Available online at: https://ijact.in

Date of Submission	19/11/2019
Date of Acceptance	10/12/2019
Date of Publication	31/12/2019
Page numbers	3525-3529 (5 Pages)

Cite This Paper: Abdulshaheed M, Hammad M, Alqaddoumi A, Obeidat Q., Mining Historical Software Testing Outcomes to Predict Future Results, 8(12), COMPUSOFT, An International Journal of Advanced Computer Technology. PP. 3525-3529.

ISSN:2320-0790

An International Journal of Advanced Computer Technology

MINING HISTORICAL SOFTWARE TESTING OUTCOMES TO PREDICT FUTURE RESULTS

Mohamed Abdulshaheed, Mustafa Hammad, Abdulla Alqaddoumi, Qasem Obeidat

Department of Computer Science, University of Bahrain, Sakheer, Kingdom of Bahrain
a.shaheed@gmail.com, {mhammad, aqaddumi, qobeidat}@uob.edu.bh

Abstract: Software bugs and program defects have significant negative effect on the cost and duration of software development process. Finding such bugs in early stages of the development process will cuts development time and maintenance costs. This investigation presents three different machine learning algorithms: K-Nearest Neighbors (KNN), Random Forest (RF), and Multilayer Perceptron (MLP) to build a new proposed software defect prediction model using different types of software performance metrics. This proposed model was tested on three public datasets obtained from NASA to assess its accuracy and revealed that the KNN was outperforms RF and MLP.

Keywords:software engineering; machine learning; prediction model; software defects; software evolution

I. INTRODUCTION

The presence of software bugs impacts significantly on software functionality, quality, and reliability. Software bugs in a production system is an unfavorable situation to the end-users and affect the organization in a term of reputation. It emerges when the software does not meet its specifications. Developing and deploying zero-bug software is difficult because the software becomes more complex.

Troubleshooting bugs is a critical activity in software development. However, every hour that the system developers spend to troubleshoot bugs is a waste of time. In addition, maintaining the system after it is launched is a costly job. Thus, building a defect prediction model is highly recommended, and it can be used to reduce costs and save time.

Software defect prediction is an evolutionary approach to discover or estimate, in prior stages, where defects may emerge. It is playing a key role in software success, as discovering defects in earlier stages of the software development life cycle will lead to an efficient and reliable software. In addition, costs can be reduced significantly when bugs are discovered and fixed in the testing stage than during maintenance stage. Besides, predicting defects will satisfy the users' needs and expectations which increases the software reputation such that it can be applied with different clients.

Building a software defect prediction model in software engineering is a challenging task. Many models have been presented to deal with software defect prediction issue [1]–[12], but the most notable one is using machine learning algorithms. These algorithms are used to predict software defects based on historical data and several software metrics.

The aim of this research is to build a software defect prediction model using different types of performance measurements, such as mean absolute error, root mean squared error, relative absolute error, root relative squared error, and correlation coefficient. This proposed model was examined by three machine learning algorithms: K-Nearest Neighbors (KNN), Random Forest (RF), and Multilayer Perceptron (MLP). These algorithms were used to examine their capabilities in fault prediction on three datasets collected during monitoring real-time software system.

The rest of this research is organized as follows: Section II presents a discussion of the related work in software fault prediction using different approaches. Section III provides an overview about the machine learning algorithms which are used in this study. A description about the dataset and evolution methodology is presented in Section IV. Experimental results are shown in Section V followed by conclusion and future work in Section VI.

II. Related Work

Many researchers used different machine learning models to predict defects in software modules such as Decision Trees [1], Naïve Bayes [2], Fuzzy Logic [3], Logistic Regression [4], Case-based Reasoning [5], Genetic Programming [6], and Artificial Neural Networks [7]–[10].

A study by Ma *et al.* [13] proposed a novel approach for software fault prediction based on balanced random forests. The proposed methodology was compared with many different classification and machine learning approaches using five defect datasets from NASA based on a set of performance measurements. The results showed that random forests can be the "best guess" among other classifier algorithms for a dataset that describes software metrics of fault components.

