AI-Powered Software Development Life Cycle: From Requirements to Maintenance

Sandeep Burte

O9 Solutions Inc, Bengaluru, India

Email: sburte@gmail.com

Abstract

The integration of artificial intelligence into software development represents a paradigmatic shift with profound implications for productivity, quality, and efficiency in modern software engineering practices. This viewpoint paper aims to examine the comprehensive application of generative AI technologies across all Software Development Life Cycle (SDLC) phases, analyze their transformative potential, and provide strategic insights for successful implementation. Through systematic analysis of recent empirical studies and practical implementations, we conducted a comprehensive review of AI applications spanning requirements analysis, system design, coding, testing, deployment, and maintenance phases. AI demonstrates significant capabilities in automating code generation, enhancing testing strategies, optimizing deployment processes, and enabling proactive maintenance approaches. Key applications include natural language processing for requirements refinement, intelligent architecture recommendations, comprehensive test suite generation, and predictive bug detection. Research Implications: Successful AI integration requires gradual implementation strategies, robust quality assurance mechanisms, and maintaining human oversight for critical decisions. Organizations must balance AI automation with human expertise, addressing ethical considerations and security concerns while fostering continuous learning approaches that align AI capabilities with organizational standards and domain-specific requirements.

Keywords

Artificial Intelligence, Software Development Life Cycle, Human-AI Collaboration, Digital Transformation, Automated Testing

1. Introduction

The software development industry stands at a pivotal moment in its evolution. Traditional methodologies, while proven effective, are increasingly challenged by the demands of rapid delivery, complex requirements, and the need for higher quality software products. The emergence of generative AI and machine learning technologies offers unprecedented opportunities to augment human capabilities throughout the entire SDLC. However, the integration of these technologies requires careful consideration of their applications, limitations, and the fundamental changes they bring to established development practices [1].

This viewpoint paper argues that AI-powered SDLC represents not merely an incremental improvement but a paradigmatic shift in software engineering. The transformation touches every aspect of development, from the initial conceptualization of requirements to the ongoing maintenance of deployed systems. Yet, successful implementation demands a nuanced understanding of where AI excels and where human expertise remains irreplaceable [2].

2. The AI Revolution in Requirements Analysis and Planning

2.1 Transforming Requirements Engineering

Requirements engineering has long been recognized as one of the most critical yet challenging phases of software development. Traditional approaches often struggle with ambiguous stakeholder communications, incomplete specifications, and the translation of business needs into technical requirements. The introduction of natural language processing (NLP) capabilities has begun to address these fundamental challenges in ways previously unimaginable [3].

Modern AI systems can analyze unstructured business descriptions and automatically generate comprehensive functional requirements. For instance, when presented with a simple statement like "We need a customer portal," AI can extrapolate this into detailed specifications encompassing user authentication mechanisms, dashboard functionalities, data management requirements, and integration touchpoints [4]. This capability represents a significant leap forward in bridging the communication gap between business stakeholders and technical teams.

Figure 1 illustrates the transformation process from business descriptions to structured requirements, showing the flow from natural language input through AI processing to generated functional specifications.

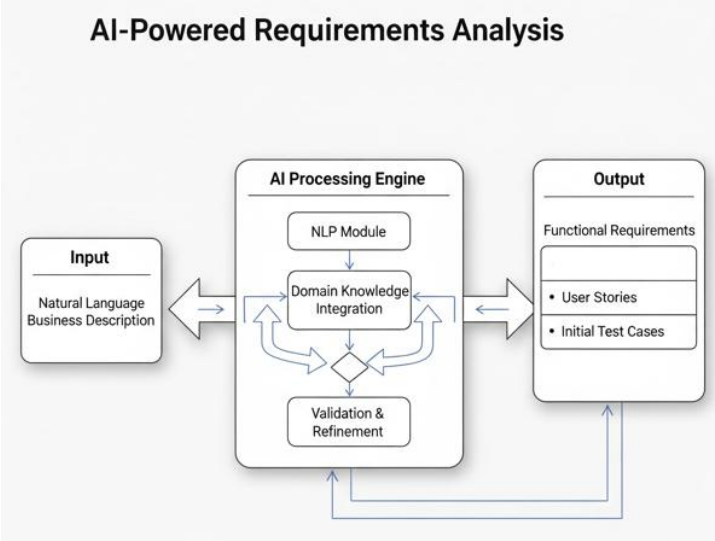

Figure 1. AI-Powered Requirements Analysis Workflow

Source: Authors Own Creation

The automated generation of user stories represents another breakthrough in requirements management. AI systems can transform basic feature descriptions into detailed, actionable user stories that follow established patterns and best practices [5]. Research in software effort estimation demonstrates that machine learning models can significantly improve the accuracy of project planning by analyzing historical data patterns and identifying potential risk factors early in the development cycle [6].

