IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 2, February 2026

DOI: 10.17148/IJARCCE.2026.15244

Assessing Object-Oriented Cognitive Complexity Metrics Using Abreu's Criteria

Dr. K. Maheswaran

Assistant Professor, Department of Computer Science, St. Joseph's College (Autonomous),

Affiliated to Bharathidasan University, Tiruchirappalli, Tamil Nadu, India - 620 002.

Abstract: Software complexity metrics provide quantitative mechanisms to evaluate essential quality attributes of software systems, including maintainability, testability, reusability, and overall design quality. Within Object-Oriented (OO) design, several metrics have been proposed to quantify structural and cognitive aspects of software complexity. Among these, Cognitive Weighted Inheritance Class Complexity (CWICC) and Interface-Based Cognitive Weighted Class Complexity (ICWCC) have been introduced to measure the cognitive burden associated with inheritance hierarchies and interface-oriented architectural constructs. This study presents a focused theoretical validation of both CWICC and ICWCC metrics using Abreu's validation criteria, a well-recognized framework for assessing the soundness and appropriateness of object-oriented design metrics. The validation process systematically examines whether these metrics satisfy Abreu's established properties for meaningful software measurement, including consistency, monotonicity, and proper representation of design characteristics. By evaluating CWICC and ICWCC against these criteria, the research aims to establish their theoretical robustness, strengthen their measurement credibility, and support their applicability in assessing cognitive complexity within object-oriented software systems.

Keywords: Complexity, Cognitive, Object-oriented metrics, Abreu's validation criteria, Metric validation.

I. INTRODUCTION

As software systems grow in size and sophistication, the challenge of managing design complexity becomes increasingly significant. Object-oriented (OO) software development, widely adopted due to its modularity and abstraction capabilities, introduces structural constructs such as inheritance, polymorphism, and interface interactions that inherently influence design complexity. Understanding and quantifying this complexity is essential because it affects maintainability, testability, reusability, and overall system quality. Software metrics play a critical role in offering quantitative measures that help researchers and industry practitioners make informed decisions about software design and evolution. The pioneering work of Chidamber and Kemerer, who proposed a suite of OO design metrics, the research community has continued to develop and refine metrics that capture various dimensions of OO complexity. However, many traditional metrics focus primarily on structural properties such as coupling, cohesion, and inheritance depth without explicitly addressing the cognitive effort required for human comprehension. Developing metrics that incorporate human understanding is important because high cognitive load can lead to increased maintenance effort, higher defect rates, and slowed development cycles. In recent years, there has been a renewed emphasis on cognitive aspects of complexity measurement. Empirical studies have investigated how cognitive complexity correlates with understandability and other quality attributes, highlighting the need to go beyond classical structural measures alone (Muñoz Barón, Wyrich & Wagner, 2020; 2023). For instance, research on cognitive complexity shows how traditional measures compare with newer cognitive metrics regarding code understandability and maintenance effort. These studies underscore that metrics sensitive to human comprehension may offer more meaningful insights for quality assessment and tool support in modern software engineering contexts.

Cognitive Weighted Inheritance Class Complexity (CWICC) and Interface-Based Cognitive Weighted Class Complexity (ICWCC) were introduced to capture the cognitive burden stemming from inheritance hierarchies and interface-based abstractions in OO designs. CWICC aims to quantify the mental effort associated with understanding inherited methods and hierarchical depth, while ICWCC targets complexity introduced by interface interactions and their implementations. These metrics augment structural evaluation by considering cognitive weights, making them potentially more reflective of real comprehension challenges in complex OO systems. The mere proposal of a metric does not guarantee its usefulness. For any metric to be practically applicable, it must be validated rigorously to demonstrate that it reliably measures the property it purports to represent. Theoretical validation frameworks, such as the criteria proposed by Abreu, define formal properties that a sound metric should satisfy. These properties include non-negativity, monotonicity, proportional sensitivity to design changes, and interpretability. Evaluating CWICC and ICWCC against Abreu's criteria allows researchers to confirm that these measures possess sound mathematical foundations and behave consistently under defined transformations, which is essential before empirical or industrial application.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 2, February 2026
DOI: 10.17148/IJARCCE.2026.15244

The recent research trends show an ongoing effort to enhance complexity measurement through new cognitive and experience-aware models, further demonstrating the active interest in this area. Newly proposed metrics that incorporate programmer experience and human-centric factors illustrate the continued evolution of complexity measurement toward more comprehensive and realistic representations of software quality. This Research focuses on the theoretical validation of CWICC and ICWCC using Abreu's validation criteria. By systematically analyzing whether these metrics satisfy established theoretical properties, the research aims to reinforce the conceptual soundness of these cognitive complexity measures and support their relevance for object-oriented software quality assessment.

