European Journal of Computer Science and Information Technology, 12 (2),22-40, 2024
Print ISSN: 2054-0957 (Print),
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Soft Computing-Based System for Performance Modeling of Object-Oriented Programming (OOP) Software

Anietie Uwah[1] and Imeh Umoren[2]
[1] Department of Computer Science, National Open University of Nigeria, Abuja, Nigeria
[2] Department of Computer Science, Akwa Ibom State University, Mkpat Enin, Nigeria
***Corresponding author: uwahanietie@gmail.com**

doi: https://doi.org/10.37745/ejcsit.2013/vol12n22240 Published April 23, 2024

Citation: Uwah A. and Umoren I. (2024) Soft Computing-Based System for Performance Modeling of Object-Oriented Programming (OOP) Software, *European Journal of Computer Science and Information Technology*, 12 (2),22-40

ABSTRACT: *Performance is a vital attribute for most software, making performance investigation an essential software engineering task. The problem is that modern applications are challenging to analyse for performance. The advancement of Software engineering has seen the application of scientific approach at various stages of Software Development Life Cycle (SDLC). New technique is available for implementation known as soft computing. This approach is a synergistic combination of artificial intelligence methodologies to model and solve real world problems that are either impossible or challenging to be modeled mathematically. The complexity, the commercial constraints and the expectation for high quality software demand, measuring the quality performance of an Object-Oriented Software (OOS). This work presents a set of testing metric (criteria) which hold the potential for ensuring remarkable improvements in quality performance of Object-Oriented software (OOS). A Prediction Model focused on identifying and evaluating the quality of the performance model based on the data set of software design components. The software design properties are comprehensive elucidated and integrating concepts of fuzzy logic and software engineering. The system enables the performance of modeling of Object-Oriented software using Fuzzy Inference System (FIS) to accurately predict the quality Object Oriented Software based on quality performance criteria such as reliability, efficiency, functionality and maintainability. Empirical validation of the results using checking data set is made to prove the usefulness of the design metrics and design quality attributes on the software prediction models. The purpose of this work is to provide development tool to solves practical problems, and focus on identifying and evaluating the quality performance of software design components. Again, the results demonstrate that, despite the research including approaches explicitly aimed at object-oriented software, there are substantial challenges in providing realistic feedback on the performance of large-scale object-oriented applications accordingly.*

KEYWORDS: SDLC, *Object Oriented Software (OOS), Efficient Prediction Model, Fuzzy Inference System (FIS)*

INTRODUCTION

Software development and engineering Industrial activities have advanced to the used of scientific techniques at various stages of the software development life cycle. These set of techniques hold the potential for remarkable improvements in the quality of software products. The complexity of modern software, the commercial constraints and the expectation for high quality products demand, measuring the quality of an object-oriented software. A research show that quality prediction focus on identifying and evaluating the performance of software files, classes and modules prior to coding stage using software design properties. Finding and correcting faults at early stage save a lot of software development cost and time. An efficient object-oriented software prediction development before coding, requires a superior prediction model, that focus on using system requirement and properties to analyze and predict prior to the coding phase so that the developers use this information for optimally performance, planned and quality assessment of the software under development. Requirements engineering and object-oriented design level metric contribute to early efficient prediction. This research work has also utilized the strengths of fuzzy logic to deal with the uncertainties and vagueness involved in the early-stage measures. Therefore, further empirical validation is necessary to prove the usefulness of the metrics and software prediction models in industrial practice. The need for efficient fault prediction model is required for useful technique for software organizations to prediction faults and evaluates performance, compared to other testing activities, verification and validation during the various phases of software development. By adopting software fault prediction model, software development team can forecast the costing at beginning of software development at a relatively lower cost. It also helps to optimize allocation of project resources and improve the quality assurance of software. This research work focus on developing an efficient fault prediction model used to enhance requirements and design constructs in object-oriented software prior to coding stage using software design properties based on fuzzy logic.

LITERATURE REVIEW

In [9], the proposed Eight relative code churn metrics (M1-M8) measuring the amount of code changes. M1 metric is computed by churned LOC (the accumulative number of deleted and added lines between a base version and a new version of a source file) divided by Total LOC. Other metrics (M2-M8) consider various normalized changes such as deleted LOC divided by total LOC, file churned (the number of changed files in a component) divided by file count, and so on. The relative churn metrics is proved as a good predictor to explain the fault density of a binary and bug-proneness.

The work in [8] extracted change metrics from the Eclipse repositories to conduct a comparative analysis between code and change metrics. The change metrics include addition and deletion in LOC similar to relative code change churn. However, a change metrics did not consider any relativeness by the total LOC and the file count but consider average and maximum values of change churn metrics. The metrics also include maximum and average of change sets (the number of files committed together) and age metrics (age of a file in weeks and the weighted age normalized by added LOC). The work concluded that change metrics are

