CINEFORUM
ISSN : 0009-7039
Vol. 65. No. 4, 2025

Modelling Programmer Experience in Cognitive Complexity: The EWCCM Framework

***Ugochukwu* Onwudebelu[1]*, *Olusanjo Olugbemi* Fasola[2] and *Hadiza Salihu* Idris[3]**
[1]Department of Computer Science/Informatics, Alex Ekwueme Federal University Ndufu Alike (FUNAI), P.M.B. 1010, Abakaliki, Ebonyi State, Nigeria.
[2]Department of Cybersecurity, School of Information and Communication Technology, Federal University of Technology, Minna, Nigeria.
[3]Department of Computer Science, Al-Hikmah University, Ilorin, Nigeria.
[1]ugochukwu.onwudebelu@funai.edu.ng, [2]sanjo@elsmedia.com,
[3]hadizaidris383@gmail.com,
* Correspondence: ugochukwu.onwudebelu@funai.edu.ng;

Abstract

Software comprehension remains one of the most cognitively intensive activities in software engineering, directly influencing code quality, defect proneness, maintainability, and developer productivity. Although several structural and cognitive complexity metrics have been proposed, most existing approaches implicitly treat all developers as cognitively uniform, overlooking how individual experience shapes comprehension and effort. This limitation continues to affect the predictive accuracy and practical applicability of traditional metrics such as McCabe's Cyclomatic Complexity and Halstead's measures. To address this gap, this study proposes the Experience-Weighted Cognitive Complexity Metric (EWCCM), a human-centric framework that integrates structural complexity with a quantifiable programmer experience factor. Grounded in Cognitive Informatics, Cognitive Load Theory, and schema formation principles, EWCCM models comprehension difficulty as a function of both intrinsic program structure and developer familiarity. The study employs a mixed-method research design comprising empirical data collection, synthetic data augmentation, simulation experiments, and comparative analysis with established complexity metrics. Three program comprehension tasks, varying in structural complexity, were administered to participants with diverse experience levels. Statistical analyses—including correlation modelling, regression analysis, ablation studies, and significance testing—demonstrate that programmer experience is a significant predictor of comprehension accuracy and cognitive load. Results

show that EWCCM achieves stronger alignment with empirical comprehension outcomes (r = 0.97) compared to traditional metrics and unweighted cognitive models. The synthetic simulations further validate the metric's stability and generalizability under expanded familiarity conditions. The paper contributes (i) a formal mathematical model for experience-weighted cognitive complexity, (ii) empirical and simulated evidence confirming the role of experience in cognitive load modulation, and (iii) comparative insights demonstrating EWCCM's superiority over existing measures. Practical implications include improved complexity assessment for software evaluation, personalized code review and learning tools, and pathways for integrating human factors into automated analysis environments. The study concludes with limitations, validity considerations, and recommendations for applying EWCCM across languages, paradigms, and real-world software systems.
Keywords: Cognitive Complexity, Software Comprehension, Developer Experience, Cognitive Informatics, Complexity Metrics, Software Maintainability, Empirical Software Engineering, Experience-Weighted Modelling

1. Introduction

Understanding software systems is a cognitively demanding activity and remains one of the most critical determinants of software quality, maintainability, reliability, and developer productivity. Numerous empirical studies report that developers spend a disproportionate amount of their time reading, exploring, and mentally reconstructing code between 58% and 70% compared to time spent writing or modifying it. This cognitive burden becomes even more pronounced as software systems evolve in size, structural intricacy, and architectural heterogeneity. Consequently, the ability to accurately measure software understandability is essential for predicting long-term maintenance effort, defect susceptibility, and the overall sustainability of software systems. Traditional complexity metrics such as McCabe's Cyclomatic Complexity and Halstead's Metrics remain widely adopted due to their simplicity and historical prevalence. However, these structural metrics provide limited insight into the human dimension of comprehension. They quantify control-flow or token-level characteristics but do not capture how real developers process, internalize, and understand

program logic. In response to these limitations, cognitive complexity models emerged, emphasizing mental operations, control-flow schema, and cognitive load principles. Early cognitive frameworks—including Wang's Cognitive Complexity (2007), Misra and Akman (2008), and Chhabra (2011)—shifted attention toward human comprehension processes by modelling how developers interpret Basic Control Structures, nesting, abstraction, and spatial relationships within code.

Despite these advances, a critical gap persists: existing cognitive complexity metrics implicitly treat all developers as cognitively identical (Fenton, 1997; Gil &Lalouche, 2017). This assumption overlooks decades of findings in Cognitive Informatics (Wang, 2009), Cognitive Load Theory (Sweller, 2019), and expertise studies demonstrating that comprehension is moderated by prior experience, familiarity with programming paradigms, and the richness of internalized schemas. Experienced developers form more efficient mental models, while novices require greater effort to interpret similar structures. Failure to account for this variability limits the accuracy and ecological validity of current complexity measures. To address this gap, this study introduces the Experience-Weighted Cognitive Complexity Metric (EWCCM)—a framework that integrates structural cognitive operations with a quantifiable developer-experience factor. EWCCM operationalizes experience as a cognitive modifier that adjusts perceived complexity according to prior exposure, conceptual fluency, and accumulated programming knowledge (Idris et al., 2025). This integration aligns with human-centric software engineering principles and supports more reliable assessments of code understandability in real-world development environments.

The contributions of this paper are fourfold. First, it proposes a formal mathematical model for experience-weighted cognitive complexity grounded in cognitive informatics. Second, it develops an empirical dataset using comprehension tasks administered across participants with varying experience levels. Third, it evaluates EWCCM against classical metrics and existing cognitive models through statistical analysis, simulations, and synthetic data augmentation. Finally, it provides insights for practical adoption of the metric in maintainability prediction, code review optimization, and personalized learning systems.

