INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 7,854 Volume 10, issue 08 (2023)

Artificial Intelligence and Machine Learning in Software Evolution and Architectural Transformation: A Theoretical Framework for Intelligent Software Engineering

Matia Kovács

Department of Computer Science, University of Budapest, Hungary

Abstract: The evolution of software systems has long been recognized as a fundamental phenomenon in computer science and software engineering. As software systems grow in complexity and scale, traditional development and maintenance practices often struggle to keep pace with the demands of modern digital ecosystems. The emergence of artificial intelligence and machine learning technologies has introduced new possibilities for automating and improving various aspects of software engineering, including software evolution, architectural transformation, software reuse, and knowledge management. This research presents an extensive theoretical investigation into the role of artificial intelligence and machine learning in addressing challenges associated with software evolution and modern architectural paradigms such as microservices.

The study synthesizes classical theories of software evolution with contemporary research on artificial intelligence–driven software engineering practices. Foundational principles, including the laws of software evolution, are examined in relation to emerging AI-based development methodologies. Particular attention is given to the migration of monolithic systems toward microservice architectures, the management of technical debt, and the role of machine learning in identifying service boundaries and architectural transformation strategies.

The methodology employed in this research involves an integrative theoretical analysis of existing literature combined with qualitative interpretive techniques to construct a conceptual framework explaining how AI technologies support intelligent software development. Findings indicate that machine learning algorithms and AI-based knowledge management systems significantly enhance software maintainability, support architectural decision-making, and facilitate the reuse of software components across evolving systems. Furthermore, AI-driven approaches provide new capabilities for analyzing legacy systems and recommending optimal strategies for migration toward modular architectures.

The discussion explores broader implications for the future of software engineering, including the development of intelligent software ecosystems capable of self-analysis and adaptive evolution. Limitations related to organizational adoption, algorithmic transparency, and the complexity of socio-technical systems are also examined. The study concludes that the integration of artificial intelligence into software engineering processes represents a transformative shift in how software systems are designed, maintained, and evolved over time.

Keywords: Artificial Intelligence in Software Engineering, Software Evolution, Machine Learning, Microservices Architecture, Software Reuse, Knowledge Management, Intelligent Software Systems.

Introduction

The development and evolution of software systems represent one of the most complex engineering activities in modern technological society. Since the earliest days of computer science, researchers and practitioners have sought to understand how software systems grow, adapt, and transform in response to changing requirements and technological environments. Unlike many traditional engineering artifacts, software systems are inherently dynamic and continuously evolving entities. This dynamic nature has led to the formulation of theoretical frameworks that describe the laws and patterns governing software evolution.

One of the most influential theoretical contributions in this area is the concept of software evolution laws, originally proposed by Meir Manny Lehman. These laws describe how large software systems evolve over time and emphasize the inevitability of continuous change as systems interact with real-world environments (Lehman, 1980). Lehman later revisited these laws and refined them, highlighting the complex feedback mechanisms that influence software development processes (Lehman, 1996). According to these principles, software systems that operate in real-world environments must evolve continuously or risk becoming progressively less useful.

The concept of software evolution highlights a fundamental challenge in software engineering: maintaining and adapting complex software systems over extended periods. As systems evolve, they often accumulate technical debt, architectural complexity, and structural inefficiencies that make maintenance increasingly difficult. This challenge is particularly evident in legacy systems that were developed using architectural paradigms that may no longer align with modern development practices.

In recent years, the rise of microservices architecture has emerged as a prominent strategy for addressing these challenges. Microservices architecture emphasizes the decomposition of large monolithic systems into smaller, independently deployable services that communicate through well-defined interfaces. This architectural style offers several advantages, including improved scalability, flexibility, and maintainability (Jamshidi et al., 2018). However, migrating from monolithic architectures to microservices introduces significant technical and organizational challenges.

One of the most critical challenges in microservice migration involves determining appropriate service boundaries. Identifying how to decompose a monolithic system into coherent services requires deep understanding of system functionality, dependencies, and organizational requirements. Machine learning techniques have recently been proposed as tools for assisting in this process by analyzing code structures and identifying logical boundaries within legacy systems (Hebbar, 2022).

At the same time, artificial intelligence technologies have begun to play an increasingly important role in software engineering practices. AI-driven tools can analyze large code repositories, identify patterns in software development processes, and provide recommendations for improving software quality and maintainability. Research has shown that machine learning techniques can support various aspects of software engineering, including defect detection, software reuse, and automated code generation (Brown et al., 2020; Williams and Lee, 2017).