better predictors than code metrics. The work of [4], applied Shannon's entropy to capture how changes are complex and proposed history complexity metric (HCM). To validate the HCM, the system built statistical linear regression models based on HCM or two change metrics, the number of previous modifications and previous faults on six open-source projects. The work was evaluated on six open-source projects showed that prediction models build using HCM outperform those using the two-change metrics. The idea adopting the Entropy concept to measure change complexity is novel but comparing models by HCM to those by only two change metrics reveals the weakness of the evaluation of HCM. In addition, evaluation was conducted in the subsystem-level rather than the file-level. The work of D'Ambros *et al*, (2010), conducted an extensive comparisons study of defect prediction metrics. It proposed work considers. code metric churn (CHU) and code Entropy (HH) metrics. In contrast to code change metrics based on the amount of lines, CHU measures the change in biweekly basis of code metrics such as CK metrics and OO metrics. Since code metric churn computes the amount of changes in bi-weekly basis, CHU captures the extent of changes more precisely than code change churn that computes the amount of changes between a base revision and a new revision. Four variants of CHU by applying decay functions (WCHU, LDCHU, EDCHU, and LGDCHU) also were proposed. While change Entropy is computed based on the count of file changes, code Entropy (HH) is computed based on the count of involved files when a certain code metric is changed. Hence, CHU, [1] defined four variants of HH by applying decay functions (HWH, LDHH, EDHH, and LGDHH). The comparison of [1] evaluation, concluded that WCHU and LDHH metrics led to good prediction results on all subjects used in their experiments. However, limitations of these novel metrics are heavy computation resources and data because it tracks bi-weekly changes from version control systems.

In [2], the proposed popularity metrics by analyzing e-mail archives by developers in a group mailing list. The main idea of popularity metrics is more discussed software artifacts in e-mail archives are more fault-prone. Table VI lists the popularity metrics. Most metrics quantify how many times a certain class are discussed in mails. The extracting metrics from e-mail archives is novel but their evaluation of the metrics shows that popularity metrics themselves did not outperform other code and process metrics. [3], proposed four ownership metrics based on authorship of a component. This study is started from the question, how much does ownership affect quality? The ownership of a component is defined by the portion of commits of the component and minor and major contributors are defined by less than and more than 5% portions of the ownership respectively. Four ownership metrics are defined a follow; MINOR (the number of minor contributors), MAJOR (the number of major contributors), TOTAL (the total number of contributors), and OWNERSHIP (portion of ownership of the contributor with the highest portion of ownership). They concluded that higher ownership leads to less fault-prone. [11]. conducted the fine-grained investigation on relationship between defects and human factors such as ownership and developer experience. The interesting finding of this study is that quality assurance should be focused on source code files touched by less experienced developers. [5], [6], proposed micro interaction metrics (MIM) extracted from Mylyn that captures developer interactions to Eclipse. The main idea of MIM is from the fact that defects could be introduced by mistakes of developers. For example, more editing time of a certain source code file may cause more bug-proneness. The work extracted metrics from Mylyn data and compared their performance with code and process metrics. The experiments, MIM outperformed code and process metrics in both classification and regression. However,

MIM is highly depended on Mylyn, a plug-in of Eclipse so that MIM might be hard to apply for other development environments that does not support the tool like Mylyn. C. Other metrics [7]. extracted developer's metrics from a developer social network that represents collaborative structure extracted from source code repositories. The developer social network found that software failure is highly correlated with developer network metrics. [9] consider constructed developer network with software modules, i.e. developer module network. This network represents how each developer contributes to each module so that the network is called 'contribution network. [9] found that the centrality measures for the contribution network can carry out prediction with post-release defects significantly. In [13]. constructed dependency (such as data and call dependencies) graphs of binaries and conducted network analysis on those dependency graphs. From various network analysis measures such as centralness, closeness, betweenness, etc. [13] built a prediction model and compared them to models constructed by code and process metrics. In the evaluation process, network measure could predict more bug-prone binaries than code and process metrics. [12]. proposed four antipattern metrics. Antipatterns are poor design of software so that there might be higher chance to introduce faults in the source code files. The evaluation of two open-source projects, antipattern metrics could improve prediction performance in terms of f-measure. [14] proposed a Handover Manageability and Performance Model for Mobile Communication Networks then formulated a model for soft handoff in CDMA networks by initiating an overlap region between adjacent cells which facilitating the derivation of handoff manageability performance model. The paper employed an empirical modeling approach to support their analytical findings, measure and investigated the performance characteristics of a typical communication network over a specific period in an established cellular communication network operator.
[10]. develop a suitable model for predicting performance impacts, usage and user satisfaction using Motadata technology in Coscharis automobile company where usage is mandatory. The work also determines the causal relationship between constructs in the developed model and their effects on employee satisfaction and job performance. The study adopted an integrated Task-technology fit (TTF) and Technology utilization, satisfaction and performance (TUSPEM) model components as the theoretical foundation. In [15], the paper proposed an enhanced approach to be considered to mitigate the time-consuming nature of disease diagnoses. Developing a framework for evaluating healthcare logistics is a conceptual equivalence of building an actual Healthcare Management System (HMS). Hence, the qualities considered when developing a system should also be considered when developing an evaluation framework

European Journal of Computer Science and Information Technology, 12 (2),22-40, 2024
Print ISSN: 2054-0957 (Print),
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

System and Model Design

System Architecture

Figure 1 below show the architecture diagram of the proposed **EPMQPOOS**:

