Static Code Analysis for Identification and Prioritization of Code Quality Remediation in a Resource-Constrained University Academic Information System

Muhammad Ridho Kurniawan Pratama[1*], Deni Utama[2], Rauhil Fahmi[3], Ali Idrus[4]

[1,2,3,4]Information Systems and Technology, Faculty of Engineering, Universitas Negeri Jakarta, Jakarta, DKI Jakarta, Indonesia
E-mail: [1*]muhammadridho@unj.ac.id, [2]deniutama@unj.ac.id, [3]rauhilfahmi@unj.ac.id, [4]aliidrus@unj.ac.id

(Received: 30 Jan 2026, revised: 24 Feb 2026, accepted: 27 Feb 2026)

Abstract

Software quality directly impacts organizational effectiveness, yet university information systems in developing nations frequently face compounded challenges, including resource constraints, limited developer expertise, and inadequate quality assurance mechanisms. Although static code analysis systematically detects software defects, translating technical findings into actionable, context-appropriate strategies remains problematic for resource-limited academic institutions. This research investigated structural quality, security vulnerabilities, and maintainability characteristics of a university academic information system, formulating contextually informed recommendations that reconcile technical assessment outcomes with institutional constraints. The case study employed four sequential phases: organizational context evaluation, quantification of code-complexity metrics, comprehensive quality assessment via static analysis, and development of context-sensitive recommendations. Correlation analysis examined relationships between structural complexity indicators and maintainability degradation. Examination of 127 PHP files revealed 9,628 quality defects, with maintainability issues accounting for 89.9%, alongside 18 critical security vulnerabilities and severe complexity, with individual methods reaching cyclomatic complexity values of 263. Strong positive correlations emerged between complexity metrics and maintainability problems ($r = 0.938$, $p < 0.001$), indicating that complexity is a reliable predictor of quality deterioration. Issue distribution patterns across all examined files suggested systemic quality degradation rather than isolated problematic modules. Findings document critical security exposures, excessive structural complexity, and pervasive maintainability deficiencies, validating significant associations between complexity and maintenance burden. The study proposes a bifurcated improvement approach encompassing tactical measures targeting immediate technical debt, including security vulnerabilities and architectural complexity, complemented by strategic interventions, providing academic information system managers with a prioritized remediation action calibrated to institutional resource constraints.

Keywords: Static Code Analysis, Software Quality, Technical Debt, University Information Systems

I. INTRODUCTION

Software quality is vital to organizational success, with poor quality causing substantial financial losses, estimated at $2.41 trillion annually in the US due to operational failures and unsuccessful projects [1]. Beyond the economic impact, software quality significantly affects workforce well-being and long-term operational sustainability [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Academic information systems in universities face additional challenges, including technical (inadequate infrastructure, outdated software), organizational (weak IT governance), financial (limited budgets), and user-related issues (resistance to change) [12], [13], [14], [15], [16]. Tools like SonarQube and PDepend facilitate systematic approaches to early defect identification and code maintainability assessment [17], [18]. However, effectively translating technical findings into practical strategies remains challenging, especially in Indonesian universities struggling with significant resource constraints [19], [20]. While existing research covers static analysis tool capabilities and technical debt measurement, it lacks guidance on applying findings in resource-constrained educational settings, a notable gap in Indonesian universities, where development must balance technical excellence with resource limitations.

To address this gap, this study systematically investigates code quality in the studied university's academic information

DOI: 10.34148/teknika.v15i1.1458

system through three research questions: examining quality issues across multiple metrics within resource-constrained environments, exploring correlations between structural complexity metrics and quality degradation in educational systems, and developing contextualized tactical and strategic recommendations addressing organizational constraints characteristic of the studied university.

Accordingly, the study follows a four-phase systematic approach to assess software quality in the university information systems under study. Phase 1 involves documenting resources, capabilities, and constraints to understand the organizational context. In Phase 2, structural attributes are quantified using code metrics through PDepend. Phase 3 uses SonarQube to identify security, reliability, and maintainability issues. Phase 4 integrates technical findings with the organizational context to develop prioritized recommendations tailored to the context. This case study contributes significantly to software engineering by empirically characterizing code quality issues in a previously underrepresented geographic and institutional area, demonstrating the impact of structural complexity on quality through correlation analysis, and highlighting the influence of organizational context on interpreting static analysis results. Additionally, it offers practical recommendations for resource-constrained institutions, ensuring a balance between technical rigor and feasibility.

II. LITERATURE REVIEW

A. Software Quality and Quality Models

Software quality reflects how well a software component meets its specified requirements and technical specifications, with attributes such as reliability, performance, efficiency, and maintainability integrated throughout the development lifecycle [21], [22], [23]. Quality is also defined as achieving design appropriateness and implementation conformity [24]. Several established quality models guide software evaluation, including ISO/IEC 9126 [25], [26], [27], [28], [29] and its successor ISO/IEC 25010 [29], [30], [31], [32], which define characteristics such as reliability, maintainability, and security; CMMI-DEV [33], [34], [35], [36], which provides comprehensive process improvement frameworks; and the SQALE method [32], [37], [38], which focuses specifically on maintainability assessment.

B. Static Code Analysis Capabilities and Adoption Challenges

Static code analysis (SCA) automatically detects bugs, security vulnerabilities, code smells, and compliance violations without executing code. Modern tools support major programming languages, integrate with CI/CD pipelines, and provide dashboard visualization for tracking technical debt [39], [40], [41], [42], and their detection accuracy improved through advanced pattern-matching and symbolic-execution techniques [43], [44]. Despite these advances, adoption barriers persist, including high false-positive rates [45], [46], [47], unclear warnings [47], [48], and scalability limitations [49]. An empirical study spanning 56 developers and 176 open-source projects found that although 66% of projects configure SCA tools, only 37% enforce their use, with 88% of enforced rules focusing solely on style conventions [50]. Furthermore, current tools lack context-aware remediation prioritization, despite evidence that severity accounts for only part of developer decision-making, with organizational policies, team composition, and application type also influencing decisions [50]. This gap underscores the need for research integrating technical static analysis with systematic assessment of organizational context to improve actionable adoption.

