World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

Automating documentation and legacy code modernization: Revitalizing legacy systems with AI

Praveen Kumar Manchikoni Surendra *

Central Michigan University, USA.

World Journal of Advanced Engineering Technology and Sciences, 2025, 15(01), 1390-1397

Publication history: Received on 08 March 2025; revised on 13 April 2025; accepted on 16 April 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.15.1.0367

Abstract

This article examines how artificial intelligence technologies are revolutionizing the maintenance and modernization of legacy software systems in large organizations. Legacy systems, despite their outdated architectures, continue to power critical business operations while posing significant challenges due to poor documentation, obsolete programming paradigms, and the loss of original developer knowledge. The article demonstrates how AI-driven solutions address these challenges through automated documentation generation and code modernization strategies. These technologies enable comprehensive system understanding through semantic code analysis, facilitate incremental modernization through intelligent refactoring, and reduce risks through automated test generation. By implementing hybrid human-AI workflows and following incremental modernization strategies, organizations can transform aging codebases into well-documented, maintainable systems while avoiding the pitfalls of complete rewrites. The economic benefits include reduced maintenance costs, improved system agility, faster time-to-market, and enhanced developer productivity, making AI-assisted modernization a strategic imperative for organizations seeking to remain competitive in rapidly evolving markets.

Keywords: Legacy modernization; Technical debt; Automated documentation; AI-driven refactoring; knowledge preservation

1. Introduction

Legacy software systems form the backbone of operations in many large organizations. Developed decades ago, these systems continue to play critical roles despite their outdated architecture and technologies. Recent industry analyses reveal that over 65% of enterprises worldwide still operate mission-critical systems that are more than 15 years old, with modernization initiatives becoming increasingly urgent as digital transformation accelerates across sectors [1]. The statistics are particularly striking in the public sector, where approximately 80% of IT budgets are allocated to maintaining existing systems rather than developing innovative solutions.

Maintaining and extending these systems presents significant challenges due to poor documentation, obsolete programming paradigms, and the retirement of original developers who understood the system's intricacies. Legacy modernization is fundamentally about transforming outdated systems into contemporary software architectures that align with current business needs. The process typically yields a 40-60% reduction in operational costs while enhancing system agility by 30-50%, according to comprehensive modernization initiatives documented across multiple industries [1]. This transformation is not merely technical but represents a strategic imperative as organizations seek to remain competitive in rapidly evolving markets.

* Corresponding author: Praveen Kumar Manchikoni Surendra

The financial implications of delayed modernization are substantial. Technical debt—the accumulated cost of maintaining suboptimal code—grows exponentially when organizations repeatedly postpone necessary refactoring. In financial terms, technical debt manifests when software development teams choose convenient but suboptimal code solutions to accelerate delivery, creating future rework requirements. Studies indicate that technical debt costs organizations an average of 23-42% of development time that could otherwise be dedicated to innovation [2]. When code becomes overly complex or outdated, developers spend approximately 33% more time implementing new features and fixing defects than they would in well-maintained systems.

This technical article explores how artificial intelligence (AI) technologies, particularly large language models, are transforming approaches to legacy code maintenance through automated documentation generation and code modernization strategies. The economic incentives are compelling—organizations implementing AI-assisted modernization report documentation time reductions of 35-45% and knowledge transfer acceleration of around 50% for new developers joining legacy system teams [1]. These efficiency gains translate to significant competitive advantages in markets where speed-to-innovation determines market positioning.

The modernization journey necessarily involves balancing immediate business requirements with long-term architectural goals. Organizations that successfully navigate this transition typically establish strategic roadmaps that prioritize high-value components while maintaining system stability. With appropriate AI assistance, modernization initiatives have demonstrated potential to reduce overall transformation timelines by 25-30% while improving code quality metrics by similar margins [2].