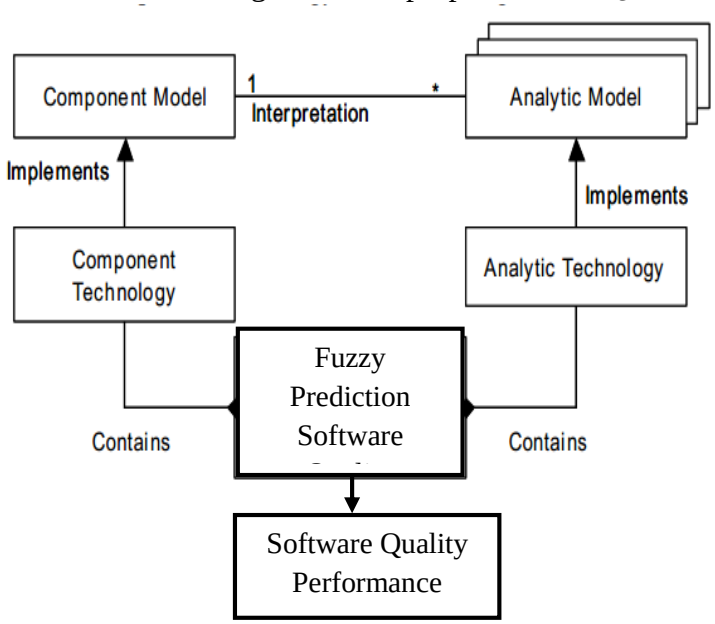

Fig. 1: Framework of EPMQPOOS using Soft Computing Model

In the above framework, Fuzzy Logic is used as the soft computing techniques to predict the quality of an object oriented software. The model uses four metrics to proffer the quality of software with emphasis on the quality performance; the four metrics are: ***Reliability, Efficiency, Functionality and Maintainability***. Figure 3.3 below shows the conceptual frame work of the fuzzy object-oriented software fault prediction.

MATERIAL AND METHODS FOR FIS IMPLEMENTATION

This section gives details of the implementation of the EPMQPOOS system. Six hundred and eighty-nine rows of previous used data derived from the operational data, i.e data get from Akwa Ibom State Transport Company (AKTC) data warehouse. The data obtained in advanced in the following major stages.

1. Data collection and Pre-processing
2. Fuzzy Inference System (FIS) Implementation

Table 1: Software Attributes to be evaluated

S/N	***QUALITY ATTRIBUTES***	***DESCRIPTION***
1.	**Reliability**	The capability of the software product to maintain a specified level of performance when used under specified conditions.
2.	**Efficiency**	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
3.	**Functionality**	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
4.	**Maintainability**	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.

A total of 682 data set as shown I Appendix A, was collected between the months of October 2013 and April 2014 from AKTC Head office, Uyo, Akwa Ibom state, Nigeria. The dataset covered data from CISCO ES-3750 Network Operating System, Transport Manager Software and Miktrotik Cloud router interface to tally ERP9. The data consist of 4 attributes as displayed in table 1 The data represents 686 instances. The attribute appears in their weight. Each weight has a total of 10. The dataset was divided into six parts. The first part and the fourth part were merged together to form the training data, the remaining parts joined together was used as checking or testing data for the system. This menu of the program displays both training data plot and checking data plot to see correlation between the both data. This stage reveals whether there is need to recognize the data or choose another data to be used for FIS. Below is a screen capture of training data and checking data plots. The checking data appears in the plot as placed superimposed on the training data. The horizontal axis is marked data set index. This index indicates the row from which that inputs data value was obtain (whether or not the input is a vector or a scalar).

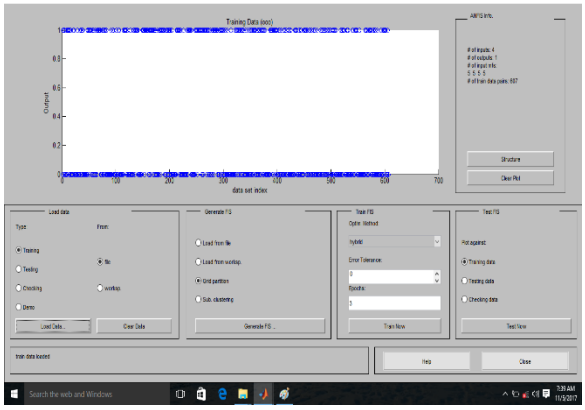

Fig. 2: Plot of training data

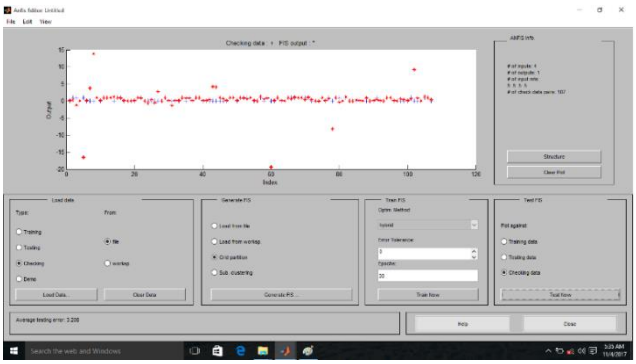

Fig. 3: Plot of checking data

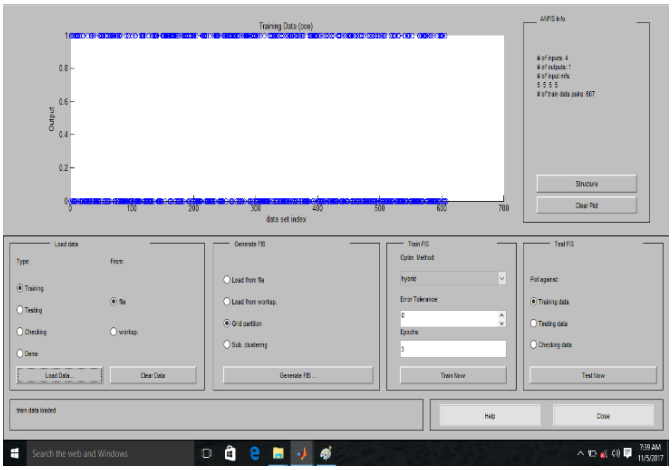

Fig. 4: A Plot of Raw Data

FIS Initialization and Generation

FIS parameters can either be initialized to one's preference, FIS can initialize the parameters automatically, and it has the following parameters: 4 inputs, 5 MFs and one linear output are showed in figure 5

