RESEARCH ARTICLE OPEN ACCESS

An Exploratory Study on Framework-Oriented AI Support for Secure SDLC Practices

G Abin Roy*, Vaishnav A Nair**, Ajesh R***, John Merfin****, Govind H*****, Karthika R S******

*(Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: godvinroyt16262@gmail.com)

** (Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: vaishnavanair@gmail.com)

*** (Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: ajeshr004@gmail.com)

**** (Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: johnmerfintvm@gmail.com)

***** (Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: govind4936u@gmail.com)

****** (Computer Science & Engineering, St Thomas Institute for Science & Technology, APJ Abdul Kalam Technological University (KTU), Trivandrum
Email: karthika.panicker88@gmail.com)

----------**********----------

Abstract:

Modern software systems are becoming increasingly complex, while security issues continue to grow across different stages of the Software Development Lifecycle (SDLC). In many development environments, security and analytical support are often addressed late in the process, which increases risk and development effort. Although artificial intelligence has shown promise in supporting software engineering activities, most existing approaches focus on specific tasks or individual SDLC phases rather than the entire lifecycle.

This paper proposes a conceptual framework for AI-assisted support in the Secure Software Development Lifecycle (Secure SDLC). The framework is developed through a review of existing research studies, industry practices, and secure development standards. Instead of presenting a working system or experimental results, the study focuses on defining high-level framework components and explaining their conceptual roles across SDLC phases such as requirements, design, development, testing, deployment, and maintenance. The proposed framework aims to support security awareness and decision-making without disrupting existing development processes.

The main contribution of this work is an implementation-independent framework that connects secure SDLC principles with AI-assisted analytical concepts. This study serves as a foundation for future research, system development, and practical validation in real-world software engineering environments.

Keywords — **Secure Software Development Lifecycle, Secure SDLC, Artificial Intelligence, AI-Assisted Analysis, Software Engineering, Decision Support Framework, Conceptual Framework**

----------**********----------

I. INTRODUCTION

Software systems play a critical role in modern society, supporting applications in finance, healthcare, education, government, and industrial sectors. As software systems grow in scale and complexity, security threats have also increased in frequency and sophistication. Security vulnerabilities introduced during early development stages often propagate throughout the lifecycle, resulting in costly fixes, operational disruptions, and reputational damage.

Traditional Software Development Lifecycle (SDLC) models primarily focus on functionality, performance, and delivery timelines. Security is frequently treated as a secondary concern or addressed only during testing or post-deployment phases. This reactive approach increases development costs and exposes organizations to preventable security risks. To address these challenges, the concept of Secure Software Development Lifecycle (Secure SDLC) has been introduced, integrating security practices across all development phases.

Recent advancements in artificial intelligence (AI) have demonstrated potential in supporting complex decision-making processes in software engineering. AI-assisted analysis can help process large volumes of development artifacts, identify risks, and provide contextual guidance. However, existing AI applications in software development often target isolated tasks and lack lifecycle-wide integration.

This paper addresses this gap by proposing a conceptual AI-assisted framework for Secure SDLC, focusing on high-level analytical support rather than concrete implementation

II. BACKGROUND AND RELATED WORK

The Software Development Lifecycle (SDLC) provides a structured approach for designing, developing, testing, and maintaining software systems; however, traditional SDLC models often treat security as a secondary concern addressed late in the development process. This limitation has led to increased interest in the Secure Software Development Lifecycle (Secure SDLC), which integrates security practices such as secure requirements, threat modeling, secure coding, and security testing across all lifecycle phases. Previous studies have shown that early incorporation of security can significantly reduce vulnerabilities and long-term remediation costs. In parallel, recent research has explored the application of artificial intelligence in software engineering to support tasks such as defect prediction, documentation analysis, and risk assessment. While these approaches demonstrate the potential of AI-assisted analysis, most existing solutions focus on isolated SDLC activities and lack a unified, lifecycle-wide perspective. Consequently, there remains a gap in research for conceptual frameworks that systematically align AI-assisted analytical support with Secure SDLC principles, motivating the need for high-level, implementation-independent models.

