INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ORIGINAL ARTICLE

Empirical Evaluation of Maintenance Effort Prediction using Inheritance and Interface-Oriented Class Complexity Metrics

K Maheswaran[1]*, **A Shreenath**[1]

1 Department of Computer Science, St Joseph's College (Autonomous), Affiliated with Bharathidasan University, Tiruchirappalli -620002, Tamil Nadu, India

OPEN ACCESS

Received: 13/04/2026

Accepted: 26/06/2026

Published: 25/07/2026

Citation: Maheswaran K, Shreenath A (2026) Empirical Evaluation of Maintenance Effort Prediction using Inheritance and Interface-Oriented Class Complexity Metrics. Indian Journal of Science and Technology 19(26): 1873-1882. https://doi.org/10.17485/IJST/v19i26.600

* **Corresponding author**.

mahes161@gmail.com; maheswaran_cs1@mail.sjctni.edu

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: The primary objective of this research is to estimate maintenance effort using complexity metrics and to demonstrate that the two selected metrics, CWICC and ICWCC, provide more accurate maintenance effort predictions compared to existing software complexity metrics. This will be validated through the application of various statistical techniques. **Method:** Two cognitive complexity metrics—Cognitive Weighted Inheritance Class Complexity (CWICC) and Interface-based Cognitive Weighted Class Complexity (ICWCC) were identified and empirically analyzed. Statistical techniques, including simple and multiple regression analysis, were applied to a collected dataset. The results were further compared with existing object-oriented metrics. **Findings:** The analysis demonstrates that cognitive complexity metrics can effectively predict software maintenance effort. The results also highlight differences in performance when compared to traditional object-oriented metrics, providing insights into their relative effectiveness. **Novelty**: This study introduces and validates two cognitive complexity metrics, CWICC and ICWCC to predict software maintenance effort. The novelty of this research work lies in establishing that the proposed metrics enable more accurate prediction of maintenance effort through closer alignment between predicted and actual maintenance time; for instance, ICWCC (160.54) and CWICC (159.09) closely approximate the observed value (162.08), outperforming WCC (139.32) and AWCC (154.29), thereby confirming their effectiveness as reliable predictors of software maintenance effort.

Keywords: Maintenance; Effort; Cognitive Complexity Metrics; Maintenance Time; Correlation; Regression

1 Introduction

Empirical validation plays a crucial role in investigating software development practices by enabling the understanding, evaluation, and improvement of software systems. It allows researchers to assess the validity of theoretical models through empirical observations, particularly in relation to key parameters such as performance,

complexity, and maintainability in software engineering [1]. Among these, maintenance is a critical quality attribute in industry, as it significantly influences overall cost and effort. Consequently, software metrics are widely employed to predict maintenance effort [2]. Software maintenance is defined as "the process of modifying a software system or its components after delivery to correct faults, improve performance or other attributes, or adapt it to a changed environment" [3].

Software complexity, measured through appropriate metrics during development, serves as a key indicator for predicting maintenance effort. In particular, design-phase complexity metrics can provide early insights into system quality and maintainability. However, traditional Object-Oriented (OO) metrics often fail to capture the true complexity of software, as they overlook cognitive aspects involved in understanding code. Syed et al. examines the evolution of software maintainability metrics from traditional approaches to Agile and DevOps environments. The authors conclude that modern maintainability assessment should combine code quality, operational, and process-oriented metrics to support continuously evolving software systems [4].

Cognitive complexity reflects the mental effort required by developers, testers, or maintenance engineers to comprehend software. It is quantified using Cognitive Weights (CWs), which represent the relative effort and time needed to understand different software structures. Metrics based on CW aim to measure this mental burden, where lower values indicate easier comprehension and better maintainability, while higher values suggest increased maintenance effort [5]. Although prior studies indicate that design-based complexity metrics can act as predictors of maintenance performance, the relationship between maintenance effort and OO metrics remains complex and often non-linear. This gap highlights the need for more effective complexity measures that better capture cognitive aspects and improves the accuracy of maintenance effort prediction. higher cognitive weight implies greater mental effort is required to comprehend software, directly influencing maintainability. In object-oriented (OO) systems, cognitive complexity is particularly significant, as it captures the effort needed by developers to understand, modify, and maintain code. Elevated cognitive complexity often results from poor design decisions, making systems difficult to manage and increasing maintenance effort. This limitation also reduces the effectiveness of traditional linear models in accurately predicting maintenance effort. To address these challenges, prior research has emphasized the use of software metrics during the development phase, enabling early detection of design and cognitive issues and thereby reducing future maintenance costs. Although numerous empirical studies have explored relationships between OO design complexity, cognition, and maintenance effort, their predictive consistency remains limited, highlighting the need for more robust and cognitively grounded metrics. Several studies have contributed to maintenance effort estimation; however, important limitations remain.

