International Journal Software Engineering and Computer Science (IJSECS)
6 (1), 2026, 313-322
Published Online April 2026 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v6i1.6823.

RESEARCH ARTICLE Open Access

Evaluating Vibe Coding as an AI-Orchestrated Development Methodology: A Case Study on Accelerating Complex Web-Based Educational Management Systems

Iqbal Muhammad Adiatma [1*]

[1*] IDN Boarding School, Bogor Regency, West Java Province, Indonesia.

**Corresponding author*: iqbalmuhammadadiatma@gmail.com.

Received: March 15, 2026; Accepted: April 17, 2026; Published: April 20, 2026.

Abstract: The emergence of generative AI has disrupted conventional software development practices, prompting considerable skepticism among IT professionals about whether such tools displace rather than augment human expertise. This study introduces "Vibe Coding" as a collaborative methodology — one in which AI operates as a capable partner, not a substitute — requiring human guidance for review, analysis, and iterative refinement of generated outputs; the primary objective is to assess whether Vibe Coding, when structured through Model Context Protocol (MCP) and schema engineering, can materially reduce development time for complex web systems — including CRUD operations, API integration, and custom business logic — relative to conventional approaches such as Waterfall. Two research questions drive the inquiry: (1) Can Vibe Coding compress development timelines for complex systems from months to days? and (2) How effective is AI as a collaborative partner in sustaining output quality through human-in-the-loop validation? A single case study approach was employed, applying the methodology to develop an ISO 9001:2015-compliant Management Information System (MIS) for Pondok Pesantren Abu Hurairah Mataram as a solo developer project, with metrics tracked across seven days including total development time, time per phase (planning, development, debugging, and deployment), proportion of AI-generated code (70–85%), prompt and iteration counts, bug frequency, debugging duration, total lines of code (LOC), and feature implementation success rate. Results show a completed system in seven days, with 70–85% of the codebase AI-generated and 15–30% manually refined for business logic, debugging, and performance tuning; human intervention effectively countered AI hallucinations throughout, repositioning the developer's role from syntax-level coding toward architectural orchestration and quality control. These findings suggest Vibe Coding raises productivity for solo developers in AI-saturated environments, though rigorous human oversight remains non-negotiable for production-grade systems.

Keywords: Vibe Coding; AI-Assisted Development; Model Context Protocol; Human-in-the-Loop; Software Productivity; Case Study.

1. Introduction

The rapid advancement of generative artificial intelligence (AI) tools — large language models (LLMs) and code assistants such as GitHub Copilot and Cursor — has fundamentally altered how software gets built. Junior developers, who have traditionally relied on entry-level tasks for skill acquisition, increasingly perceive AI as a threat to their career trajectories. Core entry-level activities — basic CRUD operations, boilerplate generation, and simple bug fixes — have become highly automatable, narrowing the experiential learning opportunities that once defined early-career development (Brynjolfsson *et al.*, 2025; Peng *et al.*, 2025). The numbers are difficult to dismiss: a Stanford Digital Economy Lab study using ADP payroll data recorded a 13–16% relative employment decline among workers aged 22–25 in AI-exposed occupations, including software development, between 2022 and 2025 (Brynjolfsson *et al.*, 2025). Separate reports document a 20–27.5% drop in software developer employment among early-career professionals, alongside rising unemployment rates for computer science graduates — 6.1–7.5% in recent U.S. data (New York Fed College Labor Market, 2025).

Several interrelated forces drive this shift. AI automation of foundational tasks erodes the traditional "learning ground" for junior developers, limiting the hands-on skill acquisition that experiential work once provided (WeAreDevelopers, 2025). Compounding this, over-reliance on AI outputs without deep technical understanding widens skill gaps in debugging, security analysis, and advanced problem-solving (LinkedIn discussions & YouTube analyses, 2025). Industry reports document a 46–53% reduction in entry-level job postings in tech sectors since 2022, as companies extract productivity gains from senior developers working alongside AI rather than hiring entry-level staff (Institute of Student Employers & Stanford studies, 2025). Mentorship structures are eroding in parallel — AI increasingly substitutes for code review and direct guidance, reducing meaningful senior-junior interaction (various industry notes, 2025). Employer expectations have shifted accordingly: junior developers are now expected to arrive with mid-level competencies, including effective AI orchestration, producing a perceived "missing career ladder" in which entry-level roles contract while senior demand persists