On another study, Boetticher [14] conducted a number of classification experiments to assess the effects of datasets in empirical software engineering. Decision Trees and Naïve Bayes models were used to analyze defect datasets from NASA. It has been observed that ten-fold cross validation is insufficient in the validation of a dataset. In the end, it has been stated that success in the evaluation of a training dataset depends on the level of difficulty of the dataset.

Challagulla *et al.* [15] applied Memory-Based Reasoning (MBR) classifier to predict software defectsin a dataset from NASA data repository (Metrics Data Program) by using different types of predicator software metrics such as McCabe, Halstead, line count, operator/operand, and branch. The assessment criteria were probability of false alarm, detection, and accuracy. A framework has been proposed that provides procedures in selecting the suitable configuration of MBR classifier.

A comparison created by Elish*et al.* [16] between the capability of support vector machine and eight statistical and machine learning methods. The datasets are obtained from four NASA software projects written in C, C++, and Java with 21 software metrics for each dataset. The results indicated that support vector machine has a higher accuracy against the performance of the compared methods.

Chidamber and Kemerer[17] developed and implemented a suite of six software metrics to assess the object-oriented design such as depth of inheritance tree and lack of cohesion in methods. Several studies have been conducted to evaluate these metrics, such as [18] and [19]. Basili *et al.*[18] measured the source code of eight projects written in C++ using the CK metrics. The Logistic Regression has been utilized to evaluate the impact of these metrics in the prediction of defect data found on object-oriented classes. The study concluded that five out of the six object-oriented metrics can be used in prediction of faulty classes.In addition, Tang *et al.* [19] studied the relationship between the CK object-oriented metrics and the object-oriented faults. CK metrics has been validated based on data from three industrial real systems developed by C++ using logic regression model. The study suggested that the CK two metrics,Weighted Methods per Class (WMC) and Response For a Class (RFC), can be good indicators for object-oriented faults. Finally, the study presented a new set of metrics that can be utilized to evaluate object-oriented faults.

Emam *et al.* [20] used Logistic Regression to evaluate object-oriented design metrics on data collected from a commercial application developed by Java. The results indicated that logistic regression has a high accuracy and export coupling (EC) metric is suited with fault-proneness.

Khoshgoftaar *et al.* [21] used Artificial Neural Networks and non-parametric discriminant techniques on a telecommunication system, which had seven million lines of code. The study compared the results between artificial neural network and nonparametric discriminant and illustrated that artificial neural network has a better accuracy rate.

Turhan*et al.* [22] examined 25 telecommunication system projects to predict fault proneness based on trained datasets from NASA by using static call graph-based ranking (CBGR) and nearest neighbor sampling. The study showed that 70% of faults can be detected by inspecting and only 3% by code using CBGR framework.

III. Machine learning Algorithms

The following is the summary of the three machine learning algorithms that are used to predict the accumulated faults of three testing and debugging datasets.

A. *K-Nearest Neighbors (KNN)*

KNN is a well-known and widely used machine learning algorithm in many fields because it is very simple.Also, KNN is considered as a lazy learning algorithm, where it use all stores training data and delays its learning until classification time [23]. Thus, this algorithm it has no specialized training phase. This algorithmfind the difference and similarity between different points in a specific area in graph by calculating the distance between these points.Euclidean distance (i.e., straight-line distance) is one of the most common distance function used in KNN [24].

B. *Random Forest (RF):*

RF generates many decision trees. For each decision tree there is a prediction result. The most repeated prediction result is selected by RF classifier to be the final class prediction result [25]. RF is one of the more accurate prediction algorithms because it can works with many decision trees and selecting the most important features for the classifier.

C. *Multilayer Perceptron (MLP):*

MLP (or Feedforward Neural Networks) is the most typical neural network model. Where it is consists of three layers: input layer, hidden mathematical layer(s), and output layer [26]. The input data is managed in the input layer and the output is store in the output layer. The number of the hidden layer can be increasedaccording to the task complexity.