A. Secure Software Development Lifecycle

Secure SDLC extends traditional SDLC by embedding security practices into each phase of software development. These practices include secure requirement identification, threat modeling, secure coding standards, security testing, and continuous monitoring. Secure SDLC emphasizes early risk identification and proactive mitigation rather than reactive patching.

Despite its advantages, Secure SDLC adoption remains inconsistent, particularly in environments with limited resources or tight delivery schedules. Many organizations struggle to operationalize security practices consistently across development phases.

B. Artificial Intelligence in Software Engineering

AI has been applied to various software engineering tasks, including defect prediction, test automation, effort estimation, and documentation analysis. Machine learning and natural language processing techniques have enabled automated analysis of code, requirements, and project artifacts.

However, most AI-based approaches focus on specific development activities rather than providing holistic lifecycle support. The lack of unified conceptual models limits their adaptability and long-term effectiveness.

C. Research Gap

Existing research highlights two major gaps:

- Limited integration between Secure SDLC practices and AI-assisted analysis.
- Lack of lifecycle-wide, implementation-independent frameworks that guide AI support across SDLC phases.

This study aims to address these gaps by proposing a conceptual framework that aligns AI-assisted analysis with Secure SDLC principles.

III. RESEARCH MOTIVATION AND OBJECTIVES

The increasing scale, complexity, and security demands of modern software systems have made software development a highly decision-intensive process. Development teams are required to balance functional requirements, security considerations, regulatory compliance, time constraints, and resource limitations across multiple phases of the Software Development Lifecycle (SDLC). In practice, these decisions are often made with incomplete information, fragmented tools, and limited visibility into security implications, which can lead to inconsistent implementation of secure development practices.

While the Secure Software Development Lifecycle (Secure SDLC) promotes early and continuous integration of security, its effective adoption remains challenging in real-world environments. Developers and project stakeholders frequently lack contextual support that connects security principles with day-to-day development activities. At the same time, recent advancements in artificial intelligence have highlighted opportunities to assist complex analytical and decision-making tasks by processing large volumes of software artifacts and development-related information. However, existing AI-driven approaches in software engineering are typically narrow in scope and do not provide cohesive lifecycle-wide support.

These challenges motivate the need for a structured and conceptual approach that aligns AI-assisted analytical support with Secure SDLC practices. By defining clear objectives and scope, this research aims to establish a foundation for integrating intelligent support mechanisms into secure software development processes in a systematic and adaptable manner.

D. Research Motivation

Modern software development involves complex decision-making across multiple phases of the Software Development Lifecycle (SDLC), where security, quality, cost, and delivery timelines must be balanced simultaneously. In many development environments, security considerations are still introduced late in the lifecycle, leading to increased vulnerabilities and higher remediation costs. Although Secure Software Development Lifecycle (Secure SDLC) practices aim to address this issue, their consistent adoption remains challenging due to fragmented processes, limited expertise, and increasing system complexity. At the same time, recent advances in artificial intelligence have demonstrated the ability to analyze large volumes of software-related information and provide contextual insights to support human decision-making. However, most AI-based solutions in software engineering focus on narrow tasks or individual lifecycle phases, offering limited support for end-to-end Secure SDLC integration. This gap motivates the need for a unified, high-level framework that conceptually integrates AI-assisted analytical support across the entire Secure SDLC without imposing implementation constraints.

E. Research Objective

The primary objectives of this research are to identify the key factors influencing the subject under study and to analyze their impacts in a systematic manner. Additionally, the research aims to provide evidence-based insights that can inform future practice and decision-making.

- To analyze existing Secure SDLC practices and AI-assisted approaches in software engineering.
- To propose a conceptual, implementation-independent framework for AI-assisted support in Secure SDLC.
- To map the roles of AI-assisted analytical support across different SDLC phases.

