ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

A Systematic Synthesis of Software Maintainability Paradigms: From Static Metrics to Predictive Intelligence

Gunawan Ismail [1], Kumala Dian Pangesti [2]*, Hendra Bayu Suseno[3], Syahira Putri Pratidina[4]

[1] Department of Information System, UIN Syarif Hidayatullah Jakarta, Indonesia
[2] Department of Law, UIN Syarif Hidayatullah Jakarta, Indonesia
[3]Department of Informatics Engineering, UIN Syarif Hidayatullah Jakarta, Indonesia
[4]Department Master of Information Technology, UIN Syarif Hidayatullah Jakarta, Indonesia

[1] gunawan.ismail24@mhs.uinjkt.ac.id
[2]*kumalaadian@gmail.com
[3]hendra.bayu@uinjkt.ac.id
[4]syahiraputripratidina24@mhs.uinjkt.ac.id

Received: 6 March 2025, Revised: 13 March 2025, Accepted: 2 April 2025, Published: 30 April 2025

Abstract

The high cost of maintenance and the demand for sustainable systems drive the need for a deeper understanding of software maintainability. Therefore, this study aims to map the latest trends, from evaluation metrics to solutions for code smells and technical debt. We applied the Systematic Literature Review (SLR) method based on the PRISMA guidelines to critically examine eight primary studies. The extracted data were then analyzed thematically to find common threads across the studies and identify underexplored areas. Our analysis highlights an interesting duality: classic metrics remain a relevant foundation, but the dominance of machine learning in predicting maintainability is increasingly undeniable, despite the accompanying imbalanced data constraints. The literature consistently confirms that code smells and technical debt are major bottlenecks to code quality, which can now be mitigated through automated refactoring strategies and prioritizing the handling of bad smells. Another crucial finding is the vital role of historical data on code evolution for prediction accuracy, as well as the emergence of a new perspective linking maintainability to energy efficiency. This study enriches the literature by consolidating empirical evidence, which not only validates previous theories but also identifies remaining research gaps. From a practical perspective, the results of this analysis can serve as a strategic reference for practitioners and academics in sharpening maintainability predictions and mitigations—key steps to reduce costs and improve software quality. As a future agenda, we recommend focusing research on developing more adaptive predictive models and deeper investigations into non-technical factors.

Keywords: Code Smells, Green IT, Machine Learning, Technical Debt, Software Maintainability.

I. Introduction

The dominance of software in the modern industrial ecosystem—from enterprise infrastructure to embedded systems—places code quality as a non-negotiable priority. In this landscape, software maintainability is not just an optional attribute, but a key determinant of system sustainability [1]. This aspect defines how flexible

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

a software is in adapting to bug fixes, feature modifications, and changes in the operational environment. In the increasingly complex digital era, software has become the backbone of nearly every industrial sector, from enterprise systems to mobile and embedded applications. High software quality is crucial, and one of the most vital quality characteristics is software maintainability [2]. Therefore, a predictive strategy to maintain maintainability levels from the early stages of development is no longer just an option, but a strategic imperative to ensure long-term efficiency.

The urgency of this issue is clearly reflected in the cost structure; the literature consistently highlights that the maintenance phase consumes the largest share of resources, often exceeding 60% of the total lifecycle cost. However, software has a natural tendency to experience entropy, or quality degradation, over time. The accumulation of unmanaged modifications leads to a surge in complexity, a situation often exacerbated by a buildup of technical debt and code smells—the residue of pragmatic design decisions made to meet deadlines [3].

In the software engineering ecosystem, maintainability defines a system's flexibility to accommodate corrections, adaptations, and feature enhancements without destroying its underlying structure [4]. The importance of this aspect cannot be underestimated; literature notes that the maintenance phase consistently monopolizes resource allocation, even consuming up to 70% of the total development lifecycle budget [5]. Unfortunately, software is subject to the phenomenon of 'software aging'; without disciplined intervention, system complexity will continue to creep up as changes accumulate. This situation is often exacerbated by technical debt and code smells [6]. Therefore, the ability to predict code quality degradation early is no longer just an option, but a strategic imperative to ensure efficient system continuity [7].