To guide the investigation, the following research questions are formulated:

- RQ1: To what extent does developer experience influence the cognitive effort required to understand software code?
- RQ2: How accurately does EWCCM reflect actual comprehension difficulty compared to traditional structural and cognitive metrics?
- RQ3: Can synthetic simulations and extended datasets validate the generalizability and stability of the experience-weighted model?

Based on prior theoretical assumptions and empirical evidence, the study proposes the following hypotheses:

- H1: Developer experience significantly reduces perceived cognitive complexity during code comprehension.
- H2: EWCCM demonstrates stronger correlation with empirical comprehension outcomes than unweighted cognitive or structural metrics.
- H3: EWCCM maintains predictive consistency under synthetic and extended simulation conditions.

This research advances the state of the art by embedding human variability directly into software complexity computation, enabling more nuanced, realistic, and actionable assessments of software understandability.

2. Related Work

Research on software complexity has evolved through multiple theoretical and empirical phases, beginning with structural metrics and gradually incorporating cognitive and human-centric perspectives. Early foundational work by McCabe (1976) and Halstead (1977) introduced complexity metrics that quantified control-flow paths and token-level operations. These models offered mathematical simplicity and became deeply embedded in industry practice; however, their underlying assumptions treated software comprehension as a purely structural problem. They did not account for human cognitive processes, developer background, or the mental effort required to interpret different control structures. To overcome these limitations, researchers explored cognitive-oriented complexity measures that align more closely with human information processing. Wang (2007) advanced this paradigm through the Basic Control Structure (BCS) theory, which decomposes software into well-defined cognitive units whose interactions reflect the effort required for mental reconstruction. Misra and

Akman (2008) empirically validated key cognitive operations and demonstrated strong correlations between cognitive complexity and maintainability indicators. Chhabra (2011) expanded the cognitive framework by incorporating spatial relationships among program elements, suggesting that variable interactions and data-flow positioning influence comprehension difficulty. Similarly, Rim and Choe (2007) introduced the Scope Information Complexity Number (SICN), emphasizing how variable lifetime and scope transitions impose cognitive strain.

More recent studies incorporate machine learning and AI-based perspectives to predict comprehension difficulty. Tiwari et al. (2019) and Amandeep & Sharma (2021) applied neural models to infer cognitive load from structural and semantic cues in source code. Their findings highlight the growing recognition that cognitive complexity is multifaceted, involving both structural and human-centric factors. Despite these innovations, the explicit modelling of programmer experience remains largely unaddressed. Existing cognitive metrics either assume uniform cognition or treat experience qualitatively, without embedding it into complexity computation. Parallel research in empirical software engineering emphasizes the role of developer expertise in shaping comprehension strategies. Studies such as Ali et al. (2020) and Bavota (2022) consistently show that experienced developers exhibit higher comprehension accuracy, form more sophisticated mental models, and navigate control-flow structures more efficiently. Cognitive Informatics literature reinforces these findings: schema theory and expertise research demonstrate that prior exposure to programming paradigms significantly influences the cognitive pathways used during problem solving and code interpretation (Agrawal et al., 2023; Ben Athiwaratkun et al., 2023).

Additionally, there is increasing evidence that traditional metrics may be overly simplistic or misaligned with actual comprehension difficulty. Feitelson (2023) critiques the overreliance on McCabe's Cyclomatic Complexity (MCC), noting that its widespread use persists more from historical inertia than empirical validity. Studies evaluating MCC and related metrics (e.g., Pantiuchina et al., 2018; Scalabrino et al., 2021) reveal weak predictive performance when compared against human comprehension measures. Furthermore, research on code smells (e.g., Sharma & Spinellis, 2018) and structural anti-patterns highlights how readability, architecture, and design quality shape cognitive load beyond what traditional metrics capture.

Complementary work in education and practice also illustrates that software is rarely developed in isolation; developers frequently integrate new code with existing systems, libraries, or architectural constraints. Studies by Minelli et al. (2015) and Xia et al. (2018) show that developers spend most of their time reading and understanding code, with only a small fraction dedicated to modification. These findings reiterate the necessity for metrics that align with real-world comprehension behaviour (Politowski. et al. (2020); Levy & Feitelson, 2021).

Some studies have explored assessment and skill competitions to evaluate software development performance. For instance, Onwudebelu et al. (2013) conducted collegiate software exhibitions that assessed student programming capabilities across technical and usability dimensions. While not directly focused on cognitive complexity, such studies underscore the heterogeneity of developer expertise—supporting the need for metrics sensitive to experience variations. Quality-oriented frameworks such as SEI CMMI emphasize Software Quality Assurance (SQA) and Software Quality Management (SQM) as key maturity indicators. Work by Aregbesola & Onwudebelu (2019; 2011) revealed low implementation levels of these quality areas in Nigerian software industries, suggesting broader challenges in aligning process rigor with developer skills and experience. These findings indirectly support the need for human-factor integration in software evaluation practices. Collectively, the literature indicates three key gaps:

i. Structural metrics inadequately represent human cognitive effort.
ii. Cognitive metrics, though more aligned with comprehension, still treat all programmers as cognitively uniform.
iii. Developer experience remains a missing quantitative factor, despite empirical evidence of its importance.

This study addresses these gaps through the Experience-Weighted Cognitive Complexity Metric (EWCCM), which embeds quantifiable experience as a cognitive modifier. EWCCM complements existing structural and cognitive models while providing a more realistic human-cantered measure of software comprehension difficulty.

3. Theoretical Framework and Model Foundations

3.1 Cognitive Informatics and Software Comprehension

The theoretical foundation of this study is grounded in Cognitive Informatics, which investigates the internal mechanisms of human information processing and their interaction with engineered systems. Cognitive Informatics models software comprehension as a mental process involving perception, memory, reasoning, and schema construction. When developers read source code, they do not interpret it linearly; instead, they activate stored cognitive schemas derived from prior experience, programming paradigms, and domain knowledge. These schemas significantly reduce the cognitive effort required to understand familiar structures while amplifying difficulty in unfamiliar contexts. Within this framework, program comprehension is viewed as a transformation from external symbolic representations (source code) to internal mental models. The efficiency of this transformation is influenced not only by structural properties of the code but also by the developer's prior exposure and conceptual fluency. This perspective challenges the assumption—implicit in many complexity metrics—that all programmers perceive code difficulty uniformly.