III. RESEARCH METHODOLOGY

A. System and Organizational Context Identification

Before conducting technical analysis, identifying the system selection and organizational context is crucial. The study focuses on the studied university's Academic Information System due to its complexity and significant role in academic processes. Organizational context analysis examines institutional constraints, workforce composition, resource availability, and operational needs, ensuring that recommendations are feasible and effective within the organizational setting.

B. Code Complexity Measurement

This phase involves quantifying structural characteristics using established software engineering metrics. Five key metrics are used: Lines of Code (LOC), Cyclomatic Complexity Number (CCN) [51], and object-oriented metrics [52] such as Weighted Methods per Class (WMC), Coupling Between Objects (CBO), and Depth of Inheritance Tree (DIT). These metrics provide a comprehensive characterization of the codebase, identifying complexity concentrations and helping assess architectural quality. Complexity metrics were extracted using PDepend, and critical methods with CCN >= 30 were identified through a researcher-defined filter of PDepend output.

C. Code Quality Assessment

Quality assessment examines security, reliability, and maintainability. Assessment performed using SonarQube Community Edition version 25.9 in Multi-Quality Rule mode, employing the default Sonar Way quality profile for PHP without modification, executed via SonarScanner CLI. Static Application Security Testing (SAST) identifies potential vulnerabilities mapped to OWASP Top 10 and Common Weakness Enumeration (CWE) classifications. The security assessment distinguished between two categories determined by SonarQube: security vulnerabilities, which represent confirmed exploitable flaws requiring mandatory remediation, and security hotspots, which denote security-sensitive code segments requiring manual expert review to determine whether contextual usage constitutes an actual risk. Reliability analysis identifies patterns that are susceptible to operational failures and runtime exceptions. Maintainability

assessment evaluates excessive complexity, code smells, duplications, and violations of coding practices. Issues are classified using a four-tier severity taxonomy. Blocker-severity issues require immediate remediation due to critical impact. High-severity issues pose substantial risks and require prioritized resolution. Medium-severity issues represent moderate concerns warranting scheduled attention. Low-severity issues indicate minor improvements with minimal immediate impact.

D. Contextualized Recommendation Formulation

The study proposes a dual-track recommendation formulation that addresses both immediate and long-term quality enhancement needs. Tactical recommendations target short-term interventions for critical issues. Strategic recommendations focus on establishing sustainable organizational practices and capabilities. This approach ensures comprehensive quality improvement by addressing both technical deficiencies and organizational constraints.

IV. RESULTS

A. System and Organizational Context

The analyzed Academic Information System at the studied university, developed circa 2017, comprises 2,438 files, including 127 spanning the Controller, Model, and Bank Integration Script layers. The technical stack consists of Nginx Web Server, PHP 5 with the CodeIgniter 3 framework, and IBM DB2 database management system, running on an AS/400 server platform. The university's ICT department manages system development and maintenance, specifically through the Information Systems division, which operates with a staffing ratio of 1 employee to 3 apprentices. Each developer serves as a full-stack engineer handling multiple systems concurrently.

B. Descriptive Statistics

The codebase analysis reveals a substantial software system with considerable scale and complexity. Table 1 summarizes the overall descriptive statistics of the analyzed codebase, including its third-party dependencies.

Table 1. Overall Codebase Statistics

Metrics	Value
Total Files	2,438
Total Classes	123
Total Methods	2,491
Total Lines of Code (LOC)	543,713
Total Non-Comment LOC (NCLOC)	380,615
Total Logical LOC (LLOC)	204,417
Total Comment Lines	163,098
Average NCLOC per File	156.1
Average Methods per Class	20.3
Average Weighted Method Count (WMC) per Class	67.7
Maximum Depth of Inheritance (DIT)	1
Average Cyclomatic Complexity (CCN) per Method	2.9

The analysis of the codebase reveals a medium-to-large-scale software system characterized by significant complexity. It consists of 2,438 files totalling 380,615 non-comment lines of code (NCLOC), yielding an average file size of 156.1 NCLOC, suggesting moderately sized files. The system comprises 123 classes with 2,491 methods, averaging 20.3 methods per class, aligning with standard object-oriented design practices. The approximately 30% comment-to-code ratio indicates adequate documentation. Despite this, the maximum depth of inheritance (DIT) is only 1, indicating a flat class hierarchy with limited use of inheritance-based abstraction in the PHP codebase.

Additionally, the average Weighted Methods per Class (WMC) is 67.7, reflecting moderate class-level complexity, while the average Cyclomatic Complexity Number (CCN) per method is 2.9, suggesting relatively simple individual methods. However, these averages might obscure concentrations of complexity that warrant further analysis.

From the 2,438 total files, 127 files spanning the Controller, Model, and Bank Integration Script layers were selected for detailed complexity and quality analysis, as these encapsulate the core custom-developed business logic. Framework core files, third-party libraries, view templates, configuration files, and static assets were excluded because they were externally maintained or did not reflect institutional development practices.

C. Complexity Analysis

The analysis explores code complexity using key metrics: NCLOC (non-comment lines of code), WMC (weighted methods per class), CCN (cyclomatic complexity number), and CBO (coupling between objects) collected using PDepend. Statistics, as shown in Figure 1, indicate variability in NCLOC, spanning from 11 to 3,551 lines, suggesting a right-skewed distribution with larger files. WMC ranges from 1 to 752, and CCN from 1 to 549, showing moderate typical complexity in the median values but revealing significant complexity peaks in the maximum values. CBO records low values, indicating limited object coupling or procedural programming. The top 10 complex files, ranked by WMC, highlight hotspots that require focus.