Managing software maintainability effectively is not only a technical requirement but also a significant business strategy to continuously strive for more accurate metrics, models, and approaches to measure and predict maintainability [8]. In the academic realm, according to T. Besker, A. Martini, and J. Bosch, the focus of research is increasingly focused on finding evaluation models that are able to capture code dynamics precisely, both through static and dynamic analysis [9]. Many studies have explored the use of machine learning techniques to predict maintainability based on various aspects of source code, but often focus on the current state of the code and ignore its evolutionary history [10]. On the other hand, the discourse on the correlation between maintainability and energy efficiency (Green IT) is still dominated by theoretical hypotheses with minimal empirical validation. [11]. Although the importance of maintainability to software energy efficiency has been highlighted theoretically, empirical evidence to test this relationship is still limited [12].

Despite numerous attempts to predict software maintainability, significant gaps in its understanding and practical application remain that need to be addressed by further research. For example, several studies have highlighted the impact of bad smells on software quality, but few have proposed methods for prioritizing bad smells based on their impact on maintainability [13]. Furthermore, while the use of historical data for maintainability prediction is intuitive, empirical research examining the contribution of historical software metric changes to predicting future maintainability trends is limited. Automated refactoring approaches utilizing fuzzy genetic methods have been introduced to address code smells and improve maintainability, but their implementation and validation on a larger scale require further exploration. [14]. This gap indicates the need for more adaptive predictive models, considering code evolution, addressing data imbalance, and providing clear guidance for prioritizing maintenance efforts. Based on this background and gap, the core question that will guide this research is: How can we identify research trends in software maintainability that are significant for improving the maintainability of large-scale software systems.

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

II. Related Work

Research on software maintainability initially centered on structural code metrics as objective indicators of internal quality. Classical object-oriented metrics such as the Chidamber & Kemerer suite and Lines of Code (LOC) remain widely validated predictors of complexity, coupling, and structural stability. Although these metrics were introduced decades ago, empirical studies consistently confirm their relevance, demonstrating that fundamental structural characteristics continue to shape maintenance effort and system evolution.

Subsequent research expanded beyond static measurement toward qualitative degradation factors, particularly code smells and technical debt. Empirical investigations show that unmanaged code smells significantly increase maintenance complexity, prompting the development of prioritization models and automated refactoring techniques, including optimization-based and fuzzy genetic approaches. In parallel, machine learning–driven predictive models have become dominant in estimating maintainability levels, leveraging ensemble algorithms such as Random Forest variants. However, class imbalance within software datasets remains a persistent limitation, requiring resampling strategies to enhance predictive robustness and reduce bias toward majority classes.

More recent studies introduce a dynamic and sustainability-oriented perspective by incorporating historical software evolution data and exploring maintainability's relationship with energy consumption. Evolution-aware models demonstrate that historical metric changes across releases provide stronger predictive power than snapshot-based approaches, recognizing maintainability as a temporal property shaped by cumulative modifications. Additionally, exploratory empirical evidence suggests that structural characteristics such as LOC may influence energy usage, extending maintainability implications beyond technical quality toward green software engineering. Despite significant advances across metrics, predictive analytics, refactoring automation, and sustainability research, existing studies remain fragmented. An integrative framework that unifies structural metrics, historical evolution, imbalance-aware prediction, and environmental considerations therefore represents a critical research opportunity

III. Research Methodology

The empirical foundation of this study employs a Systematic Literature Review (SLR) protocol as the primary research design to identify, evaluate, and interpret all available studies relevant to the formulated research questions. The SLR design, specifically following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines, was chosen for its ability to provide a transparent, replicable, and comprehensive method for gathering scientific evidence. [15], [16]. This approach systematically minimizes bias and increases the reliability of findings by establishing a rigorous search protocol, clear inclusion and exclusion criteria, and a multi-step study selection process [17], [18]. Thus, this design is well suited to achieve the research objective of building an adaptive predictive model for software maintainability, as it allows for a thorough identification of best practices, relevant metrics, and challenges that already exist in the extensive literature.

