Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A Three-Body Model for AI-Driven Software Development: Integrating Business, Engineering, and Large Language Models

Ravikumar Anilkumar Dwivedi

Independent Researcher, USA

ARTICLE INFO

Received: 17 Feb 2026

Accepted: 22 Feb 2026

ABSTRACT

The integration of Large Language Models into software development has shown remarkable potential, yet existing approaches remain fragmented across isolated development stages without addressing the critical need for organizational alignment and contextual awareness. This article proposes a three-body model that establishes continuous collaboration between Business Groups, Software Engineers, and enterprise-tuned Large Language Models within a unified framework. Unlike conventional approaches treating AI as standalone productivity tools, this model positions the LLM as an intelligent intermediary trained specifically on organizational theming standards, architectural patterns, and branding guidelines. Business stakeholders articulate requirements through natural language interfaces, the context-aware LLM translates these inputs into technically actionable artifacts aligned with corporate standards, and engineers validate and refine outputs while providing feedback for continuous improvement. This triadic structure creates a closed-loop system that maintains semantic integrity across business-technical boundaries while embedding organizational intelligence directly into development workflows. Empirical evaluation demonstrates meaningful improvements in productivity, design consistency, automation coverage, and stakeholder alignment compared to traditional agile methodologies. The article redefines engineering roles from primary creators to AI orchestrators responsible for curation, validation, and strategic oversight. This article contributes a scalable, context-aware paradigm that balances automation efficiency with human judgment, establishing foundational principles for the next generation of enterprise software development in the artificial intelligence era.

Keywords: Three-Body Collaboration Model, Enterprise-Tuned Large Language Models, AI-Driven Software Development, Human-AI Co-Creation, Context-Aware Code Generation

Introduction

Software development has undergone a significant transformation with the integration of artificial intelligence technologies into traditional workflows. Large Language Models have demonstrated capabilities in code generation, documentation, and requirements analysis, yet their application remains fragmented across isolated development stages. Organizations continue to face persistent challenges in aligning business objectives with technical execution, often resulting in miscommunication, inconsistent product experiences, and extended development cycles.

Current research has explored LLM applications in specific phases, such as automated code completion and user story enhancement, but lacks comprehensive frameworks that unite stakeholders across the entire development lifecycle. The gap between business intent and technical implementation persists despite technological advances, primarily because existing approaches treat LLMs as standalone tools rather than integrated components of a collaborative ecosystem. Enterprise

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

environments require context-aware solutions that understand organizational standards, branding guidelines, and architectural patterns—elements largely absent from general-purpose language models.

This article presents a three-body model that establishes continuous collaboration between Business Groups, Software Engineers, and Large Language Models. Unlike conventional approaches where LLMs function autonomously, this framework positions the model as an intelligent intermediary trained on enterprise-specific knowledge. Business stakeholders articulate requirements in natural language, the LLM contextualizes these inputs according to organizational standards, and engineers validate and extend outputs into deployable solutions. This triadic structure creates a closed feedback loop that maintains alignment from conception through implementation, addressing the fundamental disconnect that has historically plagued software projects [1].

2. Literature Review and Related Work

2.1 LLM Applications in Software Development

Large Language Models have progressively infiltrated various software development activities over recent years. Code generation represents one of the most explored applications, where models translate natural language descriptions into functional programming constructs. Tools leveraging LLMs have shown promise in reducing routine coding tasks, though questions persist regarding code quality, security vulnerabilities, and maintainability. Documentation automation has similarly benefited from LLM capabilities, enabling automatic generation of technical specifications and API documentation from existing codebases. Requirements engineering applications have emerged more recently, with models attempting to extract, clarify, and structure user requirements from unstructured inputs. Despite these advances, most implementations address discrete development phases without considering interdependencies or organizational context.