2.2 Predictive Risk Assessment

Perhaps one of the most valuable applications of AI in the planning phase is its ability to perform sophisticated risk assessment and project estimation. By analyzing vast repositories of historical project data, AI systems can identify patterns that human analysts might miss. This predictive capability extends to early test case generation, where AI can anticipate potential edge cases and failure scenarios based on requirements analysis alone [7].

The integration of wise reasoning principles into AI-powered planning tools ensures that technical recommendations are balanced with contextual understanding and domain expertise [8]. This approach prevents the common pitfall of over-relying on purely algorithmic solutions without considering the nuanced realities of software development projects.

3. Intelligent System Design and Architecture

3.1 Architecture Pattern Recognition and Recommendation

System design and architecture represent areas where AI's pattern recognition capabilities offer substantial value. Modern AI systems can analyze project requirements and automatically suggest optimal architectural patterns based on factors such as scalability requirements, team size, complexity constraints, and performance expectations [9]. This capability is particularly valuable for organizations that may lack deep architectural expertise or need to make rapid design decisions.

The recommendation of microservices versus monolithic architectures, for example, can be informed by AI analysis of multiple factors including team structure, deployment requirements, data consistency needs, and operational complexity tolerance. Such recommendations help development teams avoid common architectural antipatterns and establish solid foundations for their applications [10].

3.2 Automated Database and API Design

Database schema generation represents another area where AI demonstrates significant potential. By analyzing data requirements and relationships, AI systems can generate normalized database schemas complete with appropriate indexes, constraints, and relationship definitions [11]. This automation not only accelerates the design process but also helps ensure that database designs follow established best practices and optimization principles.

Similarly, API design automation through tools that generate OpenAPI specifications demonstrates how AI can bridge the gap between functional requirements and technical implementation [12]. These systems can automatically define endpoints, request/response schemas, error codes, and documentation, creating a comprehensive API specification that serves as both implementation guide and communication tool.

Figure 2 shows the decision tree and factors that AI considers when recommending architectural patterns, including inputs like team size, scalability requirements, and complexity metrics, leading to specific architectural recommendations.

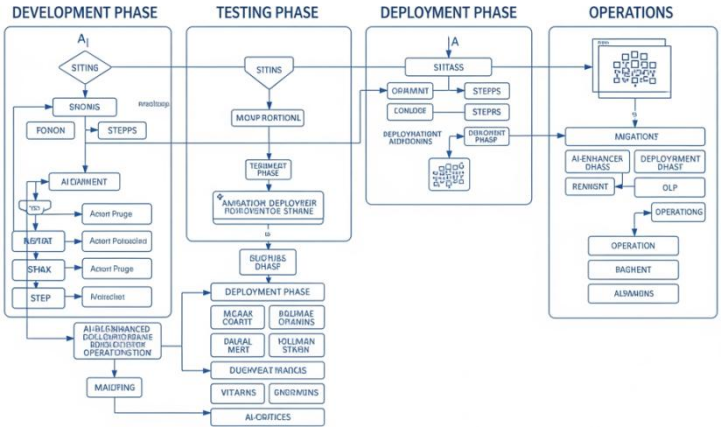

Figure 2. AI-Driven Architecture Decision Framework

Source: Authors Own Creation

4. Code Generation and Implementation Enhancement

4.1 The Evolution of Code Generation

The implementation phase of software development has perhaps seen the most visible transformation through AI integration. Code generation capabilities have evolved from simple template-based approaches to sophisticated systems that can understand context, maintain consistency, and generate production-quality code [13]. The ability to generate entire functions from comments or specifications represents a fundamental shift in how developers approach implementation tasks.

Framework setup and project scaffolding automation exemplifies how AI can eliminate tedious boilerplate creation while ensuring consistency with established patterns and best practices. Whether generating React components, Spring Boot applications, or Django projects, AI systems can create proper folder structures, configuration files, and initial implementations that provide solid starting points for development teams [14].