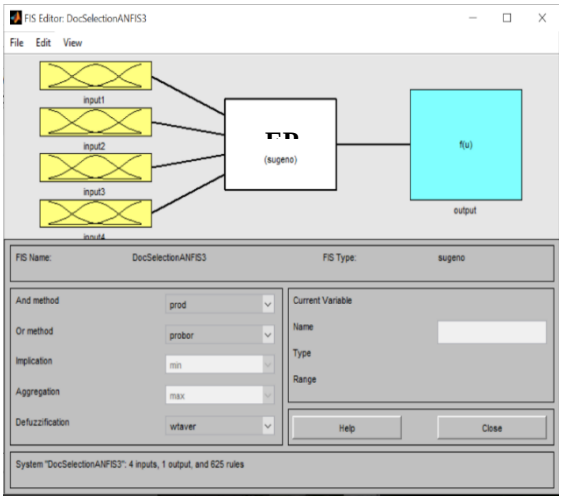

Fig. 5: FIS Inference Editor

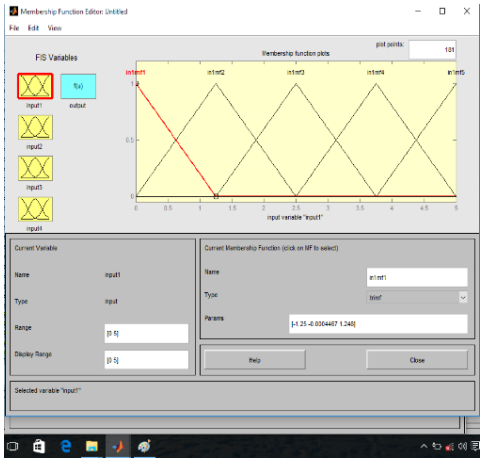

Fig. 6: Membership function Editor (Input1: Reliability)

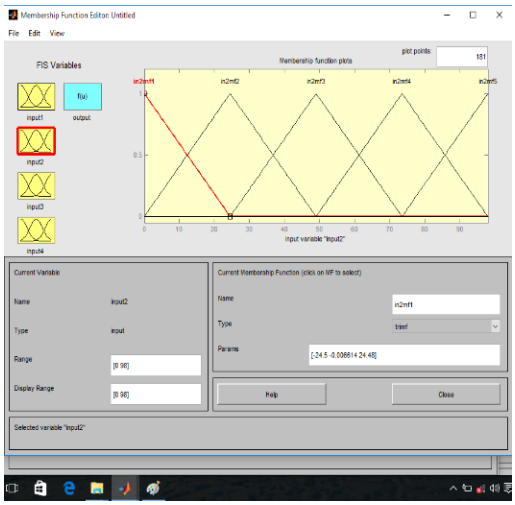

Fig. 7: Membership function Editor (Input 2: Efficiency)

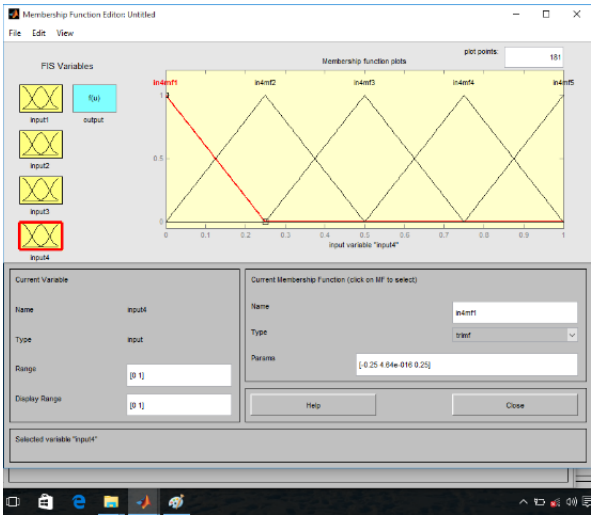

Fig. 8: Membership function Editor (Input 3: Functionality)

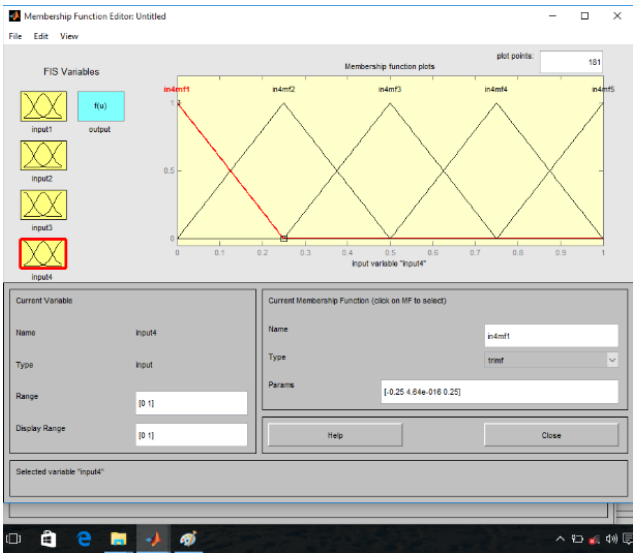

Fig. 9: Membership function Editor (Input4: Maintainability)