A. Characteristics and Reference Selection

The study identification and selection process was conducted through several systematic stages. First, an initial search was conducted in major academic databases such as IEEE Xplore, ACM Digital Library, Scopus, and SpringerLink using a combination of keywords relevant to software maintainability and maintainability prediction. The number of references screened in this initial stage was substantial, reflecting the breadth of

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

research in the field of software maintainability. The initial search resulted in 118 articles published in international journals from 1981 to 2025. However, the SLR process typically involves thousands of initial results that are then filtered through inclusion and exclusion criteria.[19], [20], [21]

Inclusion criteria focused on studies published between 2014 and 2025 that: (1) explicitly discussed software maintainability or its influencing factors (e.g., technical debt, code smells), (2) proposed or evaluated models/metrics/approaches for predicting or improving maintainability, (3) used empirical research methodologies (case studies, experiments, surveys) or systematic reviews, and (4) were published in peer-reviewed scientific journals. Exclusion criteria included studies that: (1) only discussed software quality in general without focusing on maintainability, (2) were in gray literature (e.g., theses, dissertations, blog posts) without peer review, (3) were non-systematic review articles or opinion pieces, and (4) were not relevant to the context of predicting or improving software maintainability, (5) or were reputable conference proceedings in English.

The study selection method was conducted in two stages: first, screening based on titles and abstracts to identify potentially relevant studies; second, reading the full text of the studies that passed the first stage to thoroughly verify their relevance. This process ensures that only the most relevant and high-quality studies are included in this review, reflecting current and relevant research trends on software maintainability.

B. Validity and Reliability

The validity and reliability of an SLR depend heavily on thoroughness at every stage of the process. To ensure internal validity, the research protocol was explicitly designed and documented, including a comprehensive search strategy, objective selection criteria, and standardized data extraction procedures. This helps minimize study selection and data extraction bias. External validity was achieved by ensuring broad coverage of relevant databases and publications, so that the synthesized findings can be generalized to various research contexts and practices in the field of software maintainability.

The reliability of this study was enhanced through the PRISMA guidelines, which mandate a systematic and transparent approach. The interpretation and data extraction of selected studies will be achieved by involving at least two researchers independently in the screening and data extraction process, then resolving discrepancies through discussion and consensus. The use of a reference manager also ensures consistency and accuracy of citations and bibliographies. Although some studies may have limitations in their own internal validity or reliability (e.g., imbalanced data in predictive models that could impact accuracy), this SLR will critically evaluate and address these limitations in the synthesis of findings. Focusing on empirical studies that have gone through a rigorous peer-review process also inherently supports the reliability of the data collected.

The data collection procedures in this study were systematically designed to ensure transparency and replicability. The preparation phase involved designing a detailed SLR protocol, including a clear definition of the research question, a comprehensive keyword search strategy, and specific inclusion and exclusion criteria, similar to the practice in other SLR studies. [22], [23]. The data collection period will cover the period from the protocol drafting date to the completion of the study selection process, with an estimated duration of several weeks to allow for careful screening of the significant volume of literature. The data collection will be conducted online through access to the sci-hub database, which is standard practice in systematic literature reviews. The specific technique applied for data collection is data extraction from each article that has passed the inclusion criteria, where relevant information such as the year of publication, the metrics used (e.g.,

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

Chidamber & Kemerer, lines of code), the methodology (e.g., machine learning, refactoring), the main results, and the limitations of the study are recorded in a structured manner. For example, data regarding the impact of code smells and the prioritization of bad smells on maintainability will be extracted from related studies. [24]. In addition, information regarding the handling of imbalanced data in maintainability prediction is also a focus of data extraction, considering its significance in improving model accuracy.

The data analysis methods used in this study are narrative and thematic synthesis, chosen due to their relevance in processing diverse qualitative and quantitative data from the included studies. The thematic approach allows for the identification of recurring patterns, trends, and research gaps across the collected literature. Quantitative data, such as maintainability metrics or predictive model accuracy from the surveyed studies, will be analyzed descriptively to identify the most frequently used metrics and the effectiveness of various machine learning algorithms. Qualitative data, such as findings on maintainability challenges or proposed solutions to address code smells, will be synthesized to build a comprehensive understanding of the current research landscape. This approach will also make it possible to identify how previous studies addressed issues such as imbalanced data in maintainability predictions or the relationship between maintainability and energy consumption, as well as the limitations they faced [25]. By comparing findings from various sources, this thematic analysis aims to understand and formulate recommendations for robust and relevant adaptive predictive models.