4.2 Intelligent Code Assistance and Refactoring

Data access layer generation showcases AI's ability to understand entity relationships and automatically create repository classes, Data Access Objects (DAOs), and CRUD operations [15]. This automation not only accelerates development but also ensures consistency in data access patterns across applications. The generated code typically follows established conventions and includes appropriate error handling and optimization strategies.

Unit test generation represents a particularly valuable application where AI can analyze code structure and generate comprehensive test suites covering various scenarios, edge cases, and boundary conditions [2]. The quality of AI-generated tests continues to improve as systems become better at understanding code semantics and identifying potential failure points.

Figure 3 illustrates the flow from requirements/specifications through AI-powered code generation, automated testing, and quality checks, showing how AI enhances each step of the implementation process.

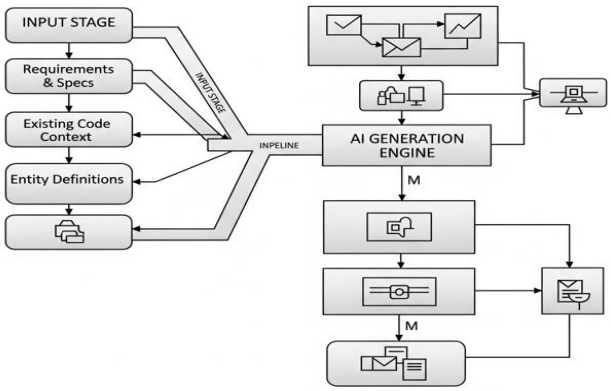

Figure 3. Code Generation and Quality Assurance Pipeline

Source: Authors Own Creation

5. Advanced Testing Strategies and Automation

5.1 Comprehensive Test Suite Generation

Testing represents one of the most promising areas for AI application in the SDLC. Modern AI systems can analyze codebases and automatically generate comprehensive test suites that cover unit tests, integration tests, and end-to-end testing scenarios [7,16]. The sophistication of these generated tests extends beyond simple happy-path scenarios to include edge cases, boundary conditions, and error handling verification.

Test data synthesis has emerged as a critical capability where AI can generate realistic test datasets that maintain referential integrity while covering various operational scenarios [17]. This includes normal operation data, edge cases, and stress testing conditions. The ability to generate meaningful test data automatically addresses one of the most time-consuming aspects of quality assurance processes.

5.2 Intelligent Test Automation and Optimization

Selenium script generation for web automation represents another breakthrough where AI can analyze UI mockups or existing web pages to create comprehensive automation scripts [7]. These scripts include intelligent element locators, robust test workflows, and adaptive strategies for handling dynamic content. Research in test case prioritization demonstrates how deep learning approaches can optimize testing efficiency by focusing attention on the most critical test scenarios [2].

Performance testing scenario generation showcases AI's ability to model realistic user behavior patterns and create sophisticated load testing scripts [18]. These scenarios can simulate various user interaction patterns, data volumes, and system stress conditions that might not be obvious to human testers but are critical for understanding system behavior under real-world conditions.

The prediction and early detection of potential bugs through code analysis represents a proactive approach to quality assurance [14]. AI systems can analyze code patterns, dependency relationships, and historical bug data to identify areas of increased risk before issues manifest in production environments [1].

6. Deployment Automation and Infrastructure Management

6.1 Infrastructure as Code and Deployment Optimization

Deployment automation through AI represents a significant advancement in DevOps practices. Infrastructure as Code (IaC) generation capabilities enable AI systems to analyze application requirements and automatically generate deployment configurations for various cloud platforms and container orchestration systems [4]. This automation ensures consistency across environments while incorporating best practices for security, scalability, and reliability.

Configuration management automation extends beyond simple template generation to include intelligent parameter optimization based on application characteristics and expected load patterns [19]. AI systems can recommend resource allocations, scaling policies, and monitoring configurations that align with application requirements and organizational constraints.

6.2 Environment Setup and Configuration Consistency

Environment setup automation addresses the common challenge of configuration drift and inconsistency across development, staging, and production environments [20]. AI-powered tools can analyze application dependencies, generate appropriate containerization strategies, and create deployment pipelines that ensure reproducible deployments across different environments.

The integration of continuous integration and deployment practices benefits significantly from AI-powered build failure prediction and optimization [12]. By analyzing historical build data and code change patterns, AI systems can predict potential integration issues and suggest preventive measures before failures occur.