None of this, however, makes AI a replacement for human developers. The more defensible position — and the one this study takes — is that AI functions best as a collaborative partner. This aligns with methodologies that foreground human-in-the-loop validation, repositioning developers from syntax-level coding to architectural orchestration, rigorous review, and quality control. Vibe Coding, popularized by Karpathy (2025), exemplifies this shift: intent-driven interaction ("vibes") takes precedence over manual coding, with AI handling implementation while humans maintain oversight. Existing literature on AI-assisted development, though growing, leaves notable gaps. Studies examining productivity gains, code quality, and tool adoption — including analyses of GitHub Copilot effects (Vaithilingam *et al.*, 2022; Sarkar *et al.*, 2022) — predominantly rely on isolated experiments or small-scale simulations rather than end-to-end real-world implementations of complex systems involving CRUD operations, API integration, and full-stack custom logic. AI is typically framed as a tool rather than a partner, with limited attention to structured frameworks for mitigating hallucinations, ensuring consistency, and reducing iteration overhead (Barke *et al.*, 2023; recent reviews on AI code generation, 2025). Actionable, systematic methodologies for controlling AI outputs through persistent context, predefined schemas, and conditional tech stacks remain scarce.

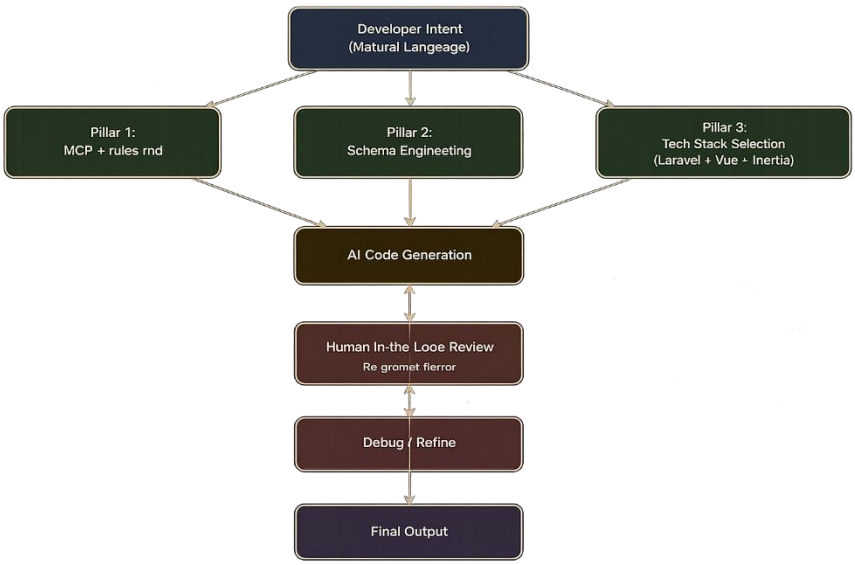

Figure 1. Framework

This study addresses these gaps by proposing and evaluating a three-pillar Vibe Coding framework:
1) Context Persistence via Model Context Protocol (MCP) — for structured, long-term AI interaction;
2) Schema Engineering — to anchor outputs in defined workflows and reduce hallucinations; and
3) Conditional Tech Stack Selection — to align AI generation with standardized, opinionated frameworks *(e.g.*, Laravel for complex MIS).

The value of this approach lies in controlled, efficient collaboration: MCP reduces prompt repetition, schema enforcement ensures consistency and maintainability, and predefined tech stacks improve compatibility and adherence to established conventions. Without these pillars, AI-assisted development risks inconsistency, excessive trial-and-error, and prolonged debugging — problems consistently observed in unstructured workflows. The primary research objective is to evaluate this framework's effectiveness in accelerating complex system development for solo developers, positioning AI as a capable partner that requires human guidance. Two specific research questions structure the inquiry:
1) RQ1: Can Vibe Coding reduce development time for complex web systems (CRUD, API integration, custom logic) compared to conventional methodologies such as Waterfall?
2) RQ2: How effective is AI as a collaborative partner in Vibe Coding for maintaining quality through human-in-the-loop validation?

Both questions are examined through a single case study: the solo development of an ISO 9001:2015-compliant Management Information System (MIS) in seven days.