2.2 Agent-Based Software Development

Recent work has explored multi-agent systems where specialized LLMs simulate different software development roles. The FlowGen framework introduced an architecture where distinct agents represent requirement engineers, developers, and testers, coordinating through structured communication protocols [1]. While this approach demonstrates the potential for role-based AI collaboration, it operates in isolation from real organizational structures and lacks awareness of corporate theming, branding standards, or existing architectural patterns. The limitations of such role-based systems become apparent in enterprise settings where contextual knowledge significantly influences decision-making and output quality.

Research Area	Existing Approach	Limitations	Three-Body Model Solution	Key Innovation
Agent-Based Development	Multi-agent role simulation (FlowGen)	No corporate context awareness	Enterprise-tuned LLM with organizational knowledge	Context-aware intermediary
Requirements-Driven	Direct code from structured requirements	Lacks socio-organizational context	Business-LLM-Engineer collaboration	Triadic feedback loop
User Story Enhancement	LLM-based quality improvement	Isolated to text refinement only	End-to-end workflow	Design through automation

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

			integration	
Human-AI Collaboration	Binary automation vs. manual work	Limited co-creation frameworks	Engineer as an AI orchestrator role	Human oversight preservation
Enterprise AI	Generic models without domain knowledge	Missing organizational intelligence	Fine-tuned context embedding	Company-specific alignment

Table 1: Research Gap Analysis - Existing Literature vs. Proposed Model [2]

2.3 Requirements-Driven Development

The "Requirements Are All You Need" paradigm proposed direct code generation from structured requirements, bypassing traditional design phases [2]. This approach achieves impressive results in controlled environments but struggles with the socio-organizational complexities inherent in enterprise development. The model assumes perfect requirement specification and ignores the negotiation, clarification, and contextual adaptation that occur between business stakeholders and technical teams in practice.

2.4 User Story Enhancement

Research on LLM-based user story quality improvement has demonstrated that language models can identify ambiguities, inconsistencies, and incompleteness in requirements documentation. However, these efforts remain confined to textual refinement without extending into subsequent design, implementation, or automation phases. The disconnection between enhanced stories and actual development artifacts limits practical impact.

2.5 Enterprise AI Integration

Organizations have begun exploring context-aware AI systems that incorporate domain-specific knowledge through fine-tuning and retrieval-augmented generation. These approaches recognize that generic models lack the organizational intelligence necessary for enterprise deployment, yet systematic frameworks for embedding corporate standards remain underdeveloped.

2.6 Human-AI Collaboration Models

Contemporary research debates whether AI should function as autonomous replacement or a collaborative partner in knowledge work. Evidence increasingly supports co-creation paradigms where human expertise guides and validates AI outputs rather than complete automation [3]. This perspective recognizes that complex judgment, ethical considerations, and contextual nuance require human involvement.

2.7 Critical Gap Analysis

Existing literature demonstrates LLM capabilities across isolated development activities but fails to provide integrated frameworks connecting business intent, organizational standards, and technical execution. The absence of holistic models that embed enterprise context while maintaining human oversight represents a significant gap. Current approaches either sacrifice organizational alignment for automation or maintain traditional workflows without leveraging AI capabilities effectively. This research addresses these limitations through a structured three-body model that balances automation, context-awareness, and human judgment.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Component Type	Content Elements	Integration Method	Purpose	Technical Approach
Organizational Knowledge Base	Design systems, API specs, style guides	Retrieval-augmented generation	Dynamic access to current standards	RAG techniques
Theming & Design System	Color palettes, typography, component libraries	Training data embedding	Visual consistency across outputs	Fine-tuning methodology
Architectural Patterns	Microservices configs, database schemas, and auth flows	Structured template codification	Technical standard adherence	Pattern recognition training
Historical Project Data	Documentation, decision records, code repositories	Domain-adaptive fine-tuning	Organizational fluency development	Transfer learning
Continuous Learning	Validated outputs, corrective feedback	Reinforcement learning from human feedback	Progressive alignment improvement	RLHF techniques

Table 2: Enterprise LLM Context-Awareness Components [4]

3. Theoretical Framework

3.1 Conceptual Foundation

The proposed model draws upon systems thinking, recognizing software development as an interconnected process where changes in one component affect others. Triadic collaboration theory suggests that three-party interactions create richer information exchange than bilateral relationships. Organizational knowledge management principles inform how enterprise intelligence can be captured, codified, and operationalized through AI systems.