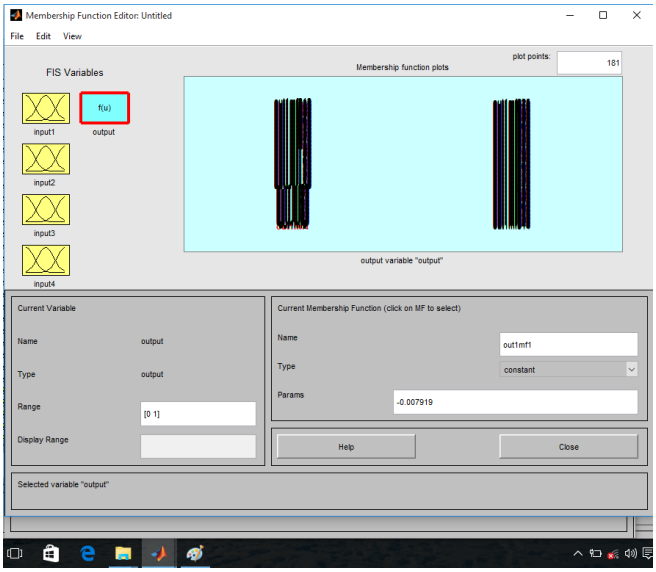

Fig. 10: Membership function Editor (Output: Quality Assurance)

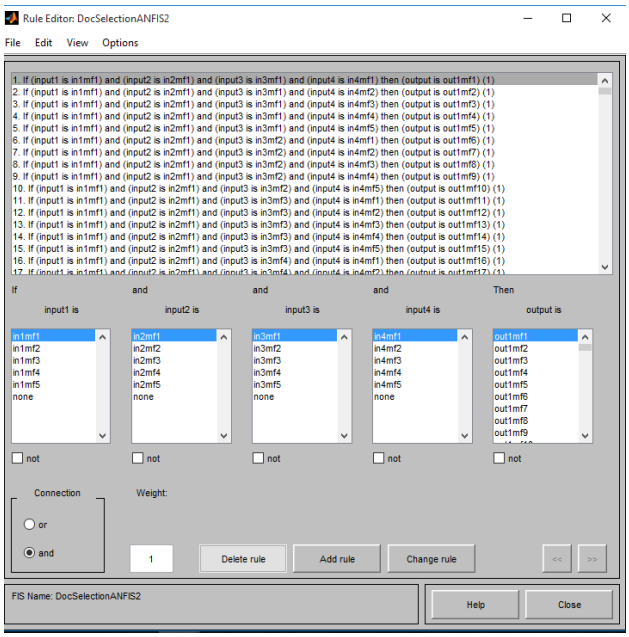

Fig. 11: FIS Rule Base Editor

Fig. 1`2: FIS Rule Analysis rule viewer

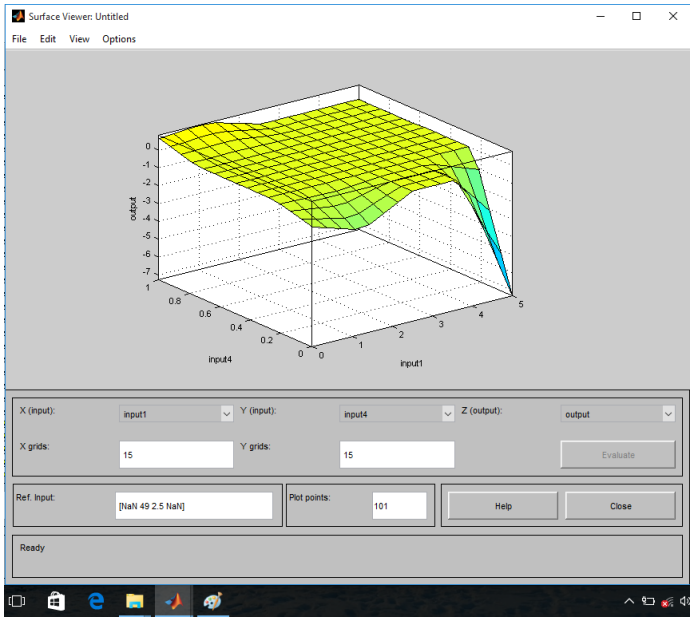

Fig. 13: FIS Surface Viewer (Var1 versus Var4: Reliability to Maintainability)

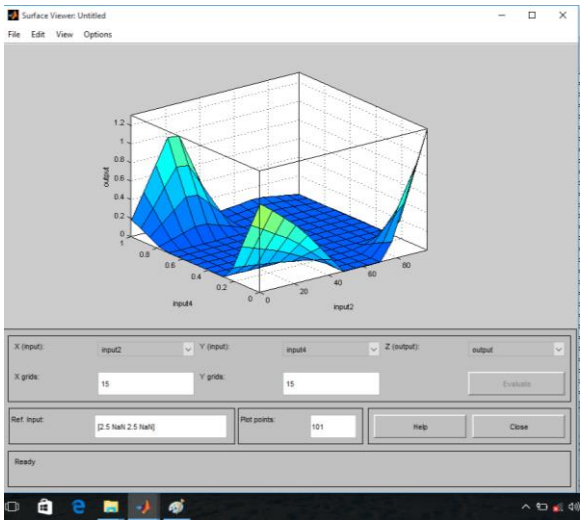

Fig. 14: FIS Surface Viewer (Var2 versus Var4: Efficiency to Maintainability)