IV. Result

A. Result

The literature shows that metrics and tools for measuring software maintainability have become a significant and systematic area of research on tool-based maintainability metrics and finds that the Chidamber & Kemerer Suite (C&K) is one of the most well-known and widely used sets of metrics, designed specifically for object-oriented approaches. Metrics in the C&K suite include Weighted Method per Class (WMC), which measures class complexity, and Coupling Between Objects (CBO), which measures dependencies between classes, both of which have a significant impact on maintainability. Furthermore, Lines of Code Modified (LOC) is also an important metric, as ongoing changes to a class can be an indicator of low maintainability. These findings underscore that despite the variety of metrics available; C&K and LOC remain relevant in evaluating maintainability.

Literature consistently recognizes code smells and technical debt as crucial factors affecting software maintainability. Alshammari and Alshayeb (2021) emphasize that bad smells are indicators of potential problems in software, and refactoring is generally performed to address them. Their research proposes a model for prioritizing bad smells based on their impact on software maintainability, which can help practitioners allocate efforts and resources more effectively. Saheb Nasagh et al. (2020) introduce an automated method for identifying and refactoring code smells using a fuzzy genetic approach, demonstrating how fixing code smells can improve software maintainability and flexibility. These findings reinforce the understanding that code smells are directly correlated with maintenance difficulty and that intervention through refactoring can mitigate this.

A prominent trend in maintainability research is the application of machine learning (ML) techniques for prediction. Malhotra and Lata (2020) show that numerous predictive models have been developed using various ML techniques to predict the maintainability of software modules or classes. However, they highlight the challenge of imbalanced data, where an imbalanced dataset can bias ML techniques in their predictions toward the majority class, ignoring instances of the minority class that may have high maintenance effort. To address this issue, their research shows that data resampling methods, including oversampling, under sampling, and hybrid resampling,

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

can improve the performance of maintainability prediction models (SMPs) developed with ML techniques. Gupta and Chug (2020) also support the use of ML approaches by proposing the Enhanced Random Forest Algorithm (Enhanced-RFA) for maintainability prediction, demonstrating significant improvements in predictive performance compared to the standard Random Forest method. This suggests that optimizing data and ML algorithms is a crucial focus in achieving accurate maintainability predictions.

Historical aspects and code evolution are also a focus in maintainability research. Gradišnik et al. (2020) examined the impact of historical software metric changes in predicting future maintainability trends. They found that the additional layer of historical changes in software metrics from previous releases contributes to better predictions of future software maintainability. This challenges the common approach of focusing solely on the current state of the code and ignoring its evolutionary history, demonstrating that historical data has significant predictive value. This study emphasizes the importance of considering code evolution for more accurate maintainability predictions.

Beyond internal technical aspects, maintainability is also beginning to be linked to broader external factors, such as energy consumption. Mancebo et al. (2021) conducted an empirical study to test whether there is a relationship between energy consumption and software maintainability. While previous theoretical studies have reinforced the idea that maintainability may influence energy use, their study aimed to empirically demonstrate this. Preliminary results indicate that metrics such as lines of code (LOC) can influence process energy consumption, suggesting a correlation between maintainability-related code characteristics and energy efficiency. While this research is more exploratory, it opens new dimensions in understanding the impact of maintainability beyond the purely technical realm. Overall, the findings from the reviewed literature indicate that the field of software maintainability continues to evolve, with a focus on developing more accurate metrics and predictive tools, addressing code smells and technical debt, as well as leveraging historical data and considering the impact on other non-functional aspects.