Bisht et al. proposed the cohesion metric and demonstrated its effectiveness in predicting reusability, but the study does not explicitly address maintenance effort or cognitive complexity [6]. Umar et al. proposed ML-PEQRM: A Machine Learning Framework for Software Quality, Reliability, and Maintenance Effort Estimation, a machine learning framework for predicting software quality, reliability, and maintenance effort. The study shows that ML-based approaches improve accuracy in estimating software maintenance and quality attributes. [7]. Manju and Pradeep Kumar Bhatia propose a fuzzy logic–based model to evaluate maintainability of object-oriented software systems. The approach integrates multiple object-oriented metrics to handle uncertainty and provide a more flexible assessment of maintainability. The study concludes that fuzzy logic improves decision-making in maintainability evaluation compared to rigid traditional metric-based methods [8]. Qullet et al. combined OO metrics with centrality measures to predict fault-proneness and severity, achieving improved performance; however, their work primarily targets defect prediction rather than maintenance effort [9]. Filo et al. established correlations between state chart metrics and cognitive understanding, but their validation is limited to specific modeling constructs [10].

Rochimah S. et al. propose a maintainability framework to ensure software quality in object-oriented programming systems. The study integrates structured maintainability metrics to systematically evaluate and improve code quality during software development. The authors conclude that a framework-based approach enhances maintainability assessment and supports long-term software reliability and evolution [11]. Shukla et al. examine various machine learning techniques for effort estimation in object-oriented software development. The study compares different learning models to improve the accuracy of predicting development effort based on software metrics. The authors conclude that learning-based approaches provide more reliable and adaptive effort estimation compared to traditional methods. [12]. Gokul Yenduri showed that OO metrics outperform traditional metrics in predicting maintainability; however, cognitive aspects are not explicitly considered [13]. Mei et al. validated metric thresholds using large-scale data, improving prediction accuracy, but their threshold-based approach does not account for cognitive effort [14]. Ofem et al. validated object-oriented based and aggregated complexity metrics, respectively, yet their studies emphasize maintainability and code complexity rather than direct maintenance effort prediction [15]. Heričko et al. highlights the Maintainability Index (MI) and proposed with different metric combinations and weightings. It empirically compares several MI variants and concludes that MI is useful for relative maintainability assessment. [16]. Recent advancements using artificial intelligence demonstrate improved predictive performance but introduce new challenges. Anita Devi et al. utilized deep neural

networks with OO metrics, achieving higher accuracy; however, the approach lacks interpretability and does not explicitly integrate cognitive complexity[17]. Similarly, Jha et al. showed that deep learning models outperform traditional techniques in predicting change-related variables, but their work prioritizes performance over understanding cognitive factors[18].

Overall, the literature indicates that while significant progress has been made using OO metrics, statistical models, and machine learning techniques and theoretical validations[19], either focus on indirect indicators such as fault-proneness and maintainability or fail to incorporate cognitive complexity explicitly. Additionally, many models assume linear relationships or lack interpretability, limiting their effectiveness in practical maintenance effort prediction. These limitations reveal a clear research gap: the absence of comprehensive, empirically validated models that integrate cognitive complexity with maintenance effort prediction.