Figure 4 shows the complete deployment pipeline from code commit through AI-powered build optimization, automated testing, infrastructure provisioning, and deployment to production, highlighting AI touchpoints throughout the process.

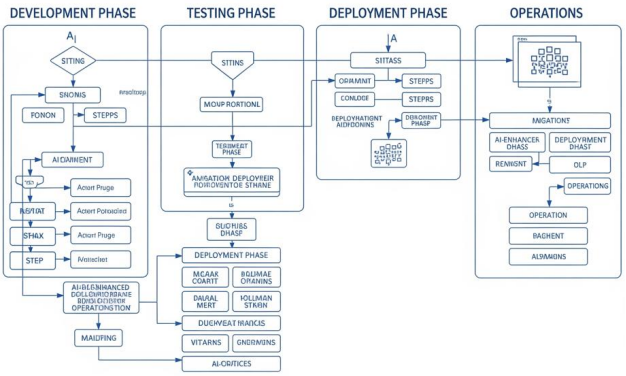

Figure 4. AI-Enhanced Deployment and Operations Pipeline
Source: Authors Own Creation

7. Maintenance and Long-term System Evolution

7.1 Proactive Bug Detection and Resolution

The maintenance phase of software systems has traditionally been reactive, addressing issues as they arise. AI-powered maintenance approaches enable proactive identification of potential problems through log analysis, error pattern recognition, and predictive modeling [1]. These systems can analyze vast amounts of operational data to identify subtle patterns that indicate emerging issues before they impact users.

Performance optimization recommendations generated through AI analysis can identify bottlenecks in code execution, database queries, and system resource utilization [21]. These recommendations often uncover optimization opportunities that might not be apparent through traditional profiling approaches, leading to significant improvements in system performance and resource efficiency.

7.2 Security and Legacy System Modernization

Security vulnerability identification through AI-powered code analysis represents a critical capability for maintaining system integrity [22]. These systems can analyze code patterns, dependency relationships, and known vulnerability databases to identify potential security risks and recommend remediation strategies. The automated nature of this analysis enables continuous security monitoring rather than periodic assessments.

Legacy code modernization through AI assistance addresses one of the most challenging aspects of software maintenance [23]. AI systems can analyze older codebases and suggest refactoring strategies that incorporate modern patterns, update deprecated APIs, and improve overall maintainability while preserving functional behavior.

8. Cross-Cutting AI Applications in SDLC

8.1 Documentation and Knowledge Management

Documentation generation represents a universally beneficial application of AI across all SDLC phases [10]. AI systems can generate comprehensive API documentation, user manuals, and technical specifications by analyzing code, comments, and system behavior. The automation of documentation creation ensures that documentation remains current and comprehensive while reducing the burden on development teams.

Inline code comment generation and README file creation provide immediate value by improving code readability and maintainability [24]. AI systems can analyze code context and generate meaningful comments that explain complex logic, document assumptions, and provide usage examples.

8.2 Code Review and Quality Assurance

Automated code review capabilities enhance traditional peer review processes by providing consistent analysis of code quality, style adherence, and potential issues [3]. AI-powered code review tools can identify security vulnerabilities, performance concerns, and maintainability issues while enforcing organizational coding standards and best practices.

The integration of hyperparameter optimization in machine learning development workflows demonstrates how AI can enhance its own development processes [3]. This creates feedback loops that continuously improve the quality and effectiveness of AI-powered development tools.

8.3 Project Management and Resource Optimization

Sprint planning assistance through AI-powered story point estimation provides more accurate project planning by analyzing historical velocity data, task complexity patterns, and team performance metrics [6,9]. These systems can identify potential bottlenecks and resource constraints before they impact project timelines.

Risk assessment and mitigation planning benefit from AI's ability to analyze multiple project variables simultaneously and identify potential risk factors that might not be apparent through traditional project management approaches [9,15]. Progress tracking and reporting automation provide real-time insights into project health and development velocity [4].

Figure 5 provides a comprehensive view of the entire SDLC with AI integration points, showing how different AI capabilities connect and support each phase, creating a cohesive ecosystem for enhanced software development.