IV. Datasets and Evolution Methodology

Three different datasets are used in this work to evaluate the accuracy of the machine learning algorithms. The datasets are collected from testing three different software systems, namely, dataset A, B, and C. The three datasets consist of each day measurement (D), detected faults (F), accumulated faults (AF), and number of test workers (W).

The evaluation criteria for evaluating the machine learning algorithms are standard numeric performance measurement: If$O$is the accumulated actual fault, $\tilde{O}$ is the accumulated predicted fault, $\bar{O}$ is the mean of O, and k is the number of test instances, then:

A. *Mean absolute error (MAE)*

It is the absolute sum of the error divided by number of predictions. The error is the difference between the accumulated actual fault and accumulated predictedfault[27].

$$MAE = \frac{1}{k}\sum_{j=1}^{k} |O_j - \tilde{O}_j|$$

B. *Root mean squared error (RMSE)*

It is the square root of sum of the square error divided by number of predictions. The error is the difference between the accumulated actual fault and accumulated predictedfault[29].

$$RMSE = \sqrt{\frac{1}{k}\sum_{j=1}^{k}\left(O_j - \tilde{O}_j\right)^2}$$

C. *Relative absolute error (RAE)*

It is the sum of the absolute error divided by sum of absolute relative error. The error is the difference between the accumulated actual fault and accumulated predictedfault, while the relative error is the difference between the accumulated actual fault and its mean [29].

$$RAE = \frac{\sum_{j=1}^{k} |O_j - \tilde{O}_j|}{\sum_{j=1}^{k} |O_j - \bar{O}_j|}$$

D. *Root relative squared error (RRSE)*

It is the square root of the square error divided by the square relative error. The error is the difference between the accumulated actual fault and accumulated predictedfault, while the relative error is the difference between the accumulated fault and its mean [29].

$$RRSE = \sqrt{\frac{\sum_{j=1}^{k}\left(O_j - \tilde{O}_j\right)^2}{\sum_{j=1}^{k}\left(O_j - \bar{O}_j\right)^2}}$$

E. *Correlation coefficient (R^2)*

Correlation coefficient shows how the accumulated actual fault and accumulated predictedfault are related. It gives value between 0 and 1. The correlation is 1 when the numbers of actual faults and predicted faults are similar, and it is zero when there is no relation between them [29] and[30].

$$R^2 = 1 - \frac{\sum_{j=1}^{k}\left(O_j - \tilde{O}_j\right)^2}{\sum_{j=1}^{k}\left(O_j - \bar{O}_j\right)^2}$$

V. ExperimentalResults

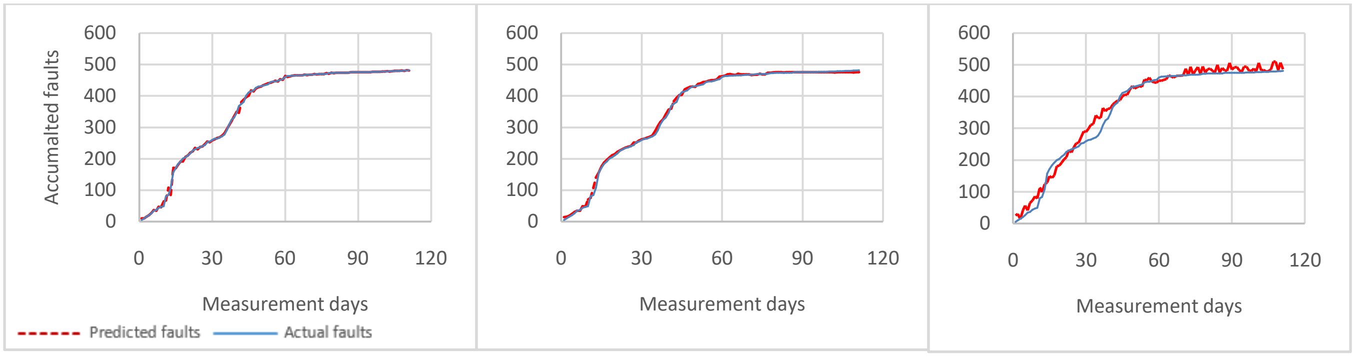
In order to evaluate the machine learning algorithm with the performance measurement, Weka 3.9.3 toolkit has been used.