- To establish a foundational model that can guide future empirical studies, system implementations, and industry validation.

IV. PROPOSED CONCEPTUAL FRAMEWORK

In response to the limitations observed in existing Secure Software Development Lifecycle (Secure SDLC) practices and the fragmented adoption of AI-assisted techniques within software engineering, this study introduces a conceptual framework aimed at providing structured analytical support across the SDLC. The framework is positioned as a complementary layer that enhances security-oriented reasoning and decision-making without disrupting established development methodologies or organizational workflows.

The proposed framework is deliberately designed to be technology-agnostic and implementation-independent, focusing on abstract functional roles rather than concrete system architectures. This design choice ensures broad applicability across varying project contexts, development models, and organizational maturity levels. By emphasizing logical structure over operational detail, the framework facilitates conceptual alignment between AI-assisted analytical capabilities and Secure SDLC activities, enabling consistent integration of security considerations throughout the lifecycle.

This section establishes the conceptual basis and scope of the proposed framework, serving as a precursor to the detailed presentation of its overall structure and constituent components in the following subsections.

A. Framework Overview

The proposed framework represents a high-level analytical support layer designed to complement and enhance existing Secure Software Development Lifecycle (Secure SDLC) practices. Rather than defining a concrete system or technical implementation, the framework is formulated as a logical and conceptual model, allowing it to remain independent of specific technologies, platforms, or development tools. This abstraction ensures that the framework can be adapted across diverse development environments, organizational structures, and software engineering methodologies.

The primary objective of the framework is to support informed decision-making by improving security awareness and risk comprehension throughout the SDLC. It provides a structured conceptual approach for integrating analytical reasoning into each lifecycle phase, enabling stakeholders to better understand potential security implications associated with design choices, development activities, and operational decisions. By maintaining a lifecycle-wide perspective, the framework promotes consistency in security consideration and encourages proactive risk identification rather than reactive mitigation.

B. Framework Components

The proposed conceptual framework is composed of a set of logical and interrelated components, each representing a distinct analytical role within the Secure Software Development Lifecycle (Secure SDLC). These components are not intended to denote physical modules or system implementations; instead, they define abstract functional layers that collectively support security-aware decision-making across the software lifecycle.

Knowledge Layer:

The Knowledge Layer serves as the foundational component of the framework, encapsulating domain knowledge related to software engineering principles, secure development standards, regulatory guidelines, and established best practices. This layer is derived from academic literature, industry frameworks, and recognized security standards, providing a structured knowledge base that informs analytical and decision-support activities throughout the SDLC.

Analysis Layer:

The Analysis Layer is responsible for the conceptual examination of software development artifacts, processes, and lifecycle activities. It facilitates the identification of potential security risks, design inconsistencies, and process-level weaknesses by applying analytical reasoning to

contextual development information. This layer supports stakeholders in understanding security implications without relying on specific analytical techniques or tools.

Recommendation Layer:

The Recommendation Layer provides high-level guidance and security-oriented suggestions that are aligned with individual SDLC phases. Rather than enforcing prescriptive actions, this layer offers contextual recommendations intended to assist developers and decision-makers in selecting appropriate security practices and mitigating identified risks in a flexible and adaptable manner.

Feedback Layer:

The Feedback Layer supports continuous improvement by conceptually integrating insights and lessons learned across different stages of the SDLC. By enabling iterative reflection on past decisions and outcomes, this layer contributes to the refinement of security awareness and analytical understanding over time, reinforcing the lifecycle-wide applicability of the framework.

V. MAPPING THE FRAMEWORK TO SECURE SDLC PHASES

The conceptual framework supports Secure SDLC phases as follows:

TABLE I
AI-Assisted Security Roles Across the SDLC

SDLC Phase	Conceptual AI-Assisted Role
Requirements	Security awareness and requirement validation
Design	Threat identification and architectural risk analysis
Development	Secure coding guidance and consistency check
Testing	Vulnerability prioritization support
Deployment	Configuration and compliance awareness
Maintenance	Continuous security monitoring guidance

This mapping demonstrates how AI-assisted concepts can enhance Secure SDLC practices without altering existing workflows.