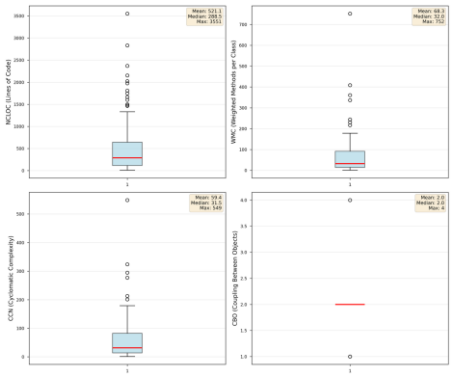

Figure 1. Complexity Metric Distribution

DOI: 10.34148/teknika.v15i1.1458

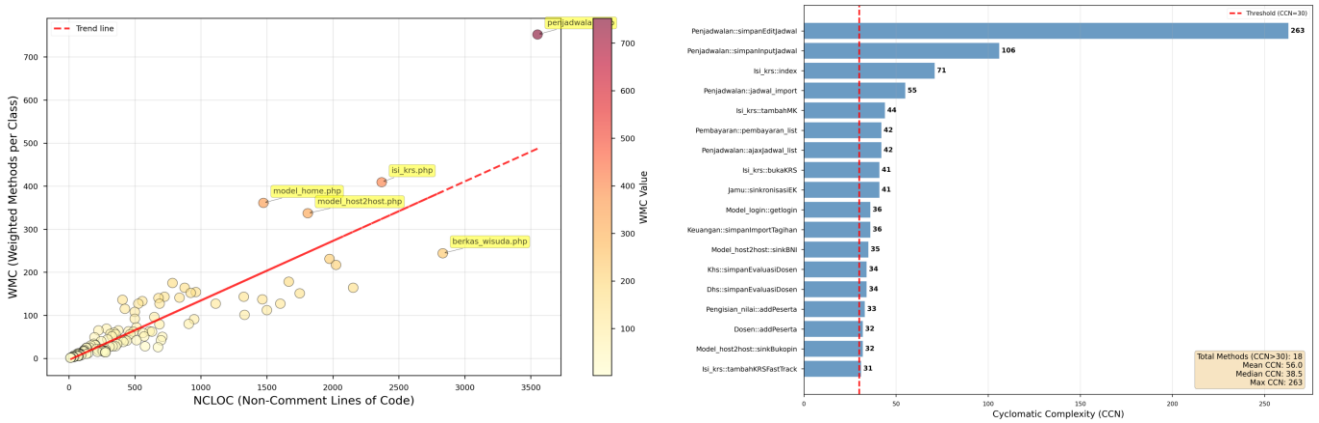

Figure 2. Structural Complexity Metrics: NCLOC-WMC Correlation and Distribution of Critically Complex Methods (CCN >= 30)

The *penjadwalan.php* controller ranks as the most complex file, with high WMC and NCLOC revealing a complexity per method far exceeding the codebase average. Similarly, *isi_krs.php* and model files such as *model_home.php* and *model_host2host.php* exhibit elevated complexity, with extensive method counts, suggesting concentrated business logic. A linear relationship between NCLOC and WMC is observed in Figure 2, indicating that complexity increases with code size.

Focusing on individual methods, those exceeding a CCN of 30 were identified and are detailed in Figure 2. Methods such as *simpanEditJadwal* exhibit significant complexity, with a CCN of 263, posing serious maintainability challenges due to deeply nested conditions. Other methods, such as *simpanInputJadwal* and *index* in the course registration controller, also exhibit high complexity, increasing defect risks, and maintenance burden. A total of 18 methods surpass the critical CCN threshold, primarily within scheduling, course registration, and bank integration, underscoring the need for immediate refactoring into smaller, more manageable units.

D. Quality Issues Analysis

The static code analysis utilizing SonarQube identified 9,628 quality issues across 127 files. The issues fall mainly into maintainability, reliability, and security categories. Each file analyzed exhibits an average of 76 issues. Maintainability problems dominate, making up 89.9% of all issues. Every file has at least one maintainability flaw, underscoring systemic code quality issues. Reliability defects affect 92.1% of files, indicating widespread flawed coding practices. Security issues, totalling 18 files (5.5% of files), demand urgent attention given their risk potential. The file *model_host2host.php* has 484 issues, indicating severe technical debt that necessitates thorough refactoring. Consequently, Figure 3 presents the distribution of quality issues by quality aspect and severity level.

Among the total defects, security issues, although fewer, are highly severe, with 10 blockers and 8 high-severity issues. These concerns include hardcoded credentials and poor SSL/TLS validation, corresponding to critical OWASP Top 10-2021 vulnerabilities in the authentication and cryptographic categories, posing risks such as unauthorized access and man-in-the-middle attacks. In contrast, reliability issues are less severe, primarily at the medium and low levels, suggesting a non-immediate failure risk.

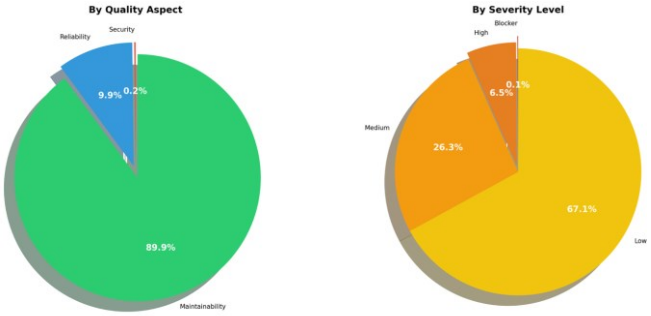

Figure 3. Quality Issues Distribution

Maintainability issues, spread across all severity levels, are predominantly low-severity (65.7%). The existence of 619 high-severity issues suggests significant technical debt affecting future development speed and code adaptability. Furthermore, Figure 4 presents the top ten files identified and analyzed for the highest quality issue density.

There is a noticeable correspondence between structurally complex files and those with high issue counts. Eight files reappear in both the high-complexity and high-issue lists, suggesting an association between complexity and issue prevalence, confirmed by correlation analysis. Notably, *model_host2host.php*, with 484 issues (99 reliability and 385 maintainability), comprises 5% of all issues. Other problematic files include *penjadwalan.php* and *isi_krs.php*, with only *model_home.php* showing security issues among the top problem files, indicating that, though severe, security concerns remain isolated.