The primary objective of this experimental study was to empirically validate the two proposed cognitive complexity metrics, CWICC and ICWCC, by examining their effectiveness in predicting software maintenance time. To address this gap, the present study proposes a maintenance effort prediction approach based on two cognitive complexity metrics, CWICC and ICWCC, introduced by Maheswaran et al.[20],[21]. The proposed approach explicitly incorporates cognitive aspects of software comprehension and evaluates their effectiveness using multiple statistical techniques. By doing so, this study aims to demonstrate that cognitively enriched complexity metrics can provide more accurate, reliable, and practically meaningful maintenance effort predictions compared to traditional OO metrics, thereby offering a more effective solution to the limitations identified in existing literature.

2 Methodology

This study adopts a controlled experimental research design to empirically evaluate maintenance performance in object-oriented software systems. The experiment is structured around four primary parameters: design complexity, maintenance task, program level, and programmer ability. These parameters are selected to capture key factors that influence software maintenance effort and performance. The experiment was conducted at Bharathidasan University, Tamil Nadu, India, with participants drawn from two academic levels: undergraduate and postgraduate students. These groups represent differing levels of programmer ability and academic exposure, allowing for comparative analysis across experience levels. Two independent experimental treatments were employed. Treatment 1 focuses on perfective maintenance, while Treatment 2 addresses corrective maintenance. Each participant was required to perform both treatments, ensuring a within-subject experimental design. For each treatment, tasks were executed under low and high condition levels, corresponding to variations in design complexity and program characteristics. By requiring all participants to complete both maintenance types under varying conditions, the experimental design enables systematic observation of maintenance performance across task types, complexity levels, and programmer ability, while controlling for individual differences [Table 1].

Table 1. Objectives and Variables of the Empirical Study

Category	Description
Primary Objective	To investigate the relationship between design complexity and maintenance tasks with maintenance performance, and to develop a prediction model for software maintenance effort.
Dependent Variable	Maintenance Performance, expressed in terms of comprehension time.
Independent Variables	Design Complexity and Maintenance Task Type.
Design Complexity Metrics	Cognitive Weighted Inheritance Complexity (CWICC) and Inheritance Cognitive Weighted Class Complexity (ICWCC).
Maintenance Task Types	Perfective Maintenance and Corrective Maintenance.
Measurement of Dependent Variable	Computed based on the number of lines of code (LOC) modified, accuracy of modifications, time required to perform modifications, and time taken to understand, develop, and modify existing programs.
Outcome	Development of an empirical prediction model for maintenance effort.

2.1 Measurement of Design Complexity

Two metrics serve as measures of design complexity (independent variable) in this study: Cognitive Weighted Inheritance Class Complexity (CWICC)[19] and Interface-based Cognitive Weighted Class Complexity (ICWCC)[20]. These metrics have been subjectively validated through comparison with existing similar metrics, namely Weighted Class Complexity (WCC) and

Attribute Weighted Class Complexity (AWCC), and have demonstrated superior performance. The CWICC metric emphasizes different levels of inheritance hierarchy and assigns cognitive weights accordingly. The CWICC metric for a class can be computed using Equation 1. The ICWCC metric incorporates the complexity of user-defined interfaces implemented by a class, as represented in Equation 2.

The cognitive class complexity metric CWICC has been enhanced by taking into account the inheritance complexity of a class. It illustrates the different levels of the inheritance hierarchy, showing how cognitive weights are assigned. If a class has 'p' attributes, 'q' methods, and 'r' inherited classes, then the CWICC metric for that class can be calculated as shown in equation (1).

$$CWICC = \sum_{i=1}^{p} AC_i + \sum_{j=1}^{q} MC_j + \sum_{k=1}^{r} IICC_k \tag{1}$$

In this context, AC, MC, and IICC represent attribute, method, improved inherited class complexity respectively. AC is utilized to assess the complexity of the attribute within the class using equation (2)

$$AC = (PDT * W_b) + (DDT * W_d) + (UDDT * W_u) \tag{2}$$

The terms PDT, DDT, and UDDT represent distinct categories of data type attributes. Their associated cognitive weights—denoted as W_b, W_d, and W_u, respectively fall within a scale ranging from 1 to 3, where W_b corresponds to the lowest level of cognitive effort and W_u to the highest. Complexity of the method is calculated based on the following equation (3) given below.

$$MC = \sum_{j=1}^{q} \left[\prod_{k=1}^{m} \sum_{i=1}^{n} W_c(j,k,i)\right] \tag{3}$$