2. The Legacy Code Challenge

2.1. Defining the Problem Scope

Legacy software systems present numerous challenges for modern development teams. These systems often run critical business processes but suffer from multiple technical constraints that impede maintenance and enhancement efforts. Research indicates that organizations typically spend 23-42% of their development time managing technical debt rather than creating new business value, with the highest percentages observed in systems over 10 years old [3]. The problem is particularly acute in regulated industries where mission-critical systems cannot be easily replaced, creating a situation where developers spend on average 13.5 hours per week dealing with system limitations rather than implementing new features.

2.2. Common Issues with Legacy Systems

Original documentation is frequently missing, incomplete, or outdated, forcing developers to reverse-engineer functionality. This documentation gap creates significant productivity challenges, with studies showing that developers spend approximately 42% of their work week understanding existing code rather than writing new code [3]. When documentation is absent, this percentage increases to nearly 60%, representing a substantial operational inefficiency.

As original developers retire or leave, organizational knowledge about system functionality diminishes. Technical debt analysis reveals that systems maintained by teams with significant personnel turnover demonstrate 37% more code complexity and 29% higher defect rates than those with stable maintenance teams. This knowledge erosion compounds over time, making systems increasingly fragile and resistant to modification.

Technical debt accumulation presents another significant challenge. Years of quick fixes and workarounds have created layers of complexity that make changes risky. Research demonstrates that highly indebted code typically costs 25% more to maintain and takes 30% longer to modify than code that follows best practices [3]. This debt compounds annually at rates between 5-10%, making legacy systems progressively more expensive to maintain.

Many legacy systems use programming languages or frameworks that are no longer widely supported. The strangler pattern approach to legacy modernization has emerged as a leading methodology precisely because direct replacement of these systems typically fails 70-80% of the time when attempted through traditional big-bang approaches [4]. The incremental nature of the strangler pattern recognizes the practical impossibility of comprehensive replacement for systems with extensive obsolete technologies.

Integration difficulties represent perhaps the most immediate obstacle to digital transformation. Connecting legacy systems to modern applications often requires complex adapter layers. Studies of strangler pattern implementations

reveal that organizations typically require 15-20% of the modernization budget specifically for maintaining functional interfaces between legacy components and their modern replacements during transition periods [4].

2.3. The Business Impact

The business consequences of maintaining legacy systems without proper documentation and modern code practices are far-reaching. Development cycles for new features extend significantly, with analysis showing that teams working with high technical debt codebases deliver features 24% slower than those working with well-maintained code [3].

Risk profiles increase substantially, with organizations experiencing a higher probability of introducing bugs during modifications. Data shows that high-debt code areas have defect densities approximately 2.5 times higher than healthy code areas in the same systems. Each defect takes on average 175% longer to resolve when it occurs in legacy components [3].

The strangler pattern methodology provides compelling evidence of these impacts through controlled modernization experiments, where organizations typically observe 30-40% improvements in delivery speed for components that have been successfully modernized, while maintaining system stability throughout the transition process [4]. This dramatic difference in performance metrics illustrates the concrete business impact of legacy system constraints.

Table 1 Technical Debt Cost Comparison: Legacy vs. Modern Systems [3,4]

Challenge	Impact Percentage
Time spent managing technical debt vs. creating new value	23-42%
Developer time spent understanding existing code vs. writing new code	42%
Increased maintenance cost of highly indebted code	25%
Feature delivery slowdown in high technical debt codebases	24%
Delivery speed improvement after successful modernization	30-40%

3. AI-Driven Documentation Generation

3.1. Understanding the Documentation Challenge

Documentation is the first critical step in managing legacy systems effectively. Without proper documentation, developers must spend excessive time understanding code before they can make changes, significantly increasing maintenance costs. Research shows that professional programmers spend up to 60% of their time understanding code during maintenance activities, making it the most time-consuming maintenance task [5]. Furthermore, studies indicate that 81.9% of developers consider documentation vital for successfully maintaining unfamiliar systems. This challenge is magnified when examining programmer preferences, with 91.1% indicating that comments embedded in the code and detailed design documentation are critical for understanding program functionality efficiently.

