural Networks", *ACM SIGSOFT Software Engineering Notes*, Vol. 29, No. 5, pp. 1-6, 2004.

[16] Atchara Mahaweerawat, Peraphon Sophatsathit, Chidchanok Lursinsap and Petr Musilek, "Fault Prediction in Object-Oriented Software using Neural Network Techniques", *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 3, pp. 312-322, 2006.

[17] T.M. Khoshgoftaar and R.M. Szabo, "Using Neural Network to Predict Software Faults during Testing", *IEEE Transactions on Reliability*, Vol. 45, No. 3, pp. 456-462, 1996.

[18] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepoh and S.J. Aud, "Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System", *IEEE Transactions on Neural Networks*, Vol. 8, No. 4, pp. 902-909, 1997.

[19] Qiu-suo Hu and Cheng Zhong, "Model of Predicting Software Module Risk based on Neural Network", *Computer Engineering and Applications*, Vol. 43, No. 18, pp. 106-110, 2007.

[20] Yan Zhao, Cheng Zhong, Zhi Li and Tie Yan, "Object Oriented Software Fault Proneness Prediction using Support Vector Machine", *Computer Engineering and Science*, Vol. 30, No. 11, pp. 115-117, 2008.

[21] Saida Benlarbi, Khaled El Emam and Nishith Geol, "Issues in Validating Object-Oriented Metrics for Early Risk Prediction", *Proceedings of 10th International Symposium on Software Reliability Engineering*, pp. 1-7, 1999.

[22] N.E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 675-689, 1999.

[23] P. Bellini, "Comparing Fault-Proneness Estimation Models", *Proceedings of 10th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 205-214, 2005.

[24] Giovanni Denaro, "Estimating Software Fault-Proneness for Tuning Testing Activities", *Proceedings of 22nd International Conference on Software Engineering*, pp. 1-7, 2000.

[25] A. Mahaweerawat, "Fault-Prediction in Object Oriented Software's using Neural Network Techniques", *IEEE Transactions on Software Engineering*, Vol. 30, No. 3, pp. 711-718, 2004.

[26] Y. Ma and L. Guo, "A Statistical Framework for the Prediction of Fault-Proneness", *IEEE Transactions on Software Engineering*, Vol. 32, No. 5, pp. 651-662, 2006.

[27] Thomas Zimmermann and Nachiappan Nagappan, "Predicting Defects using Social Network Analysis on Dependency Graphs", *Proceedings of International Conference on Software Engineering*, pp. 1-7, 2008.

[28] Audris Mockus, Nachiappan Nagappan and Trung T. Dinh Trong, "Test Coverage and Post-Verification Defects: A Multiple Case Study", *Proceedings of ACM-IEEE International Conference on Empirical Software Engineering and Measurement*, pp. 23-29, 2009.

[29] Cagatay Catal and Banu Diri, "A Systematic Review of Software Fault Prediction Studies", *Journal of Expert Systems with Applications*, Vol. 36, No. 4, pp. 7346-7354, 2009.

[30] Jonas Boberg, "Early Fault Detection with the Model-based Testing", *Proceedings of 7th ACM SIGNPLAN workshop on ERLANG*, pp. 9-20, 2008.

[31] Bindu Goel and Yogesh Singh, "Emperical Investigation of Metrics for Fault Prediction on Object Oriented Software", *Computer and Information Science*, Vol. 131, pp. 255-265, 2008.

[32] T.M. Khoshgoftaar, E.B. Allen, F.D. Ross, R. Munikoti, N. Goel, and A. Nandi, "Predicting Fault-Prone Modules with Case-Based Reasoning", *Proceedings of 8th International Symposium on Software Engineering*, pp. 27-35, 1997.

[33] Min Gu Lee and Theresa L. Jefferson, "An Empirical Study of Software Maintenance of a Web-based Java Application", *Proceedings 21st IEEE International Conference on Software Maintenance*, pp. 455-463, 2005.

[34] Marco D. Ambros and Michle Lanza, "Software Bugs and Evolution: A Visual Approach to Uncover Their Relationship", *Proceedings of IEEE Conference on Software Maintenance and Reengineering*, pp. 229-238, 2006.

[35] George E. Stark, "Measurements for Managing Software Maintenance", I *Proceedings of IEEE Conference on Software Maintenance*, pp. 4-8, 1996.

[36] T.M. Khoshgoftaar and J.C. Munson, "Predicting Software Development Errors using Complexity Metrics", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, pp. 253-261, 1990.

[37] T. Menzies, K. Ammar, A. Nikora and S. Stefano, "*How Simple is Software Defect Prediction?*", Kluwer Academic Publishers, 2003.

[38] K. Eman, S. Benlarbi, N. Goel and S. Rai, "Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components", *Journal of Systems Software*, Vol. 55, No. 3, pp. 301-310, 2001.

[39] Salah Bouktif, Houari Sahraoui and Giuliano Antoniol, "Simulated Annealing for Improving Software Quality Prediction", *Proceedings of 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 1893-1900, 2006.

[40] Ping Guo and Michael R. Lyu, "Software Quality Prediction using Mixture Models with EM Algorithm", *Proceedings of 1st Asia Pacific Conference on Quality Software*, pp. 69-78, 2000.

[41] Yue Jiang, Bojan Cukic, Tim Menzies and Nick Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction", *Proceedings of 4th International Workshop on Predictor Models in Software Engineering*, pp. 227-234, 2008.

[42] A. Kaur, A.S. Brar and P.S. Sandhu, "An Empirical Approach for Software Fault Prediction", *Proceedings of International Conference on Industrial and Information Systems*, pp. 261-265, 2010.

[43] NASA, "NASA Independent Verification and Validation Facility", Available at: https://www.nasa.gov/sites/default/files/166681main_NASA_Annual_Report_2005.pdf.

[44] M.A. Hall, "Correlation-based Feature Subset Selection for Machine Learning", PhD Dissertation, Department of Computer Science, The University of Waikato, 1998.

[45] M. Praneesh and K. Mahalakshmi, "Object Oriented Approach for Analysis of Software Fault Prediction using K-Jensen Shannon Entropy Model based Clustering Algorithm", *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 6, No. 1, pp. 21-29, 2017.

[46] M. Serdar Bicer and Banu Diri, "Defect Prediction for Cascading Style Sheets", *Applied Soft Computing*, Vol. 49, No. 2, pp. 1078-1084, 2016.

[47] Raed Shatnawi, "Deriving Metrics Thresholds using Log Transformation", *Journal of Software: Evolution and Process*, Vol. 27, pp. 95-113, 2015.

[48] Ruchika Malhotra, "A Systematic Review of Machine Learning Techniques for Software Fault Prediction", *Applied Soft Computing*, Vol. 27, pp. 504-518, 2015.

[49] Ruchika Malhotra, "Comparative Analysis of Statistical and Machine Learning Methods for Predicting Faulty Modules", *Applied Soft Computing*, Vol. 21, pp. 286-297, 2014.