A. Complexity and Quality Issues Correlation Analysis

A Pearson correlation analysis was used to examine the relationship between code complexity metrics and quality issues, revealing linear associations. A heatmap, as

DOI: 10.34148/teknika.v15i1.1458

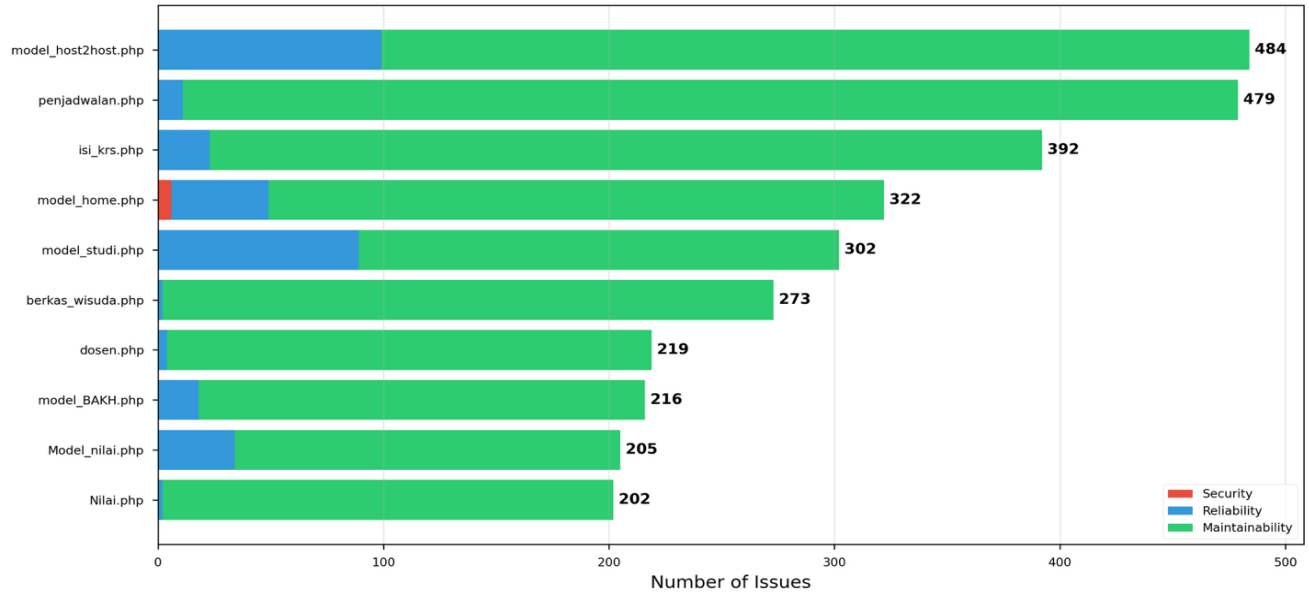

Figure 4. Top Ten Files with Highest Defect Counts

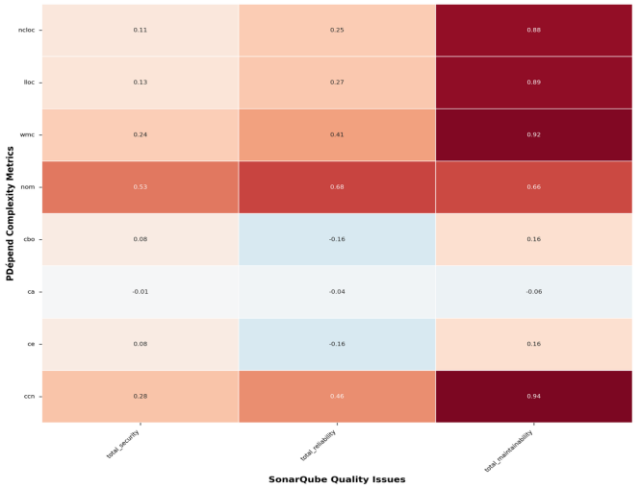

Figure 5. Complexity and Quality Issues Correlation Heatmap

demonstrated in Figure 5, illustrates the magnitude and significance of these associations, indicating the impact of structural complexity on code quality degradation. Correlation analysis was conducted at the file level (n = 127), with each file's aggregate complexity metrics from PDepend (e.g., total cyclomatic complexity, NCLOC, WMC) correlated against the corresponding file-level issue counts identified by SonarQube across security, reliability, and maintainability categories.

The analysis indicates very strong positive correlations between complexity metrics and maintainability issues. Cyclomatic complexity (CCN) shows the strongest link with maintainability issues ($r = 0.938$, $p < 0.001$), explaining 88% of the variance ($r^2 = 0.88$). This suggests focusing on reducing cyclomatic complexity to improve maintainability. Other metrics, such as WMC (Weighted Methods per Class), also exhibit a very strong correlation ($r = 0.923$), followed by LLOC ($r = 0.891$) and NCLOC ($r = 0.880$). These findings confirm that larger, complex codebases tend to accumulate more maintainability issues.

The number of methods (NOM) shows moderate positive correlations across reliability ($r = 0.675$), maintainability ($r = 0.662$), and security ($r = 0.526$), highly significant ($p < 0.001$). This indicates that files with many methods may have more quality issues, suggesting possible violations of the Single Responsibility Principle. Reliability issues show moderate correlations with complexity metrics, such as CCN ($r = 0.463$) and WMC ($r = 0.413$), suggesting that complex code is prone to bugs. Security issues correlate weakly with complexity metrics: NOM ($r = 0.526$), CCN ($r = 0.283$), and WMC ($r = 0.244$), suggesting that security concerns are more influenced by coding practices and security awareness than structural complexity.

B. Security Hotspots Analysis

SonarQube identified 275 security hotspots across 127 analyzed files. These hotspots require manual review by skilled personnel to determine appropriate remediation measures. The hotspots are categorized by security risk, highlighting areas that could facilitate unauthorized access or expose sensitive information. Figure 6 categorizes these hotspots by security category and their distribution.

The analysis reveals that “Insecure Configuration” is the largest category, with 86 hotspots, accounting for 31.3% of the total. This points to system misconfigurations that could lead to unauthorized access. Following closely are authentication-related issues, with 72 instances, making up 26.2% of the hotspots. These suggest weaknesses in access control and user identity verification. Other identified categories include miscellaneous security issues (47 hotspots, 17.1%), concerns relating to data encryption (42 hotspots,

