International Journal of Engineering and Techniques - Volume 12 Issue 2, March-April - 2026

VIBE CODING IN PRACTICE: MOTIVATIONS, FUTURE ANDCHALLENGES

Noufin P[1], Sumi M[2]

1(MCA, Nehru College of Engineering and Research Centre, Thrissur, Kerala, India
Email: noufinp321@gmail.com)

2(1(MCA, Nehru College of Engineering and Research Centre, Thrissur, Kerala, India
Email: sumimurali118@gmail.com)

Abstract:

Recent progress in large language models (LLMs) and AI code generation tools, such as GitHub Copilot and ChatGPT, is fundamentally reshaping the landscape of software engineering by shifting the developer's role from manual syntax authoring to natural language orchestration. This emerging paradigm, frequently referred to as "vibe coding," prioritizes rapid experimentation and goal-oriented outcomes by allowing users to generate executable code through iterative prompting with minimal technical review. While this approach significantly lowers entry barriers for non-programmers and accelerates prototyping for professionals, it introduces a critical "speed-quality paradox" characterized by code that is often fragile, insecure, or difficult to maintain. Empirical evidence indicates that while a majority of practitioners experience an immediate success flow, a significant portion of AI-generated codebases contain hidden vulnerabilities or technical debt, exacerbated by a widespread tendency to skip rigorous quality assurance in favor of "run-and-see" validation. Furthermore, over-reliance on these tools may lead to the erosion of fundamental problem-solving skills and a growing "QA crisis" where developers struggle to debug complex failures when AI-driven logic breaks down. As the industry moves toward deeper integration of autonomous AI agents, the future of software development will likely depend on a balanced framework that combines the velocity of vibe coding with robust human-in-the-loop oversight and governance to ensure long-term system reliability and security.

Keywords ---- Large Language Models (LLMs) , Vibe Coding , AI Code Generation , Natural Language Programming , Software Engineering Automation , Rapid Prototyping.

INTRODUCTION

The rapid evolution of large language models (LLMs), accessible through advanced AI code generation platforms, is fundamentally reshaping the landscape of modern software engineering. These sophisticated tools allow developers to describe complex functionalities using natural language and receive executable code in return, significantly accelerating routine tasks. By shifting the focus from manual syntax construction to high level orchestration and supervision, these technologies have effectively lowered the entry barriers for individuals with limited programming experience. However, as these tools become deeply integrated into daily workflows, new and distinct coding practices are beginning to emerge that differ significantly from traditional methodologies. Within this wave of rapid technological adoption, a specific practice known as "vibe coding" has surfaced as a prominent new paradigm. First defined in early 2025, vibe coding describes a development approach where users rely on AI tools to build software based on the "vibe" or general intent of their natural language descriptions, often without maintaining a comprehensive understanding of the underlying source code. This trend is gaining significant traction in high-stakes environments, evidenced by reports indicating that a quarter of emerging startups now utilize codebases generated almost entirely by AI. Unlike conventional AI-assisted programming, this method prioritizes velocity and experimental iteration over deep technical comprehension

Within this wave of rapid technological adoption, a specific practice known as "vibe coding" has surfaced as a prominent new

paradigm. First defined in early 2025, vibe coding describes a development approach where users rely on AI tools to build software based on the "vibe" or general intent of their natural language descriptions, often without maintaining a comprehensive understanding of the underlying source code. This trend is gaining significant traction in high-stakes environments, evidenced by reports indicating that a quarter of emerging startups now utilize codebases generated almost entirely by AI. Unlike conventional AI-assisted programming, this method prioritizes velocity and experimental iteration over deep technical comprehension

The primary drivers for adopting this intent-driven approach are rooted in the pursuit of extreme efficiency and democratization. For professional engineers, it serves as a powerful mechanism for rapid prototyping and the validation of creative concepts at speeds previously unattainable. For non-technical domain experts, it provides the necessary accessibility to transform ideas into functional applications without the requirement of formal computer science training. By treating code as a byproduct of intent rather than the primary artifact, vibe coding redefines the concept of developer productivity and broadens participation in the digital economy. Despite these transformative benefits, the shift toward intuition-based development introduces a critical "speed-quality paradox" that threatens long-term system integrity. Research highlights significant risks associated with unverified code, including the introduction of security vulnerabilities and the accumulation of technical debt. Because this practice often involves minimal review of generated logic, it can lead to the creation of "vulnerable developers" who possess functional software but lack the expertise to diagnose or remediate complex failures when they inevitably occur. This reduced human oversight raises serious concerns regarding the reliability and sustainability of applications built on "vibes" alone.