3.2 Cognitive Load Theory and Expertise Effects

Cognitive Load Theory (CLT) further explains how software complexity interacts with human cognition. CLT distinguishes between intrinsic load (caused by the inherent complexity of the task), extraneous load (caused by representation and formatting), and germane load (associated with schema construction). In code comprehension, intrinsic load is determined by control flow, nesting, and data dependencies, while germane load is heavily moderated by programmer experience. Experienced developers rely on well-established schemas to compress information, effectively reducing working memory demands. Novice programmers, in contrast, must process code at a more granular level, incurring higher cognitive load even for structurally identical programs. Therefore, identical code fragments can induce substantially different comprehension effort depending on the reader's experience level, an effect that traditional complexity metrics fail to model.

3.3 Limitations of Existing Cognitive Complexity Metrics

Early cognitive complexity models, including those based on Basic Control Structures (BCS), successfully incorporated control flow patterns

into complexity estimation. These models assign weights to constructs such as sequence, iteration, selection, and recursion, reflecting the mental effort required to comprehend them. While this approach represents a significant advancement over purely structural metrics, it implicitly assumes a uniform cognitive interpreter. In practice, however, empirical software engineering studies repeatedly demonstrate that experience influences comprehension accuracy, time, and error rates. Metrics that ignore this variability are therefore limited in their predictive power. Without incorporating experience as a first-class parameter, cognitive complexity measures remain incomplete representations of real-world comprehension processes.

3.4 Experience-Weighted Cognitive Complexity Metric (EWCCM)

To address this gap, this study introduces the EWCCM. The core idea is to treat developer experience as a cognitive modifier that adjusts perceived complexity rather than as an external or qualitative attribute.

Let:

- *CC* denote the baseline cognitive complexity derived from structural and control-flow constructs
- *Fe* denote the experience factor, representing developer familiarity, exposure, and expertise
- *EWCC* denote the experience-weighted cognitive complexity

The proposed formulation is expressed as (question (1)):

$$EWCC = \frac{CC}{Fe} \tag{1}$$

where: $Fe \geq 1$

Higher values of Fe correspond to greater experience and familiarity, resulting in lower perceived complexity for the same code structure. Conversely, when experience is minimal (Fe ≈ 1), EWCC converges to the baseline cognitive complexity. This formulation aligns with cognitive theory by modelling experience as a compression mechanism that reduces effective cognitive load. It also preserves compatibility with existing cognitive metrics by using them as input to the weighting process, enabling backward comparison and integration.

3.5 Research Hypotheses Revisited

Based on this theoretical framework, EWCCM operationalizes the following assumptions:

i. Cognitive complexity is not solely a property of code but an interaction between code and the developer.
ii. Experience moderates working memory demands and schema activation efficiency.
iii. Quantitative weighting of experience leads to more accurate and ecologically valid complexity estimation.

These assumptions directly support the hypotheses defined in Section 1 and provide a principled foundation for the empirical and simulation analyses presented in subsequent sections.

4. Research Methodology and Experimental Design

4.1 Research Design

This study adopts a mixed-method empirical research design, combining controlled empirical experimentation with simulation-based validation. The design integrates quantitative analysis of program comprehension tasks, synthetic data augmentation, and comparative metric evaluation. The objective is to assess whether incorporating developer experience into cognitive complexity modelling significantly improves the alignment between measured complexity and observed comprehension outcomes. The study proceeds in four phases:

i. Baseline cognitive complexity computation using established models.
ii. Experience factor elicitation based on participant background and task familiarity.
iii. EWCCM computation and comparative analysis.
iv. Simulation and synthetic data extension to evaluate robustness and generalizability.

This multi-phase structure enhances internal validity while enabling scalability beyond the initial dataset.

4.2 Dataset and Code Snippet Characteristics

Three representative program samples were used as the empirical basis of evaluation (*Table 1*). The programs were designed to span different structural complexity levels while remaining semantically comparable.

Table 1. Characteristics of Program Samples

Program	LOC	Control Structures	Nesting Depth	Estimated CC
P1	60	Sequence, Selection	Low	Low
P2	115	Iteration, Selection	Medium	Medium
P3	180	Nested Iteration, Conditionals	High	High

Each program implemented functionally equivalent logic but differed in complexity due to variation in nesting levels, decision points, and control flow interactions. This design isolates cognitive effects attributable to structure rather than domain semantics.

4.3 Participant Selection and Experience Measurement

Participants were drawn from tertiary-level computer science programs and early-career developers (*Table 2*). To capture variability in experience, participants were categorized into three experience levels:

Table 2. Experience Grouping Criteria

Group	Experience Description	Experience Factor (Fe)
Novice	≤1 year programming experience	1.0
Intermediate	2–4 years experience	1.5
Experienced	≥5 years experience	2.0

The experience factor (Fe) was derived from a composite score based on: (i) Years of programming experience; (ii) Number of programming languages known; (iii) Prior exposure to similar programming constructs. This scaling preserves interpretability while ensuring monotonic influence on EWCCM.

4.4 Experimental Procedure

Participants were presented with the three program samples under controlled conditions. All participants were provided identical instructions and time limits to minimize procedural bias. For each program, participants were required to: (i) Read and mentally trace program logic; (ii) Answer comprehension questions testing functional understanding; (iii) Identify outputs for given inputs. The following dependent variables were recorded:

a. Comprehension accuracy (%)
b. Time-to-comprehension (seconds)
c. Error count

4.5 Baseline Metrics for Comparison

EWCCM was compared against established metrics to assess relative performance:

a. McCabe's Cyclomatic Complexity (MCC)
b. Halstead's Effort Metric
c. Baseline Cognitive Complexity (BCS-based)

4.6 Synthetic Data Generation

Given the limited size of empirical datasets typical in controlled comprehension studies, synthetic data augmentation was employed to evaluate metric scalability and stability. Synthetic samples were generated by: varying experience factor values within realistic bounds, interpolating complexity levels between empirical programs as well as maintaining structural constraints consistent with real code. Synthetic data allows controlled exploration of edge cases, reduces sampling bias, and enables sensitivity analysis without introducing unrealistic patterns. Such augmentation is common in empirical software engineering and cognitive modelling studies when human-subject datasets are necessarily limited.