The integration of artificial intelligence into software engineering represents a significant paradigm shift in the field. Traditional software development processes have largely relied on manual design decisions and human expertise. While these approaches remain essential, the increasing complexity of software systems has created a need for automated tools that can assist developers in analyzing and managing large-scale systems. Artificial intelligence provides a powerful set of techniques for addressing this need.

Another important dimension of software engineering research concerns knowledge management within development organizations. As software projects grow in scale and complexity, the ability to capture, organize, and reuse knowledge becomes increasingly important. Effective knowledge management practices enable organizations to leverage past experience and improve the efficiency of future development efforts (Garcia et al., 2022). Artificial intelligence technologies can play a key role in supporting knowledge management by analyzing documentation, code repositories, and development histories to extract valuable insights.

Software reuse represents another important area where artificial intelligence can contribute significantly. The ability to reuse existing software components reduces development time, improves reliability, and promotes standardization across projects. Research on AI-based approaches to software reuse has demonstrated that machine learning techniques can identify reusable components and recommend their integration into new systems (Davis et al., 2021).

The emergence of software intelligence as a research field further reflects the growing importance of AI-driven approaches in software engineering. Software intelligence refers to the use of artificial intelligence techniques to analyze software artifacts and generate actionable insights that improve development processes and system quality (Harris et al., 2023). This concept encompasses a wide range of applications, including predictive analytics for software quality, automated architectural analysis, and intelligent development assistants.

Despite the growing interest in AI-driven software engineering, several theoretical and practical challenges remain. One of the most important challenges involves integrating AI technologies with existing software development methodologies. Software engineering is not solely a technical discipline but also a socio-technical process involving collaboration among developers, stakeholders, and organizations. The successful adoption of AI-driven tools therefore

requires careful consideration of organizational structures and innovation processes.

Research on organizational innovation provides valuable insights into how new technologies can be integrated into complex organizational environments. Studies on innovation management emphasize the importance of aligning technological capabilities with organizational structures and cultural practices (Tushman and Nadler, 1986). These insights are particularly relevant for the adoption of AI-driven development tools, which often require significant changes in workflows and decision-making processes.

Another important challenge involves evaluating the reliability and effectiveness of AI-based software engineering tools. As with any automated system, it is essential to ensure that AI-driven recommendations are accurate, transparent, and aligned with engineering best practices. Techniques for measuring agreement among evaluators, such as statistical measures used in qualitative research validation, provide useful methodologies for assessing the reliability of AI-generated insights (Landis and Koch, 1977).

Qualitative research methods also play an important role in understanding the human and organizational dimensions of AI adoption in software engineering. Focus group interviews and similar qualitative research techniques enable researchers to explore how developers and organizations perceive and interact with AI-driven tools (Dilshad and Latif, 2013). These methods provide valuable insights into the practical challenges and opportunities associated with integrating artificial intelligence into development processes.

Given the diverse range of technological and organizational factors involved in AI-driven software engineering, there is a need for comprehensive research that integrates theoretical insights from multiple domains. Such research can provide a holistic understanding of how artificial intelligence technologies influence software evolution, architectural transformation, and development practices.

This study addresses this need by developing a theoretical framework that integrates classical theories of software evolution with contemporary research on artificial intelligence and machine learning in software engineering. The research examines how AI technologies can support the evolution of software systems, facilitate architectural migration toward microservices, and enhance knowledge management and software reuse practices.

The objectives of this research include analyzing the relationship between software evolution laws and modern AI-driven development practices, examining the role of machine learning in supporting architectural transformation and system modularization, exploring how AI technologies contribute to knowledge management and software reuse, and identifying challenges and future research directions for intelligent software engineering systems.

Through this comprehensive theoretical investigation, the study aims to contribute to the ongoing development of intelligent software engineering methodologies capable of addressing the challenges of modern software ecosystems.

Methodology

The methodological framework adopted in this research is grounded in a qualitative and theoretical research approach designed to synthesize insights from foundational software engineering theories and contemporary research on artificial intelligence applications in software development. Rather than relying on experimental datasets or computational simulations, the study emphasizes conceptual integration and interpretive analysis of scholarly literature. This approach is particularly suitable for examining complex interdisciplinary phenomena such as the intersection of software evolution theory, artificial intelligence technologies, and architectural transformation strategies.