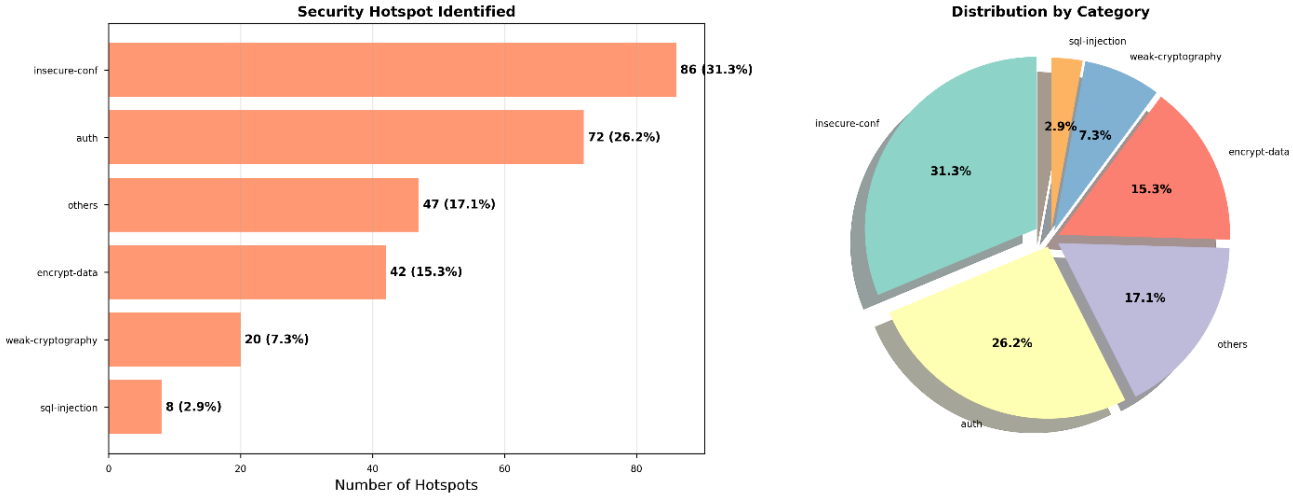

Figure 6. Security Hotspots Category

15.3%), weak cryptography implementations (20 hotspots, 7.3%), and SQL injection vulnerabilities (8 hotspots, 2.9%). The fact that insecure configurations and authentication issues account for 57.5% of the identified hotspots underscores the critical need for thorough security configuration evaluation and for enhancing authentication mechanisms to mitigate exploitation risks.

V. DISCUSSIONS

A. Complexity and Maintainability Relationship

The analysis reveals a significant correlation (r = 0.938) between cyclomatic complexity (CCN) and maintainability issues, validating the theoretical link between structural complexity and code quality. This aligns with McCabe's work [51] and other studies indicating that complex code is harder to handle [53], [54], [55], [56], [57]. Notably, methods like *simpanEditJadwal* exhibit catastrophic complexity (CCN of 263), highlighting a critical breach of software engineering norms. Recommended thresholds suggest a CCN of 10 or less for maintainability [51], [58], [59], with 18 methods with CCN >= 30 requiring urgent refactoring. Complexity concentration in scheduling and course registration modules signals a need for better modular design, rather than monolithic methods.

B. Pervasive Maintainability Issues

The 8,654 maintainability issues across 127 files indicate systemic quality deficiencies stemming from the absence of coding standards enforcement and automated quality checks. Prevalent code smells and technical debts suggest inadequate peer review processes. This accumulated technical debt reflects prioritization of rapid development over code quality, compounded by gaps in contemporary coding practices. Targeted remediation of concentrated problems, such as the 484 issues in *model_host2host.php*, through enhanced unit-test coverage and automated quality gates could generate substantial quality improvements.

C. Reliability and Security Implications

Beyond complexity and maintainability issues, the analysis identified 956 reliability issues across most files. Reliability issues typically include functional defects, syntax and semantic errors, logical defects, and various runtime errors [60], [61]. Additionally, 18 critical security vulnerabilities were detected, indicating significant governance challenges. Reliability defects correlate moderately with complexity (r = 0.463) and stem from poor error handling and resource management. Security concerns, particularly in authentication (55% of issues), pose significant risks and require immediate action aligned with OWASP guidelines to prevent unauthorized access and data breaches. Implementing strong cryptographic controls and thorough manual reviews of 275 security hotspots is essential.

D. Technical Debt Quantification and Remediation Priority

Remediation effort for the 9,628 detected quality defects totals approximately 2,407 person-hours (60 person-weeks), based on SonarQube's 5-minute-per-trivial-issue and 30-minute-per-major-issue estimates, with security review integration extending this to 62 person-weeks. This technical debt burden diminishes development efficiency, amplifies onboarding costs, and restricts innovation. Table 2 presents a prioritized remediation plan calibrated to the studied university's resource constraints, particularly its 1:3 permanent-to-apprentice developer ratio. Actions are sequenced by a composite assessment of severity, implementation effort, and expected impact, distinguishing quick wins (Priorities 1 and 2), achievable within existing capacity, from longer-term interventions requiring sustained investment. This prioritization operationalizes the tactical–strategic framework by providing actionable entry points for resource-constrained institutions.

E. Information Systems Management Implication

System quality at the studied university is substantially shaped by workforce structures that feature one permanent

DOI: 10.34148/teknika.v15i1.1458

Table 2. Prioritized Remediation Actions

Priority	Action	Severity	Effort	Impact
1	Remediate 18 critical security vulnerabilities (hardcoded credentials, SSL/TLS validation)	Blocker	Low (~1 person-week)	Eliminates immediate exploitation risk
2	Implement automated quality gates in CI/CD pipeline	Medium	Low (~1 person-week)	Prevents new technical debt accumulation
3	Refactor top 5 critically complex methods (CCN >= 30, e.g., *simpanEditJadwal*)	High	Moderate (~4 person-weeks)	Reduces defect density in scheduling and course registration modules
4	Address high-severity maintainability issues in top 5 defect-dense files	High	High (~12 person-weeks)	Resolves 30% of total maintainability debt
5	Establish structured code review and mentoring program for apprentice developers	Medium	Low (organizational)	Builds long-term capability within the 1:3 staffing model

developer for every three student developers. This staffing model contributes to the 62 person-weeks of accumulated technical debt, reflecting organizational decisions that prioritize rapid feature delivery over sustainable practices. Addressing these challenges necessitates enhanced talent management strategies, including converting apprentices to permanent positions, recruiting senior developers to balance junior-to-senior ratios, formalizing competency-based career progression frameworks, and implementing organizational capability enhancement initiatives. The strong correlations between complexity and maintainability metrics offer IS leaders predictive mechanisms for proactive quality management, while detected security vulnerabilities indicate governance deficiencies demanding executive action to safeguard sensitive academic data. Given resource limitations, institutions must adopt tailored IS quality strategies that mitigate immediate risks while cultivating long-term organizational capacity, thereby improving project outcomes while fulfilling educational missions through student professional development.