VI. DISCUSSION

The proposed conceptual framework places strong emphasis on **clarity, adaptability, and generalizability**, which are essential characteristics for supporting Secure Software Development Lifecycle (Secure SDLC) practices across diverse development environments. By maintaining an implementation-agnostic design, the framework avoids reliance on specific technologies, programming paradigms, or analytical tools. This design choice enhances its applicability across organizations with varying technical infrastructures, resource availability, and security maturity levels, enabling flexible adoption without imposing rigid implementation constraints.

Rather than functioning as a replacement for developers, architects, or security professionals, the framework is positioned as a **supportive decision-assistance construct**. It aims to augment human judgment by promoting structured analytical reasoning and continuous security awareness throughout the SDLC. By encouraging consistent consideration of security implications at each lifecycle phase, the framework helps reduce fragmented or reactive security practices and supports a more proactive and informed development approach.

Furthermore, the conceptual nature of the framework allows it to evolve alongside emerging technologies, development methodologies, and organizational practices. This adaptability positions the framework as a foundational reference model that can guide future empirical studies, system implementations, and industry-driven refinements, contributing to the advancement of secure and intelligence-aware software engineering practices.

VII. LIMITATIONS

This study is purely conceptual and does not include system implementation, datasets, or experimental validation. The framework has not been empirically evaluated in real-world development environments. As a result, its practical effectiveness remains to be validated through future research.

VIII. FUTURE WORK

While this study establishes a conceptual foundation for AI-assisted support in the Secure Software Development Lifecycle (Secure SDLC), several directions remain open for future investigation. One potential extension of this work involves the implementation of the proposed conceptual framework as a software-based system, enabling practical exploration of its analytical and decision-support capabilities within real development environments.

Future research may also focus on empirical validation through controlled experiments, longitudinal studies, or industry case analyses to assess the framework's effectiveness in improving security awareness, decision quality, and lifecycle consistency. Evaluating the framework across organizations with varying sizes, domains, and security maturity levels would provide deeper insight into its adaptability and practical relevance.

Additionally, subsequent studies could explore the integration of quantitative metrics and automated analytical techniques to support objective evaluation of security risks and development outcomes. Such extensions would enhance the framework's analytical rigor and facilitate measurable assessment, thereby strengthening its applicability for both academic research and industrial adoption.

IX. CONCLUSION

This paper presented a conceptual framework for AI-assisted support in the Secure Software Development Lifecycle. By aligning AI-assisted analytical concepts with Secure SDLC practices, the framework addresses gaps in existing research and provides a structured foundation for future development. The proposed model emphasizes security awareness, decision support, and lifecycle consistency while remaining implementation-independent. This work contributes to advancing secure and intelligence-aware software development practices.

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to God Almighty for granting us the strength, wisdom, and perseverance to successfully complete this work.

We sincerely thank Dr. A. G. Mathew, Principal of St. Thomas Institute for Science and Technology, for his constant encouragement and for providing a supportive academic environment that enabled the successful completion of this research. We also extend our gratitude to Mr. Anup Mathew Abraham, Head of the Department of Computer Science and Engineering, for his valuable guidance and continuous support throughout this project.

We are especially thankful to Ms. Pooja P. Raj, Project Coordinator, for her constructive suggestions, academic supervision, and timely assistance, which were instrumental in shaping this work. We also acknowledge the faculty members of the Department of Computer Science and Engineering for their encouragement and cooperation.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 10th ed., Pearson Education, 2016.
[2] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.
[3] K. Schwaber and J. Sutherland, *The Scrum Guide*, Scrum.org, 2020.
[4] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, vol. 31, no. 5, pp. 23–31, 1998.
[5] T. Menzies and B. Cukic, "When to Test Less," *IEEE Software*, vol. 17, no. 5, pp. 107–112, 2000.
[6] J. Bosch, "Continuous Software Engineering: An Introduction," *IEEE Software*, vol. 31, no. 6, pp. 15–18, 2014.
[7] F. Palomba et al., "Crowdsourcing-Based Code Smell Detection," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 599–615, 2019.
[8] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD*, pp. 785–794, 2016.
[9] S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," *Proceedings of ICSE*, IEEE, 2019.
[10] A. Aurum and C. Wohlin, *Engineering and Managing Software Requirements*, Springer, 2005.