The methodological design involves several interconnected stages that collectively enable the development of a comprehensive theoretical framework. These stages include literature synthesis, conceptual analysis, interpretive evaluation, and qualitative validation of theoretical insights. By systematically examining research contributions from multiple domains, the study constructs an integrative perspective on how artificial intelligence and machine learning influence the evolution of software systems and architectural design practices.

The first stage of the methodology involves an extensive review and synthesis of relevant literature. The selected references represent foundational contributions to software evolution theory, contemporary research on artificial intelligence applications in software engineering, and studies related to architectural migration toward microservices. These works collectively provide the theoretical and empirical foundations necessary for constructing an integrated conceptual model.

The literature synthesis process begins with the examination of classical theories of software evolution. Lehman's laws of software evolution provide a fundamental framework for understanding how software systems evolve in response to changing requirements and environmental conditions (Lehman, 1980). According to these laws, large software systems that interact with real-world environments must undergo continuous adaptation to remain useful and relevant. Failure to evolve results in progressive decline in system value and usability.

Lehman's subsequent work revisited these laws and emphasized the importance of feedback mechanisms in software development processes (Lehman, 1996). Software evolution is not a linear process but rather a dynamic interaction between technical systems, organizational structures, and user requirements. This perspective highlights the complexity of managing long-term software evolution and underscores the need for tools that can support adaptive system maintenance.

Complementing the study of software evolution theory is the examination of research paradigms in computer science. Wegner's work on research paradigms provides valuable insights into the methodological foundations of computing research and the importance of interdisciplinary approaches (Wegner, 1976). This perspective informs the methodological design of the present study by emphasizing the integration of theoretical and applied research perspectives.

The second stage of the methodology focuses on analyzing contemporary research related to artificial intelligence and machine learning in software engineering. Machine learning techniques have been increasingly applied to software development tasks such as defect detection, code analysis, architectural decision support, and software reuse. Studies examining these applications highlight the potential of AI technologies to enhance developer productivity and improve software quality (Brown et al., 2020; Williams and Lee, 2017).

In addition to technical applications, the methodology also considers research on knowledge management within software engineering organizations. Effective knowledge management is essential for capturing and disseminating expertise across development teams. Artificial intelligence technologies can facilitate this process by analyzing large repositories of software artifacts and extracting relevant knowledge for reuse in future projects (Garcia et al., 2022).

Another important methodological component involves examining research related to architectural migration and microservices adoption. Microservices architecture represents a significant shift in how large software systems are designed and maintained. Instead of building monolithic applications, developers decompose systems into smaller services that can be independently developed and deployed (Jamshidi et al., 2018). This architectural paradigm offers numerous advantages but also introduces challenges related to system decomposition and service boundary identification.

Research examining the migration from monolithic architectures to microservices highlights the complexity of this transformation process. Architectural migration often requires extensive analysis of system dependencies, organizational structures, and technical constraints (Capuano and Muccini, 2022). Artificial intelligence techniques can assist in this process by analyzing system structures and recommending optimal decomposition strategies.

The methodology also incorporates studies examining the relationship between architectural migration and technical debt. Technical debt refers to the accumulation of design and implementation decisions that may simplify short-term development but create long-term maintenance challenges. Migrating to microservices architecture may reduce technical debt by promoting modularity and clearer service boundaries (Lenarduzzi et al., 2020).

To complement the theoretical analysis of technical literature, the methodology integrates qualitative research perspectives. Qualitative methods provide valuable insights into the human and organizational dimensions of software engineering practices. Focus group interviews, for example, allow researchers to explore how developers perceive and interact with AI-driven tools and architectural transformation strategies (Dilshad and Latif, 2013).

Qualitative validation techniques are also used to assess the reliability of interpretive findings. Methods for measuring agreement among evaluators provide useful frameworks for ensuring consistency in qualitative research interpretations (Landis and Koch, 1977). Although the present study does not involve empirical data collection, these validation frameworks inform the interpretive analysis by emphasizing methodological rigor and transparency.

The final stage of the methodology involves conceptual integration. Insights derived from the literature analysis are synthesized into a coherent theoretical framework that explains how artificial intelligence technologies influence software evolution and architectural transformation. This integrative approach enables the identification of key themes

and relationships among different research domains.

Through this multi-stage methodological approach, the study constructs a comprehensive theoretical understanding of intelligent software engineering practices. The following sections present the results and discussion derived from this integrative analysis.