LITERATURE REVIEW

Zviel-Girshin et al. [2024]: This study examines the adoption of AI code generation tools among novice developers, revealing that while such technologies bolster student confidence, they often lead to a superficial grasp of underlying programming concepts. The research suggests that an over-reliance on automated outputs may ultimately impede long term learning outcomes.Zi et al. [2025]: This research identifies significant comprehension barriers among computer science students utilizing Large Language Models (LLMs). The findings indicate a low success rate in code explanation tasks, attributed primarily to automation bias and the unfamiliarity of styles generated by autonomous systems.Feldman and Anderson [2024]: This paper investigates the utility of AI tools for non-technical users, noting that while these individuals can produce functional software, they face substantial difficulties in articulating precise prompts and verifying the accuracy of the resulting codeHowever, this assumption breaks down in highly distributed and dynamic environments. As the number of users, devices, and transactions increases, centralized systems face growing risks from single points of failure, insider threats, and coordinated cyber-attacks. Minor misconfigurations or breaches can propagate rapidly across the infrastructure, leading to large-scale data compromise. Much like statistical patterns failing in small samples, centralized trust models struggle to maintain security guarantees under real-world operational complexity.

Ferino et al. [2025]: The authors explore the workflow of junior developers using GitHub Copilot, observing a heavy reliance on trial-and-error methodologies. The study highlights a common tendency to accept algorithmic suggestions without a comprehensive technical review of the logic.Vaithilingam et al. [2022]: This study highlights a paradox in developer experience where users express high satisfaction with AI assistants but encounter significant friction when attempting to debug or remediate errors within the generated source code.Fu et al. [2023]: This empirical analysis focuses on the security implications of AI-driven development, identifying frequent vulnerabilities within codebases hosted on public repositories. The authors emphasize that the speed advantages of these tools must be balanced with rigorous security audits.Ahmed Fawzy et al. [2025]: This work provides a comprehensive synthesis of "vibe coding" as an intuition-driven practice. By reviewing firsthand practitioner accounts, the study identifies a critical research gap concerning the long-term maintainability and ethical implications of software built primarily through natural language interaction.Kruse et al. [2024]: This investigation analyzes the intricacies of prompt engineering, specifically how developer experience shapes the quality of generated outputs. While the study underscores the necessity of precise prompt design, it notes that traditional structured approaches are often bypassed in intuition-led practices like vibe coding, where trial-and-error dominates the workflow.Domenico Cotroneo et al. [2025]: In a large-scale comparative study, the authors evaluate the structural differences between human-written and AI-generated code. Their findings indicate that while AI systems offer significant advantages in boilerplate generation, they frequently introduce higher levels of complexity and latent defects that complicate long-term system evolution.

METHODOLOGY

The systematic investigation of vibe coding is grounded in a comprehensive analysis of practitioner experiences and industry trends to capture the authentic behavioral patterns associated with this emerging paradigm. By utilizing a grey literature review (GLR) methodology, the research synthesizes firsthand accounts from diverse sources, including technical blogs, professional forums, and community discussions. This approach allows for the collection of real-world data that traditional academic literature may not yet encompass, specifically focusing on the transition from syntax-driven programming to goal-oriented, conversational development. The data collection process involves identifying key motivations, such as development velocity and accessibility, alongside the challenges encountered during rapid iteration. Quantitative metrics were derived from a review of 101 distinct sources, revealing specific trends in user experience, such as the "instant success flow" versus the frequency of project abandonment. This analytical framework assesses user perceptions regarding the quality, security, and maintainability of AI-generated code, while also examining the trade-offs inherent in skipping conventional quality assurance practices.

To ensure a rigorous evaluation, the methodology extends to investigating the quality assurance protocols—or lack thereof—applied to unverified code. This includes a detailed examination of how users delegate debugging tasks back to AI tools and the resulting impact on long-term software sustainability and technical debt. The findings are then used to generate insights that inform responsible AI integration, aiming to improve both educational approaches and industrial management of AI-driven development workflows. The analytical framework further extends to an evaluation of the "speed-quality paradox" by investigating the trade-offs between rapid functional deployment and long-term software sustainability. This involves a thematic analysis of quality assurance practices, specifically focusing on the shift from structured testing to "run-and-see" validation and the delegation of debugging tasks back to autonomous tools. By examining user experiences ranging from "instant success flows" to project breakdowns and abandonment, the methodology identifies the critical risks associated with unverified code, such as security vulnerabilities and the accumulation of technical debt. This comprehensive approach ensures that the study not only captures the efficiency gains of intent-driven development but also addresses the resulting "QA crisis" and the potential for skill erosion among practitioners

RESULTS AND DISCUSSIONS

The study highlights a major transformation in software engineering through intent-driven development, where traditional line-by-line coding is replaced by high-level natural language instructions. By leveraging large language models to generate structural code, development becomes more accessible and efficient, enabling individuals from diverse backgrounds to participate in application creation. This approach encourages rapid prototyping and iterative experimentation, especially in agile and startup environments, through a “describe–generate–run” workflow. It also promotes cross-functional collaboration, allowing non-technical stakeholders to directly contribute to the development process and ensuring that final products better reflect the team’s original vision.