The IICC is calculated by adding the complexities of different levels of inheritance hierarchy, reused method and attribute complexity as shown in the equation (4).

$$IICC = CDC + \sum_{k=1}^{s} RMC_k + \sum_{i=1}^{t} RAC_i \tag{4}$$

The 's' is the number of inherited methods, 't' is the number of inherited attributes, CDC is the cognitive depth Complexity which is calculated in equation (5). RMC and RAC denote the complexity due to reused methods and attributes.

$$CDC = DC * (CW_l) \tag{5}$$

Where DC is the depth of the class from the root class. CW_l is the cognitive weight of the particular level in the inheritance hierarchy which is tabulated in following Table 2. The cognitive weights for various inheritance levels are calculated based on the cognitive phenomenon described by Wang and the calibrations for the various levels of inheritance hierarchy.

Table 2. Cognitive weight for different inheritance level

Level	One	Two	Three	Four	Five	Six
Cognitive Weight	1	2	3	5	8	12

In CWICC, class complexity is enhanced by accounting for inheritance. It emphasizes the varying levels within the inheritance hierarchy, which form the basis for assigning cognitive weights. However, this metric does not consider the complexity introduced by user-defined interfaces when measuring class complexity. To overcome the limitation of CWICC, a metric called ICWCC is proposed. If the class has 'p'-attributes, 'q'-methods, 'r'- reused classes and 's' user-defined interfaces then the ICWCC metric of a class can be computed as given in equation. (6).

$$ICWCC = \sum_{i=1}^{p} AC_i + \sum_{j=1}^{q} MC_j + \sum_{k=1}^{r} IICC_k + \sum_{k=1}^{s} IIC_k \tag{6}$$

Where, AC, MC, IICC, and IIC stands for complexity of attribute, method, improved inherited class complexity and interface implemented complexity respectively.

The Interface Implemented Complexity (IIC) is used to calculate the complexity due to the various kinds of interface implemented in the class by using equation. (7). The cognitive weights assigned to interfaces are based on the taxonomy of cognitive phenomena proposed by Wang, as outlined in Table 3. The SI, EI, and NI denote Simple, Extended and Nested Interfaces.

Table 3. Cognitive Weights for the different types of Interfaces

Interface Types	Cognitive Weights
SI	3
EI	4
NI	5

$$IIC = (NSII * CWs) + NMSI + (NEII * CWe) + NMEI + (NNII * CWn) + NMNI \quad (7)$$

Here, NSII is the number of simple interfaces implemented, NEII is the number of extended interfaces implemented, and NNII is the number of nested interfaces implemented. Also, NMSI, NMEI, and NMNI are the number of methods defined in simple, extended and nested interfaces, respectively. The Cognitive weights i.e. CWs, CWe, CWn are simple, extended and nested interface weights.

2.2 Experimental Design

A total of 123 participants were recruited from both undergraduate (UG) and postgraduate (PG) programs to participate in this controlled experiment. Participant selection was based on two primary criteria: (1) programming experience and (2) academic performance during the semester. This selection approach ensured a reasonable level of homogeneity in programming competence while maintaining sufficient variance to examine the potential moderating effect of participant type on maintenance time. Prior to the experiment, all participants received standardized instructions regarding the experimental procedures, maintenance tasks, and time recording protocols. Participants were randomly assigned to experimental conditions to minimize selection bias and ensure internal validity.

Two independent treatments representing distinct maintenance task types were administered in this experiment: perfective maintenance (Treatment 1) and corrective maintenance (Treatment 2). Each treatment was further subdivided into low and high complexity conditions based on the CWICC and ICWCC metric values. The experimental data for the Treatment 1 and 2 is given in the Appendix A and B.

2.2.1. Treatment 1: Perfective Maintenance (Tri_Area System)

The first treatment involved a "Tri_Area" system designed to calculate the area of different types of triangles, including equilateral, isosceles, and scalene triangles. Participants were tasked with adding new routines to compute the area of additional triangle types based on specified user requirements. This treatment simulated real-world perfective maintenance scenarios where functionality enhancements are requested without altering the core system logic.