Results

The integrative analysis conducted in this research reveals several significant patterns regarding the interaction between artificial intelligence technologies, software evolution principles, and architectural transformation strategies. The findings demonstrate that the convergence of machine learning techniques and classical software engineering theories has begun to reshape how developers approach the design, maintenance, and evolution of complex software systems.

One of the most important results emerging from the literature synthesis is the confirmation that the principles of software evolution articulated by Lehman remain highly relevant in contemporary software engineering environments. Modern software systems continue to exhibit the dynamic characteristics described in the laws of software evolution, including continuous growth, increasing complexity, and the necessity for ongoing adaptation (Lehman, 1980). However, the tools available to manage these evolutionary processes have expanded significantly with the introduction of artificial intelligence technologies.

Machine learning algorithms now enable automated analysis of software artifacts at scales that were previously impractical. These capabilities allow developers to detect patterns in codebases, development histories, and system architectures that may indicate emerging maintenance challenges or architectural weaknesses. By identifying such patterns early in the software lifecycle, AI-driven tools can support proactive decision-making and reduce the risk of uncontrolled complexity growth.

Another important finding concerns the relationship between artificial intelligence and software reuse. Research indicates that machine learning techniques can significantly improve the identification and classification of reusable software components (Davis et al., 2021). By analyzing structural and semantic similarities among software artifacts, AI systems can recommend components that may be reused in new development contexts. This capability enhances development efficiency and promotes the creation of standardized architectural patterns.

The analysis also highlights the role of artificial intelligence in supporting knowledge management within software development organizations. Large software projects generate extensive documentation, code repositories, and development histories that collectively represent valuable organizational knowledge. AI-based knowledge management systems can analyze these resources to extract insights that inform future development efforts (Garcia et al., 2022). These insights may include best practices for architectural design, recurring patterns in software defects, and effective strategies for managing system evolution.

Another key result concerns the role of artificial intelligence in architectural migration processes. The transition from monolithic architectures to microservices has become a prominent trend in modern software engineering. This transformation is driven by the need for greater scalability, flexibility, and maintainability in large-scale software systems (Jamshidi et al., 2018). However, the migration process is often complex and resource-intensive.

Machine learning techniques can assist developers in identifying appropriate service boundaries within legacy systems. By analyzing code dependencies, communication patterns, and functional relationships among system components, AI-driven tools can recommend decomposition strategies that align with microservices architectural principles (Hebbar, 2022). These recommendations provide valuable guidance for architects seeking to restructure large systems without introducing unnecessary fragmentation.

The results also reveal that migrating to microservices architecture may contribute to the reduction of technical debt in certain contexts. Technical debt often arises from tightly coupled components and poorly defined system boundaries. Microservices architectures encourage modularity and clearer separation of concerns, which can improve system maintainability (Lenarduzzi et al., 2020). However, the benefits of microservices migration depend heavily on the quality of the decomposition strategy and the organizational readiness to manage distributed systems.

Another significant finding relates to the organizational dimension of AI adoption in software engineering. Research on innovation management suggests that successful integration of new technologies requires alignment between technological capabilities and organizational structures (Tushman and Nadler, 1986). AI-driven software engineering

tools must therefore be integrated into development workflows in ways that complement existing processes rather than disrupt them.

Finally, the analysis indicates that the emergence of software intelligence as a research domain reflects a broader transformation in software engineering practice. Software intelligence systems combine artificial intelligence techniques with traditional software engineering methodologies to create development environments capable of analyzing and improving their own processes (Harris et al., 2023). These systems represent an important step toward the realization of adaptive software ecosystems capable of continuous self-improvement.

Discussion

The findings of this research highlight the profound implications of integrating artificial intelligence and machine learning technologies into software engineering practices. The intersection between classical theories of software evolution and modern AI-driven development tools reveals a significant shift in how software systems are conceptualized, managed, and transformed over time. This shift reflects not only technological advancements but also deeper changes in the epistemological foundations of software engineering as a discipline.

One of the most significant theoretical implications concerns the reinterpretation of Lehman's laws of software evolution in the context of intelligent development environments. Lehman's original formulation emphasized that software systems evolve continuously in response to changing requirements and environmental conditions (Lehman, 1980). The laws also highlighted the inevitability of increasing complexity as systems grow and adapt. In traditional software development environments, managing this complexity required substantial human expertise and manual analysis.

Artificial intelligence technologies introduce new mechanisms for managing software evolution by enabling automated analysis of system structures, development histories, and operational behaviors. Machine learning algorithms can identify patterns in software artifacts that indicate emerging complexity or architectural weaknesses. These insights allow developers to intervene earlier in the evolution process, potentially preventing the uncontrolled growth of technical debt and system fragility.