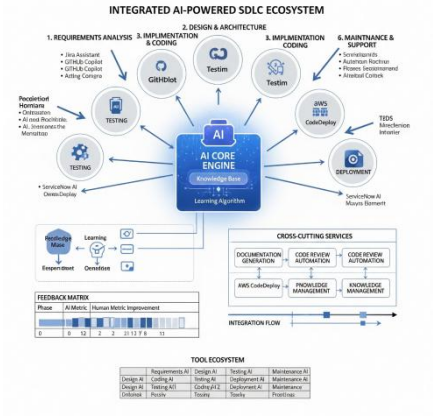

Figure 5. Integrated AI-Powered SDLC Ecosystem
Source: Authors Own Creation

9. Implementation Strategy and Best Practices

9.1 Gradual Integration Approach

Successful AI integration in SDLC requires a thoughtful, gradual approach that begins with low-risk applications and progressively expands to more critical areas [16]. Starting with code completion and documentation generation provides immediate value while allowing teams to develop familiarity with AI-powered tools. This approach minimizes disruption to existing workflows while demonstrating tangible benefits that build organizational confidence in AI capabilities.

Tool integration strategies should focus on seamlessly incorporating AI-powered capabilities into existing development environments [17]. Popular tools like GitHub Copilot, Amazon Code Whisperer, and various testing platforms provide entry points for AI adoption without requiring significant infrastructure changes or workflow disruptions.

9.2 Quality Assurance and Human Oversight

Human oversight remains critical for ensuring that AI-generated content meets quality, security, and business requirements [11]. This is particularly important for critical business logic, security-sensitive code, and architectural decisions that have long-term implications. The implementation of quality gates at each phase ensures that AI-generated content undergoes appropriate review and validation before proceeding to subsequent development phases.

Continuous learning approaches that train AI models on organizational coding patterns, standards, and domain-specific requirements improve the relevance and quality of AI-generated content over time [3,8]. This customization ensures that AI tools align with organizational practices and produce outputs that integrate seamlessly with existing codebases and development standards.

10. Challenges and Considerations

10.1 Technical and Organizational Challenges

Despite the significant benefits of AI integration in SDLC, several challenges must be addressed for successful implementation [18]. Technical challenges include ensuring the reliability and consistency of AI-generated code, managing the computational resources required for AI processing, and integrating AI tools with existing development infrastructure.

Organizational challenges encompass change management, skill development, and the cultural shifts required to effectively leverage AI capabilities [9,13]. Development teams must develop new skills for working with AI tools while maintaining their core technical competencies. Organizations must also address concerns about job displacement and establish clear guidelines for AI usage in critical development activities.

10.2 Ethical and Security Considerations

The use of AI in software development raises important ethical and security considerations [19]. Code generation tools may inadvertently introduce vulnerabilities or biases present in their training data. Organizations must establish robust review processes and security scanning procedures to identify and address potential issues in AI-generated code.

Intellectual property concerns arise when AI tools generate code based on training data that may include proprietary or copyrighted material [20]. Clear policies and legal frameworks are necessary to address ownership and liability issues related to AI-generated content.

11. Future Perspectives and Research Directions

11.1 Emerging Technologies and Capabilities

The future of AI-powered SDLC will likely see continued advancement in natural language understanding, enabling more sophisticated requirements analysis and automated documentation generation [21,24]. Improved code understanding capabilities will enhance AI's ability to work with complex, legacy codebases and provide more accurate refactoring recommendations.

Integration with emerging technologies such as quantum computing, edge computing, and advanced blockchain systems will require AI tools to evolve and adapt to new programming paradigms and architectural patterns [22]. The development of domain-specific AI models tailored to industries or application areas will provide more relevant and accurate assistance for specialized development projects [8,23].

11.2 Research and Development Opportunities

Current research in fault-tolerant systems and advanced testing methodologies points toward more robust and reliable AI-powered development tools [1,2]. The integration of transfer learning and deep learning approaches promises to improve the adaptability and effectiveness of AI tools across different development contexts and organizational environments [3,25].

The development of explainable AI capabilities will enable developers to better understand and validate AI-generated recommendations and code [11]. This transparency is crucial for building trust in AI tools and ensuring that their outputs can be effectively reviewed and maintained by human developers.

12. Conclusion

The integration of AI into the Software Development Life Cycle represents a transformative opportunity that extends far beyond simple automation. While AI excels at pattern recognition, code generation, and optimization tasks, the most successful implementations will be those that thoughtfully combine AI capabilities with human expertise and judgment.