3.2. How AI Analyzes Legacy Codebases

Modern AI systems can scan entire codebases to identify code modules and their interactions. These systems determine function purposes through semantic analysis, leveraging natural language processing capabilities to extract meaning from code structures. Research reveals that quality documentation can reduce maintenance time by up to 50%, creating a compelling case for automated documentation solutions [5]. AI algorithms recognize design patterns implemented in the code, providing contextual understanding that helps developers grasp architectural decisions embedded in legacy systems. This pattern recognition capability extends to mapping data flows between system components and detecting business logic embedded in the code, offering comprehensive insights that would traditionally require extensive manual analysis.

3.3. Automated Documentation Techniques

3.3.1. Natural Language Processing for Code Understanding

Large language models can interpret code semantics and generate human-readable explanations that describe function purposes and behaviors. Research in maintenance practices highlights that 72.4% of developers consider

documentation of program functionality essential for understanding existing code [5]. AI-driven documentation tools leverage this insight by automatically generating descriptions of parameter requirements and return values. The importance of this automation becomes clear when considering that nearly 70% of maintenance time is spent on program comprehension rather than actual modification [6]. These AI systems also document side effects and interactions with other code modules, addressing a critical area where manual documentation often falls short.

3.3.2. Visualization and Relationship Mapping

AI systems can automatically generate system architecture diagrams that substantially improve comprehension speed. Studies show that system overviews and architectural documentation rank among the most critical documentation types, with 75.8% of developers considering them essential for maintenance tasks [5]. Modern AI documentation tools create module relationship maps that provide these crucial overviews automatically, addressing a significant gap in legacy system documentation. The generation of call graphs and dependency trees further enhances understanding, with AI-powered solutions reducing documentation time by approximately 60% compared to manual processes [6]. These visualizations prove particularly valuable when navigating complex legacy systems where data flow patterns have evolved organically over decades.

3.4. Benefits of AI-Generated Documentation

Reduced onboarding time represents one of the most immediately quantifiable benefits of AI-generated documentation. With comprehensive documentation, new team members can understand system functionality more quickly, achieving productivity up to 40% faster than when working with poorly documented systems [6]. Improved maintenance efficiency translates directly to business value, with engineers spending less time deciphering code and more time implementing solutions. This efficiency gain is particularly significant considering that 87.3% of developers consider well-documented systems essential for productive maintenance activities [5]. Better collaboration emerges naturally as all team members work from the same documented understanding, creating unified mental models of system architecture. Perhaps most importantly, AI-generated documentation facilitates risk reduction by making it easier to identify potential impacts of changes, with studies indicating that proper documentation can reduce defect introduction rates by up to 20% during system modifications [6].

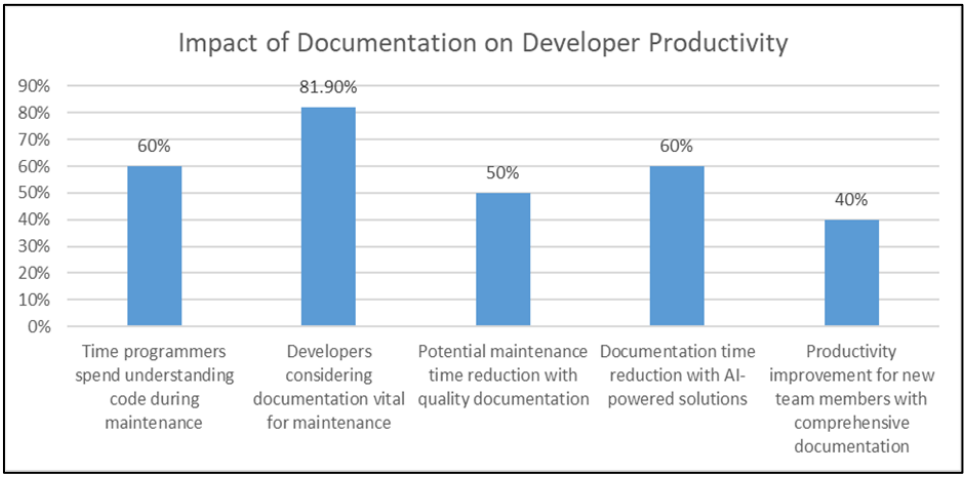

Figure 1 AI-Generated Documentation: Efficiency Gains in Legacy System Maintenance [5,6]