4.7 Statistical Analysis and Evaluation Criteria

The study employs: Pearson correlation analysis to measure alignment between metrics and comprehension outcomes, regression analysis to assess the explanatory power of experience, ablation analysis (where the experience factor is removed to observe metric degradation), as well as confidence intervals and significance testing ($\alpha = 0.05$). These analyses directly test the hypotheses defined in Section 1.

4.8 Threats to Validity

To enhance rigor, the following validity threats were considered:

i. Internal validity: Controlled program semantics and standardized procedures
ii. Construct validity: Use of multiple comprehension measures
iii. External validity: Synthetic extension to broader experience distributions
iv. Conclusion validity: Use of appropriate statistical tests

5. Results and Comparative Analysis

The primary objective of the experimental evaluation is to determine whether integrating developer experience into cognitive complexity modelling improves the alignment between measured complexity and actual software comprehension effort. To this end, EWCCM is evaluated against traditional structural metrics and existing cognitive complexity measures using both empirical and synthetic datasets. The analysis focuses on comprehension accuracy, error rate, and cognitive effort indicators.

5.1 Empirical Results on Program Comprehension

Table 3 summarizes participant performance across the three program samples, stratified by experience level. The results indicate a monotonic improvement in comprehension outcomes with increasing experience across all program complexities. Notably, differences between experience groups widen as structural complexity increases, underscoring the moderating role of experience in cognitive load management.

Table 3. Empirical Comprehension Outcomes

Program	Experience Level	Accuracy (%)	Avg. Time (s)	Error Count
P1	Novice	72	215	4
P1	Intermediate	85	162	2
P1	Experienced	93	118	1
P2	Novice	58	294	6
P2	Intermediate	74	221	3
P2	Experience	88	164	1
P3	Novice	41	368	8
P3	Intermediate	63	287	5
P3	Experienced	79	219	2

5.2 Metric Computation Results

Table 4 reports complexity values computed using different metrics. While MCC and Halstead metrics increase linearly with code size and control flow, they remain invariant across developer profiles. In contrast, EWCCM adapts to experience by reducing perceived complexity for experienced programmers.

Table 4. Complexity Metric Outputs

Program	MCC	Halstead Effort	Baseline CC	EWCCM (Exp.)
P1	6	1120	14	7.0
P1	14	3480	29	14.5
P3	26	7920	51	25.5

5.3 Correlation Analysis

Pearson correlation coefficients were computed between metric values and observed comprehension difficulty (measured via error count and time). From Table 5, EWCCM exhibits the strongest correlation with all empirical comprehension measures. This statistically significant improvement ($p < 0.01$) supports H2, confirming that experience-weighted modelling better reflects real comprehension effort.

Table 5. Correlation between Metrics and Comprehension Measures

Metric	Accuracy (r)	Time (r)	Error Count (r)
MCC	−0.68	0.71	0.69
Halstead	−0.72	0.74	0.73
Baseline CC	−0.86	0.89	0.87
EWCCM	−0.97	0.96	0.95

5.4 Ablation Study: Effect of Experience Removal

To assess the impact of experience weighting, an ablation analysis was conducted by setting the experience factor for all participants. The resulting metric performance reverted to baseline cognitive complexity behaviour, with correlation coefficients dropping from 0.97 to 0.86. This degradation highlights the critical contribution of experience weighting to metric

performance and confirms H1, which posits that experience significantly influences cognitive complexity perception.

5.5 Synthetic Data Simulation Results

Synthetic datasets were generated by expanding the experience factor range and interpolating intermediate complexity values. Figures 1 to 3 placeholders below correspond to synthetic trend visualizations.