However, this accelerated development model presents notable challenges related to quality, security, and sustainability. AI-generated code, though produced quickly, may contain hidden bugs, vulnerabilities, and architectural weaknesses, particularly when users skip thorough reviews and testing. The tendency to rely on AI for debugging and validation can lead to technical debt and fragile systems, raising concerns about a potential quality assurance gap. Additionally, excessive dependence on automated tools may erode fundamental programming skills among students and professionals. Therefore, while intent-driven development serves as a powerful innovation enabler, it should complement—not replace—traditional engineering practices, with strong human oversight and structured governance ensuring long-term reliability and security.

CONCLUSION

The incorporation of artificial intelligence into the software development lifecycle marks a significant shift in how digital products are designed and built. Instead of concentrating solely on manual coding and syntax, developers can now communicate complex requirements through natural language, allowing AI systems to generate and refine the underlying code. This evolution signals a move from rigid, code-centric practices to more adaptive, intent-based workflows that emphasize achieving functional results over writing every line manually.

One of the key benefits of this transformation is the substantial boost in productivity and speed. By automating routine tasks such as generating boilerplate code, performing initial debugging, and preparing documentation, AI enables developers to concentrate on strategic design decisions and innovative problem-solving. This advantage is especially impactful in fast-paced environments like startups, research initiatives, and competitive industries, where rapid development and quick deployment are critical. Additionally, the use of natural language tools lowers entry barriers, allowing students, domain experts, and non-technical contributors to actively engage in software creation, while AI assistants in education serve as interactive guides that enhance learning efficiency.

One of the key benefits of this transformation is the substantial boost in productivity and speed. By automating routine tasks such as generating boilerplate code, performing initial debugging, and preparing documentation, AI enables developers to concentrate on strategic design decisions and innovative problem-solving. This advantage is especially impactful in fast-paced environments like

startups, research initiatives, and competitive industries, where rapid development and quick deployment are critical. Additionally, the use of natural language tools lowers entry barriers, allowing students, domain experts, and non-technical contributors to actively engage in software creation, while AI assistants in education serve as interactive guides that enhance learning efficiency.

However, heavy dependence on AI-generated outputs introduces important concerns related to code quality, security, and foundational understanding. Relying solely on automated suggestions can result in inconsistent structures, unnoticed vulnerabilities, and weaker comprehension of system architecture. Therefore, while intent-driven development offers powerful support and accelerates innovation, it should complement—not replace—core engineering expertise. Responsible use, supported by careful human oversight, structured testing, and adherence to fundamental principles, will ensure that this evolving approach strengthens rather than undermines the future of software engineering.

REFERENCES

[1] Andrej Karpathy, "There's a new kind of coding I call 'vibe coding'," X (formerly Twitter), 2025.

[2] Ahmed Fawzy, Amjed Tahir, and Kelly Blincoe, "Vibe Coding in Practice: Motivations, challenges, and future outlook - A Grey literature Review," Zenodo, 2025.

[3] Yujia Fu et al., "Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study," ACM TOSEM, 2025.

[4] Domenico Cotroneo, Cristina Improta, and Pietro Liguori, "Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity," arXiv preprint arXiv:2508.21634, 2025.

[5] Hammond Pearce et al., "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," Communications of the ACM, vol. 68, no. 2, pp. 96–105, 2025.

[6] Shraddha Barke, Michael B. James, and Nadia Polikarpova, "Grounded copilot: How programmers interact with code generating models," PACMPL, vol. 7, OOPSLA1, pp. 85–111, 2023.

[7] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," Information and Software Technology, vol. 106, pp. 101–121, 2019.

[8] Gergely Orosz and Elin Nilsson, "Vibe Coding as a Software Engineer," The Pragmatic Engineer, 2025.

[9] James Prather et al., "'It's weird that it knows what I want': Usability and interactions with Copilot for novice programmers," ACM TOCHI, vol. 31, no. 1, 2023.

[10] Namanyay Goel, "Vibe Coding is a Dangerous Fantasy," nmn.gl blog, 2025