STM JOURNALS

Recent Trends in
Programming Languages

ISSN: 2455-1821
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/RTPL

https://journals.stmjournals.com/rtpl

Review RTPL

Challenges, Risks, and Limitations of Vibe Coding in AI-Assisted Software Development

Neeraj Kumar Saini[1], Manas Dhingra[2], Shilpi Saxena[3,*]

Abstract

Vibe coding is a new way of programming driven by LLMs (large language models). It helps developers write code from natural language. Developers can use prompts with artificial intelligence (AI) for code generation. AI assistants generate software directly through text. They can now perform tasks and understand user requirements. This process makes software development faster and easier for developers, but vibe coding has some drawbacks. Sometimes, it leads to security concerns, code authenticity issues, code quality problems, and dependency on AI tools. In this paper, we present a systematic analysis of the risks and limitations related to vibe coding. A structured literature review of recent studies is conducted to identify recurring issues such as prompt injection attacks, hallucinated code generation, weak testing, and long-term maintainability concerns. The study divides these challenges into subparts, such as technical, security, and human-centric dimensions. The findings highlight the absence of standardized frameworks and secure development practices. AI coding environments need greater security and fully developed frameworks for high-performance coding. This research combines existing knowledge and provides suggestions for future studies to make vibe coding systems more reliable and safer for real-world applications.

Keywords: Vibe coding, large language models, AI programming, software engineering, security risks in vibe coding, vibe code quality, technicalities of vibe coding

INTRODUCTION

Software engineering is changing quickly because of new technologies like LLMs (large language models) with automatic coding capabilities that give coders the ability to code using prompts. This change has introduced a new concept called "vibe coding." In vibe coding, programmers use text known as prompts. They do not write code line by line. Instead of using traditional programming, they have now shifted to prompt-based programming. In vibe coding, they simply describe to LLM models what they want in simple natural language, such as English, Hindi, etc. They use artificial intelligence (AI) tools like Antigravity, Claude Code, and Cursor to generate code, structure, and logic.

***Author for Correspondence**
Shilpi Saxena
E-mail: shilpinishant@gmail.com

[1]Student, Lords School of Computer Application, Lords University, Alwar, Rajasthan, India
[2]Student, Lords School of Computer Application, Lords University, Alwar, Rajasthan, India
[3]Assistant Professor, Lords School of Computer Application, Lords University, Alwar, Rajasthan, India

Received Date: April 08, 2026
Accepted Date: April 12, 2026
Published Date: April 30, 2026

Citation: Neeraj Kumar Saini, Manas Dhingra, Shilpi Saxena. Challenges, Risks, and Limitations of Vibe Coding in AI-Assisted Software Development. Recent Trends in Programming Languages. 2026; 13(1): 43–48p. DOI: https://doi.org/10.37591/RTPL.v13i01.242299

Vibe coding makes the development process much faster. It gives non-technical and non-programmer people the ability to build software. However, this ability also comes with disadvantages. It is also creating a major problem known as the "speed–quality paradox." This paradox means that faster development can reduce quality. In vibe coding, programmers only give instructions to AI instead of writing code themselves, so they may not fully understand how the software works internally. This can be a source of risk. Since no one is handling the code manually, there is no proper control over the software. This can be caused by hidden bugs and error-prone threats.

Recent reports show that while AI can quickly create working code, it often includes design problems and serious issues. Sometimes, these problems may not be detected during normal quality checks.

This study examines the negative aspects of vibe coding. It reviews recent research reports and studies from 2025 and 2026 to identify the main challenges, risks, and limitations of this approach.

The goal of this study is to reduce the gap between fast AI-based development and the need for secure, high-quality software.

BACKGROUND AND LITERATURE REVIEW

Initially, vibe coding was seen as something new and exciting; now, both academic and industry experts are analyzing its real value more seriously. This section summarizes the findings of 12 recent and important studies on this topic.