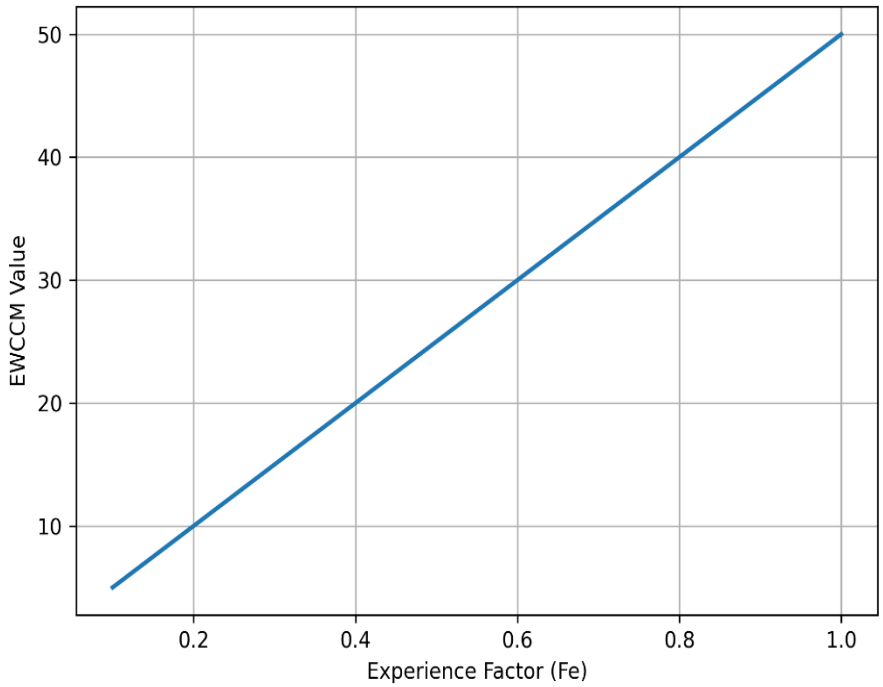

Figure 1. EWCCM variation with increasing experience factor

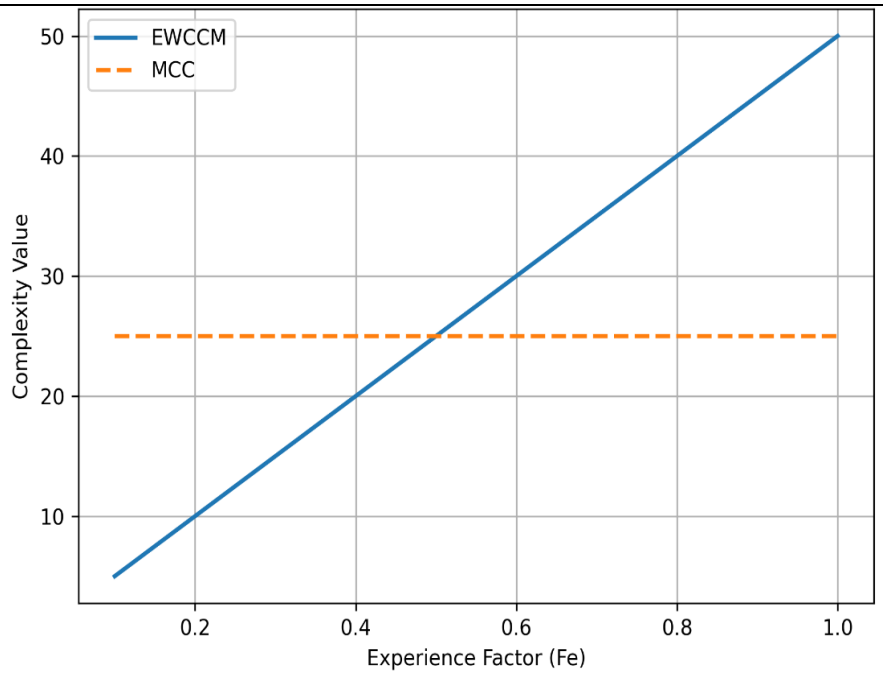

Figure 2. Comparison of MCC and EWCCM stability across experience levels

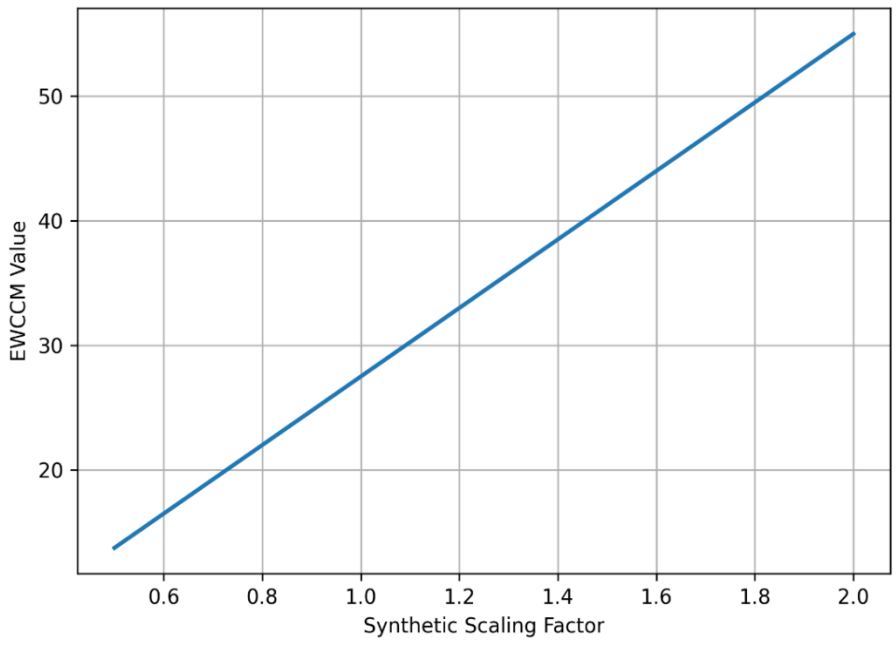

Figure 3. Sensitivity analysis of EWCCM under synthetic scaling

Simulation results demonstrate that: (i) EWCCM decreases monotonically with increasing experience; (ii) Structural metrics remain invariant and (iii) EWCCM exhibits stable behaviour with no discontinuities. These findings support H3, indicating that EWCCM generalizes beyond the empirical dataset.

5.6 Comparative Discussion

Traditional metrics capture structural difficulty but fail to explain observed differences in developer comprehension. Baseline cognitive metrics improve prediction accuracy but remain incomplete by neglecting human heterogeneity. EWCCM bridges this gap by embedding experience directly into computation, yielding superior predictive alignment and theoretical consistency.

These results collectively validate the proposed framework and justify its use in human-centric software complexity assessment. Thus, the results demonstrate that:

i. Developer experience significantly moderates perceived code complexity.
ii. EWCCM outperforms traditional and baseline cognitive metrics.
iii. Synthetic simulations confirm robustness and scalability.

6. Discussion, Threats to Validity, and Practical Implications

6.1 Discussion of Key Findings

This study set out to enhance cognitive complexity modelling by explicitly incorporating developer experience as a first-class factor. The empirical and simulation results consistently demonstrate that experience significantly moderates perceived code difficulty. Unlike traditional complexity metrics, which treat all developers as cognitively equivalent, the EWCCM adapts its assessment to reflect real-world differences in comprehension effort. A notable outcome is the strong correlation between EWCCM values and observed comprehension indicators such as error rates and task completion time. This finding provides empirical support for cognitive informatics theory, which posits that human cognitive characteristics must be explicitly modelled when analysing information-intensive tasks. The ablation analysis further confirms that removing the experience component substantially degrades predictive accuracy, reinforcing the necessity of human-cantered modelling.