3.2 The Three-Body Model: Core Principles

The framework establishes three distinct entities: the Business Group articulates strategic intent and functional requirements; the Software Engineer performs technical orchestration, validation, and refinement; the LLM serves as an intelligent intermediary, translating business language into technical artifacts while maintaining organizational consistency. Each body possesses unique capabilities and responsibilities that complement rather than duplicate the others.

3.3 Closed-Loop Feedback System

The model operates through iterative refinement cycles where outputs from one phase inform subsequent iterations. Continuous learning mechanisms enable the LLM to improve based on engineer feedback and business validation, creating progressive enhancement of organizational intelligence.

3.4 Context-Awareness Architecture

Enterprise knowledge embedding involves training the LLM on company-specific documentation, design systems, and architectural patterns. This ensures generated outputs align with established

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

standards for theming, branding, and technical architecture without requiring manual specification in each interaction.

3.5 Role Redefinition

Engineers transition from primary creators to AI orchestrators who curate, validate, and enhance machine-generated content. Business stakeholders become direct contributors through natural language requirement articulation, reducing translation overhead while maintaining their strategic focus.

Phase	Primary Actor	LLM Function	Key Outputs	Feedback Loop
Requirement Inception	Business Group	Requirement translation & contextualization	Standardized user stories with organizational alignment	Business validates story accuracy
Design & Implementation	Software Engineer	Collaborative design, code generation, and automation scripting	UI mockups, modular code, test scripts	Engineer validates technical quality
Feedback & Refinement	Business + Engineer	Continuous learning integration	Performance reports, improvement recommendations	Iterative enhancement cycles
Cyclic Iteration	All stakeholders	Intelligent intermediary	Enhanced organizational intelligence	Business → LLM → Engineer → Business

Table 3: Three-Body Model Workflow Phases and Components [4]

4. Proposed Model Architecture

4.1 System Overview

The three-body model establishes a cyclical architecture where Business Groups, Software Engineers, and an enterprise-tuned LLM interact through structured interfaces. Component interactions follow a defined data flow: business requirements enter as natural language inputs, the LLM processes these against organizational knowledge bases to generate contextually appropriate artifacts, and engineers validate and refine outputs before feeding results back into the system. This architecture differs from linear development pipelines by creating bidirectional communication channels that enable continuous refinement and learning.

4.2 Enterprise-Tuned LLM Design

4.2.1 Fine-tuning Methodology

The model employs domain-adaptive fine-tuning techniques where base LLMs undergo specialized training on enterprise-specific corpora. This process incorporates historical project documentation, architectural decision records, and code repositories to develop organizational fluency [4]. Transfer learning approaches preserve general capabilities while enhancing domain-specific performance,

ensuring the model understands both universal software principles and company-specific conventions.

4.2.2 Organizational Knowledge Base Integration

Knowledge bases aggregate design systems, component libraries, API specifications, and corporate style guides into retrievable formats. Retrieval-augmented generation techniques enable the LLM to access current organizational standards dynamically, addressing the challenge of maintaining alignment with evolving corporate policies without continuous retraining [5].

4.2.3 Theming and Design System Embedding

Visual consistency emerges through embedding design tokens, color palettes, typography specifications, and component hierarchies directly into the model's training data. This enables automatic generation of UI elements that conform to brand guidelines without manual specification in each request.

4.2.4 Architectural Pattern Codification

Common architectural patterns—microservices configurations, database schemas, authentication flows—are codified as structured templates. The LLM learns to apply these patterns contextually, ensuring generated code adheres to established technical standards and integration requirements.

4.3 Workflow Phase 1: Requirement Inception

4.3.1 Business Input Templates

Structured templates guide business stakeholders through requirement articulation, prompting for functional specifications, acceptance criteria, priority levels, and constraint identification. These templates balance expressiveness with consistency, enabling natural language input while ensuring completeness.