2.2.2. Treatment 2: Corrective Maintenance (Eb_Bill System)

The second treatment utilized an "Eb_Bill" system that calculates electricity bills based on unit consumption. Participants were required to identify and correct errors in the billing calculation method according to a given tariff structure and consumption units. This treatment represented typical corrective maintenance activities involving bug identification and resolution.

The dependent variable, maintenance time, was recorded for each participant across both treatments and complexity levels. Table 4 presents the Mean Maintenance Time (MMT) in minutes for the two treatments across low and high complexity conditions.

Table 4. Mean Maintenance Time for Two Treatments

Treatments	Mean Maintenance Time (MMT) [in minutes]	
	Low Complexity	High Complexity
Treatment 1 (Perfective)	70.84	89.78
Treatment 2 (Corrective)	129.98	162.17

As evident from Table 4, corrective maintenance tasks required substantially more time than perfective maintenance tasks across both complexity levels. Additionally, high complexity conditions consistently resulted in longer maintenance times compared to low complexity conditions, suggesting a potential interaction effect between task type and complexity level.

2.3 Data Collection Procedure

Participants were required to complete the assigned maintenance tasks within a controlled laboratory environment. Maintenance time was measured from the moment participants began examining the source code until they submitted the modified, tested, and validated solution. All submissions were evaluated for correctness and completeness to ensure that recorded maintenance times reflected successful task completion rather than premature or incomplete attempts. The structural and complexity characteristics of both treatments are detailed in Table 5. These characteristics include the number of classes, methods, interfaces, and the corresponding CWICC and ICWCC metric values for each complexity level.

Table 5. Characteristics of the Two Treatments

Attributes	Treatment 1 – Perfective (Tri_Area)		Treatment 2 – Corrective (Eb_Bill)	
	Low	High	Low	High
#Classes	2	1	2	3
#Methods	5	17	7	14
#Interfaces	1	2	2	2
CWICC	14	38	26	72
ICWCC	19	46	35	86

The complexity manipulation was achieved by varying the architectural design of the systems. For instance, in Treatment 1 (perfective maintenance), the high complexity condition featured 17 methods compared to only 5 methods in the low complexity condition, resulting in CWICC values of 38 and 14, respectively. Similarly, Treatment 2 exhibited higher metric values in the high complexity condition (CWICC = 72, ICWCC = 86) compared to the low complexity condition (CWICC = 26, ICWCC = 35).

The study investigated whether higher cognitive complexity, as measured by these metrics, is associated with increased maintenance effort. To assess the statistical significance of differences in maintenance time between high- and low-complexity program versions, one-way Analysis of Variance (ANOVA) tests were conducted. The ANOVA procedure enabled us to determine whether the mean maintenance times of the two complexity groups differed significantly.

In addition, simple correlation analysis was performed to evaluate the strength and direction of the relationship between each cognitive complexity metric (CWICC and ICWCC) and maintenance time. This analysis provided further insight into the individual predictive capability of each metric. To enhance the validity and generalizability of the findings, the dataset was partitioned into two subsets: a model-building dataset (80%) and a holdout dataset (20%) [16]. The model-building dataset was used to develop and evaluate the statistical models, while the holdout dataset served to validate the robustness and predictive stability of the results. The consistency of findings across these subsets increases confidence in the reliability of the proposed metrics and the conclusions drawn from the analysis.

3 Results and Discussion

3.1 Maintenance Effort Prediction Using Simple Linear Regression Analysis

To investigate the predictive capability of the complexity metrics, simple linear regression analysis was performed. Regression analysis provides a quantitative framework for modeling the relationship between a dependent variable (Y) and an independent variable (X). In this study, maintenance time was treated as the dependent variable, while each cognitive complexity metric served as an independent variable. Separate regression models were developed for each metric using the linear form $Y = a + bX$ where Y represents the Predicted Maintenance Time (PMT), X denotes the respective complexity metric, a is the intercept, and b is the regression coefficient.