II. REVIEW OF LITERATURE

Software complexity assessment has remained a central research focus within software engineering for several decades. Cognitive complexity metrics aim to measure the mental effort required to understand the software maintenance. Recent years have seen a surge of interest in software complexity measurement, particularly in frameworks that extend beyond traditional size- or control-flow-based metrics to capture cognitive and structural aspects of software design. A growing body of research indicates that incorporating human-centric perspectives (e.g., understandability, perceived difficulty) into complexity measurement provides additional insights into software quality beyond classical structural metrics alone [1, 2]. A foundational extension of complexity measurement is Cognitive Complexity, designed to reflect code understandability rather than control-flow paths. Cognitive Complexity has been empirically validated on large datasets of comprehension tasks and subjective developer assessments, showing strong correlations with developer effort and perceived difficulty [3].

Empirical evaluation of Cognitive Complexity and traditional metrics (e.g., McCabe's Cyclomatic Complexity) has continued, with several experiments demonstrating that combinations of metrics often perform better than any single metric in predicting code understandability or fault proneness [3][4][11]. Lavazza *et al.* reported that although both structural and cognitive metrics contribute to understandability prediction, integrating additional information (e.g., change and process factors) improves predictive accuracy [2]. Studies on code reviews further emphasize the utility of complexity metrics in practice. Tools that surface complexity-related hotspots assist reviewers in identifying parts of the code that are difficult to maintain or understand, enhancing sustainability and review effectiveness [10][12]. Recent research has pursued diverse extensions and alternatives to conventional metrics. Machine learning-enhanced approaches that incorporate new metrics based on class complexity, coupling, and cohesion have shown improved performance in fault prediction models compared to classical metrics alone [5][9][13]. Other work has proposed cognitive complexity metrics for evaluating design characteristics of microservices architectures, demonstrating that cognitive measures are extensible beyond traditional OO code to system-level design assessment [8][14]. From a structural standpoint, hybrid complexity measures that combine cyclomatic and other structural weights have been introduced to more effectively reflect software quality in automated defect prediction contexts [4][15].

Large-scale empirical analyses of static code metrics have highlighted the challenges, redundancies, and inconsistencies in existing metric standards. These studies motivate research on metric standardization and the construction of robust measurement frameworks that can generalize across millions of open-source software projects [9][16]. Esposito *et al.* found that early-career developers' perceived understandability of code with varying levels of complexity only modestly correlated with common metrics, indicating that additional cognitive factors should be incorporated into measurement models [6]. Complementing human-oriented studies, research in algorithmic metrics is progressing with new rule-based and machine learning techniques to compute or refine complexity measures for educational and practical scenarios. For instance, novel rule-based algorithms have been proposed for measuring software complexity with enhanced precision in understanding and learning contexts [7]. Alharthi presented a systematic procedure for validating inheritance metrics from the QMOOD model using Weyuker's nine properties. The analysis showed that most inheritance metrics satisfy Weyuker's axioms, confirming their theoretical validity and suitability [17].

Sabharwal introduced novel coupling metrics, subsequently validating them through rigorous application of Briand's theoretical framework [18]. Singh proposed an innovative structural complexity metric, demonstrating its validity through comprehensive evaluation against Briand's established properties [19]. Luhaybi et al. conducted a thorough validation of Li's inheritance metrics by systematically applying Briand's theoretical properties [20]. Collectively, these works reflect a trend toward nuanced, multi-faceted complexity measurement, blending traditional structural metrics, cognitive perspectives, machine learning insights, and practical validation to improve software design assessment and engineering practice. Despite significant advances in object-oriented (OO) complexity measurement, the need for theoretically validated class-level metrics remains evident. Several theoretical validation frameworks have been proposed in the literature, notably Weyuker's properties and Briand's property-based criteria. In addition to these established

DOI: 10.17148/IJARCCE.2026.15244

approaches, this research focuses on another recognized theoretical validation framework Abreu's criteria, proposed by Luís Abreu—to rigorously evaluate and validate the CWICC and ICWCC metrics.