The evidence suggests that AI-powered SDLC can significantly enhance productivity, improve code quality, and accelerate development timelines. However, realizing these benefits requires careful attention to implementation strategies, quality assurance processes, and the fundamental principle that AI should augment rather than replace human capabilities.

Organizations that approach AI integration with realistic expectations, appropriate safeguards, and a commitment to continuous learning will be best positioned to leverage these technologies effectively. The future of software development lies not in the replacement of human developers with AI systems, but in the creation of powerful partnerships that combine the best capabilities of both.

As we move forward, the software development industry must continue to evolve its practices, tools, and methodologies to fully harness the potential of AI while maintaining the critical thinking, creativity, and domain expertise that remain uniquely human. The successful integration of AI into SDLC represents not just a technological advancement, but a fundamental evolution in how we conceive, create, and maintain the software systems that drive our digital world.

The journey toward AI-powered software development is complex and multifaceted, requiring ongoing research, experimentation, and adaptation. However, the potential benefits – improved productivity, higher quality software, and more efficient development processes – make this journey not just worthwhile, but essential for organizations seeking to remain competitive in an increasingly digital landscape.

References

[1] Amin, A. A., Iqbal, M. S., & Shahbaz, M. H. (2023). Development of intelligent fault-tolerant control systems with machine learning, deep learning, and transfer learning algorithms: A review. Expert Systems with Applications, 238, 121956. https://doi.org/10.1016/j.eswa.2023.121956

[2] Behera, A., & Acharya, A. A. (2025). An effective GRU-based deep learning method for test case prioritization in continuous integration testing. Procedia Computer Science, 258, 4070-4083. https://doi.org/10.1016/j.procs.2025.04.658

[3] Gutiérrez-Avilés, D., Jiménez-Navarro, M. J., Torres, J. F., & Martínez-Álvarez, F. (2025). MetaGen: A framework for metaheuristic development and hyperparameter optimization in machine and deep learning. Neurocomputing, 637, 130046. https://doi.org/10.1016/j.neucom.2025.130046

[4] Han, J. Y., Hsu, C. R., & Huang, C. J. (2024). Automated progress monitoring of land development projects using unmanned aerial vehicles and machine learning. Automation in Construction, 168, 105827. https://doi.org/10.1016/j.autcon.2024.105827

[5] Hennebold, C., Klöpfer, K., Lettenbauer, P., & Huber, M. (2022). Machine learning based cost prediction for product development in mechanical engineering. Procedia CIRP, 107, 264-269. https://doi.org/10.1016/j.procir.2022.04.043

[6] Jadhav, A., & Shandilya, S. K. (2023). Reliable machine learning models for estimating effective software development efforts: A comparative analysis. Journal of Engineering Research, 11(4), 362-376. https://doi.org/10.1016/j.jer.2023.100150

[7] Khaliq, Z., Farooq, S. U., & Khan, D. A. (2022). A deep learning-based automated framework for functional user interface testing. Information and Software Technology, 150, 106969. https://doi.org/10.1016/j.infsof.2022.106969

[8] Khuat, T. T., Bassett, R., Otte, E., Grevis-James, A., & Gabrys, B. (2024). Applications of machine learning in antibody discovery, process development, manufacturing and formulation: Current trends, challenges, and opportunities. Computers and Chemical Engineering, 182, 108585. https://doi.org/10.1016/j.compchemeng.2024.108585

[9] Lin, C. T., & Huang, S. J. (2024). Technical risk model of machine learning based software project development - A multinational empirical study using modified Delphi-AHP method. Information and Software Technology, 171, 107449. https://doi.org/10.1016/j.infsof.2024.107449

[10] Marcolini, A., Bussola, N., Arbitrio, E., Amgad, M., Jurman, G., & Furlanello, C. (2022). histolab: A Python library for reproducible digital pathology preprocessing with automated testing. SoftwareX, 20, 101237. https://doi.org/10.1016/j.softx.2022.101237

[11] Namvar, M., Intezari, A., Akhlaghpour, S., & Brienza, J. P. (2022). Beyond effective use: Integrating wise reasoning in machine learning development. International Journal of Information Management, 69, 102566. https://doi.org/10.1016/j.ijinfomgt.2022.102566

[12] Saidani, I., Ouni, A., Chouchen, M., & Mkaouer, M. W. (2020). Predicting continuous integration build failures using evolutionary search. Information and Software Technology, 128, 106392. https://doi.org/10.1016/j.infsof.2020.106392