To evaluate model adequacy, the adjusted coefficient of determination (Adjusted R^2) was used. Adjusted R^2 indicates the proportion of variance in maintenance time explained by the complexity metric while accounting for model complexity. Higher values reflect better predictive performance. The summarized regression results are presented in Table 6. All models yielded statistically significant results with $p < 0.00001$ at the 99% confidence level. Accordingly, the null hypothesis (H_{03}) is rejected and the alternative hypothesis ($H\square_3$) is accepted, confirming the existence of a statistically significant linear relationship between complexity metrics and maintenance time. The predictive accuracies (Adjusted R^2 values) are as follows:

1. WCC: 31.8%
2. AWCC: 50.5%
3. CWICC: 60.8%
4. ICWCC: 65.5%

The proposed metrics, particularly ICWCC, demonstrate substantially higher explanatory power compared to the existing metrics. This indicates that CWICC and ICWCC account for a larger proportion of variance in maintenance time and thus provide more reliable predictions of maintenance effort. The resulting maintenance effort prediction models are expressed as:

1. PMT = 71.5258 + 1.4817 × WCC
2. PMT = 71.2126 + 1.2291 × AWCC
3. PMT = 64.4296 + 1.3223 × CWICC
4. PMT = 58.6507 + 1.1916 × ICWCC

These models were applied to compute Predicted Maintenance Time (PMT) for both existing and proposed metrics. Additionally, Mean Maintenance Time (MMT) was calculated for the 20% holdout validation dataset. The results are presented in Table 6.

Table 6. PMT & MMT of metrics calculation for both treatments

PMT (80%) *(in Minutes)*	**Treatment 1 – Perfective (Tri_Area)**		**Treatment2 – Corrective (Eb_Bill)**	
	HIGH	**LOW**	**HIGH**	**LOW**
Existing WCC	127.53	89.21	139.32	98.05
AWCC	117.64	87.10	154.29	95.65
Proposed CWICC	114.38	82.83	159.09	98.61
ICWCC	113.15	81.16	160.54	100.12
MMT (20%)	87.17	68.77	162.08	130.27

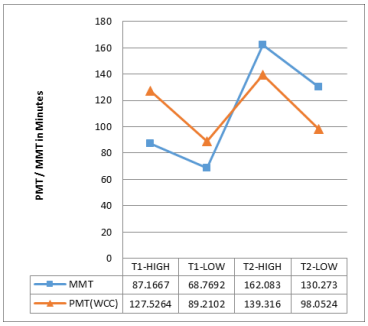

Fig 1. Comparison of PMT and MMT for WCC

The graphical comparisons between Predictive Maintenance Time (PMT) and Mean Maintenance Time (MMT) clearly indicate that the proposed metrics, CWICC and ICWCC, produce predictions that are consistently closer to the actual maintenance time than the existing metrics, WCC and AWCC. In the Figures 1 and 2, larger deviations between PMT and MMT are observed for WCC and AWCC, reflecting lower prediction accuracy. Figures 3 and 4 show that CWICC and ICWCC yield PMT values that closely follow the MMT trend, indicating improved alignment. The Figure 5 further reinforces this observation by directly comparing all metrics, where CWICC and ICWCC demonstrate minimal deviation from MMT across