2. Related Work

The integration of AI into software development has attracted sustained scholarly attention, with LLMs demonstrating measurable potential in code generation, testing, and productivity across the software development lifecycle (SDLC). Studies have examined AI's role in automation opportunities and integration practices (Niclavose, 2025), while comprehensive reviews document advancements in AI-assisted code generation, automated testing, and security considerations — acknowledging AI's capacity to accelerate development while introducing new categories of risk (Zhang *et al*. 2025; DORA, 2025). Within this broader trajectory, hallucinations in LLM-generated code represent one of the more consequential concerns — instances where models produce plausible but incorrect, syntactically flawed, logically erroneous, or insecure outputs. Agarwal *et al*. (2024) introduced CodeMirage, a benchmark dataset of 1,137 hallucinated Python snippets generated by GPT-3.5 on HumanEval and MBPP tasks, categorizing hallucinations into types including dead or unreachable code, syntactic incorrectness, logical errors, robustness failures, and security vulnerabilities. Security-related issues were particularly prevalent. Subsequent work expanded on this, developing taxonomies for repository-level hallucinations and identifying contributing factors (Various, 2024), with empirical findings indicating that hallucinated code contains vulnerabilities in up to 48% of cases — particularly in security-sensitive contexts (Zhang *et al*. 2025). That figure alone warrants serious attention from practitioners and researchers alike.

Human-in-the-loop (HiL) approaches have emerged as the primary mitigation strategy, and their application extends directly to the concerns raised above. The HiLDe framework (González *et al*., 2025) employs intentional decoding with user interaction to reduce vulnerabilities in generated code by incorporating human feedback during the generation process, emphasizing iterative refinement where developers guide LLMs to align outputs with specific intents — improving reliability and reducing error rates in practice. Complementing this, the Model Context Protocol (MCP), introduced by Anthropic in late 2024, addresses context persistence and external tool integration by enabling secure, bi-directional communication between LLMs and data sources, mitigating context loss in large projects through standardized discovery and invocation of tools (Anthropic, 2024; Hou *et al*., 2025). Industry analyses describe MCP as a foundational standard for agentic AI, facilitating persistent rules and reducing fragmentation in AI workflows (Cloudflare, 2025; Snyk, 2025; Thoughtworks, 2025). Despite its promise, MCP adoption remains nascent, with limited empirical evaluations in full-scale development scenarios — a gap this study directly addresses.

Several substantive gaps persist in the existing literature. Most studies concentrate on productivity metrics, code quality, or isolated tool adoption — often through controlled experiments rather than end-to-end real-world implementations of complex systems involving CRUD operations, API integration, and custom business logic (DORA, 2025; various reviews, 2025). AI as a collaborative partner requiring structured human oversight receives limited treatment, and few frameworks offer actionable methodologies for controlling outputs through persistent context, schema constraints, and conditional tech stacks. The primary weaknesses of current AI coding tools are well-documented: hallucinations producing plausible but incorrect or insecure code with vulnerability rates reaching 48%, context loss in large-scale projects leading to inconsistencies,

insufficient comprehension of complex business requirements, and over-reliance risks that demand rigorous human validation for security and maintainability. This study contributes by proposing a three-pillar Vibe Coding framework that addresses these gaps directly — (1) Context Persistence via MCP for structured, long-term AI interaction; (2) Schema Engineering to constrain outputs and reduce hallucinations; and (3) Conditional Tech Stack Selection to promote consistency and adherence to established conventions — and by applying this framework in a real-world solo development case to provide empirical evidence of its efficacy in accelerating complex system development while maintaining meaningful human-AI partnership.

3. Methodology

This section outlines the research design, data collection procedures, and evaluation criteria applied in assessing the Vibe Coding framework. The methodology was structured to ensure replicability and analytical rigor within the constraints of a solo-developer, real-world implementation context.