III. FORMAL DEFINITIONS OF PROPOSED COGNITIVE COMPLEXITY METRICS

Assessing software complexity has long been a primary focus in the field of software engineering, as it significantly influences program understandability, maintainability, reliability, and testing efforts. Traditional complexity metrics, such as lines of code, cyclomatic complexity, and coupling/cohesion measures, provide useful insights but often fail to account for the unique complexities introduced by object-oriented (OO) paradigms. In OO systems, elements like attributes, methods, inheritance, and interfaces contribute significantly to the cognitive load that developers experience when trying to understand, extend, or reuse classes. Consequently, there is a need for metrics that not only capture structural characteristics but also take into account the cognitive effort required to process these elements. To fill this void, Maheswaran et al. introduced two innovative metrics that quantify OO class complexity by assigning cognitive weights to its structural components. The metrics are the Cognitive Weighted Inheritance Class Complexity (CWICC) and the Interface-Based Cognitive Weighted Class Complexity (ICWCC) [21, 22].

Both metrics aim to evaluate complexity beyond mere size-based measurements, effectively capturing the mental workload associated with more intricate inheritance hierarchies and the use of interfaces. 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).

$$\mathrm{CWICC} = \sum_{i=1}^{p} \mathrm{AC_i} + \sum_{j=1}^{q} \mathrm{MC_j} + \sum_{k=1}^{r} \mathrm{IICC_k} \qquad \ldots\ldots(1)$$

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)

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

The terms PDT, DDT, and UDDT represent distinct categories of data type attributes. Their associated cognitive weights—denoted as *Wb*, *Wd*, and *Wu*, respectively fall within a scale ranging from 1 to 3, where *Wb* corresponds to the lowest level of cognitive effort and *Wu* to the highest. Complexity of the method is calculated based on the work of Wang [23] using the equation (3) provided.

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

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).

$$\mathrm{IICC} = \mathrm{CDC} + \sum_{k=1}^{s} \mathrm{RMC_k} + \sum_{i=1}^{t} \mathrm{RAC}_i \qquad .....(4)$$

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.

$$CDC = DC * (CW_l) \qquad .....(5)$$

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 I.

TABLE I COGNITIVE WEIGHT FOR DIFFERENT INHERITANCE LEVELS

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

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 2, February 2026

DOI: 10.17148/IJARCCE.2026.15244

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.

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).

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

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 II.

TABLE II COGNITIVE WEIGHTS FOR THE DIFFERENT TYPES OF INTERFACES

Interface Types	Cognitive Weights
SI	3
EI	4
NI	5

The calibrations of the eights of the above interfaces are discussed elaborately in the next section. The SI, EI, and NI denote Simple, Extended and Nested Interfaces.

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

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.

IV. THEORETICAL VALIDATION OF A METRICS AND DISCUSSION

In software engineering, particularly in software metrics validation, Weyuker's and Briand's properties are formal criteria used to evaluate the theoretical soundness and usefulness of software metrics. These properties help determine whether a metric behaves in a logical, consistent, and discriminative way, ensuring that it provides meaningful and valid measurements for software attributes such as complexity, inheritance, interface, cohesion, coupling, etc.

A) Validation using Abreu's Criteria

Weyuker's properties provide essential conditions for defining a robust complexity metric, but they are not sufficient on their own. Therefore, to establish the robustness and reliability of a metric, it must undergo further validation against additional criteria, such as Abreu's criteria. These criteria were applied to assess the CWICC and ICWCC metrics, with the results detailed in Table III. The CWICC and ICWCC are metrics designed to measure cognitive complexity with a focus on inheritance interfaces along with the existing parameters like attributes and method complexity.

Criterion_1: Metrics must have a clear, formal definition

object-oriented programming (OOP). They are formally defined in terms of cognitive weightings assigned to class inheritance hierarchies, encapsulating how inheritance impacts the mental load on a programmer. These definitions

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 2, February 2026

DOI: 10.17148/IJARCCE.2026.15244

are typically documented in metric formulation literature. Hence the metrics are formally defined with specific computation rules. Hence Criterion-1 fulfilled.

Criterion_2: Metrics that are not based on size should remain unaffected by the system's size

The CWICC and ICWCC are designed as cognitive complexity metrics focusing on class inheritance, not system size. While the total complexity of a system may grow with additional classes, these metrics aim to reflect the cognitive load of each class or inheritance structure, irrespective of the overall system size. While the cognitive load measured may increase with the system, the metrics focus on the inheritance structure, not the overall size. Hence Criterion-2 partially fulfilled.