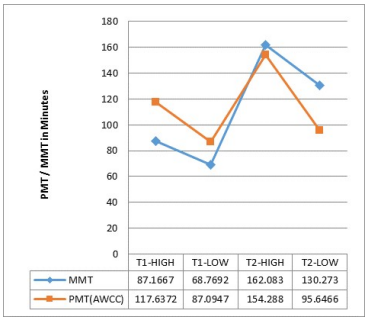

Fig 2. Comparison of PMT and MMT for AWCC

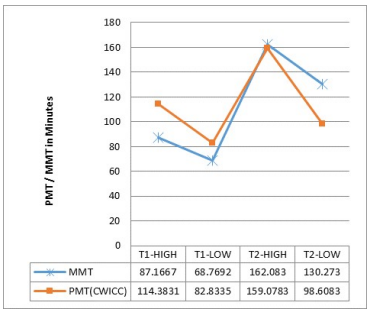

Fig 3. Comparison of PMT and MMT for CWICC

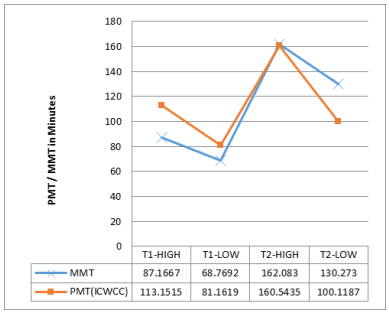

Fig 4. Comparison of PMT and MMT for ICWCC

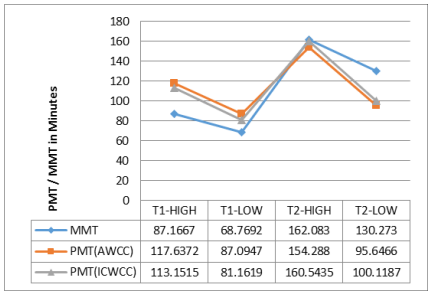

Fig 5. Comparison of PMT of AWCC, CWICC and ICWCC with MMT

different cases. This consistent closeness suggests that the proposed metrics have higher predictive accuracy, better stability, and stronger generalization capability. Overall, the charts confirm that CWICC and ICWCC are more effective in estimating software maintenance effort than traditional object-oriented metrics.

Collectively, these results indicate that CWICC and ICWCC provide a more robust representation of cognitive complexity. They not only capture structural characteristics of code more effectively but also serve as superior predictors of maintenance effort. Therefore, it can be concluded that the proposed metrics outperform the traditional metrics WCC and AWCC in terms of reliability, predictive capability, and practical applicability in software maintenance assessment

4 Conclusion

This research paper establishes that the proposed cognitive complexity metrics, CWICC and ICWCC, significantly enhance the prediction of software maintenance effort compared to traditional metrics such as WCC and AWCC. Quantitative evidence shows that predicted maintenance times using CWICC (159.09) and ICWCC (160.54) closely approximate the observed Mean Maintenance Time (162.08), whereas WCC (139.32) and AWCC (154.29) exhibit larger deviations. Similar trends across multiple cases, supported by stronger correlations and higher coefficients of determination (R^2), confirm the superior predictive accuracy, stability, and explanatory power of the proposed metrics.

The novelty of this work lies in demonstrating that CWICC and ICWCC provide a more precise estimation of maintenance effort by achieving closer alignment between predicted and actual maintenance time. These metrics effectively capture cognitive and structural complexity, making them reliable indicators for maintenance effort assessment. From a practical perspective, it is recommended that software practitioners integrate CWICC and ICWCC into effort estimation models and maintenance planning tools to improve resource allocation, cost estimation, and project scheduling. Future work can extend this research by validating the proposed metrics on larger and heterogeneous datasets, exploring their applicability across diverse programming paradigms, and incorporating advanced machine learning techniques to further enhance prediction accuracy and generalization capability.

5 Acknowledgements

The author gratefully acknowledges the Department of Science and Technology (DST-FIST), Government of India, for providing financial support toward the development of infrastructure at St. Joseph's College (Autonomous), Tiruchirappalli – 620 002.