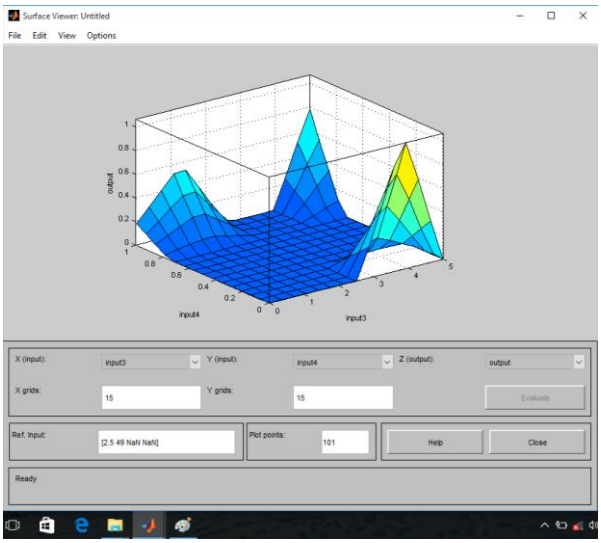

Fig. 15: FIS Surface Viewer (Var3 versus Var4: Functionality to Maintainability)

FIS Results Analysis

The results of clustering algorithms in the form of matrix are cluster center (C) and sigma(σ) which will be used to determine the values of Gauss membership function parameter. The steps of fuzzy rule extraction of the cluster are:

i. Calculate the degree of membership by using the Gauss membership function. Degree of membership of a data point Xi on the K-th cluster is shown below
ii. Creates a matrix U of size n x (cluster number * number of attributes) by
iii. Multiplying the membership degree of each data i in cluster k by each attribute j of the data
iv. The results of matrix U is applied to the least square error method to obtain the output parameter coefficient, to calculate coefficient K can be used in Least Square Error method using pseudo matrix.

Inference system is capable of constructing input-output. Mapping accurately based on both human knowledge and stipulated input-output data pairs. However, once a fuzzy model is developed, in most cases its needs to undergo an optimization process. The aim optimizing and refining is two folds; the model structures and parameters.

FIS Training Procedure

Figure 16 shows the FIS training editor which is made up of four major parts namely: load data, Generate FIS, Test FIS Output and FIS information. Figure 17 shows the training error, figure 18 shows the checking window, and figure 19 shows the training output. The training with back propagation is indicated in figure 19 with training output at 300 epochs.