Table I presents the performance measurement results for the evaluation of datasets A, B, and C. These results showed that KNN and RF machine learning algorithms have the best performance results, while MLP has the worst performance in all dataset.

The predicted and actual accumulated faults for the test instances for dataset A, dataset B, and dataset C based on the aforementioned three machine learning algorithms are shown in Fig.1, Fig. 2, and Fig. 3 respectively.

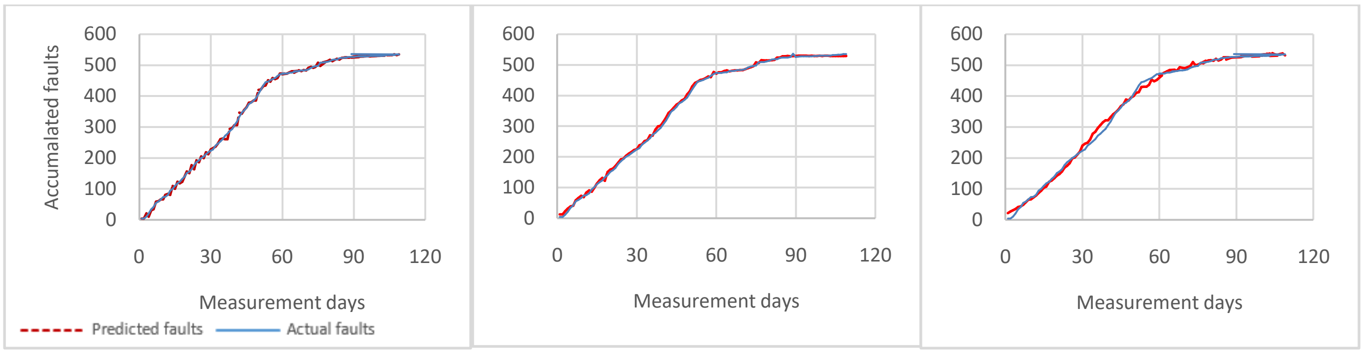
Table I: Performance Measurement Results for Dataset A, B, and C

	Dataset A					**Dataset B**					**Dataset C**				
Algorithms	***MAE***	***RMSE***	***RAE***	***RRSE***	***R²***	***MAE***	***RMSE***	***RAE***	***RRSE***	***R²***	***MAE***	***RMSE***	***RAE***	***RRSE***	***R²***
KNN	2.4009	4.9899	1.8537	3.319	0.9994	3.8853	5.6164	2.4855	3.2013	0.9995	3.8478	6.1987	5.4915	7.7145	0.9969
RF	3.8422	6.0254	2.9665	4.0079	0.9994	4.3953	5.4592	2.8118	3.1118	0.9997	4.1332	6.9066	5.8987	8.5955	0.9974
MLP	15.7285	20.9983	12.1438	13.9671	0.9919	7.6678	10.4495	4.9052	5.9562	0.9983	13.3191	16.7993	19.0087	20.9072	0.9769