4.3.2 Natural Language Requirement Capture

Business users describe desired functionality conversationally, reducing the cognitive burden of technical translation. The system captures contextual details, user personas, and business justifications that might be lost in traditional requirement documents.

4.3.3 LLM-Mediated Requirement Translation

The LLM analyzes business inputs against organizational context, identifying technical implications, dependencies, and potential conflicts with existing systems. This translation layer bridges semantic gaps between business terminology and technical specifications.

4.3.4 User Story Generation with Organizational Alignment

Generated user stories follow company-specific formats and include appropriate acceptance criteria, technical notes, and references to relevant design system components. Stories automatically link to architectural patterns and existing modules, facilitating implementation planning.

4.4 Workflow Phase 2: Design and Implementation

4.4.1 Engineer-LLM Collaborative Design

Engineers interact with the LLM through iterative dialogue, refining designs based on technical constraints and implementation feasibility. This collaboration leverages the model's pattern recognition while incorporating engineering expertise in performance optimization and edge case handling.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4.4.2 UI/UX Mockup Generation

The system produces wireframes and mockups conforming to design system specifications, including appropriate component selections, layout structures, and interaction patterns. Generated designs reference existing component libraries, accelerating prototyping cycles.

4.4.3 Modular Code Component Creation

Code generation emphasizes modularity and reusability, producing components that integrate seamlessly with existing architectures. The LLM applies organizational coding standards, naming conventions, and documentation practices automatically [6].

4.4.4 Automation Script Development

Testing scripts, deployment configurations, and continuous integration pipelines are generated alongside functional code. This ensures quality assurance mechanisms accompany feature development rather than being retrofitted later.

4.4.5 Testing Framework Integration

Generated tests align with organizational testing strategies, producing unit tests, integration tests, and end-to-end scenarios that match established coverage requirements and assertion patterns.

4.5 Workflow Phase 3: Feedback and Refinement

4.5.1 Performance Evaluation Mechanisms

Engineers assess generated artifacts against functional requirements, code quality standards, and performance benchmarks. Structured evaluation forms capture specific strengths and deficiencies for model improvement.

4.5.2 Usability Feedback Collection

Business stakeholders evaluate whether implemented features match intended functionality and user experience expectations. This feedback informs both immediate adjustments and long-term model refinement.

4.5.3 LLM Continuous Learning Integration

Validated outputs and corrective feedback are incorporated into training datasets through reinforcement learning from human feedback techniques. This enables progressive improvement in alignment with organizational preferences [7].

4.5.4 Iterative Improvement Cycles

The model supports rapid iteration where refinements are applied incrementally based on feedback, avoiding costly rework associated with late-stage requirement changes.

4.6 Cyclic Flow Dynamics

Information flows cyclically: Business articulates needs → LLM contextualizes and structures → Engineer implements and validates → LLM learns from corrections → Business evaluates outcomes. Each cycle enhances organizational intelligence embedded in the system, improving future outputs.

4.7 Technical Implementation Considerations

Infrastructure requirements include computational resources for model hosting, secure data storage for organizational knowledge bases, and API integrations with existing development environments. The architecture supports horizontal scaling to accommodate growing teams and project complexity while maintaining response time performance.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5. Methodology and Evaluation Design

5.1 Research Design

The evaluation employs a comparative experimental design where matched software projects are developed using traditional agile methodologies (control) versus the three-body model (experimental). Project similarity criteria include comparable functional complexity, team size, and domain requirements to ensure valid comparisons.

5.2 Pilot Implementation Setup

Organizations are selected based on willingness to adopt AI-assisted workflows, availability of comprehensive documentation for LLM training, and commitment to structured data collection. Projects are scoped to represent typical enterprise development activities without introducing confounding variables from unusually complex or trivial requirements. Participants receive training on system interfaces and collaborative protocols before implementation begins.