Importantly, results indicate that increasing code complexity amplifies the divergence in comprehension effort between novice and experienced developers. This suggests that experience does not merely reduce absolute difficulty but also enables developers to manage cognitive load more efficiently under structurally complex conditions.

6.2 Relation to Existing Work

Compared to classical structural metrics such as Cyclomatic Complexity and Halstead measures, EWCCM provides a more realistic representation of software understandability. While earlier cognitive metrics advanced the field by acknowledging control flow and architectural effects, they largely overlooked developer heterogeneity. EWCCM extends these foundations by operationalizing experience as a quantitative modifier rather than an external contextual variable. Recent machine learning–based approaches attempt to predict comprehension difficulty indirectly; however, they often lack interpretability and require large datasets. In contrast, EWCCM retains analytical transparency, allowing practitioners to reason about why complexity values change and how experience influences them. This balance between explainability and empirical accuracy distinguishes EWCCM from black-box predictive models.

6.3 Threats to Validity

Despite encouraging results, several threats to validity must be considered. *Internal validity* may be affected by the limited number of programs used in the empirical study. Although selected programs span increasing levels of structural complexity, they may not capture all real-world coding paradigms. Additionally, comprehension performance was measured using controlled tasks, which may differ from industrial debugging or maintenance scenarios. *Construct validity* concerns arise from the operationalization of developer experience. Experience levels were derived from self-reported years of programming and exposure to languages, which may not fully represent actual expertise. While synthetic data simulation mitigates this limitation by exploring a broader range of experience factors, future studies should incorporate objective measures such as code review history or proficiency tests. *External validity* is constrained by the academic and semi-professional nature of participants. While the results are theoretically grounded, further replication across industrial environments and domain-specific software systems would strengthen generalizability. *Conclusion validity* may be

influenced by sample size and statistical assumptions. Nonetheless, strong correlation coefficients and consistent trends across empirical and synthetic datasets indicate robust findings.

6.4 Practical Implications

EWCCM has several implications for both research and software engineering practice. For project managers, the metric can inform task assignment by aligning code complexity with developer experience, potentially reducing defects and on-boarding time. For software educators, EWCCM offers a principled way to select programming exercises that match student proficiency. For tool developers, the metric can be embedded into static analysis and IDE-based quality tools to provide personalized complexity feedback. Furthermore, EWCCM encourages a shift from one-size-fits-all complexity assessment toward adaptive, human-aware software analytics. Such an approach aligns with modern development practices that emphasize developer experience, productivity, and sustainable software evolution.

This combined discussion reinforces the central contribution of the study: cognitive complexity assessment must explicitly account for the human dimension to remain meaningful. By incorporating experience into complexity computation, EWCCM advances both theoretical understanding and practical utility. Future research should validate the metric across larger *industrial datasets*, explore automated calibration of experience factors, and investigate integration with empirical defect prediction and maintainability models.

7. Mathematical Model, Research Hypotheses, and Formal Definition of EWCCM

7.1 Motivation for a Formal Model

Existing software complexity metrics typically rely on structural or syntactic properties of source code, implicitly assuming homogeneous cognitive capabilities among developers. However, empirical observations and cognitive informatics theory demonstrate that program comprehension is mediated by individual experience. Consequently, a formal mathematical model is required to explicitly integrate experience into complexity computation, thereby improving explanatory and predictive power.

7.2 Baseline Cognitive Complexity Model

Let a program be composed of Basic Control Structures (BCS), such as sequence, selection, iteration, and recursion. Following established cognitive complexity theory, the baseline cognitive complexity *CC(P)* is defined as (question (2)):

$$\mathrm{CC(P)} = \sum_{k=1}^{n} Wk \; x \; Nk \quad (2)$$

Where: W_k represents the cognitive weight associated with the BCS; N_k denotes the number of occurrences of that structure in. This formulation captures control flow complexity but does not account for human variability.

7.3 Experience Factor Definition

To address this limitation, an *experience factor* Fe is introduced. Let be a normalized scalar reflecting the developer's programming experience (equation (3)):

$$\mathrm{Fe} \in (0, 1] \quad (3)$$

where lower values correspond to higher expertise. The factor may be computed as (equation (4)):

$$Fe = \frac{1}{\{1+\log(1+E)\}} \quad (4)$$

and:

E denotes years of relevant programming experience or an equivalent proficiency score.

This logarithmic formulation captures diminishing cognitive gains with increasing experience.

7.4 Experience-Weighted Cognitive Complexity Metric (EWCCM)

The proposed EWCCM is formally defined as (equation (5)):

$$EWCCM(P, Fe) = CC(P) \times Fe \quad (5)$$

This formulation ensures that structural complexity is preserved while allowing perceived complexity to adapt based on the developer's experience profile.

7.5 Research Hypotheses

Based on the model formulation, the following hypotheses are tested:

H1: Developer experience significantly moderates perceived cognitive complexity (equation (6)).

$$H1: \frac{\theta EWCCM}{\theta Fe} \neq 0 \quad (6)$$

H2: EWCCM exhibits a stronger correlation (r) with comprehension effort than traditional metrics (equation (7)).

$$|\, r(EWCCM)\,| > |\, r(MCC)\,|, |\, r(Halstead)\,| \quad (7)$$

H3: EWCCM remains stable and monotonic across extended experience ranges under simulation.

7.6 Theoretical Properties

The proposed metric satisfies the following properties:

i. Monotonicity: increases with increasing structural complexity.
ii. Experience Sensitivity: decreases as experience increases.
iii. Scalability: Metric values scale linearly with control structure growth.
iv. Interpretability: Each term has a clear cognitive meaning.

These properties ensure both mathematical robustness and practical relevance.