[13] Tran-The, T., Heo, E., Lim, S., Suh, Y., Heo, K. N., Lee, E. E., Lee, H. Y., Kim, E. S., Lee, J. Y., & Jung, S. Y. (2023). Development of machine learning algorithms for scaling-up antibiotic stewardship. International Journal of Medical Informatics, 181, 105300. https://doi.org/10.1016/j.ijmedinf.2023.105300

[14] Tunukovic, V., McKnight, S., Pyle, R., Wang, Z., Mohseni, E., Pierce, S. G., Vithanage, R. K. W., Dobie, G., MacLeod, C. N., Cochran, S., & O'Hare, T. (2024). Unsupervised machine learning for flaw detection in automated ultrasonic testing of carbon fibre reinforced plastic composites. Ultrasonics, 140, 107313. https://doi.org/10.1016/j.ultras.2024.107313

[15] Vaghasiya, J., Khan, M., & Bakhda, T. M. (2024). A meta-analysis of AI and machine learning in project management: Optimizing vaccine development for emerging viral threats in biotechnology. International Journal of Medical Informatics, 195, 105768. https://doi.org/10.1016/j.ijmedinf.2024.105768

[16] Villarroel Ordenes, F., & Silipo, R. (2021). Machine learning for marketing on the KNIME Hub: The development of a live repository for marketing applications. Journal of Business Research, 137, 393-410. https://doi.org/10.1016/j.jbusres.2021.08.036

[17] Wossnig, L., Furtmann, N., Buchanan, A., Kumar, S., & Greiff, V. (2024). Best practices for machine learning in antibody discovery and development. Drug Discovery Today, 29(7), 104025. https://doi.org/10.1016/j.drudis.2024.104025

[18] Cheng, H., Liu, B. J., Sun, X., & Qiu, X. (2025). Data-efficient creativity evaluation in museum cultural creative products: A machine learning framework for data-driven decision-making in product development. Expert Systems with Applications, 297, 129014. https://doi.org/10.1016/j.eswa.2025.129014

[19] Diéguez-Santana, K., & González-Díaz, H. (2023). Machine learning in antibacterial discovery and development: A bibliometric and network analysis of research hotspots and trends. Computers in Biology and Medicine, 155, 106638. https://doi.org/10.1016/j.compbiomed.2023.106638

[20] Eugster, R., Orsi, M., Buttitta, G., Serafini, N., Tiboni, M., Casettari, L., Reymond, J. L., Aleandri, S., & Luciani, P. (2024). Leveraging machine learning to streamline the development of liposomal drug delivery systems. Journal of Controlled Release, 376, 1025-1038. https://doi.org/10.1016/j.jconrel.2024.10.065

[21] Helleckes, L. M., Hemmerich, J., Wiechert, W., von Lieres, E., & Grünberger, A. (2022). Machine learning in bioprocess development: From promise to practice. Trends in Biotechnology, 41(6), 817-835. https://doi.org/10.1016/j.tibtech.2022.10.010

[22] Iannacchero, M., Löfgren, J., Mohan, M., Rinke, P., & Vapaavuori, J. (2024). Machine learning-assisted development of polypyrrole-grafted yarns for e-textiles. Materials & Design, 249, 113528. https://doi.org/10.1016/j.matdes.2024.113528

[23] Liu, Q., Zuo, S. M., Peng, S., Zhang, H., Peng, Y., Li, W., Xiong, Y., Lin, R., Feng, Z., Li, H., Yang, J., Wang, G. L., & Kang, H. (2024). Development of machine learning methods for accurate prediction of plant disease resistance. Engineering, 40, 100-110.

[24] Murray, J. D., Lange, J. J., Bennett-Lenane, H., Holm, R., Kuentz, M., O'Dwyer, P. J., & Griffin, B. T. (2023). Advancing algorithmic drug product development: Recommendations for machine learning approaches in drug formulation. European Journal of Pharmaceutical Sciences, 191, 106562. https://doi.org/10.1016/j.ejps.2023.106562

[25] Zhang, G., Borgert, T., Stoffelen, C., & Schmitz, C. (2025). High-throughput and explainable machine learning for lacquer formulations: Enhancing coating development by interpretable models. Progress in Organic Coatings, 205, 109265. https://doi.org/10.1016/j.porgcoat.2025.109265