F. Limitations

Several limitations should be considered when interpreting these findings. As a single-case analysis conducted on a specific technological stack (PHP 5, CodeIgniter 3, IBM DB2), the generalizability of quantitative findings may be constrained, and different platforms may exhibit distinct quality profiles. The analysis focused on 127 files representing core application logic, but quality characteristics of excluded components, such as view templates and configuration files, remain unexamined. The recommendations have not undergone expert validation, and the cross-sectional design cannot capture temporal changes in quality or the longitudinal effects of implemented interventions. Additionally, the remediation effort estimates are derived from SonarQube's default time approximations, which may not fully reflect actual effort in this organizational context.

VI. CONCLUSIONS

This study conducted a comprehensive static code analysis on a resource-constrained university's academic information system, revealing 9,628 quality defects, 18 critical security vulnerabilities, and severe structural complexity across 127 PHP files. Strong correlations between complexity metrics and maintainability problems ($r = 0.938$, $p < 0.001$) confirm complexity as a reliable predictor of quality decline. A dual-track improvement strategy is proposed: the tactical track prioritizes immediate security hardening, automated quality gates, and refactoring of critically complex modules, while the strategic track addresses long-term sustainability through talent development, quality-centric culture, and incremental architectural modernization. Future research directions include developing context-aware static analysis frameworks validated through structured expert assessment, expanding empirical investigation across multiple institutions, and adopting longitudinal methodologies to analyze temporal relationships between organizational practices and code quality trajectories.

REFERENCES

[1] O. Gordieiev, D. Gordieieva, A. Rainer, and O. Pishchukhina, "Relationship between factors influencing the software development process and software defects," in *2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, IEEE, Oct. 2023, pp. 1–7. doi: 10.1109/DESSERT61349.2023.10416445.

[2] A. Nabot and A. Al-Qerem, "Impact of software quality on organizational performance," *Array*, vol. 27, p. 100476, Sep. 2025, doi: 10.1016/j.array.2025.100476.

[3] A. Al-Qerem *et al.*, "Enhancing Organizational Performance: Synergy of Cyber-Physical Systems, Cloud Services, and Crowdsensing," *International Journal of Crowd Science*, vol. 9, no. 1, pp. 44–55, Jan. 2025, doi: 10.26599/IJCS.2023.9100033.

[4] I. Obi, J. Butler, S. Haniyur, B. Hassan, M.-A. Storey, and B. Murphy, "Identifying Factors Contributing to

'Bad Days' for Software Developers: A Mixed-Methods Study," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, Apr. 2025, pp. 1–11. doi: 10.1109/ICSE-SEIP66354.2025.00006.

[5] A. G. Erol, M. Yilmaz, and P. M. Clarke, "Sleepless in the Code: Exploring the Relationship Between Occupational Anxiety and Sleep Patterns in the Software Industry," in *Communications in Computer and Information Science*, vol. 2179 CCIS, 2024, pp. 33–47. doi: 10.1007/978-3-031-71139-8_3.

[6] M. M. Ahmed, M. Khudari, A. M. Hussein, and J. Jais, "Quality of Work Life, Job Enrichment and their Impact on Employee Retention: Exploratory Research in Private Colleges in Baghdad," *WSEAS TRANSACTIONS ON BUSINESS AND ECONOMICS*, vol. 20, pp. 848–858, Apr. 2023, doi: 10.37394/23207.2023.20.78.

[7] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes, "Time pressure in software engineering: A systematic review," *Inf. Softw. Technol.*, vol. 121, p. 106257, May 2020, doi: 10.1016/j.infsof.2020.106257.

[8] R. Gilal, M. Omar, and M. M. Rejab, "The key factors contribute to time pressure in software development projects: A review," *International Journal of ADVANCED AND APPLIED SCIENCES*, vol. 10, no. 10, pp. 155–165, Oct. 2023, doi: 10.21833/ijaas.2023.10.018.

[9] B. Johnson, T. Zimmermann, and C. Bird, "The Effect of Work Environments on Productivity and Satisfaction of Software Engineers," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 736–757, Apr. 2021, doi: 10.1109/TSE.2019.2903053.

[10] S. Khalid, U. Qamar, U. Rahseed, and W. H. Butt, "Relationship between Organization's Physical and Psychological Environment and Employees' Mental Satisfaction: An Empirical Analysis of Employee Turnover in Software Industry," in *2022 16th International Conference on Open Source Systems and Technologies (ICOSST)*, IEEE, Dec. 2022, pp. 1–6. doi: 10.1109/ICOSST57195.2022.10016849.

[11] A. Subarno, C. D. S. Indrawati, and P. Ninghardjanti, "The influence of physical and non-physical factors on employee well-being in office spaces: A comprehensive study of the work environment," *IOP Conf. Ser. Earth Environ. Sci.*, vol. 1462, no. 1, p. 012023, Mar. 2025, doi: 10.1088/1755-1315/1462/1/012023.

[12] L. Piersiala, "Digital Technological Solutions in Knowledge Transfer in Higher Education," *European Conference on Knowledge Management*, vol. 25, no. 1, pp. 659–664, Sep. 2024, doi: 10.34190/eckm.25.1.2379.

[13] E. Acheampong and J. D. De-Graft, "Investigation into the Challenges Associated with the Delivery of Library Services on Mobile Technology Platform," *IAFOR Journal of Literature & Librarianship*, vol. 9, no. 1, pp. 78–93, Jul. 2020, doi: 10.22492/ijl.9.1.04.

[14] O. A. Taufik, O. Y. Pamungkas, S. Suprapto, A. K. Ahmad, D. W. Andini, and P. Cahyandaru, "Mapping Determinants of Success through Information Systems in Higher Education: A Structural Equation Modeling Approach," *Educational Process International Journal*, vol. 14, no. 1, 2025, doi: 10.22521/edupij.2025.14.67.

[15] M. K. Asamoah and J. D. Ansong, "Building resilience in higher education by addressing infrastructure and financial resource challenges in Ghana," *SN Social Sciences*, vol. 5, no. 7, p. 78, Jun. 2025, doi: 10.1007/s43545-025-01110-z.