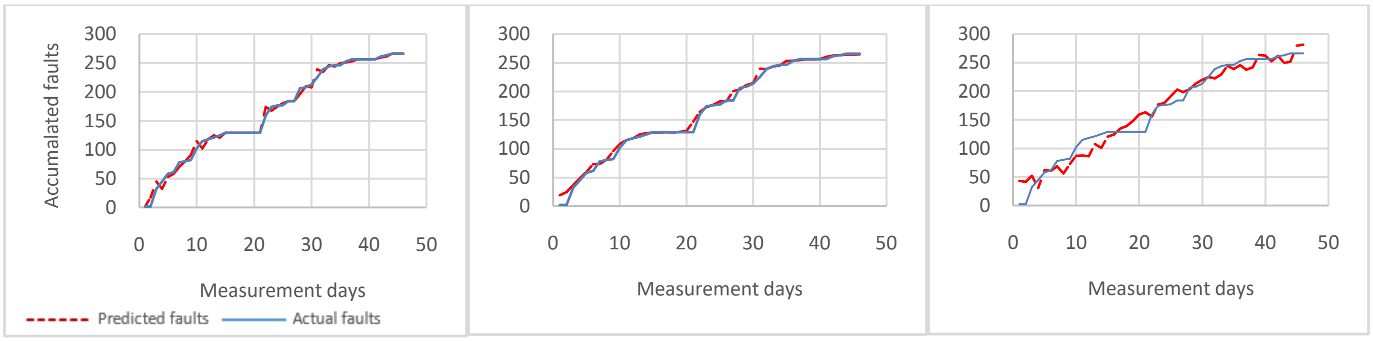
(a) KNN (b) RF (c) MLP

Fig.1 Predicted faults and actual faults for dataset A.



(a) KNN (b) RF (c) MLP

Fig.2 Predicted faults and actual faults for dataset B.



(a) KNN (b) RF (c) MLP

Fig.3 Predicted faults and actual faults for dataset C.

Table II shows a comparison between the three selected machine learning algorithms used in this work with other techniques that were used on the same datasets. The work in [28] used linear auto regression (AR) model and power model (POWM) in order to evaluate the three datasets with RMSE performance measurement.
For dataset A and C, KNN achieved the best result among other algorithms. However, MLP has the highest estimation error over other machine learning algorithms. For dataset B, AR model achieved a better estimation rate than the others.

Table II: RMSE measurement for machine learning algorithms, AR model, and POWR model

Dataset	Machine learning algorithms			Approach used in [28]	
	KNN	**RF**	**MLP**	**AR model**	**POWM**
Dataset A	4.9899	6.0254	20.9983	6.1365	32.3550
Dataset B	5.6164	5.4592	10.4495	3.2686	22.2166
Dataset C	6.1987	6.9066	16.7993	8.5462	11.9446

VI. Conclusion and Future Work

Software fault prediction is used to measure the software quality before releasing it. In this paper, accumulated faults from testing three software systems have been used to estimate the future accumulated faults using three types of machine learning algorithms. The machine learning algorithms have been evaluated by the errors between the actual and predicted accumulated faults using five types of performance measurements. A comparison between the results of this work with others that used the same testing datasets also has been provided. The comparison shows that some techniques performance such as KNN, RF, and AR are competitors between each other.

As a future work, we may study the impact of number of test workers in prediction faults described in the dataset A, B, and C using the machine learning algorithms with extra performance measurements such as accuracy, confusion matrix, precision, recall, and F-measures. In addition, this proposed software defect prediction model can be used in arecent study [31] to help the software tester in evolving the software test suite to accommodate the changes on code (i.e., new software version).

VII. References

[1] T. M. Khoshgoftaar and N. Seliya, "Software quality classification modeling using the SPRINT decision tree algorithm," Int. J. Artif. Intell. Tools, vol. 12, no. 03, pp. 207–225, 2003.

[2] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, 2007.

[3] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 85–90.

[4] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," IEEE Trans. Softw. Eng., vol. 33, no. 6, pp. 402–419, 2007.

[5] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components," J. Syst. Softw., vol. 55, no. 3, pp. 301–320, 2001.

[6] M. Evett, T. Khoshgoftar, P. Chien, and E. Allen, "GP-based software quality prediction," in Proceedings of the Third Annual Conference Genetic Programming, volume, 1998, pp. 60–65.

[7] T. J. McCabe, "A complexity measure," IEEE Trans. Softw. Eng., no. 4, pp. 308–320, 1976.

[8] M. E. R. Bezerra, A. L. I. Oliveira, P. J. L. Adeodato, and S. R. L. Meira, "Enhancing RBF-DDA algorithm's robustness: Neural networks applied to prediction of fault-prone software modules," in IFIP International Conference on Artificial Intelligence in Theory and Practice, 2008, pp. 119–128.