The finding that metrics such as the Chidamber & Kemerer Suite (C&K) and Lines of Code (LOC) remain important indicators of software maintainability strongly supports the existing literature on code quality measurement. Ardito et al. (2020) explicitly highlights C&K as one of the most well-known and widely used metric sets, consistent with the notion that complexity and coupling are fundamental factors in determining the maintainability of object-oriented systems. The modified LOC, as an indicator of change frequency, also reinforces the theory that instability or constant change in a component is often inversely related to its optimal maintainability. The relevance of these metrics, despite being introduced decades ago, demonstrates that the fundamental principles underlying software maintainability remain relevant and provide a solid foundation for further research. These findings reinforce the belief that maintainability evaluation should begin with the analysis of basic code metrics, which can provide an initial snapshot of the software's internal health. This reinforcement is crucial for practitioners seeking early indicators to identify areas requiring maintenance attention.

The application of machine learning (ML) techniques to software maintainability prediction has become a dominant trend, confirming the ongoing efforts to automate and improve the accuracy of the estimation process. However, findings regarding the imbalanced data challenge raised by Malhotra and Lata (2020) identify crucial limitations in many existing prediction models. This demonstrates a discrepancy between the potential of ML and real-world data challenges, where imbalanced datasets can lead to model bias and neglect of minority classes, often cases with high maintainability effort. Solutions through data resampling methods, such as those proposed by Malhotra and Lata (2020), or the use of enhanced algorithms such as the Enhanced Random Forest Algorithm (Enhanced-RFA) by Gupta and Chug (2020), directly address this issue. The consistency between these two studies in acknowledging the problem and offering solutions based on data preprocessing or more robust algorithms suggests that research is moving towards more accurate and reliable predictions, even under challenging data conditions. The theoretical implication is that maintainability prediction models should explicitly account for data distribution, and the practical implication is that developers need to consider resampling techniques when applying

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

ML models for maintainability estimation.

The findings confirm the negative impact of code smells and technical debt on software maintainability align with broad consensus in the software engineering literature. Alshammari and Alshayeb (2021) specifically proposed a model for prioritizing bad smells based on their impact on maintainability, which is an important development because it not only identifies the problem but also offers strategies to address maintenance resource constraints. This approach expands understanding beyond simply recognizing the presence of code smells to providing actionable insights about which bad smells should be prioritized for refactoring to maximize maintainability improvements. Meanwhile, the fuzzy genetic automatic refactoring method introduced by Saheb Nasagh et al. (2020) provides an automated solution for addressing code smells, demonstrating how technology can assist practitioners in improving code maintainability and flexibility. The theoretical implication of these findings is the importance of integrating code smell impact analysis into a more comprehensive maintainability prediction model, and the practical implication is the recommendation to adopt automated tools that can support efficient code smell identification and refactoring.

Gradišnik et al.'s (2020) study, which highlights the contribution of historical software metric changes in predicting future maintainability trends, marks an important shift from a static to a dynamic focus in maintainability evaluation. This contrasts with previous studies that tended to ignore code history and rely solely on current conditions. This finding is particularly relevant because maintainability is not a static property; it continuously evolves as code changes occur. By demonstrating that "an additional layer of historical changes in software metrics from previous releases contributes to better predictions of future software maintainability," this study fills an important gap in the literature. The theoretical implication is that any robust maintainability prediction model should integrate the temporal dimension and historical nature of code changes, not just snapshot analysis. For practitioners, this means that version control systems (VCSs) and the historical data they contain should be more actively utilized for predictive maintainability analysis.

Mancebo et al.'s study (2021) exploring the relationship between software maintainability and software energy consumption opens an important new dimension in the literature. While this study is exploratory and represents an initial attempt to empirically test a relationship previously only theoretically established, the preliminary finding that Lines of Code (LOC) can affect process energy consumption suggests an interesting correlation. This expands the understanding of maintainability's impact beyond internal quality metrics and development efficiency, reaching the increasingly important areas of sustainability and green computing. While there is no broad consensus or strong empirical evidence in the reviewed literature regarding this relationship, these findings point to a promising future research direction, where maintainability can be viewed as a determining factor in software energy efficiency. The practical implication is that efforts to improve maintainability can indirectly contribute to reducing the carbon footprint of software, adding economic and environmental value to the initiative.