References

1) Lavazza L. An empirical study on software understandability and its dependence on code characteristics. *Empirical Software Engineering*. 2023;28(155):1–24. https://doi.org/10.1007/s10664-023-10396-7.
2) Hussien N. Practical implementation and analysis of software metrics impact on maintainability in open-source systems. *AlKadhim Journal for Computer Science*. 2025;3(4):87–99. https://doi.org/10.61710/kjcs.v3i4.135.
3) Vescan A, Daniel BA. Software maintainability prediction based on change metric using neural network models. *Engineering Applications of Artificial Intelligence*. 2025;144:1–12. https://doi.org/10.1016/j.engappai.2025.110032.
4) Syed-Mohamad SM, Ngah A, Ali AM, Keikhosrokiani P. Measuring software maintainability: An analysis of evolving metrics across development paradigms. *Journal of Advanced Research in Applied Sciences and Engineering Technology*. 2026;63(2):181–195. https://doi.org/10.37934/araset.63.2.181195.
5) Lavazza L, Abualkishik AZ, Liu G, Morasca S. An empirical evaluation of the Cognitive Complexity measure as a predictor of code understandability. *Journal of Systems and Software*. 2023;197:111561. https://doi.org/10.1016/j.jss.2022.111561.
6) Bisht B, Gandhi P. Empirical validation of variable method interaction cohesion metric (VMICM) for enhancing reusability of object-oriented software. *International Journal of Electrical and Computer Engineering Systems*. 2023;14(10):1097–1105. https://doi.org/10.32985/ijeces.14.10.2.
7) Sreeramkumar T, Karthika OR, Jayapratha C, Kumar JNA, Govindaprabhu GB. The IntelliEstimator: Estimating maintenance cost and prediction of software quality, reliability, and maintenance using stacking RFCXGB classifier. *International Research Journal of Multidisciplinary Scope*. 2025;6(2):710–737. https://doi.org/10.47857/irjms.2025.v06i02.03365.
8) Manju, Bhatia PK. Maintainability model of object-oriented software based on fuzzy logic approach. *International Journal of Engineering Science Invention*. 2019;8(5):76–83. https://doi.org/10.51584/ijrias.2019.v08i05.76.
9) Ouellet A, Badri M. Combining object-oriented metrics and centrality measures to predict faults in object-oriented software: An empirical validation. *Journal of Software: Evolution and Process*. 2023;36(4):1–37. https://doi.org/10.1002/smr.2548.
10) Filo TGS, Bigonha MAS, Ferreira KAM. Evaluating thresholds for object-oriented software metrics. *Journal of the Brazilian Computer Society*. 2024;30(1):315–346. https://doi.org/10.5753/jbcs.2024.3373.
11) Rochimah S, Hadiningrum TR, Mardiana BD, Siahaan DO, Akbar RJ. A maintainability framework to ensure the software quality in object-oriented programming. *IEEE Access*. 2025;13:195796–195821. https://doi.org/10.1109/ACCESS.2025.3633265.
12) Shukla S, Kumar S. Study of learning techniques for effort estimation in object-oriented software development. *IEEE Transactions on Engineering Management*. 2024;71:4602–4618. https://doi.org/10.1109/TEM.2022.3217570.
13) Yenduri G, Veeranjaneyulu N. An empirical analysis of software maintainability metrics: Object-oriented approach versus traditional. *International Journal of Advanced Intelligence Paradigms*. 2025;30(5):419–432. https://doi.org/10.1504/ijaip.2025.149746.

14) Mei Y, Rong Y, Liu G, Yang, Lu H. Deriving thresholds of object-oriented metrics to predict defect-proneness of classes: A large-scale meta-analysis. *International Journal of Software Engineering and Knowledge Engineering*. 2023;33(5):651–695. https://doi.org/10.1142/S0218194023500110.
15) Ofem P, Isong B, Lugayizi F. Metrics for evaluating and improving transparency in software engineering: An empirical study and improvement model. *SN Computer Science*. 2024;5(8):1–35. https://doi.org/10.1007/s42979-024-03471-3.
16) Hericko T, Sumak B. Exploring maintainability index variants for software maintainability measurement in object-oriented systems. *Applied Sciences*. 2023;13:2972–3008. https://doi.org/10.3390/app13052972.
17) Devi A, Charaya S, Maan M. Software maintainability prediction for object-oriented systems using deep learning. *Industrial Engineering Journal*. 2023;15(11):5–8. Available from: https://iejournal.in/assets/pdf/11)%20IEJ%20Nov%202023-5-8.pdf.
18) Jha S, Kumar R, Son LH. Deep learning approach for software maintainability metrics prediction. *IEEE Access*. 2019;7:61840–61855. https://doi.org/10.1109/ACCESS.2019.2913349.
19) Maheswaran K. Validating object-oriented cognitive complexity metrics using Briand's properties. *International Journal of Research and Innovation in Applied Sciences*. 2026;11(2):1512–1524. https://dx.doi.org/10.51584/IJRIAS.2026.110200141.
20) Maheswaran K, Aloysius A. Cognitive weighted inherited class complexity metric. *Procedia Computer Science*. 2018;125:297–304. https://doi.org/10.1016/j.procs.2017.12.040.
21) Maheswaran K, Aloysius A. An interface-based cognitive weighted class complexity measure for object-oriented design. *International Journal of Pure and Applied Mathematics*. 2018;118(18):2771–2778. Available from: https://acadpubl.eu/jsi/2018-118-8/articles/18c/57.pdf.