[9] P. C. Pendharkar, "Exhaustive and heuristic search approaches for learning a software defect prediction model," Eng. Appl. Artif. Intell., vol. 23, no. 1, pp. 34–40, 2010.

[10] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," Inf. Softw. Technol., vol. 49, no. 5, pp. 483–492, 2007.

[11] Hammad, M., Alqaddoumi, A. and Al-Obaidy, H., 2019. "Predicting Software Faults Based on K-Nearest Neighbors Classification.", International Journal of Computing and Digital Systems, 8(5), pp.462-467.

[12] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, and Fatima Alsarayrah. 2018. "Software bug prediction using machine learning approach.", International Journal of Advanced Computer Science and Applications, 9(2).

[13] Y. Ma, L. Guo, and B. Cukic, "A statistical framework for the prediction of fault-proneness," in Advances in Machine Learning Applications in Software Engineering, IGI Global, 2007, pp. 237–263.

[14] G. D. Boetticher, "Improving credibility of machine learner models in software engineering," in Advances in Machine Learning Applications in Software Engineering, IGI Global, 2007, pp. 52–72.

[15] V. U. B. Challagulla, F. B. Bastani, and I. L. Yen, "A unified framework for defect data analysis using the MBR technique," Proc. - Int. Conf. Tools with Artif. Intell. ICTAI, pp. 39–46, 2006.

[16] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," J. Syst. Softw., vol. 81, no. 5, pp. 649–660, 2008.

[17] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476–493, 1994.

[18] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Trans. Softw. Eng., vol. 22, no. 10, pp. 751–761, 1996.

[19] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in Proceedings sixth international software metrics symposium (Cat. No. PR00403), 1999, pp. 242–249.

[20] K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," J. Syst. Softw., vol. 56, no. 1, pp. 63–75, 2001.

[21] T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," IEEE Trans. Neural Networks, vol. 8, no. 4, pp. 902–909, 1997.

[22] B. Turhan, G. Kocak, and A. Bener, "Data mining source code for locating software bugs: A case study in telecommunication industry," Expert Syst. Appl., vol. 36, no. 6, pp. 9986–9990, 2009.

[23] S. KR, "Microarray Data Classification Using Support Vector Machine," Int. J. Biometrics Bioinforma., vol. 5, no. 1, pp. 10–15, 2011.

[24] L. Jiang, Z. Cai, D. Wang, and S. Jiang, "Survey of improving k-nearest-neighbor for classification," in Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), 2007, vol. 1, pp. 679–683.

[25] E. Jedari, Z. Wu, R. Rashidzadeh, and M. Saif, "Wi-Fi based indoor location positioning employing random forest classifier," in 2015 international conference on indoor positioning and indoor navigation (IPIN), 2015, pp. 1–5.

[26] J. C. Patra, R. N. Pal, B. N. Chatterji, and G. Panda, "Identification of nonlinear dynamic systems using functional link artificial neural networks," IEEE Trans. Syst. man, Cybern. part b, vol. 29, no. 2, pp. 254–262, 1999.

[27] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," IEEE Trans. Softw. Eng., vol. 15, no. 3, pp. 345–355, 1989.

[28] A. Sheta and D. Rine, "Modeling Incremental Faults of Software Testing Process Using AR Models," in the Proceeding of 4th International Multi-Conferences on Computer Science and Information Technology (CSIT 2006), Amman, Jordan, 2006, vol. 3.

[29] Y. kumar and G. Sahoo, "Analysis of Parametric & Non Parametric Classifiers for Classification Technique using WEKA," Int. J. Inf. Technol. Comput. Sci., vol. 4, no. 7, pp. 43–49, 2012.

[30] "machine learning - How to interpret error measures? -Cross Validated," 2017. [Online]. Available: https://stats.stackexchange.com/questions/131267/how-to-interpret-error-measures/272904. [Accessed: 19-May-2019].

[31] Alsolami, N., Obeidat, Q., Alenezi, M., 2019. Empirical Analysis of Object-Oriented Software Test Suite Evolution. International Journal of Advanced Computer Science and Applications, 10(11). (in press)