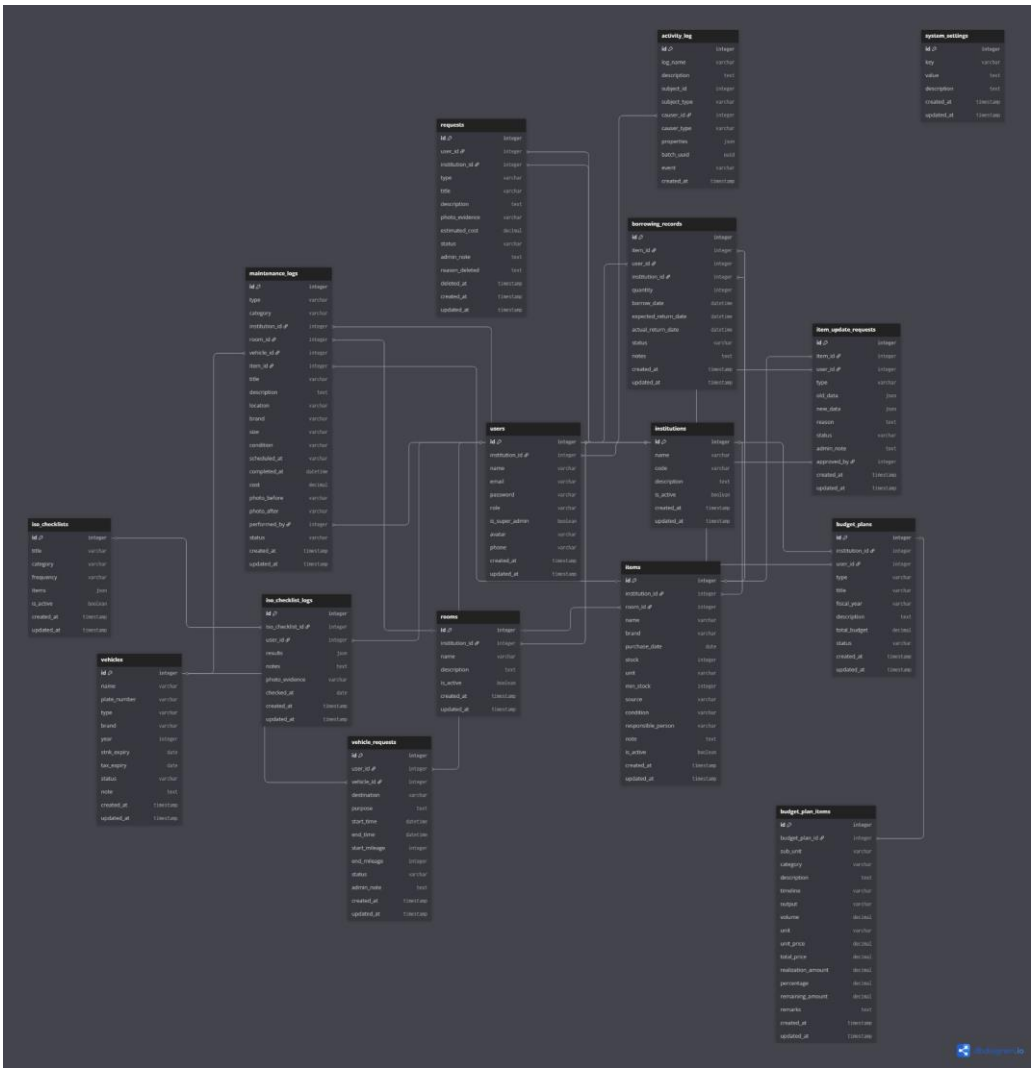

Figure 2. Entity-Relationship Diagram (ERD) of the MIS system illustrating multi-tenancy and module relationships.

This study adopts a hybrid research approach combining a single case study with an experimental evaluation to assess the Vibe Coding framework's effectiveness in accelerating complex web system development. The single case study focuses on the real-world implementation of an ISO 9001:2015-compliant Management Information System (MIS) for Pondok Pesantren Abu Hurairah Mataram, where the researcher served as both lead developer and project manager. This design aligns with established software engineering research practice: single-case studies are appropriate for exploratory investigations of contemporary phenomena in their natural context, particularly when the researcher has full access to the development lifecycle and seeks depth over breadth (Yin, 2018; Runeson *et al.*, 2012). The experimental component

evaluates the framework's performance against conventional methodologies — specifically Waterfall — by measuring time reduction and quality indicators across an end-to-end development process.

The selection of a single case was deliberate and justified on several grounds. Full access to the entire development lifecycle enabled detailed, accurate observation of AI-human interactions, prompting patterns, and iterative refinements — elements difficult to capture in multi-case settings without sacrificing analytical depth. The system itself exhibited high representational complexity, encompassing multi-tenancy, role-based access control (RBAC), CRUD operations, API integration, and custom business logic, making it a suitable test of the framework's robustness under realistic conditions. The organizational context of Pondok Pesantren Abu Hurairah further provided structured workflows aligned with quality management standards (ISO-like processes), offering a clear baseline for evaluation. As an exploratory initial study, the single-case approach also serves as a foundation for future multi-case replications, consistent with software engineering methodology guidelines emphasizing progressive generalization (Runeson *et al.*, 2012).

Development time was measured chronologically over seven days, from February 5 (start of engineering documentation at 09:00) to functional deployment. Tracking encompassed time per phase — planning and schema engineering, development, debugging, and deployment — recorded manually in Notion; Git version control activity (45 commits as indicators of progress); prompt logs from AI tools (Cursor and LLM interactions); and engineering documentation including ERD, schema, and workflow diagrams. Data collected across these sources included full prompt interaction logs, Git history with code changes and diffs, bug lists identifying errors during testing, MCP workflow screenshots, and AI-generated code alongside final manual outputs. Taken together, these sources enabled quantitative analysis of efficiency, AI contribution, and debugging frequency — mirroring common software engineering productivity metrics that combine time tracking with commit history and interaction logs (Peng *et al.*, 2023; arXiv 2406.17910).