Criterion_3: Metrics should use a consistent unit system

The CWICC and ICWCC metrics use cognitive weights as their measurement unit. These weights are applied consistently across classes to determine the cognitive load induced by inheritance. The unit of measurement (cognitive weight) is applied consistently across all entities, allowing for comparison between classes. Hence Criterion-3 fulfilled.

Criterion_4: Metrics should be derivable early in the project lifecycle

The CWICC and ICWCC rely on knowledge of inheritance structures, which typically emerge during the design phase. While they may not be obtainable during requirements gathering, they can be calculated as soon as the class inheritance structure is defined. The metrics are obtainable at the design stage but not necessarily very early in the development life cycle. Hence Criterion-4 Partially fulfilled.

Criterion_5: Metrics should be adaptable to smaller-scale systems

The CWICC and ICWCC metrics can be applied to individual classes or subsystems within a larger project, allowing them to be scalable down to smaller structures within the system. This flexibility supports evaluating cognitive load at multiple levels of abstraction. These metrics are down-scalable to individual classes and can adapt to smaller subsystems. Hence Criterion-5 satisfied.

Criterion_6: Metrics should be straightforward to calculate

The CWICC and ICWCC calculations primarily involve traversing inheritance structures and applying cognitive weights, making them computationally feasible with modest overhead. The metrics are straightforward to compute once inheritance hierarchies and cognitive weights are established. Hence Criterion-6 satisfied.

Criterion _7: Metrics should be independent of programming language

The CWICC and ICWCC are based on class inheritance and cognitive complexity, they can theoretically apply to any object-oriented programming language with inheritance support. The metrics are not specific to any programming syntax. The metrics are theoretically language-independent and can be used across languages that support inheritance. Hence Criterion-7 fulfilled.

TABLE III ABREU'S CRITERIONS AGAINST PROPOSED METRICS

Abreu's Criterion	CWICC & ICWCC Metric Status
Criterion_1	Fulfilled
Criterion_2	Partially Fulfilled
Criterion_3	Fulfilled
Criterion_4	Partially Fulfilled
Criterion_5	Fulfilled
Criterion_6	Fulfilled
Criterion_7	Fulfilled

Table III presents the validation of CWICC and ICWCC against Abreu's criteria. In this study, the metrics CWICC and ICWCC were rigorously examined using Abreu's established evaluation framework to determine their validity and reliability. The assessment focused on both their theoretical foundation and their applicability in real-world object-oriented systems. Through systematic verification, each metric was analyzed against the essential properties expected of object-oriented complexity measures, ensuring compliance with the fundamental criteria. The findings indicate that CWICC and ICWCC uniquely satisfy the minimum required characteristics of object-oriented complexity metrics as defined by Abreu's standards. This outcome confirms their formal consistency and mathematical robustness. Moreover, the measures effectively capture the cognitive influence of core object-oriented components, including attributes, methods, inheritance structures, and interfaces. By accurately reflecting how these elements contribute to class

DOI: 10.17148/IJARCCE.2026.15244

complexity, the metrics provide a comprehensive perspective on structural and cognitive load within software systems. Given these strengths, CWICC and ICWCC can be confidently applied to evaluate and compare the cognitive complexity of object-oriented designs. Their adoption can support more informed design decisions, enhance maintainability assessments, and strengthen overall quality assurance processes.

V. CONCLUSION

The comprehensive validation confirms that CWICC and ICWCC are both theoretically rigorous and practically meaningful measures of object-oriented cognitive complexity. Their adherence to established evaluation principles enhances their credibility and reliability in academic and industrial contexts. Beyond satisfying formal criteria, the metrics offer practical benefits by improving transparency in complexity evaluation and promoting better design discipline. They also provide a consistent basis for comparative studies, benchmarking, and future empirical research in software metrics. Overall, the validation process demonstrates that CWICC and ICWCC are theoretically sound, practically applicable, and reliable indicators of object-oriented cognitive complexity. Their compliance with established evaluation criteria reinforces their credibility and positions them as valuable tools for researchers and practitioners seeking systematic and meaningful complexity assessment in software engineering.

REFERENCES

[1] M. Munoz Baron, M. Wyrich, and S. Wagner, "An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability", Proc. ACM/IEEE Int. Symposium on Empirical Software Engineering and Measurement (ESEM), 2020.