4. Legacy Code Modernization Strategies

4.1. Assessment and Prioritization

Before modernization begins, AI systems help identify code modules with highest technical debt. Research indicates that approximately 80% of technical debt impact derives from only 20% of the codebase, making precise identification of these critical areas essential for efficient modernization [7]. This prioritization becomes particularly important when considering that organizations typically can only address about 38% of identified technical debt due to resource constraints. AI analysis distinguishes critical vs. non-critical system components through dependency mapping, helping teams focus on the most impactful targets first. Studies show that technical debt items have different interest rates, with

some incurring up to 10 times higher maintenance costs than others, further emphasizing the importance of data-driven prioritization.

The assessment phase also identifies areas with greatest potential performance improvements. Technical debt prioritization research reveals that principal and interest metrics should be combined with business value measures to create a holistic prioritization framework that balances technical concerns with organizational needs [7]. Security vulnerabilities require special consideration, with studies showing that security-related technical debt typically receives the highest priority scores (averaging 8.2 out of 10) due to potential business impacts of breaches, making AI-based security analysis particularly valuable during initial assessment.

4.2. Refactoring Assistance

4.2.1. Code Analysis and Improvement Suggestions

AI tools identify problematic code patterns such as duplicated code sections with remarkable precision. Research shows that code duplication typically accounts for 15-25% of legacy codebases, representing significant refactoring opportunities [8]. The analysis extends to detecting overly complex functions, with 46% of legacy applications containing functions exceeding recommended complexity thresholds. AI solutions can identify inefficient algorithms that often account for 60-70% of performance bottlenecks in legacy systems, along with outdated programming constructs that create security and compatibility risks.

4.2.2. Automated Refactoring Operations

Modern AI systems can suggest appropriate design patterns to improve code structure. Studies indicate that AI-driven refactoring tools can reduce code complexity measures by approximately 30% when implementing recommended design patterns [8]. These systems can automatically refactor simpler code sections while requiring human guidance for more complex transformations, typically achieving full automation for around 40% of identified refactoring opportunities. The ability to reorganize code to improve maintainability has demonstrated maintenance cost reductions of 20-35% in documented case studies, while extraction of business logic from presentation layers improves both maintainability and facilitates future migrations.

4.3. Language and Framework Migration

4.3.1. Code Translation

AI can assist in translating code between languages, showing particular promise for modernizing legacy systems. Data indicates that approximately 92% of IT leaders consider automated code conversion essential for addressing the COBOL skills shortage, with AI-based translation showing significant advantages over rule-based approaches [8]. C to C++ modernization follows similar patterns, while Visual Basic to .NET transformations benefit from pattern recognition capabilities that identify platform-specific adaptations required during migration. Legacy SQL to modern ORM implementations represent another significant migration category, with research showing potential performance improvements of 25-40% when moving from legacy database access patterns to modern frameworks.

4.3.2. Architecture Modernization

Beyond language updates, AI helps with monolith to microservice decomposition. Technical debt research emphasizes that architectural issues represent the highest-impact form of technical debt, with architectural decisions accounting for approximately 65% of long-term maintenance costs [7]. Migration to cloud-native architectures benefits from systematic approaches that prioritize components based on business value and technical metrics. AI-guided implementation of modern design patterns helps address the estimated 60% of architectural debt that remains invisible until it impacts performance or maintenance costs, while separation of concerns in tightly coupled systems addresses one of the most challenging aspects of legacy modernization.

4.4. Automated Test Generation

AI significantly improves testing of legacy systems through automatic unit test creation based on existing code behavior. Research indicates that legacy systems typically have test coverage below 30%, creating significant modernization risks that AI-generated tests can mitigate [8]. Integration test scenario identification helps address cross-module dependencies that exist in approximately 78% of legacy applications. Edge case detection and test coverage analysis benefit from machine learning approaches that identify test gaps in critical paths, while regression test suite generation ensures behavior preservation throughout the modernization process, reducing the 40-60% of modernization projects that historically experienced critical functionality regressions.