Framework success was measured through six Key Performance Indicators (KPIs), informed by prior AI-assisted development studies: (1) Development Time Reduction — comparison of the actual 7-day timeline against conventional estimates of 3–6 months for systems of comparable complexity; (2) AI Contribution Ratio — proportion of AI-generated code (70–85%), with 15–30% manual refinement for logic, debugging, and optimization; (3) Iteration Efficiency — number of prompts and refinements per feature; (4) Bug Density — bugs per module or feature; (5) Feature Completion Rate — percentage of features implemented without major revisions; and (6) Debugging Time — time spent on error correction. These KPIs draw from evidence that AI can reduce repetitive task time by 30–50% (arXiv 2406.17910) and overall coding time by up to 55% in structured contexts (Emergent Mind summaries & related studies). To control AI behavior and reduce output uncertainty, MCP rules were defined in a rules.md file governing tech stack constraints, multi-tenancy rules, and agentic workflows. The three-pillar structure underpinning these rules is illustrated below.

```
# AGENT MODE: SIM URT PAH - MATARAM ARCHITECT

## 1. IDENTITY & GOAL
You are a Lead Developer for "SIM URT PAH - MATARAM". Your goal is to build a high-performance, agent-ready Management System for 28 internal
institutions at Pondok Pesantren Abu Hurairah.

## 2. THE TECH STACK (STRICT)
- Framework: Laravel 12
- Frontend: Vue 3 (Inertia.js bridge)
- Database: MySQL (strictly managed via migrations)
- CSS: Tailwind CSS (Primary Color: #C9A658)
- NO FILAMENT: All CRUDs and Dashboards must be manually coded for maximum control.

## 3. MULTI-TENANCY & RBAC
- Every resource (items, requests) MUST be filtered by `institution_id`.
- Roles: 'admin' (URT Division) vs 'karyawan' (28 specific institutions).
- Use Laravel Breeze for Auth, but customize the login to include a Searchable Dropdown for institutions.

## 4. AGENTIC WORKFLOW RULES (ANTIGRAVITY SPECIFIC)
- **Autonomous Tasking:** When asked to create a feature, analyze the ERD first. If it involves 'items', ensure 'item_update_requests' is used for any
stock updates.
- **Audit Trail:** You must implement `spatie/laravel-activitylog` for every model action.
- **Photo Handling:** Every 'request' must handle image uploads to `public/storage/reports`.

## 5. DESIGN TOKENS
- Main Branding: #C9A658
- Layout: Mobile-responsive sidebar (important for mobile usage in the field).
- UI Components: Use Svelte 5 runes ($state, $props) for state management.

## 6. PROMPT SHORTCUTS
- [SIM-GEN]: Generate a standard Laravel + Svelte component for this project.
- [SIM-LOG]: Check activity_logs consistency for the last generated feature.
```

Figure 3. Three-Pillar Vibe Coding Framework emphasizing structured human-AI collaboration through context persistence, schema engineering, and conditional tech stack selection.

Hallucinations were managed through structured prompting and context reinforcement across three steps: providing complete error context including console logs and code snippets; requesting root cause analysis based on system relations; and demanding solutions aligned with the existing architecture. An example prompt applied during bug fixing reads as follows:

> *"Saya menemukan bug pada tombol X yang tidak men-trigger fungsi. Berikut log error dari console dan terminal. Analisis root cause berdasarkan struktur code dan relasi antar komponen, lalu berikan solusi yang sesuai dengan arsitektur sistem."*

This methodology ensures replicability while addressing the exploratory nature of evaluating an emerging AI-orchestrated framework in a real-world solo development context. The complete source code — including Laravel backend, Vue frontend components, MCP rules.md, and database migrations — is available in a public GitHub repository at https://github.com/iqbaladiatma/sim-pah, following open science principles with structured directories, comprehensive README documentation, installation instructions, and an MIT license for reusability.

4. Result and Discussion

4.1 Results

The application of the Vibe Coding framework resulted in the successful development and deployment of an ISO 9001:2015-compliant Management Information System (MIS) within seven days as a solo developer project. The process was divided into distinct phases, with systematic tracking of time, AI contribution, and output metrics across each stage.