[16] F. Hamad, M. Al-Fadel, and H. Fakhouri, "The provision of smart service at academic libraries and associated challenges," *Journal of Librarianship and Information Science*, vol. 55, no. 4, pp. 960–971, Dec. 2023, doi: 10.1177/09610006221114173.

[17] S. Mat'ašová, M. Chovanec, R. Rusnáková, and D. Čatloch, "Enhancing Code Quality Through Static Analysis: Optimizing the Flava Tool for Detecting Code Smells and Errors Using SonarQube Integration," in *2024 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, IEEE, Oct. 2024, pp. 431–438. doi: 10.1109/ICETA63795.2024.10850845.

[18] M. Adnan and M. Afzal, "A novel approach to super quality software development using workflows," in *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, Oct. 2016, pp. 1–3. doi: 10.1109/UEMCON.2016.7777855.

[19] U. Rosyidi, "Managing Corporate Higher Education: Indonesia's Greatest Challenge," *The New Educational Review*, vol. 53, no. 3, pp. 39–48, Sep. 2018, doi: 10.15804/tner.2018.53.3.03.

[20] L. D. Prasojo, L. Yuliana, and L. A. Prihandoko, "Research Performance in Higher Education: A PLS-SEM Analysis of Research Atmosphere, Collaboration, Funding, Competence, and Output, Especially for Science and Engineering Facilities in Indonesian Universities," *ASEAN Journal of Science and Engineering*, vol. 5, no. 1, pp. 123–144, Jan. 2025, doi: 10.17509/ajse.v5i1.81224.

[21] D. Azar, H. Harmanani, and R. Korkmaz, "A hybrid heuristic approach to optimize rule-based software quality estimation models," *Inf. Softw. Technol.*, vol. 51, no. 9, pp. 1365–1376, Sep. 2009, doi: 10.1016/j.infsof.2009.05.003.

[22] R. Ozakinci and A. Tarhan, "The Role of Process in Early Software Defect Prediction: Methods, Attributes and Metrics," 2016, pp. 287–300. doi: 10.1007/978-3-319-38980-6_21.

[23] J. Voas and W. W. Agresti, "Software quality from a behavioral perspective," *IT Prof.*, vol. 6, no. 4, pp. 46–50, Jul. 2004, doi: 10.1109/MITP.2004.45.

[24] J. S. Challa, A. Paul, Y. Dada, V. Nerella, and P. R. Srivastava, "Quantification of Software Quality Parameters Using Fuzzy Multi Criteria Approach," in

2011 International Conference on Process Automation, Control and Computing, IEEE, Jul. 2011, pp. 1–6. doi: 10.1109/PACC.2011.5978957.

[25] T. Wahyuningrum and K. Mustofa, "A Systematic Mapping Review of Software Quality Measurement: Research Trends, Model, and Method," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 5, p. 2847, Oct. 2017, doi: 10.11591/ijece.v7i5.pp2847-2854.

[26] L. Aversano and M. Tortorella, "Evaluating the Quality of Free/Open Source Systems: A Case Study," in *(eds) Enterprise Information Systems. ICEIS 2010. Lecture Notes in Business Information Processing*, vol. 73, Springer, Berlin, Heidelberg, 2011, pp. 119–134. doi: 10.1007/978-3-642-19802-1_9.

[27] Kilsup Lee and Sung Jong Lee, "A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development," in *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*, IEEE, 2005, pp. 20–25. doi: 10.1109/SNPD-SAWN.2005.7.

[28] N. F. I. Abdul Hamid and M. K. Hasan, "Software Quality Model for Telecommunication Industry in Malaysia," *J. Teknol.*, vol. 63, no. 1, pp. 13–19, Jul. 2013, doi: 10.11113/jt.v63.1354.

[29] A.-J. Molnar and S. Motogna, "Longitudinal Evaluation of Open-Source Software Maintainability," *ENASE 2020 - Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.00447

[30] W. Mallouli, "Security Testing as part of Software Quality Assurance: Principles and Challenges," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, Apr. 2022, pp. 29–29. doi: 10.1109/ICSTW55395.2022.00019.

[31] B. Wang, G. Ping, and D. Lv, "Multi-objective Software Quality Optimization Evaluation Model Based on Genetic Algorithm," in *2025 International Conference on Artificial Intelligence and Engineering Management (ICAIEM)*, IEEE, Jun. 2025, pp. 11–15. doi: 10.1109/ICAIEM66060.2025.11139275.

[32] A.-J. Molnar, A. Neamţu, and S. Motogna, "Longitudinal Evaluation of Software Quality Metrics in Open-Source Applications," in *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, SCITEPRESS - Science and Technology Publications, 2019, pp. 80–91. doi: 10.5220/0007725600800091.

[33] Y. Ghanim, "Toward a Specialized Quality Management Maturity Assessment Model," in *Proceedings of the 2nd Africa and Middle East Conference on Software Engineering*, New York, NY, USA: ACM, May 2016, pp. 1–8. doi: 10.1145/2944165.2944166.

[34] M. M. Alam, S. I. Priti, K. Fatema, M. Hasan, and S. Alam, "CMMI in Practice: Enhancing Organizational Performance Through Holistic Approaches," in *Studies in Big Data*, vol. 163, 2024, pp. 15–28. doi: 10.1007/978-3-031-73632-2_2.

[35] Y. Alqadri, E. K. Budiardjo, A. Ferdinansyah, and M. F. Rokhman, "The CMMI-Dev Implementation Factors for Software Quality Improvement," in *Proceedings of the 2020 2nd Asia Pacific Information Technology Conference*, New York, NY, USA: ACM, Jan. 2020, pp. 34–40. doi: 10.1145/3379310.3379327.

[36] S. L. Ramírez-Mora, H. Oktaba, and J. Patlán Pérez, "Group maturity, team efficiency, and team effectiveness in software development: A case study in a CMMI-DEV Level 5 organization," *Journal of Software: Evolution and Process*, vol. 32, no. 4, Apr. 2020, doi: 10.1002/smr.2232.

[37] R. Ferenc, P. Hegedűs, and T. Gyimóthy, "Software Product Quality Models," in *Evolving Software Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 65–100. doi: 10.1007/978-3-642-45398-4_3.