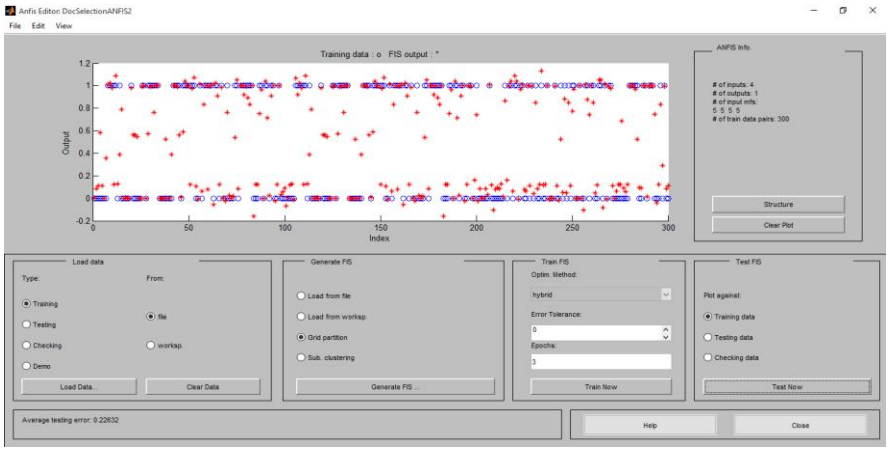

Fig. 16: FIS Training Window

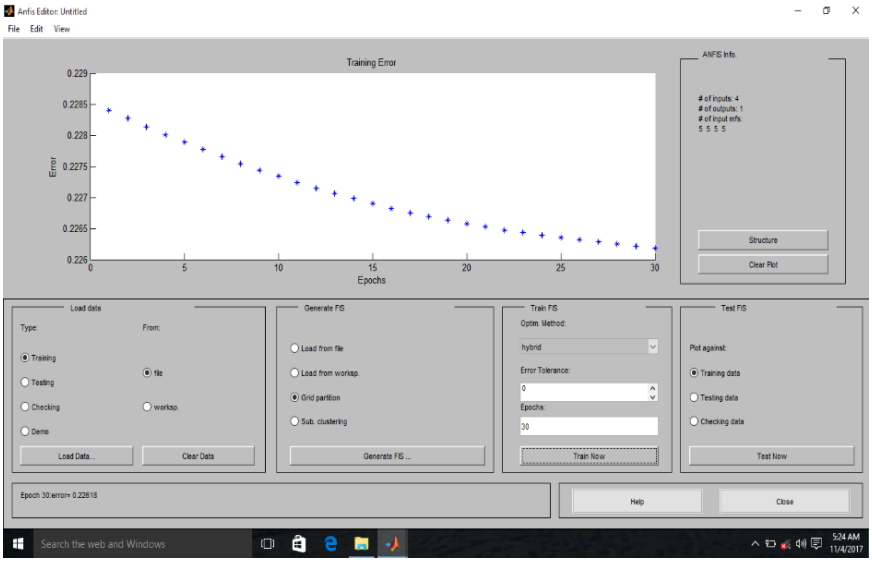

Fig. 17: FIS Training Error window

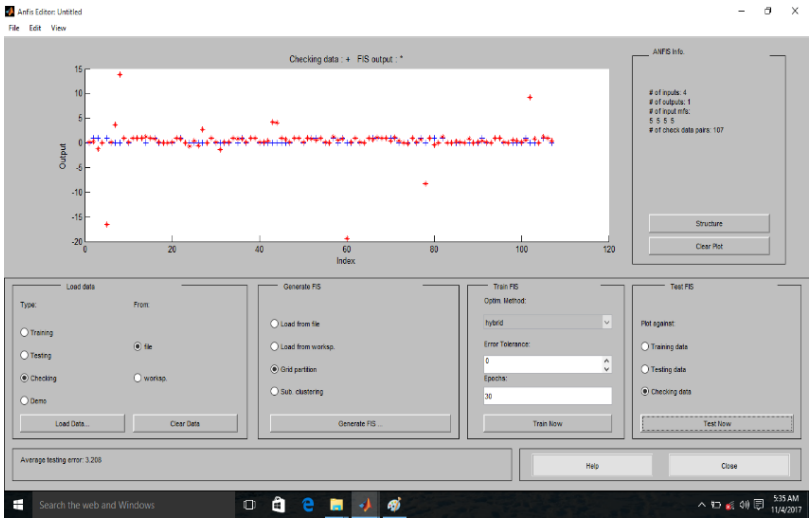

Fig. 18: FIS checking Window

The checking data are represented by the (+) sign the number of checking data pairs are 300.

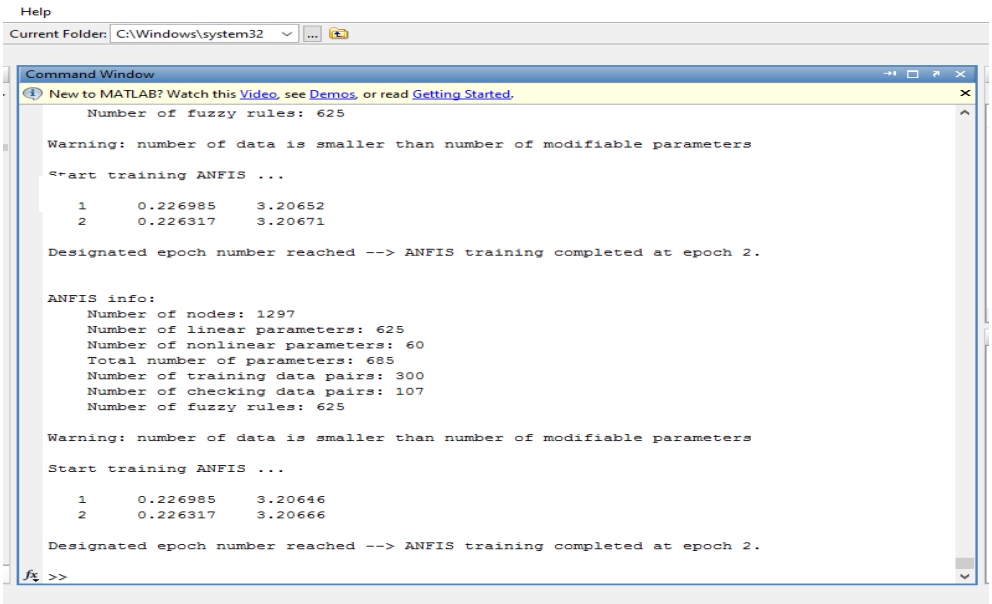

Fig. 19: FIS Training Output

Table 2: FIS training Information

S/N	Parameters	Sub clustering method
1	Number of nodes	1200
2	Linear parameters	625
3	Nonlinear parameters	60
4	Total number of parameters	685
5	Number of training data pairs	300
6	Number of checking data pairs	107
7	Total number fuzzy rules	625
8	Training mean square error	0.326985
9	Validation mean square error	0.043453
10	Testing mean square error	0.304532

From table 4.2, at epoch 300, the testing error value of 0.304532 is observed between the computed data and the desired output. The observed error value is far greater the error tolerance of 0.0001 specified in the train FIS. The idea behind using a checking data set for model validation is that after a certain point in the training, the model begins over fitting the training data set. In principle, the model error for the checking data set tends to decrease as the training takes place up to the points that over fitting begins, and then the model error for the checking data suddenly increases. Over fitting is accounted for by testing the FIS trained on the training data against the checking data, and chosen the membership function parameter to be those associated with the minimum checking error if these errors indicate model over fitting.

Testing and Integration

The system was tested in a window-based environment and the output was analyzed to find the correlate between the various input variables and their corresponding effect on the output variable. The Mathlab FIS platform was used to analyze the model of **EPMQPOOS**. Fuzzy based system and the new **EPMQPOOS** based system, the two outputs were correlated and the differences were noted as shown in Frame 5.1 below.