4.1.1 Development Phases and Timeline

The development was structured chronologically across five phases, as summarized in Table 1. On Day 1 alone, approximately 30% of the system was implemented, reflecting the early momentum enabled by structured AI orchestration through pre-defined MCP rules and schema constraints.

Table 1. Development Phase Timeline and Key Outcomes

Phase	Description	Duration	Key Outcomes
Day 1 – Foundation	System workflow design, database schema, authentication setup, basic UI	1 day	~30% system implemented; schema & auth complete
Day 2–3 – Core Development	Core features: CRUD operations, inventory system, multi-tenancy, RBAC	2 days	Primary business logic operational
Day 4–5 – Feature Expansion	Advanced modules: ISO procedures, reporting, request system	2 days	Full feature set expanded
Day 6 – Optimization	Debugging, refactoring, performance improvements	1 day	Stability enhanced
Day 7 – UI Improvement & Deployment	Final UI/UX refinements and production deployment	1 day	System live and functional

This timeline represents a substantial acceleration relative to conventional full-stack development for systems of comparable complexity, which typically requires 3–6 months.

4.1.2 Project Statistics and Metrics

Key quantitative data collected throughout the development process are summarized in Table 2. The volume of custom files, database migrations, and ISO-compliant modules underscores the system's complexity and the framework's capacity to sustain output quality at scale.

Table 2. Project Statistics and Quantitative Metrics

Metric	Value	Description
Total Prompts	~45	AI interactions across all phases
Total Lines of Code (LOC)	37,875	Excluding vendor dependencies and libraries
Custom Files	250+	Controllers, models, Vue components, etc.
Git Commits	45	Version control activity indicators
Database Migrations	66	Structured schema development
ISO Modules	39	Complex procedural modules compliant with standards
Controllers	23	Business logic layer
Activity Logs	184	Audit trail records for traceability

4.1.3 AI Contribution per Phase

AI generation rates varied by phase, with higher contribution percentages observed in structurally constrained tasks such as ISO module generation and UI modernization. Table 3 presents the breakdown by phase.

Table 3. AI Contribution Rate and LOC by Development Phase

Phase	Focus	Prompts	AI Contribution (%)	Approximate LOC
Foundation	Setup & Authentication	5	90	~4,500
Core System	Inventory & RBAC	8	95	~6,500
ISO Modules	39 Procedures	15	100	~12,500
UI Modernization	Frontend Design	12	98	~10,000
Optimization	Excel Export & Logs	5	100	~4,375

70–85% of the codebase was AI-generated, with 15–30% manual intervention for business logic adjustments, debugging, and optimization. This exceeds reported industry averages, where AI contributes approximately 41% of code in 2025–2026 and typically requires higher rates of manual refinement (Index.dev, 2025; various studies, 2026).

4.1.4 Bug Identification and Resolution

A total of approximately five major bugs were identified during development and testing, including errors in Excel import/export functionality, mobile responsiveness issues, institution account-related errors, and minor integration inconsistencies. Most were resolved through AI-assisted debugging — specifically, prompted root cause analysis — combined with targeted manual interventions such as data structuring and spreadsheet normalization. No major refactoring was required, indicating that the framework's structured approach to context persistence and schema constraints effectively limited error propagation across modules.

4.1.5 Supporting Artifacts

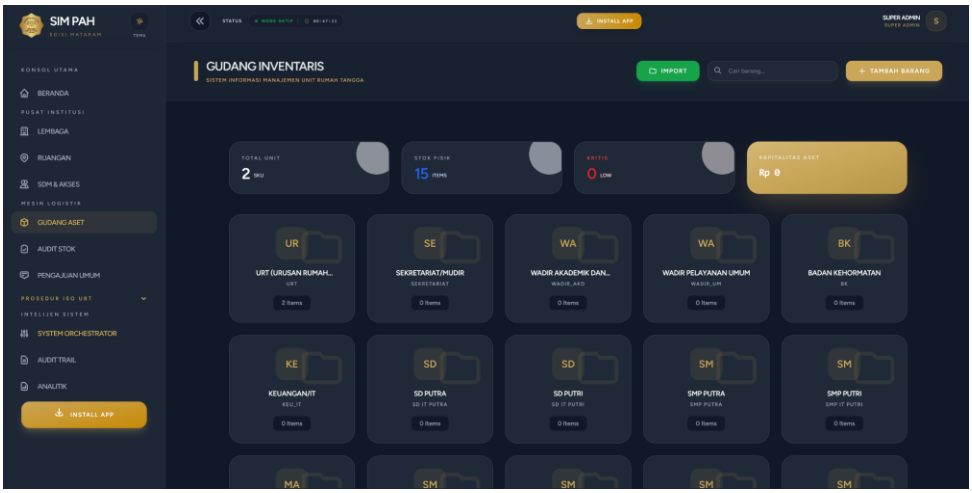

Figure 4. MIS dashboard interface demonstrating implemented modules including inventory, reporting, and request management.