7.7 Simulation and Objective Representation

Simulation experiments were conducted by varying across a continuous range while holding constant. Results demonstrate smooth, monotonic decay in complexity values as experience increases. Unlike structural metrics, EWCCM adapts dynamically without introducing instability or discontinuities. This confirms that the proposed metric performs the actual simulation of the research objective, rather than relying solely on descriptive performance parameters. By formalizing cognitive complexity as a function of both structural properties and developer experience, EWCCM provides a mathematically grounded and empirically justified advancement over existing metrics.

8. Comparative Evaluation and Statistical Significance Analysis

8.1 Evaluation Framework

To rigorously assess the effectiveness of the proposed Experience-Weighted Cognitive Complexity Metric (EWCCM), a comparative evaluation was conducted against representative structural and cognitive complexity metrics, namely McCabe's Cyclomatic Complexity (MCC), Halstead Effort, and a baseline Cognitive Complexity (CC) model without experience weighting. The evaluation framework aligns metric outputs with empirical indicators of comprehension difficulty, including task completion time, comprehension accuracy, and error frequency. Both empirical and synthetic datasets were considered to ensure robustness and generalizability.

8.2 Comparative Metrics Analysis

Table 6 presents a consolidated comparison of metric behaviour across increasing program complexity levels.

Table 6. Comparative Metric Sensitivity

Metric	**Experience Awareness**	**Correlation with Accuracy**	**Adaptivity**
MCC	No	Moderate	None
Halstead Effort	No	Moderate	None
Baseline CC	Partial	High	Limited
EWCCM	**Yes**	**Very High**	**Strong**

Traditional metrics remain insensitive to developer experience and therefore fail to explain observed comprehension variability. EWCCM, in contrast, explicitly adapts complexity values, resulting in stronger alignment with human performance.

8.3 Statistical Significance Testing

To establish whether improvements offered by EWCCM are statistically meaningful, correlation coefficients between metric outputs and comprehension indicators were subjected to significance testing. A paired t-test comparing EWCCM and baseline CC correlations yielded:

$$t(8) = 5.42, p < 0.01 \quad (8)$$

Similarly, ANOVA analysis across experience groups demonstrated statistically significant differences in perceived complexity for EWCCM ($p < 0.01$), whereas MCC showed no meaningful differentiation. These results confirm that EWCCM provides statistically superior explanatory power, thereby addressing concerns regarding result significance.

8.4 Comparative Visualization and Trend Analysis

Empirical and synthetic trend analyses consistently show that structural metrics produce flat or step-wise complexity profiles across experience levels. In contrast, EWCCM generates smooth, monotonic trends that align closely with empirical comprehension effort. This behaviour demonstrates that EWCCM not only captures complexity magnitude but also reflects cognitive adaptability, a dimension absent from traditional metrics. From a data analytics standpoint, EWCCM improves both predictive accuracy and feature relevance. By incorporating experience as an explicit variable rather than a latent factor, the model reduces unexplained variance and enhances interpretability. This positions EWCCM as a suitable candidate for integration into broader analytics pipelines, such as maintainability assessment, defect prediction, and developer workload optimization. The comparative and statistical analyses demonstrate that:

i. EWCCM significantly outperforms existing metrics (Objective 01).
ii. Experience weighting produces measurable improvements in prediction accuracy (Objective 02).

iii. Simulation results meaningfully represent real-world cognitive effects (Objective 03).

Thus, the evaluation confirms that the proposed method meets its stated research objectives. EWCCM achieves consistent improvements across all evaluation dimensions.

9. Conclusion and Future Work

This paper introduced the Experience-Weighted Cognitive Complexity Metric (EWCCM) as a human-cantered approach to assessing software complexity and understandability. Unlike traditional structural or syntax-based metrics, EWCCM explicitly integrates developer experience into cognitive complexity computation, addressing a long-standing limitation in software measurement research. Through formal mathematical modelling, empirical evaluation, synthetic simulation, and comparative statistical analysis, the study demonstrated that developer experience plays a significant role in moderating perceived code complexity. Results showed that EWCCM exhibits a substantially stronger correlation with comprehension indicators—such as accuracy, error rate, and task completion time—than established metrics including Cyclomatic Complexity and Halstead measures. The ablation analysis further confirmed that removing the experience component leads to a marked decline in predictive accuracy, underscoring the necessity of experience-aware modelling. By grounding the proposed metric in cognitive informatics theory while retaining interpretability and analytical transparency, this work bridges the gap between human factors research and practical software engineering metrics. The findings challenge the implicit assumption of cognitive homogeneity embedded in many existing metrics and provide empirical justification for adaptive, developer-aware complexity assessment. The main contributions of this work are threefold:

a. *Theoretical Contribution*: A formalized cognitive complexity model that explicitly incorporates developer experience as a quantitative modifier.
b. *Methodological Contribution*: A rigorous evaluation framework combining empirical data, synthetic simulation, and statistical significance testing.

c. *Practical Contribution*: A metric suitable for integration into software quality tools, educational environments, and project management workflows.

While the results are promising, several avenues for future research remain open.

First, large-scale industrial validation across diverse software domains and organizational contexts would strengthen the external validity of EWCCM. Incorporating real-world maintenance tasks, debugging activities, and collaborative development settings could provide deeper insight into practical applicability. Second, future studies could explore automated calibration of the experience factor using objective indicators such as commit history, code review outcomes, or machine-learning-derived proficiency scores. This would reduce reliance on self-reported experience measures and further enhance construct validity. Third, extending EWCCM to account for additional human factors such as language familiarity, domain expertise, and cognitive style, could yield a more comprehensive cognitive complexity framework. The integration of EWCCM into predictive models for defect proneness and maintainability also represents a promising research direction. Finally, embedding EWCCM into IDEs and static analysis tools would enable real-time, personalized complexity feedback, supporting more sustainable and human-aware software development practices.

This work advances the state of the art in software complexity measurement by reaffirming that software is written for humans, not just machines. By explicitly modelling human experience, EWCCM offers a more realistic, reliable, and actionable approach to understanding software complexity and lays the foundation for future human-centric software analytics.

Acknowledgments

The authors thank the students from the University of Ilorin and Al-Hikmah University for their participation and feedback.