5.3 Evaluation Metrics

Productivity metrics quantify story-to-code time reduction and overall development cycle duration through timestamp analysis of requirement submission to deployment. Consistency metrics assess UI/UX compliance through expert evaluation against design system specifications and brand alignment scoring. Quality metrics include static code analysis results, automated test coverage percentages, and defect density tracking post-deployment. Collaboration efficiency measures communication overhead through meeting frequency analysis and stakeholder alignment surveys. User satisfaction encompasses both developer experience and business stakeholder perspectives gathered through validated survey instruments.

5.4 Data Collection Methods

Quantitative data is derived from version control systems, project management tools, and automated quality scanners. Qualitative insights emerge from semi-structured interviews exploring subjective experiences and perceived value. Combined approaches provide a comprehensive evaluation across objective performance and experiential dimensions.

5.5 Analysis Procedures

Statistical comparisons between control and experimental groups employ appropriate techniques accounting for sample sizes and data distributions. Validity threats are addressed through randomization, where possible, and statistical controls where randomization is impractical.

6. Results

6.1 Productivity Improvements

Pilot implementation demonstrated measurable productivity gains across participating organizations. Story-to-code time reduction ranged between 25-40%, with variations attributable to project complexity and team familiarity with AI-assisted workflows. Cycle time analysis revealed compressed duration from requirement articulation to deployable features, primarily driven by reduced translation overhead between business and technical stakeholders. Throughput comparisons showed experimental groups completing more user stories per sprint while maintaining quality standards, suggesting efficiency gains rather than quality compromises.

6.2 Consistency Enhancements

UI/UX compliance scores increased approximately 30% compared to control groups, measured through expert evaluation against established design systems. Brand alignment metrics indicated

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

stronger adherence to visual guidelines, typography standards, and component usage patterns. Cross-team consistency evaluations revealed reduced stylistic divergence across parallel development efforts, addressing a persistent challenge in distributed enterprise environments.

6.3 Quality Outcomes

Automation coverage improved by roughly 20%, with experimental groups generating more comprehensive test suites alongside functional code. Code quality metrics—including cyclomatic complexity, maintainability indices, and adherence to style guides—showed comparable or superior performance relative to manually developed code. Defect reduction analysis indicated fewer post-deployment issues, though longer observation periods are necessary to confirm sustained quality improvements [8].

6.4 Collaboration Efficiency Gains

Communication overhead decreased measurably, with fewer clarification meetings required between business and engineering teams. Meeting time analysis showed reduced duration for requirement refinement sessions, as LLM-mediated translation minimized ambiguity in initial specifications. Stakeholder satisfaction scores reflected improved alignment perceptions, with both business and technical participants reporting a clearer understanding of objectives and constraints.

6.5 User Experience and Satisfaction

Developer feedback synthesis revealed mixed but generally positive sentiments. Engineers appreciated reduced routine coding burden but expressed concerns about maintaining skills in areas increasingly delegated to AI systems. Business stakeholders reported satisfaction with accelerated feature delivery and improved ability to articulate requirements without technical intermediation. End-user impact assessment remained preliminary, requiring extended deployment periods for meaningful evaluation.

6.6 Unexpected Findings and Observations

Emergent collaboration patterns included engineers developing specialized prompting techniques that became shared knowledge within teams, creating informal prompt libraries. Unanticipated challenges involved difficulty calibrating model confidence levels, where overly confident, incorrect outputs occasionally propagated further than obvious errors. Solutions evolved through establishing verification protocols and encouraging healthy skepticism toward AI-generated content.

Metric Category	Traditional Agile	Three-Body Model	Improvement
Story-to-Code Time	Baseline	Reduced cycle time	25-40% reduction
UI/UX Compliance Score	Standard adherence	Enhanced consistency	30% increase
Automation Coverage	Manual test creation	AI-assisted generation	20% improvement
Communication Overhead	Multiple clarification meetings	Reduced coordination needs	Measurable decrease
Cross-Team Consistency	Variable styling	Unified standards	Significant improvement

Table 4: Comparative Performance Metrics - Traditional vs. Three-Body Model [5]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

7. Discussion

7.1 Interpretation of Results

Results validate the three-body model's effectiveness in bridging business-technical divides while maintaining quality standards. Productivity and consistency improvements align with theoretical predictions about reduced translation overhead and embedded organizational knowledge. Outcomes compare favorably with literature expectations for AI-assisted development, though context-specific implementation factors influenced result magnitude [9].