For future research, it is recommended to:

1. Conduct large-scale empirical studies that directly test adaptive predictive models for maintainability that integrate historical data and imbalanced data handling methods on real-world datasets from large-scale systems.
2. Develop and validate automated tools that can identify and prioritize code smells based on their specific maintainability impact, considering different programming languages and architectures.
3. Further investigate the relationship between software maintainability and energy consumption, with more controlled experimental studies to identify causal correlations and environmental implications.
4. Explore the role of non-technical factors, such as team development practices and organizational culture, in influencing software maintainability, which may be understudied in the current literature.
5. Expand the literature review to include other types of technical debt beyond code smells, as well as mitigation strategies. These recommendations aim to fill the identified gaps and deepen our understanding

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

of software maintainability from multiple perspectives, both theoretical and practical.

B. Discussion

This study conducted a systematic review of the literature on software maintainability, identifying key trends, metrics, and approaches used for its prediction and improvement. Key findings confirm that classic metrics such as the Chidamber & Kemerer Suite (C&K) and Lines of Code (LOC) remain relevant as fundamental maintainability indicators, providing a strong foundation for code quality analysis. Significantly, the study shows that the application of machine learning (ML) techniques has become the dominant method for maintainability prediction, although the challenge of imbalanced data in datasets often reduces model accuracy. To address this, the use of resampling methods or enhanced ML algorithms has been shown to improve prediction performance. Furthermore, the study highlights the crucial impact of code smells and technical debt on maintainability, with automated refactoring approaches and bad smell prioritization models emerging as effective solutions. The importance of historical data and code evolution in predicting future maintainability is also emphasized, challenging the exclusive focus on current code conditions. Finally, the study highlights the emerging relationship between maintainability and broader aspects such as energy consumption, opening up new research areas. This study's significant contribution lies in its comprehensive synthesis of these trends, providing a more structured understanding of the software maintainability research landscape. Theoretically, it strengthens existing frameworks on software metrics and re-informs predictive models by emphasizing the importance of historical dynamics and data imbalance management. Practically, these findings provide valuable guidance for practitioners to proactively identify and mitigate maintainability risks. The ability to predict maintainability early through optimized metrics and ML can save substantial maintenance costs, allocate resources more efficiently, and improve overall software quality. Prioritizing code smells and automated refactoring strategies offers practical tools for development teams to effectively and sustainably manage technical debt.

While this review provides insightful insights, several limitations should be acknowledged, particularly due to its nature as a SLR that relies on existing data. Therefore, recommendations for future research include: (1) Development and validation of new adaptive predictive models that explicitly integrate historical data and imbalance management techniques on large-scale datasets. (2) Larger empirical studies to test the causal relationship between software maintainability and energy consumption. (3) Exploration of the role of non-technical factors such as team practices and organizational culture on maintainability. (4) In-depth investigation of the effectiveness of various existing automated tools for code smell identification and refactoring. These recommendations aim to fill the identified research gaps, deepen understanding, and facilitate the development of more practical solutions to improve software maintainability in the future.

V. Conclusion

This study systematically synthesized contemporary research on software maintainability to map the transition from traditional metric-based evaluation toward predictive intelligence–driven approaches. The findings confirm that classical structural indicators such as the Chidamber & Kemerer (C&K) metrics and Lines of Code (LOC) remain foundational in assessing internal software quality. However, maintainability research has significantly evolved beyond static measurement, incorporating machine learning–based prediction models, automated refactoring strategies, and historical evolution analysis. The dominance of predictive analytics demonstrates a paradigm shift in which maintainability is no longer assessed solely as a descriptive property but increasingly treated as a forecastable and manageable attribute of large-scale software systems.

The review also highlights critical challenges that shape the current research landscape. Class imbalance in maintainability datasets continues to undermine prediction reliability, necessitating resampling strategies and

ITDM: Information Technology Development And Management
Volume 1, (1) 2025, p. 53-62
E-ISSN: XXXX-XXXX

enhanced ensemble algorithms to improve model robustness. Additionally, the persistent impact of code smells and technical debt underscores the importance of prioritization frameworks and automated mitigation techniques to allocate maintenance resources effectively. The integration of historical metric changes further reveals that maintainability should be conceptualized as a dynamic and temporal characteristic rather than a static condition. These findings collectively reinforce the need for adaptive predictive models capable of capturing structural complexity, evolutionary trends, and data distribution constraints simultaneously.