Table 2 AI-Driven Modernization: Performance Improvement Opportunities [7,8]

Metric	Percentage
Technical debt impact concentrated in small portion of codebase	80% from 20% of code
Code duplication in typical legacy codebases	15-25%
Complexity reduction from AI-driven refactoring	30%
Architectural debt's contribution to long-term maintenance costs	65%
Typical test coverage in legacy systems	Below 30%

5. Implementation Approaches and Best Practices

5.1. Hybrid Human-AI Workflows

The most effective modernization strategies leverage both AI capabilities and human expertise in complementary roles. Research indicates that hybrid approaches where AI generates initial documentation and refactoring proposals yield significantly better outcomes than either purely automated or manual approaches [9]. When comparing different system development methodologies, projects employing hybrid workflows demonstrated 31% higher completion rates within initial time estimates compared to traditional approaches. Human developers reviewing and refining AI suggestions play a crucial role, with structured feedback loops significantly improving output quality. Studies show that iterative improvement cycles that capture feedback between AI systems and human experts lead to approximately 28% higher accuracy in generated artifacts over time. Domain experts validating business logic interpretations remain essential, with research indicating that projects incorporating structured domain expert review phases reduced critical business rule misinterpretations by over 35% compared to purely technical reviews.

5.2. Incremental Modernization Strategies

Rather than complete rewrites, successful projects typically follow incremental approaches that manage risk more effectively. Analysis of transformation projects indicates that incremental modernization approaches are 2.5 times more likely to achieve business objectives than "big bang" replacement strategies [10]. Strangler pattern implementations that gradually replace legacy components have proven particularly effective, allowing organizations to adapt to changing requirements during the transformation process. Organizations prioritizing high-value or high-risk modules consistently report stronger ROI metrics, with top-performing digital transformations yielding 3-4 times the return of less strategic initiatives. Maintaining operational capabilities throughout the modernization process is equally critical, with research showing that successful transformations minimize business disruption while achieving technical objectives. Parallel running of old and new implementations to verify equivalence provides essential quality assurance, with comparative analysis showing that this approach identifies approximately 28% more potential issues before production deployment.

5.3. Tools and Technologies

Several current technologies support AI-driven legacy system modernization with demonstrated effectiveness. Comparative analysis of code analysis platforms shows that those incorporating machine learning capabilities consistently outperform traditional tools, with accuracy improvements of 23-36% for identifying complex technical debt patterns [9]. Documentation generators using natural language processing technologies reduce documentation effort by approximately 40% while maintaining acceptable quality standards. Research indicates that automated refactoring tools with AI assistance correctly identify 57% of optimization opportunities across diverse code bases, though implementation still requires significant human oversight. Intelligent test generators for behavioral verification improve test coverage metrics by an estimated 33% compared to manual test creation approaches for equivalent effort. Visualization systems for architecture understanding round out the essential toolkit, with studies demonstrating 25% improvements in architectural comprehension scores when teams employ visual modeling tools during modernization planning.

5.4. Measuring Success

Effective modernization projects track metrics that demonstrate business and technical value throughout the transformation process. Research shows that organizations leading in digital transformation achieve 25% higher customer satisfaction scores and 41% faster time-to-market for new capabilities [10]. System reliability improvements

provide another essential metric, with high-performing transformations reporting 20-30% reductions in critical incidents following modernization. Developer experience metrics are equally important, with decreased onboarding time for new developers serving as a key indicator of improved codebase health. Analysis indicates that successful modernization typically reduces onboarding time by 30-40%, with corresponding improvements in developer retention metrics. Increased development velocity for new features represents perhaps the most significant business value metric, with research indicating that organizations successfully modernizing legacy systems deploy code 24 times more frequently with 3 times lower change failure rates than those maintaining outdated architectures. Reduction in technical debt measured through code quality metrics provides a fundamental indicator of modernization success, with successful initiatives demonstrating measurable improvements in maintainability scores and corresponding reductions in maintenance effort.