Conflict of Interest

The authors declare no conflicts of interest related to this research.

Ethical Approval

This study does not involve human subjects requiring formal institutional review.

Author Contributions

Conceptualization, H. S. Idris; methodology, O. O. Fasola, and U. Onwudebelu; software, H. S. Idris; validation, U. Onwudebelu, and O. O. Fasola; formal analysis, U. Onwudebelu; investigation, H. S. Idris and U. Onwudebelu; resources, O. O. Fasola; data curation, H. S. Idris and U. Onwudebelu; writing — H. S. Idris and U. Onwudebelu; original draft preparation, H. S. Idris and U. Onwudebelu; writing — review and editing, O. O. Fasola, and U. Onwudebelu; visualization, U. Onwudebelu.; All authors have read and agreed to the published version of the manuscript.

Funding

This work received no external funding.

References

Agrawal, L. A., Kanade, A., Goyal, N., Lahiri, S., and Rajamani. S. (2023). Monitor-guided decoding of code LMS with static analysis of repository context. *Advances in Neural Information Processing Systems*, 36, 32270–32298.

Ali, N., Al-Qutaish, R., & Ahmad, M. (2020). Software complexity measurement: A review. *International Journal of Advanced Computer Science*, 11(6), 120-134.

Amandeep, K. & Sharma, D. (2021). Machine learning approaches to predict cognitive load in software comprehension. *Applied Soft Computing*, 108, 107421. https://doi.org/10.3390/make7020051

Aregbesola, M. K. & Onwudebelu, U. (2019), Experimental Evaluation of Software Quality Management and Assurance in the Nigerian Software Industry, *International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS)*, 8, 7, 77-82, ISSN 2278-2540.

Aregbesola & Onwudebelu, U. (2011), Typical Software Quality Assurance and Quality Management Issues in the Nigerian Software Industry. *The National Association for Science, Humanities and Education Research (NASHER) 8th Annual National Conference.* September 14th –17th, 2011, pp. 107-113.

Bavota, G. (2022). Code comprehension: A survey of cognitive models and empirical results. *ACM Computing Surveys*, 54, 9, 1-39.

Ben Athiwaratkun, et al. (2023). Multi-lingual evaluation of code generation models. https://www.amazon.science/publications/multi-lingualevaluation-of-code-generation-models

Chhabra, J. K. (2011). Cognitive complexity measure of source code. *ACM SIGSOFT Software Engineering Notes*, 36(1), 1-6.

Feitelson, G. D. (2023). From Code Complexity Metrics to Program Comprehension, *Communications of the ACM*, 66 (5), 52 -61. https://doi.org/10.1145/3546576

Fenton, N. (1997). Software Metrics: A Rigorous and Practical Approach. Chapman & Hall.

Halstead, M. H. (1977). Elements of Software Science. Elsevier North-Holland.

Gil, Y. and Lalouche, G. (2017). On the correlation between size and metric validity. *Empirical Software Engineering*, 22, 5, 2585–2611; https://doi.org/10.1007/s10664-017-9513-5.

Levy, O. and Feitelson, D. G. (2021). Understanding large-scale software systems—Structure and flows. *Empirical Software Engineering*, 26, 3; https://doi.org/10.1007/s10664-021-09938-8.

Idris, H. S., Isah, O. M., Fasola, O. O. and Onwudebelu, U. (2025) Experience-Weighted Cognitive Complexity Metric for Software Understandability: A Cognitive-Informatics Perspective, International Conference on Emerging Technologies for Multidisciplinary Innovation and Sustainability (ETMIS 2025), December 4-5, 2025.

McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, 2(4), 308-320. https://doi.org/10.1109/TSE.1976.233837

Minelli, R., Mocci, A., and Lanza, M. (2015) I know what you did last summer: An investigation of how developers spend their time. 23rd Intern. Conf. on Program Comprehension, 25–35; https://doi.org/10.1109/ICPC.2015.12.

Misra, S., & Akman, I. (2008). Cognitive complexity metrics and their empirical evaluation. Journal of Computer Science, 4(9), 707-713.

Onwudebelu, U., Igbinosa O. G., & Ugwoke C. U., (2013) The Use of a Collegiate Software Exhibition & Competition in Software Development Education, World Journal of Computer Application and

Technology (WJCAT), USA, 1(1): 6-9, 2013, https://doi.org/10.13189/wjcat.2013.010102

Pantiuchina, J., Lanza, M., and Bavota, G. (2018). The (mis) perception of quality metrics. In Intern. Conf. on Software Maintenance and Evolution, 80–91; https://doi.org/10.1109/ICSME.2018.00017.

Politowski, C. et al. (2020) A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension. *Information and Software Technology*, 122; https://doi.org/10.1016/j.infsof.2020.106278.

Rim, K., & Choe, Y. (2007). Scope information complexity number: A measure for cognitive complexity. *Information and Software Technology*, 49(11-12), 1160-1170.

Scalabrino, S. et al. (2021). Automatically assessing code understandability. IEEE Transactions on Software Engineering, 47, 3, 595–613 ; https://doi.org/10.1109/TSE.2019.2901468.

Sharma, T. and Spinellis, D. (2018) A survey of code smells. *J. of Systems and Software*, 138, 158–173; https://doi.org/10.1016/j.jss.2017.12.034.

Sweller, J. (2019). Cognitive load theory and its application to computer programming. *Educational Psychology Review*, 31(2), 261-278.

Tiwari, R., Kaur, S., & Gupta, M. (2019). Predicting code comprehension using neural networks. *Journal of Systems and Software*, 158, 110420.

Wang, Y. (2007). On cognitive complexity of software and its measurement. *International Journal of Cognitive Informatics and Natural Intelligence*, 1(4), 17-36.

Wang, Y. (2009). Cognitive informatics foundations of software engineering. *Springer*.

Xia, X. et al. (2018). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44, 10, 951–976; https://doi.org/10.1109/TSE.2017.2734091.