In a broader perspective, the emerging linkage between maintainability and energy consumption expands the conceptual boundaries of the field, positioning maintainability within the broader discourse of sustainable and green software engineering. Although empirical validation remains limited, preliminary evidence suggests that improving structural quality may contribute not only to reduced maintenance costs but also to enhanced energy efficiency. Future research should therefore focus on developing integrated, evolution-aware, and imbalance-sensitive predictive frameworks, while also exploring interdisciplinary connections with sustainability and organizational factors. By consolidating fragmented streams of research, this study contributes a structured understanding of the maintainability paradigm and offers strategic direction for both researchers and practitioners seeking long-term software sustainability.

REFERENCES

[1] I. Sommerville, Software engineering, 10. ed. Boston, Munich: Pearson, 2016.

[2] R. Saheb Nasagh, M. Shahidi, and M. Ashtiani, “A fuzzy genetic automatic refactoring approach to improve software maintainability and flexibility,” Soft Comput., vol. 25, no. 6, pp. 4295–4325, Mar. 2021, doi: 10.1007/s00500-020-05443-0.

[3] M. Fowler and K. Beck, Refactoring: improving the design of existing code, 28. printing. in The Addison-Wesley object technology series. Boston: Addison-Wesley, 2013.

[4] T. Alshammari and M. Alshayeb, “Toward a Software Bad Smell Prioritization Model for Software Maintainability,” Arab. J. Sci. Eng., vol. 46, no. 9, pp. 9157–9177, Sep. 2021, doi: 10.1007/s13369-021-05766-6.

[5] M. Gradišnik, T. Beranič, and S. Karakatič, “Impact of Historical Software Metric Changes in Predicting Future Maintainability Trends in Open-Source Software Development,” Appl. Sci., vol. 10, no. 13, p. 4624, Jul. 2020, doi: 10.3390/app10134624.

[6] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162),” Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/DAGREP.6.4.110.

[7] T. Alshammari and M. Alshayeb, “Toward a Software Bad Smell Prioritization Model for Software Maintainability,” Arab. J. Sci. Eng., vol. 46, no. No. 9, pp. 9157–9177, Sep. 2021, doi: https://doi.org/10.1007/s13369-021-05766-6.

[8] S. Gupta and A. Chug, “Software Maintainability Prediction Using an Enhanced Random Forest Algorithm,” J. Discrete Math. Sci. Cryptogr., vol. 23, no. No. 2, pp. 441–449, Feb. 2020, doi: https://doi.org/10.1080/09720529.2020.1728898.

[9] T. Besker, A. Martini, and J. Bosch, “Impact of Architectural Technical Debt on Daily Software Development Work — A Survey of Software Practitioners,” in 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria: IEEE, Aug. 2017, pp. 278–287. doi: 10.1109/SEAA.2017.16.

[10] R. Malhotra and K. Lata, “Using Ensembles for Class-Imbalance Problem to Predict Maintainability of Open-Source Software,” Int. J. Reliab. Qual. Saf. Eng., vol. 27, no. No. 05, p. 2040011, Oct. 2020, doi: https://doi.org/10.1142/S0218539320400112.

[11] B. C. Mourão, L. Karita, and I. Do Carmo Machado, “Green and Sustainable Software Engineering -

a Systematic Mapping Study," in Proceedings of the XVII Brazilian Symposium on Software Quality, Curitiba Brazil: ACM, Oct. 2018, pp. 121–130. doi: 10.1145/3275245.3275258.

[12] J. Mancebo, C. Calero, and F. García, "Does maintainability relate to the energy consumption of software? A case study," Softw. Qual. J., vol. 29, no. 1, pp. 101–127, Mar. 2021, doi: 10.1007/s11219-020-09536-9.