The results are supported by tangible documentation artifacts, including MIS dashboard screenshots showing functional CRUD, reporting, and request modules; MCP rules.md file excerpts defining tech stack constraints, multi-tenancy rules, and agentic workflows; Git commit history logs tracking 45 incremental commits; and prompt interaction logs capturing 45+ AI sessions and iterative refinements. These artifacts provide transparency and enable independent validation of the development process.

4.2 Discussion

The findings demonstrate that the Vibe Coding framework not only accelerates code production but also sustains the implementation of structurally complex systems under real-world conditions. The high AI contribution rate — reaching 90–100% in later, more constrained phases — aligns with emerging evidence that structured prompting and persistent context can substantially enhance generative AI efficiency (Peng *et al.*, 2023; Index.dev, 2025). These results affirm RQ1: Vibe Coding substantially reduces development time for complex web systems relative to traditional methodologies. RQ2 is similarly supported, as human-in-the-loop validation — encompassing prompt refinement, manual debugging, and architectural oversight — effectively mitigated AI hallucinations and ensured production-grade output quality.

The framework's effectiveness stems from a fundamental shift in the developer's role: from syntax authoring to AI orchestration. By defining persistent behavioral rules via MCP, anchoring outputs in pre-engineered schemas, and selecting opinionated stacks (Laravel + Vue/Inertia) to minimize generative inconsistency, the developer reduced the degrees of freedom available to the model — thereby constraining the solution space and improving output predictability. The low bug count (approximately five major issues) and their rapid resolution further indicate that structured human-AI partnership can maintain system stability even under accelerated delivery conditions. However, several limitations must be acknowledged. The single-case study design restricts generalizability; results are derived from one specific organizational context — Pondok Pesantren Abu Hurairah — with domain-specific requirements (ISO procedural modules, multi-institution management) that may not translate directly to other settings such as commercial startups or large enterprises. The technology stack was constrained to Laravel 12, Vue 3, Inertia.js, and MySQL, limiting applicability to alternative ecosystems *(e.g.*, MERN, Django, or Spring Boot). The study also relies on a single developer who simultaneously acted as project manager, introducing potential developer-centric bias in prompting strategies and self-evaluation. Furthermore, outcomes are highly dependent on prompting proficiency; less experienced practitioners may encounter higher iteration overhead or residual errors. Finally, while security concerns were addressed through manual review, the framework does not inherently guarantee secure-by-design outputs — AI-generated code can introduce vulnerabilities such as XSS, SQL injection, and insecure dependencies if not rigorously validated, as evidenced by studies reporting elevated security risks in unexamined LLM-generated code (Agarwal *et al.*, 2024; various security reports, 2025).

5. Conclusion

This study demonstrates that Vibe Coding, when structured through a three-pillar framework — Context Persistence via Model Context Protocol (MCP), Schema Engineering, and Conditional Tech Stack Selection — constitutes an effective human-AI partnership methodology for accelerating the development of complex web systems. Applied in a real-world solo developer context, the framework enabled the completion of an ISO 9001:2015-compliant Management Information System for Pondok Pesantren Abu Hurairah in seven days — a timeline that contrasts sharply with conventional full-stack development estimates of 3–6 months for systems of comparable complexity involving CRUD operations, API integration, multi-tenancy, RBAC, and procedural modules.

The primary research objective — to evaluate Vibe Coding as a collaborative approach in which AI functions as a capable partner rather than a replacement tool — was achieved. The framework successfully shifted the developer's role from manual syntax authoring to architectural orchestration, quality control, and iterative refinement. Key findings corroborate both research questions: AI contribution reached 70–85% overall and 90–100% in structurally constrained phases, substantially exceeding reported industry averages of approximately 41% in 2025–2026; hallucinations were effectively mitigated through persistent context, schema constraints, and human-in-the-loop validation, yielding only five major bugs requiring resolution; and the seven-day delivery timeline affirms that structured AI orchestration can dramatically compress development cycles without sacrificing system integrity. These outcomes confirm that Vibe Coding significantly reduces development time for complex systems relative to conventional methodologies (RQ1), and that AI proves highly effective as a collaborative partner when guided by structured human oversight (RQ2).