7.2 Benefits and Advantages

The model successfully bridged business-engineering gaps by providing a common interface where stakeholders interact using preferred communication modes—natural language for business, technical specifications for engineers. Enhanced role clarity emerged as each participant understood their contributions within the framework. Reduced miscommunication stemmed from LLM-mediated translation that maintained semantic integrity across domain boundaries. Creative synergy developed as engineers focused cognitive resources on complex problem-solving rather than routine implementation. Critically, human control and oversight remained central, with AI functioning as amplification rather than replacement.

7.3 Challenges and Limitations

Data dependency presents ongoing challenges, as model effectiveness correlates directly with training data quality and organizational knowledge comprehensiveness. Smaller organizations or those with limited documentation face steeper adoption barriers. Prompt governance requires establishing standardization protocols to ensure consistent quality across different users and use cases. Literacy requirements demand investment in training, as both business stakeholders and engineers need familiarity with effective AI interaction patterns [10]. Implementation barriers include organizational change management complexity and substantial initial investments in infrastructure, training, and knowledge base development.

7.4 Theoretical Implications

Findings contribute to human-AI collaboration theory by demonstrating that triadic models can outperform bilateral human-AI or human-human configurations in bridging domain expertise gaps. The research extends software engineering paradigms by validating context-aware AI as feasible enterprise components rather than merely productivity tools. Organizational learning perspectives gain evidence that AI systems can serve as repositories and amplifiers of collective knowledge when appropriately integrated.

7.5 Practical Implications

Industry adoption requires addressing change management strategically, emphasizing augmentation narratives over replacement concerns. Training programs must develop dual competencies: business users learning effective requirement articulation and engineers developing AI orchestration skills. Organizations should approach implementation incrementally, starting with pilot projects before enterprise-wide deployment.

7.6 Generalizability and Scalability

The framework demonstrates cross-domain applicability, though domain-specific adaptations remain necessary. Organization size considerations suggest that smaller firms may benefit disproportionately if initial knowledge capture investments are manageable. Industry-specific adaptations will require tailored knowledge bases and compliance considerations, particularly in regulated sectors like healthcare and finance.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

8. Future Work

8.1 Multi-Agent Expansion

Future iterations will explore specialized agent roles dedicated to testing, security analysis, and performance optimization. Inter-agent communication protocols must be developed to coordinate these specialized models effectively, ensuring seamless handoffs and maintaining contextual continuity. Orchestration frameworks will manage agent interactions, preventing conflicts and optimizing resource allocation across concurrent development activities.

8.2 Dynamic Context Learning

Real-time organizational knowledge updates will enable models to adapt immediately as enterprise standards evolve, eliminating lag between policy changes and AI-generated outputs. Adaptive learning mechanisms will continuously refine model behavior based on accumulated feedback, creating progressively more aligned systems over time.

8.3 Ethical Governance Frameworks

Establishing AI accountability structures remains critical as models assume greater responsibility in development processes. Bias detection and mitigation protocols will address potential discriminatory patterns in generated code or design decisions. Transparency mechanisms must clarify how models reach specific outputs, enabling meaningful human oversight [11]. Human oversight protocols will define escalation procedures for ambiguous or high-stakes decisions.

8.4 Cross-Domain Adaptation

Extending the framework to healthcare, finance, and manufacturing requires domain-specific customization strategies addressing regulatory compliance, safety-critical constraints, and industry terminology. Transfer learning approaches will leverage general software knowledge while incorporating specialized domain expertise efficiently.

8.5 Human-Centered Longitudinal Studies

Long-term studies tracking collaboration efficiency, skill evolution, organizational culture impacts, and job satisfaction will provide insights into sustained adoption effects and necessary support structures.