[2] L. Lavazza et al., "An Empirical Study on Software Understandability and Its Dependence on Code Characteristics", Empirical Software Engineering, vol. 28, no. 155, 2023.

[3] N. Hussien, "Practical Implementation and Analysis of Software Metrics Impact on Maintainability in Open Source Systems", AlKadhim Journal for Computer Science, vol. 3, no. 4, 2025.

[4] L. D. Cernau, "Unveiling Hybrid Cyclomatic Complexity: A Comprehensive Analysis and Evaluation as an Integral Feature in Automatic Defect Prediction Models", arXiv, 2025.

[5] V. S. S. Sambaturu, "Evolution Analysis of Software Quality Metrics in an Open Source Java Project: A Case Study on TestNG", arXiv preprint, 2025.

[6] T. K. Esposito, A. Janes, T. Kilamo, and V. Lenarduzzi, "Early Career Developers' Perceptions of Code Understandability", IEEE Access, vol. 13, 2025.

[7] A. V. Gorchakov, L. A. Demidova, and P. N. Sovietov, "A Rule Based Algorithm for Measuring Software Complexity in Educational Digital Environments", Computers, vol. 13, no. 3, 2024.

[8] F. H. Vera Rivera, "Cognitive Complexity Points: A Metric to Evaluate the Design of Microservices Based Applications", Ingenieria y Competitividad, vol. 26, no. 1, 2024.

[9] S. Jin and M. Zhang, "Software Code Quality Measurement: Implications from Metric Distributions", arXiv:2307.12082v4, 2024.

[10] J. M. Willenbring, "The Utility of Complexity Metrics During Code Reviews for CSE Software Projects", Future Generation Computer Systems, vol. 160, pp. 65–75, 2024.

[11] F. Alaswad and E. Poovammal, "Software Quality Prediction Using Machine Learning", Materials Today: Proceedings, vol. 62, 2022.

[12] Y. A. Pelayo Lomeli and J. P. G. Vázquez, "Comparative Evaluation of Machine Learning Algorithms for Predicting Defects Using Object-Oriented and Software Complexity Metrics", Artificial Intelligence and Quantum Computing: Early Innovations, vol. 1, 2025.

[13] N. Medeiros, "Vulnerable Code Detection Using Software Metrics and Machine Learning", IEEE Access, vol. 8, 2020.

[14] L. D. Cernau, "A Hybrid Complexity Metric in Automatic Software Defects Prediction", Proc. 17th International Conference on Software Technologies, 2022.

[15] K. Kuszczynski and M. Walkowski, "Comparative Analysis of Open-Source Tools for Conducting Static Code Analysis", Sensors, vol. 23, no. 18, 2023.

[16] M. Alharthi, "Theoretical Validation of Inheritance Metric in QMOOD Against Weyuker's Properties", Journal of Information Processing Systems, vol. 17, pp. 1250–1263, 2021.

[17] S. Sabharwal and S. Nagpal, "Theoretical and Empirical Validation of Coupling Metrics for Object-Oriented Data Warehouse Design", Arabian Journal for Science and Engineering, vol. 43, no. 8, pp. 4119–4138, 2018.

[18] T. Singh, V. Patidar, and M. Singh, "A Novel Metric for Assessing Structural Complexity of Data Warehouse Requirements Models", Expert Systems with Applications, vol. 255, pp. 1–10, 2024.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 2, February 2026
DOI: 10.17148/IJARCCE.2026.15244

[19] A. M. A. Luhaybi and W. Aljedaibi, "Using Briand's Properties to Theoretically Validate Li Inheritance Metrics", American Academic and Scholarly Research Journal, vol. 12, no. 4, pp. 1–6, 2020.
[20] K. Maheswaran and A. Aloysius, "Cognitive Weighted Inherited Class Complexity Metric", Procedia Computer Science, vol. 125, pp. 297–304, 2018.
[21] K. Maheswaran and A. Aloysius, "An Interface-Based Cognitive Weighted Class Complexity Measure for Object-Oriented Design", International Journal of Pure and Applied Mathematics, vol. 118, no. 18, pp. 2771–2778, 2018.
[22] Y. Wang, "A New Measure of Software Complexity Based on Cognitive Weights", Canadian Journal of Electrical and Computer Engineering, vol. 28, no. 2, pp. 1–6, 2003.