[13] M. Gradišnik, T. Beranič, and S. Karakatič, "Impact of Historical Software Metric Changes in Predicting Future Maintainability Trends in Open-Source Software Development," Appl. Sci., vol. 10, no. 13, p. 4624, Jul. 2020, doi: 10.3390/app10134624.

[14] R. Saheb Nasagh, M. Shahidi, and M. Ashtiani, "A fuzzy genetic automatic refactoring approach to improve software maintainability and flexibility," Soft Comput., vol. 25, no. 6, pp. 4295–4325, Mar. 2021, doi: 10.1007/s00500-020-05443-0.

[15] M. J. Page et al., "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews," BMJ, p. n71, Mar. 2021, doi: 10.1136/bmj.n71.

[16] A. F. Herrera Ortiz et al., "A Practical Guide to Perform a Systematic Literature Review and Meta-analysis," Princ. Pract. Clin. Res. J., vol. 7, no. 4, pp. 47–57, Dec. 2021, doi: 10.21801/ppcrj.2021.74.6.

[17] N. Shaheen et al., "Appraising systematic reviews: a comprehensive guide to ensuring validity and reliability," Front. Res. Metr. Anal., vol. 8, Dec. 2023, doi: 10.3389/frma.2023.1268045.

[18] Y. Riyadi, M. Wahidin, and A. Elanda, "Systematic Literature Review Implementasi Service Operation Dalam Kerangka Kerja Information Technology Infrastructure Library (ITIL) di Indonesia: Tren Penelitian, Manfaat dan Tantangan," J. Interkom J. Publ. Ilm. Bid. Teknol. Inf. Dan Komun., vol. 17, no. 2, pp. 81–97, Jul. 2022, doi: 10.35969/interkom.v17i2.232.

[19] P. Candra Susanto, D. Ulfah Arini, L. Yuntina, J. Panatap Soehaditama, and N. Nuraeni, "Konsep Penelitian Kuantitatif: Populasi, Sampel, dan Analisis Data (Sebuah Tinjauan Pustaka)," J. Ilmu Multidisplin, vol. 3, no. 1, pp. 1–12, Apr. 2024, doi: 10.38035/jim.v3i1.504.

[20] N. Lestari, P. Paidi, and S. Suyanto, "A systematic literature review about local wisdom and sustainability: Contribution and recommendation to science education," Eurasia J. Math. Sci. Technol. Educ., vol. 20, no. 2, p. em2394, Feb. 2024, doi: 10.29333/ejmste/14152.

[21] P. M. Khristodas, K. Palanivelu, A. Ramachandran, J. Anushiya, B. A. K. Prusty, and S. Guganesh, "ASSESSMENT OF CLIMATE-INDUCED SEA-LEVEL RISE SCENARIOS AND ITS INUNDATION IN COASTAL ODISHA, INDIA," Appl. Ecol. Environ. Res., vol. 20, no. 4, pp. 3393–3409, 2022, doi: 10.15666/aeer/2004_33933409.

[22] R. I. Williams, L. A. Clark, W. R. Clark, and D. M. Raffo, "Re-examining systematic literature review in management research: Additional benefits and execution protocols," Eur. Manag. J., vol. 39, no. 4, pp. 521–533, Aug. 2021, doi: 10.1016/j.emj.2020.09.007.

[23] S. Young et al., "PROTOCOL: Searching and reporting in Campbell Collaboration systematic reviews: An assessment of current methods," Campbell Syst. Rev., vol. 17, no. 4, Dec. 2021, doi: 10.1002/cl2.1208.

[24] T. Alshammari and M. Alshayeb, "Toward a Software Bad Smell Prioritization Model for Software Maintainability," Arab. J. Sci. Eng., vol. 46, no. 9, pp. 9157–9177, Sep. 2021, doi: 10.1007/s13369-021-05766-6.

[25] R. Malhotra and K. Lata, "An Empirical Study on Predictability of Software Maintainability Using Imbalanced Data," Softw. Qual. J., vol. 28, no. No. 4, pp. 1581–1614, Dec. 2020, doi: https://doi.org/10.1007/s11219-020-09525-y.