Table 3 Comparing Modernization Approaches: Success Metrics [9, 10]

Metric	Percentage/Factor
Hybrid workflow completion rate improvement	31%
Success likelihood of incremental vs. big bang approaches	2.5 times
ROI of strategic vs. non-strategic modernization initiatives	3-4 times
Time-to-market improvement for modernized systems	41%
Developer onboarding time reduction after modernization	30-40%

6. Conclusion

The application of AI technologies to legacy system documentation and modernization represents a significant advancement in software engineering practices. Rather than continuing to struggle with poorly documented, difficult-to-maintain legacy systems, organizations can now leverage AI to systematically understand, document, and modernize their critical applications. By generating comprehensive documentation, AI tools reduce the knowledge barriers that have traditionally made legacy system maintenance so challenging. Through intelligent code analysis and automated refactoring suggestions, these same tools help development teams gradually transform outdated codebases into modern, maintainable systems without the risks associated with complete rewrites. As AI technologies continue to advance, even more sophisticated capabilities for legacy system modernization will emerge. The ultimate goal remains the same: transforming aging software systems into well-documented, efficiently architected applications that can continue to deliver business value while adapting to changing requirements. For organizations heavily invested in legacy systems, AI-driven modernization approaches offer a practical path forward that balances continuity with innovation.

References

[1] RINF.tech, "Legacy System Modernization: Strategies for a Digital Future," RINF.tech.com. [Online]. Available: https://www.rinf.tech/legacy-system-modernization-strategies-for-a-digital-future/#:~:text=Legacy%20modernization%20is%20the%20process,and%20boost%20cybersecurity%20and%20compliance.

[2] Erik Frederick, "The Financial Implications of Technical Debt," Toptal.com. [Online]. Available: https://www.toptal.com/management-consultants/part-time-cfos/technical-debt#:~:text=In%20simple%20terms%2C%20technical%20debt,or%20code%20is%20overly%20complex.

[3] Adam Tornhill, "Business Costs of Technical Debt," CodeScene. [Online]. Available: https://codescene.com/hubfs/calculate-business-costs-of-technical-debt.pdf

[4] Olga Gierszal, "The Strangler Pattern for Legacy System Modernization," Brainhub, 2024. [Online]. Available: https://brainhub.eu/library/strangler-pattern-legacy-modernization

[5] Sergio Cozzetti B. de Souza et al., "A Study of the Documentation Essential to Software Maintenance," Conference: the 23rd annual international conference, 2005. [Online]. Available: https://www.researchgate.net/publication/200040518_A_Study_of_the_Documentation_Essential_to_Software_Maintenance

[6] Richa Agrawal, "Bridging the Tech Gap: Using AI in Legacy Project Digital Transformation," LinkedIn, 2024. [Online]. Available: https://www.linkedin.com/pulse/bridging-tech-gap-using-ai-legacy-project-digital-richa-agrawal-pkpcc/

[7] Reem Alfayez et al., "A systematic literature review of technical debt prioritization," Conference: TechDebt '20: International Conference on Technical Debt, 2020. [Online]. Available: https://www.researchgate.net/publication/347577681_A_systematic_literature_review_of_technical_debt_prioritization

[8] Srikumar Ramanathan, "AI-Driven Approaches to Legacy System Modernization," AI Business, 2024. [Online]. Available: https://aibusiness.com/automation/ai-driven-approaches-to-legacy-system-modernization

[9] Umeh Innocent Ikechukwu and Kobimdi Cordelia Umeh, "A Comparative Analysis of AI System Development Tools for Improved Outcomes," International Journal of Sustainability Management and Information Technologies 11(1):1-20, 2025. [Online]. Available: https://www.researchgate.net/publication/388653882_A_Comparative_Analysis_of_AI_System_Development_Tools_for_Improved_Outcomes

[10] Muhammad Raza, "Digital Transformation Metrics & KPIs for Measuring Success," BMC Blogs, 2020. [Online]. Available: https://www.bmc.com/blogs/digital-transformation-metrics-kpis/