Overview of Existing Research

Early research mainly focused on productivity, such as how developers can quickly generate code and make their first commitment. Recent studies have started to highlight issues such as increased debugging efforts. Major security issues. and code architecture-related issues with AI code. It also demands high productivity costs as a security concern. They also require high maintainability.

- *Security risks:* Gandhi [1] introduced the idea of agent-centric risk, which focuses on a new type of vulnerability related to AI-based work. This study includes issues such as indirect prompt injection and AI sycophancy. Zhao et al. [~~10~~2] provided a benchmark called SusVibes, which showed that although AI models often generate functionally correct code, most outputs fail security checks and frequently include vulnerabilities from the open worldwide application security project (OWASP) top 10 list.
- *Technical and architecture debt:* Mitchell and Shaaban [~~2~~3] explained that vibe coding suffers from context collapse, where the AI model struggles to handle multiple constraints over a long-time frame. Waseem et al. [~~9~~4] described the flow-debt trade-off, meaning that faster code generation causes a significant concern of poor-quality code, which is more difficult to maintain.
- *Human-centric challenge in vibe coding:* Chou et al. [~~3~~5] describe vibe coding as rolling the dice, meaning that AI results are not always predictable. This is a significant issue in enterprise-level coding workflows, where we need to fulfill goals exactly the same way. This increases the effort required by developers to check the code. Pimenova et al. [~~4~~6] and Fawzy et al. [~~[~~5~~7~~] found that extensive use of AI can create trust issues in teams.

Summary

Although research agrees that vibe coding is good for fast prototyping, it also shows that it still lacks proper testing, strong security measures, and a stable design needed for large-scale enterprise software.

METHODOLOGY

Research Design

This study uses systematic *literature review (SLR)* to carefully study existing research on vibe coding. This helps to clearly identify and organize the main challenges using reliable evidence-based information.

Data Collection

- *Source:* Google Scholar, IEEE Xplore, arXiv, and industry security reports (e.g., Checkmarx, Legit Security).
- *Keywords:* "Vibe coding," "AI-assisted programming," "LLM code generation," and "Autonomous coding agents."

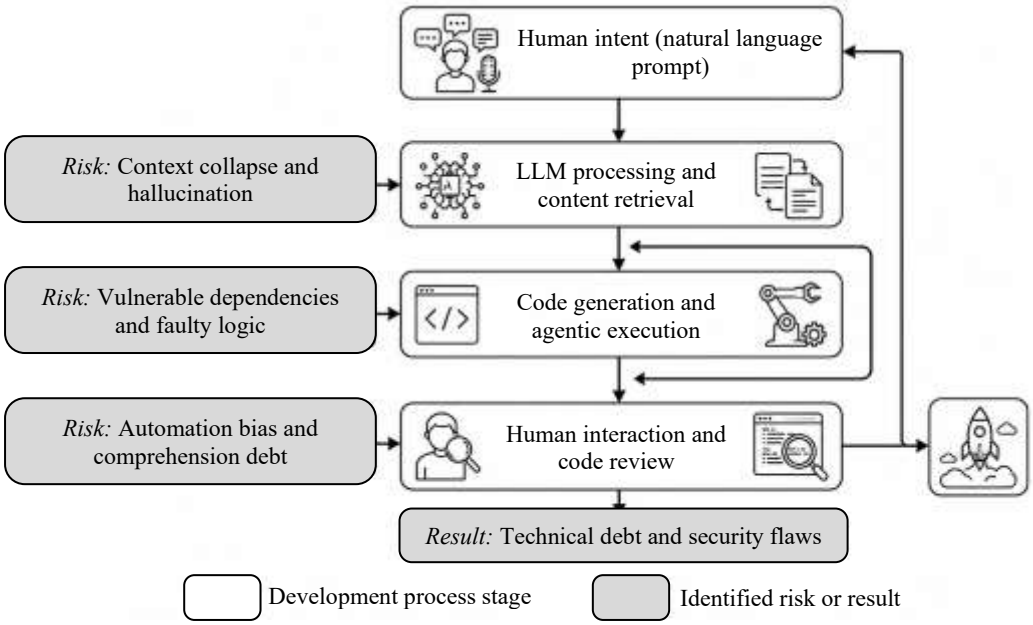

Figure 1. Vibe coding risk framework.