The study contributes a practical, replicable blueprint for solo developers navigating the generative AI era. Limitations — including the single-case design, technology stack specificity (Laravel 12, Vue 3, Inertia.js), and developer-centric bias — constrain immediate generalizability and should be addressed in subsequent research through multi-case validation, dedicated security evaluations, and cross-stack experimentation. Vibe Coding is not a substitute for fundamental programming competence or human judgment; it is a transformative methodology that amplifies developer capability. As generative AI continues to mature, sustained success will depend on disciplined orchestration, persistent context management, and an unwavering commitment to quality assurance — positioning the developer not as a passive consumer of AI output, but as its most critical collaborator.

References

Agarwal, V., Pei, Y., Alamir, S., & Liu, X. (2024). CodeMirage: Hallucinations in code generated by large language models. *arXiv*. https://doi.org/10.48550/arXiv.2408.08333

Anthropic. (2024, November 25). *Introducing the Model Context Protocol*. https://www.anthropic.com/news/model-context-protocol

Brynjolfsson, E., Li, D., & Raymond, L. (2025). Generative AI at work. *The Quarterly Journal of Economics*, *140*(2), 889–942. https://doi.org/10.1093/qje/qjae044

Daniotti, S., Wachs, J., Feng, X., & Neffke, F. (2026). Who is using AI to code? Global diffusion and impact of generative AI. *Science*, eadz9311. https://doi.org/10.1126/science.adz9311

DORA (Google Cloud). (2025). *State of AI-assisted software development report*. Google Cloud.

González, E. A., Rothkopf, R., Lerner, S., & Polikarpova, N. (2025). HiLDe: Intentional code generation via human-in-the-loop decoding. In *Proceedings of the 2025 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 222–233). IEEE. https://doi.org/10.1109/VL-HCC65237.2025.00032

Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). Model Context Protocol (MCP): Landscape, security threats, and future research directions. *arXiv*. https://arxiv.org/abs/2503.23278

Index.dev. (2025). *Developer productivity statistics with AI tools*. https://www.index.dev/blog/developer-productivity-statistics-with-ai-tools

International Organization for Standardization. (2015). *ISO 9001:2015 quality management systems — Requirements*. https://www.iso.org/standard/62085.html

Iqbal, M. A. (2026). *SIM URT PAH - Mataram: Vibe Coding MIS implementation* [Source code]. GitHub. https://github.com/iqbaladiatma/sim-pah

Karpathy, A. [@karpathy]. (2025, February). There's a new kind of coding I call "vibe coding"... [Post]. *X*. https://x.com/karpathy/status/1886192184808149383

Laravel. (2026). *The PHP framework for web artisans*. Retrieved March 17, 2026, from https://laravel.com

Niclavose, S. (2025). *AI-driven software development: Opportunities and good practices* [Master's thesis, Malmö University]. DiVA Portal. https://www.diva-portal.org/smash/record.jsf?pid=diva2:1996184

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv*. https://doi.org/10.48550/arXiv.2302.06590

Ray, P. P. (2025, March). *Vibe coding: The complete guide to AI-powered development in 2025–2026*. Medium. https://medium.com/@basukori8463/vibe-coding-the-complete-guide-to-ai-powered-development-in-2025-26-7810d2136dc2

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). *Case study research in software engineering: Guidelines and examples*. Wiley.

Thoughtworks. (2025). *From vibe coding to context engineering: Technology radar 2025*. https://www.thoughtworks.com/radar

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022, April). 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* (pp. 1–7). ACM. https://doi.org/10.1145/3491101.3519665

Yin, R. K. (2018). *Case study research and applications: Design and methods* (6th ed.). Sage.

Zhang, Z., Wang, C., Wang, Y., Shi, E., Ma, Y., Zhong, W., & Zheng, Z. (2025). LLM hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proceedings of the ACM on Software Engineering*, *2*(ISSTA), 481–503. https://doi.org/10.1145/3728894.

Appendix

ISO 9001:2015 Clause Coverage. The implemented system addresses Clauses 7.5 (Documented Information), 8.2–8.5 (Requirements, Design, and Production Control), 9.1 (Monitoring and Measurement), and 10.2 (Nonconformity and Corrective Action) through structured workflows and audit trails.

Supplementary Materials. Full prompt logs, MCP rules.md excerpts, and ERD/database schema documentation are available in the project repository under the /docs directory. A live demonstration is available upon request.

GitHub Repository. https://github.com/iqbaladiatma/sim-pah.