This capability aligns with Lehman's later observations regarding the importance of feedback loops in software development processes (Lehman, 1996). AI-driven tools effectively enhance these feedback mechanisms by providing continuous analysis of software systems and generating actionable insights for developers. In this sense, artificial intelligence functions as an extension of the evolutionary feedback systems described in classical software evolution theory.

Another important dimension of the discussion concerns the role of artificial intelligence in facilitating architectural transformation, particularly in the context of microservices migration. The transition from monolithic architectures to microservices represents a major structural change in how software systems are organized and maintained. While microservices architectures offer significant advantages in terms of scalability and modularity, they also introduce new complexities related to distributed communication, service orchestration, and operational management (Jamshidi et al., 2018).

The process of decomposing a monolithic system into microservices requires careful analysis of functional dependencies, data flows, and organizational requirements. Artificial intelligence techniques provide powerful tools for performing this analysis by examining large volumes of code and identifying clusters of functionality that may correspond to potential service boundaries. Machine learning-assisted service boundary detection techniques represent an important step toward automating aspects of architectural design (Hebbar, 2022).

However, the discussion also highlights several limitations associated with relying on AI-driven architectural recommendations. Software architecture decisions often involve trade-offs that extend beyond purely technical considerations. Factors such as organizational structure, team expertise, regulatory requirements, and long-term strategic objectives play important roles in architectural design. While machine learning algorithms can analyze technical dependencies, they may not fully capture the broader socio-technical context in which architectural decisions are made.

This observation underscores the importance of viewing artificial intelligence as a supportive tool rather than a replacement for human expertise in software engineering. AI-driven systems can augment the analytical capabilities of architects and developers, but final decision-making authority must remain with human stakeholders who possess a comprehensive understanding of organizational objectives and contextual constraints.

Another important theme emerging from the discussion is the relationship between artificial intelligence and software reuse. Software reuse has long been recognized as a key strategy for improving development efficiency and reducing redundancy. However, identifying reusable components within large codebases has historically been a challenging task due to the complexity and heterogeneity of software artifacts.

Machine learning techniques offer new approaches to this problem by enabling automated classification and clustering of software components based on structural and semantic characteristics. These capabilities allow AI systems to identify potential reuse opportunities that may not be immediately apparent to human developers (Davis et al., 2021). As a result, organizations can leverage existing software assets more effectively and accelerate development processes.

The discussion also explores the implications of artificial intelligence for knowledge management within software development organizations. Large-scale software projects generate vast amounts of information, including code repositories, design documents, issue tracking records, and operational logs. Managing this information effectively is essential for maintaining organizational knowledge and supporting informed decision-making.

AI-based knowledge management systems can analyze these information sources and extract insights that support software development activities. For example, machine learning algorithms can identify recurring patterns in software defects, recommend architectural patterns based on historical success rates, and provide contextual guidance to developers during implementation tasks (Garcia et al., 2022). These capabilities enhance organizational learning and promote the accumulation of institutional knowledge.

Another important aspect of the discussion concerns the organizational challenges associated with adopting artificial intelligence technologies in software engineering. The successful integration of AI-driven tools requires more than technical implementation; it also involves significant changes in organizational culture, workflows, and decision-making processes. Research on innovation management emphasizes that technological innovations must be aligned with organizational structures and practices to achieve sustainable adoption (Tushman and Nadler, 1986).

For example, developers must develop trust in AI-generated recommendations before incorporating them into critical development decisions. This trust can be fostered through transparent algorithms, clear explanations of model outputs, and continuous evaluation of system performance. Additionally, organizations must provide training and support to help developers understand how to use AI-driven tools effectively.

The discussion also acknowledges methodological limitations in the current body of research on AI-driven software engineering. Many studies focus primarily on technical performance metrics without fully addressing the human and organizational factors that influence technology adoption. Qualitative research methods, such as focus group interviews, can provide valuable insights into these dimensions by exploring how developers perceive and interact with AI-driven tools (Dilshad and Latif, 2013).

Another limitation involves the need for robust evaluation frameworks to assess the reliability of AI-generated insights. Techniques for measuring observer agreement in qualitative research provide useful analogies for evaluating consistency in AI system outputs (Landis and Koch, 1977). Future research should explore methods for validating AI recommendations through collaborative evaluation processes involving both automated systems and human experts.