It was observed:

Frame 5.1: Result of the ten (10) outputs

FIS Output	Optimized Result
0.429906542	0.066084112
0.242990654	0.10346729
0.093457944	0.981971963
0.822429907	-0.466626168
0.102803738	0.271691589
0.121495327	0.48664486
0.355140187	0.140850467
0.242990654	0.346457944
0.336448598	-0.064757009
0.018691589	0.271691589

In the result above, it was observed that the optimized FIS output is better than the previous system's output by 0.253. This makes the EPMQPOOS-fuzzy based a better technique to implement in Software Quality Models better than other models.

DISCUSSION

This work considers, ISO/IEC 9126 model is used in defining and measuring the software quality performance in different perspectives by specifying and evaluating a set of design quality attributes. However, this prominent model provides more of frameworks for assessing and predicting the software quality performance than concrete models ready for practitioners to apply. Technically, two major tasks are involved when predicting a particular software quality performance; identifying the factors or the internal quality attributes contributing to the intended design components; and identifying the functional relationship between the design properties and the corresponding design metrics. That is the discovery of gem of this research work. Applying this technique will enable developers to efficiently analysis, design, and built evaluate quality performance of any software. Consequently, the results show that the fuzzy technique improves the prediction accuracy rates by optimized FIS output

CONCLUSION AND RECOMMENDATIONS

This work proposed a novel approach for the early prediction of software quality performance before coding. This technique is developed with the integration of concepts of artificial intelligence and software quality engineering. The proposed system uses Fuzzy Inference System to accurately predict the quality of the **EPMQPOOS** model. The factors which affect the quality of that software are fed into the FIS as inputs, and doing the necessary computations, it predicts the level of the quality performance of the software. The key benefit of this proposed technique is its ease of use and flexibility as it can take any number of inputs and generate any number of rules based. Next, we will empirical validation the result of the software prediction models using FIS. In future, we will work on hybrid learning algorithm by combination of two soft-computing methods of Artificial Neural Network ANN and Fuzzy Logic for hi-tech software quality

ACKNOWLEDGMENTS:

The authors would like to acknowledge the effort of anonymous reviewers and colleagues whose remarks have influenced the final shape of this article. We also acknowledge the services of Department of Computer Science, Faculty of Sciences, National Open University of Nigeria. Abuja, for their enormous resources and contributions during this research work.

Author Contributions: Conceptualization, A. Uwah, I. Umoren; methodology, I. Umoren; Software, A. Uwah; validation; I. Umoren, formal analysis, A. Uwah; investigation, A. Uwah; resources, I. Umoren, A. Uwah, data curation, A. Uwah; writing original draft preparation, A. Uwah, I. Umoren; writing review and editing, I. Umoren, A. Uwah; visualization, A. Uwah; Supervision, I. Umoren; project administration, I. Umoren, funding acquisition:

All authors have read and agreed to the published version of the manuscript.

Funding: This paper did not receive funding.

Institutional Review Board Statement: Not applicable.
Informed Consent Statement: Not applicable.
Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

REFERENCES

[1] Ambros, M, D, Lanza, M. & R. Robbes. (2012) Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software Engineering, 17(4-5):531–577,

[2] Bacchelli, A., D'Ambros, M., & Lanza, M. (2010). Are popular classes more defect prone? In *Fundamental Approaches to Software Engineering: 13th International Conference, FASE 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings 13* (pp. 59-73). Springer Berlin Heidelberg.

[3] Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code! examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE), pp 4–14

[4] Hassan, A. E. (2009, May). Predicting faults using the complexity of code changes. In *2009 IEEE 31st international conference on software engineering* (pp. 78-88). IEEE.

[5] Lee et al. (2011) Micro interaction metrics for defect prediction. In SIGSOFT '11/FSE-19: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering.

[6] Lee, T., Nam, J., Han, D., Kim, S., & In, H. P. (2011, September). Micro interaction metrics for defect prediction. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 311-321).

[7] Meneely, A., Williams, L., Snipes, W., & Osborne, J. (2008, November). Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (pp. 13-23).

[8] Moser, R., Pedrycz, W. & Succi, G. (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of the 30th international conference on Software engineering, ICSE '08, pages 181–190.

[9] Nagappan, N. and Ball, T. (2005) Use of Relative Code Churn Measures to Predict System Defect Density. Proceedings of the 27th International Conference on Software Engineering, St Louis, 15-21 May 2005, 284-292.

\https://doi.org/10.1145/1062455.1062514

[10] Osang, F. B., & Umoren, I. (2018). Modelling Motadata Software Application Use and Performance in an Automobile Service Company in Nigeria: The Roles of FIT and Voluntariness. *Information Systems & Development Informatics Journal*, *9*(2), 61-72.

[11] Rahman, F., & Devanbu, P. (2011, May). Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 491-500).

[12] Taba et al. 2013 Predicting bugs using antipatterns. In ICSM, pages 270–279, 20

[13] T. Zimmermann and N. Nagappan, (2008). "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 531–540, ACM, Leipzig, Germany.

[14] Umoren I., Asagba P. and Owolabi O. (2014). Handover Manageability and Performance Modeling in CDMA Mobile Communication Networks, International Institute for Science, Technology and Education (IISTEE) and Computing Information Systems Development Informatics and Allied research - Journal (CISDA). Vol 5 No. 1, ISSN: 2167 -1710, page 27-42. www.cida.com

[15] Umoren I., Usua G. and Osang F. (2019). Analytic Medical Process for Ophthalmic Pathologies Using Fuzzy C-Mean Algorithm, Article Innovations in Systems and Software Engineering.