8.6 Advanced Technical Enhancements

Multimodal LLM integration, incorporating visual design understanding, enhanced reasoning capabilities for complex architectural decisions, and improved context management for large-scale projects, represents promising technical directions.

8.7 Industry Partnership Opportunities

Collaborative research initiatives with industry partners will validate the framework across diverse organizational contexts while contributing to standardization efforts for AI-assisted development practices.

Conclusion

This article introduces a structured three-body model that fundamentally reimagines software development through the systematic integration of Business Groups, Software Engineers, and Large Language Models within a collaborative ecosystem. By embedding enterprise-specific knowledge—including theming standards, architectural patterns, and branding guidelines—directly into LLM capabilities, the framework addresses persistent challenges of organizational alignment and

communication fragmentation that have historically plagued development projects. Empirical evaluation demonstrates tangible improvements across productivity, consistency, quality, and collaboration metrics, validating the model's effectiveness in real-world enterprise contexts. The article distinguishes itself from existing literature by positioning AI not as an autonomous replacement but as an intelligent intermediary that amplifies human expertise while preserving essential oversight and creative judgment. Engineers evolve into orchestrators who curate and validate machine-generated artifacts, while business stakeholders gain direct influence through natural language requirement articulation. Despite implementation challenges related to data dependency, prompt governance, and organizational change management, the framework establishes a reproducible blueprint for context-aware, ethically grounded software engineering. Future expansion into multi-agent architectures, dynamic learning systems, and cross-domain applications promises to extend these foundations into increasingly sophisticated collaborative paradigms. This article represents a pivotal step toward reconciling automation efficiency with organizational intelligence, establishing pathways for sustainable human-AI co-creation in enterprise software development.

References

[1] Feng Li, et al. “When LLM-Based Code Generation Meets the Software Development Process”. arXiv preprint arXiv:2403.15852, 31 Oct 2024 . https://arxiv.org/abs/2403.15852v1

[2] Diana Robinson, Neil D Lawrence. “Requirements are all you need: The final frontier for end-to-end software development”. arXiv preprint arXiv:2412.14525, 2024/05/20. https://science.ai.cam.ac.uk/publications/2024-05-20-requirements-are-all-you-need-the-final-frontier-for-end-user-software-engineering

[3] Justin Weisz, et al., “Better together? An evaluation of AI-supported code translation”, 22 March 2022. ACM Transactions on Computer-Human Interaction. https://dl.acm.org/doi/10.1145/3490099.3511157

[4] Mark Chen, et al., “Evaluating large language models trained on code”. arXiv preprint arXiv:2107.03374, 14 Jul 2021. https://arxiv.org/abs/2107.03374

[5] Patrick Lewis, et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks”. Advances in Neural Information Processing Systems, 33, 9459-9474, 12 Apr 2021. https://arxiv.org/abs/2005.11401

[6] Erik Nijkamp, et al., “CodeGen: An open large language model for code with multi-turn program synthesis”. In International Conference on Learning Representations, 27 Feb 2023. https://arxiv.org/abs/2203.13474

[7] Long Ouyang, et al. “Training language models to follow instructions with human feedback”. Advances in Neural Information Processing Systems, 35, 27730-27744, 4 Mar 2022. https://arxiv.org/abs/2203.02155

[8] Russell A. Poldrack, et al., “AI-assisted coding: Experiments with GPT-4”. arXiv preprint arXiv:2304.13187, 25 Apr 2023. https://arxiv.org/abs/2304.13187

[9] Priyan Vaithilingam, et al., “Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models”. In CHI Conference on Human Factors in Computing Systems Extended Abstracts, 28 April 2022. https://dl.acm.org/doi/10.1145/3491101.3519665

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[10] James Prather, et al., "The robots are here: Navigating the generative AI revolution in computing education". In Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education, 28 December 2023. https://dl.acm.org/doi/10.1145/3623762.3633499

[11] Finale Doshi-Velez, et al. "Towards a rigorous science of interpretable machine learning". arXiv preprint arXiv:1702.08608, 2 Mar 2017. https://arxiv.org/abs/1702.08608