Looking forward, the future of intelligent software engineering is likely to involve increasingly sophisticated interactions between human developers and AI-driven development environments. These environments may incorporate advanced software intelligence systems capable of continuously analyzing system performance, predicting maintenance needs, and recommending architectural improvements.

Such systems represent an important step toward the realization of adaptive software ecosystems that evolve dynamically in response to changing requirements and operational conditions. However, achieving this vision will require continued research into the theoretical foundations of software evolution, the development of robust AI algorithms, and the design of organizational processes that support collaborative human-AI interaction.

Conclusion

The integration of artificial intelligence and machine learning technologies into software engineering represents one of the most significant developments in the evolution of the discipline. This research has examined how classical theories of software evolution intersect with contemporary AI-driven development practices, providing a comprehensive theoretical framework for understanding intelligent software engineering systems.

The analysis confirms that the principles of software evolution articulated by Lehman remain highly relevant in modern development environments. Software systems continue to evolve continuously in response to changing requirements, and managing this evolution remains a central challenge for developers and organizations. Artificial intelligence technologies offer powerful tools for addressing this challenge by enabling automated analysis of software artifacts, architectural structures, and development processes.

Machine learning techniques have demonstrated significant potential in areas such as software reuse, knowledge management, defect detection, and architectural migration. These technologies enable developers to analyze large codebases more effectively and identify patterns that inform better design and maintenance decisions. In particular, AI-driven approaches to service boundary detection provide valuable support for organizations seeking to migrate legacy systems toward microservices architectures.

The study also highlights the importance of organizational and human factors in the adoption of AI-driven software engineering tools. Successful integration of these technologies requires alignment between technological capabilities and organizational structures, as well as the development of trust and transparency in AI-generated recommendations.

While artificial intelligence offers powerful capabilities, it should be viewed as a complement to human expertise rather than a replacement for it. Architectural design decisions and strategic development planning involve complex socio-technical considerations that extend beyond purely technical analysis.

Future research should continue exploring the development of intelligent software ecosystems that combine automated analysis with human creativity and judgment. By integrating classical software engineering theory with modern artificial intelligence technologies, the discipline can move toward a new paradigm of adaptive, knowledge-driven software development.

References

1. Brown, M., White, L., & Green, R. (2020). Machine learning in software engineering. Journal of Systems and Software.

2. Capuano, R., & Muccini, H. (2022). A systematic literature review on migration to microservices: A quality attributes perspective. IEEE International Conference on Software Architecture Companion.

3. Davis, E., Wilson, B., & Lee, S. (2021). Artificial intelligence for software reuse: A literature review. ACM Computing Surveys.

4. Dilshad, R. M., & Latif, M. I. (2013). Focus group interview as a tool for qualitative research: An analysis. Pakistan Journal of Social Sciences.

5. Garcia, R., Martinez, P., & Clark, D. (2022). Knowledge management in software engineering: State of the art and future trends. Journal of Systems and Software.

6. Harris, K., Thompson, S., & Miller, J. (2023). Software intelligence: Concepts, applications, and challenges. IEEE Transactions on Software Engineering.

7. K. S. Hebbar, "MACHINE LEARNING-ASSISTED SERVICE BOUNDARY DETECTION FOR MODULARIZING LEGACY SYSTEMS," International Journal of Applied Engineering & Technology, vol. 04, no.02, pp. 401-414, Sep. 2022, https://romanpub.com/resources/ijaet-v4-2-2022-48.pdf

8. Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software.

9. Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. Biometrics.

10. Lehman, M. M. (1980). Programs, life cycles and the laws of software evolution. Proceedings of the IEEE.

11. Lehman, M. M. (1996). Laws of software evolution revisited. Proceedings of the European Workshop on Software Process Technology.

12. Lenarduzzi, V., Lomio, F., Saarimäki, N., & Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt? Journal of Systems and Software.

13. Slamaa, A. A., El-Ghareeb, H. A., & Saleh, A. A. (2021). A roadmap for migration system-architecture decision by neutrosophic-ANP and benchmark for enterprise resource planning systems. IEEE Access.

14. Smith, T., & Brown, K. (2018). AI-driven software development: A case study. Journal of Software and Systems Engineering.

15. Tushman, M., & Nadler, D. (1986). Organizing for innovation. California Management Review.

16. Wegner, P. (1976). Research paradigms in computer science. International Conference on Software Engineering.

17. Williams, R., & Lee, C. (2017). The role of machine learning in modern software engineering. IEEE Software.