[38] J.-L. Letouzey, "The SQALE method for evaluating Technical Debt," in *2012 Third International Workshop on Managing Technical Debt (MTD)*, IEEE, Jun. 2012, pp. 31–36. doi: 10.1109/MTD.2012.6225997.

[39] D. Nikolic, D. Stefanovic, D. Dakic, S. Sladojevic, and S. Ristic, "Analysis of the Tools for Static Code Analysis," in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, IEEE, Mar. 2021, pp. 1–6. doi: 10.1109/INFOTEH51037.2021.9400688.

[40] D. Stefanovic, D. Nikolic, D. Dakic, I. Spasojevic, and S. Ristic, "Static Code Analysis Tools: A Systematic Literature Review," in *Annals of DAAAM and Proceedings of the International DAAAM Symposium*, vol. 31, no. 1, 2020, pp. 0565–0573. doi: 10.2507/31st.daaam.proceedings.078.

[41] T. Sonnekalb *et al.*, "A Static Analysis Platform for Investigating Security Trends in Repositories," in *2023 IEEE/ACM 1st International Workshop on Software Vulnerability (SVM)*, IEEE, May 2023, pp. 1–5. doi: 10.1109/SVM59160.2023.00005.

[42] A. K. Singh, D. Tomar, V. Ammasai, and U. Manogaran, "Enhancing software development efficiency: A study of static code analysis tools integrated with version control systems," in *AIP Conference Proceedings*, 2025, p. 020012. doi: 10.1063/5.0262307.

[43] S. Ghosh and R. Raj, "Fine-Tuning Large Language Models for Context-Aware Static Code Analysis and Defect Classification," in *Automating Software Defect Detection Through Machine Learning and LLMs*, IGI Global Scientific Publishing, 2025, pp. 249–288. doi: 10.4018/979-8-3373-4460-7.ch008.

[44] T. Akinyemi, E. Solomon, A. Woubie, and K. Lippert, "A Comprehensive Review of Static Memory

Analysis," *IEEE Access*, vol. 12, pp. 170204–170226, 2024, doi: 10.1109/ACCESS.2024.3482253.

[45] D. Liu, J. Calver, and M. Craig, "A Static Analysis Tool in CS1: Student Usage and Perceptions of PythonTA," in *Proceedings of the 26th Australasian Computing Education Conference*, New York, NY, USA: ACM, Jan. 2024, pp. 172–181. doi: 10.1145/3636243.3636262.

[46] E. A. AlOmar, S. A. AlOmar, and M. W. Mkaouer, "On the use of static analysis to engage students with software quality improvement: An experience with PMD," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE, May 2023, pp. 179–191. doi: 10.1109/ICSE-SEET58685.2023.00023.

[47] M. Nachtigall, M. Schlichtig, and E. Bodden, "Evaluation of Usability Criteria Addressed by Static Analysis Tools on a Large Scale," in *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, Bonn: Gesellschaft für Informatik e.V., 2023, pp. 95–96.

[48] M. Nachtigall, M. Schlichtig, and E. Bodden, "A large-scale study of usability criteria addressed by static analysis tools," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA: ACM, Jul. 2022, pp. 532–543. doi: 10.1145/3533767.3534374.

[49] D. Nikolic, A. Todoric, D. Dakic, T. Vuckovic, and D. Stefanovic, "Challenges in Integrating Static Code Analysis into Domain-Specific Language," in *Annals of DAAAM and Proceedings of the International DAAAM Symposium*, vol. 34, no. 1, 2023, pp. 0335–0340. doi: 10.2507/34th.daaam.proceedings.045.

[50] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1419–1457, Mar. 2020, doi: 10.1007/s10664-019-09750-5.

[51] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[52] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.

[53] H. A. N. Nikeshala, "Transforming Software Complexity Analysis with the ECB Metric," in *2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT)*, IEEE, Oct. 2024, pp. 156–164. doi: 10.1109/CONISOFT63288.2024.00029.

[54] T. H. A. N. N, "Software Complexity Analysis with the Advanced CB Metric," in *2024 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, IEEE, Nov. 2024, pp. 136–141. doi: 10.1109/ICT4DA62874.2024.10777285.

[55] Y. Tashtoush, N. Abu-El-Rub, O. Darwish, S. Al-Eidi, D. Darweesh, and O. Karajeh, "A Notional Understanding of the Relationship between Code Readability and Software Complexity," *Information*, vol. 14, no. 2, p. 81, Jan. 2023, doi: 10.3390/info14020081.

[56] Y. Hu, H. Jiang, and Z. Hu, "Measuring code maintainability with deep neural networks," *Front. Comput. Sci.*, vol. 17, no. 6, p. 176214, Dec. 2023, doi: 10.1007/s11704-022-2313-0.

[57] G. Rakić, M. Tóth, and Z. Budimac, "Toward recursion aware complexity metrics," *Inf. Softw. Technol.*, vol. 118, p. 106203, Feb. 2020, doi: 10.1016/j.infsof.2019.106203.

[58] M. Alenezi, M. Zarour, and M. Akour, "Complexity and Nesting Evolution in Open Source Software Systems: Experimental Study," *Recent Advances in Computer Science and Communications*, vol. 13, no. 4, pp. 572–578, Oct. 2020, doi: 10.2174/2213275912666190204134206.

[59] A. V. Gorchakov, L. A. Demidova, and P. N. Sovietov, "A Rule-Based Algorithm and Its Specializations for Measuring the Complexity of Software in Educational Digital Environments," *Computers*, vol. 13, no. 3, p. 75, Mar. 2024, doi: 10.3390/computers13030075.

[60] A. Raninen, T. Toroi, H. Vainio, and J. J. Ahonen, "Defect Data Analysis as Input for Software Process Improvement," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7343 LNCS, 2012, pp. 3–16. doi: 10.1007/978-3-642-31063-8_2.

[61] R. Ghafoor Hussain, K.-C. Yow, and M. Gori, "Leveraging an Enhanced CodeBERT-Based Model for Multiclass Software Defect Prediction via Defect Classification," *IEEE Access*, vol. 13, pp. 24383–24397, 2025, doi: 10.1109/ACCESS.2024.3525069.