[11] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," *EBSE Technical Report*, Keele University, 2007.
[12] T. Menzies, Z. Milton, B. Turhan, B. Cukic, and Y. Jiang, "Defect Prediction from Static Code Features," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–22, 2007.
[13] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the Reverse Engineering of UML Sequence Diagrams," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 642–663, 2006. → Supports SDLC traceability and modeling.
[14] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society, 2014.
[15] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Search-Based Software Engineering," *Information and Software Technology*, vol. 54, no. 10, pp. 1045–1061, 2012.
[16] F. Chollet, *Deep Learning with Python*, Manning Publications, 2018.
[17] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed., Morgan Kaufmann, 2011.
[18] A. Abran et al., "Software Engineering Measurement," *IEEE Software*, vol. 19, no. 6, pp. 23–28, 2002.
[19] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.
[20] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
[21] P. Jalote, *An Integrated Approach to Software Engineering*, Springer, 2005.
[22] R. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed., McGraw-Hill, 2014.
[23] M. Fowler and J. Highsmith, "The Agile Manifesto," *Software Development Magazine*, 2001.
[24] A. Mockus, D. Weiss, and P. Zhang, "Understanding and Predicting Effort in Software Projects," *IEEE Software*, vol. 20, no. 6, pp. 53–60, 2003.
[25] S. McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.
[26] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
[27] B. W. Boehm et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
[28] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., Addison-Wesley, 2013.
[29] M. Lehman and L. Belady, *Program Evolution: Processes of Software Change*, Academic Press, 1985.
[30] P. Bourque, R. E. Fairley, and I. Society, *SWEBOK Guide V3.0*, IEEE Computer Society, 2014.
[31] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Wiley, 2009.
[32] D. L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.
[33] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed., Wiley, 2011.
[34] N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," *Proceedings of ICSE*, IEEE, 2006.
[35] H. Kagdi, M. L. Collard, and J. I. Maletic, "A Survey and Taxonomy of Approaches for Mining Software Repositories," *Journal of Software Maintenance and Evolution*, vol. 19, no. 2, pp. 77–131, 2007.
[36] T. Mens and S. Demeyer, *Software Evolution*, Springer, 2008.
[37] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2011.
[38] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps*, IT Revolution Press, 2018.
[39] S. McConnell, *Code Complete*, 2nd ed., Microsoft Press, 2004.
[40] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, 3rd ed., Addison-Wesley, 2013.
[41] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
[42] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008.
[43] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed., Pearson, 2015.
[44] J. R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, 1999.
[45] K. Beck et al., *Extreme Programming Explained: Embrace Change*, 2nd ed., Addison-Wesley, 2004.
[46] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.
[47] N. Fenton and M. Neil, *Risk Assessment and Decision Analysis with Bayesian Networks*, CRC Press, 2012.
[48] A. Meneely, B. Smith, and L. Williams, "Validating Software Metrics: A Bayesian Approach," *Proceedings of ESEM*, ACM/IEEE, 2011.
[49] Y. Kamei et al., "A Large-Scale Empirical Study of Just-in-Time Quality Assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.
[50] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 176–192, 2006.
[51] R. K. Yin, *Case Study Research: Design and Methods*, 5th ed., Sage Publications, 2014.
[52] S. Kim, E. J. Whitehead Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.
[53] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.
[54] M. G. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*, Springer, 2006.