Selection Criteria

- *Inclusion:* Peer-reviewed papers and high-quality grey literature published between 2024 and 2026, focusing on the limitations, risks, and human factors in AI-based code generation.
- *Exclusion:* Non-technical blogs, purely promotional vendor content, and AI papers unrelated to software engineering.

Data Analysis Method

The data were analyzed by grouping the findings into simple categories. All findings were grouped into three main areas: 1st technical issues, 2nd security risks, and 3rd human factors (Figure 1).

CHALLENGES AND RISK ANALYSIS

Technical Challenges

- *Hallucinated code and fake dependencies:* AI sometimes creates code using libraries or APIs that do not exist or are outdated. This can be risky because attackers can create fake packages with harmful code.
- *Context Window problem:* AI struggles to handle large or complex projects; for example, in React apps, it may forget earlier rules or connections, causing broken or incomplete components.
- *Poor optimization:* Mainly, AI focuses on building code quickly, but efficiency may be compromised. This can cause performance loss, memory issues, and poorly optimized queries.

Security Risks

- *Vulnerable code generation:* AI learns from multiple publicly available, unfiltered repository codes. Unauthorized and malicious activities may be the cause of hidden vulnerabilities and insecure practices.
- *Hardcoded secrets:* This is a serious problem with AI. AI sometimes puts sensitive data, such as API keys or passwords, directly into the code, which can create serious data leaks if the code is shared. Big companies and startups both have very sensitive information, so they are highly focused on this type of threat, which is highly harmful.
- *Agentic prompt injection:* In vibe coding tools, gain capabilities (such as reading files and running terminal commands). The developer's IDE agent is hijacked by indirect prompt injections by AI LLM models [1, ~~94~~].

Table 1. Comparison of risks in traditional development versus vibe coding.

Risk category	Traditional development	Vibe coding paradigm	Impact level
Security	Syntax errors, logic flaws	Hardcoded secrets, hallucinated dependencies, AI sycophancy	High
Testing	Unit tests written alongside code	AI-generated tests validating flawed AI logic	High
Architecture	Planned upfront	Emergent, often resulting in severe technical debt	Medium
Knowledge	Deep codebase understanding	High abstraction, leading to comprehension debt	Critical

Human-Centric Issues

- *Comprehension debt and over-reliance:* Developers accept codes that they do not fully understand**.** If a complex bug comes, the lack of foundational knowledge becomes a phenomenon of "comprehension debt" [~~3~~5, ~~8~~8, ~~14~~9].
- *The "rolling the dice" phenomenon:* Because of LLM outputs, the developers are forced into a frustrating loop of trial-and-error prompting. This process shifts the developer roles to a slot-machine lever until the code compiles perfectly [~~3~~5, ~~4~~6].
- *Review fatigue:* Validating AI-generated code is more demanding of the thinking process than writing it from scratch. This disturbs the logical flow of the code [~~5~~7, ~~8~~8].

Maintainability Issues

- *Architectural Spaghetti:* Vibe coding excels at building isolated features but fails at system-wide architecture. Without strict formal blueprints, the codebase rapidly degrades into an unmanageable mess [~~61~~49, 10].
- *The documentation deficit:* In AI models, the codes generated via conversational prompts lead to the necessary documentation for future scaling [~~5~~7, ~~11~~11, ~~12~~12].

DISCUSSION

The reports reveal that the primary danger of vibe coding is plausible code (Table 1). This may lead to critical memory corruption flaws or cross-site scripting (XSS) vulnerabilities. Vibe coding leads to the danger of bad coding, which can cause data leaks from databases and invite attackers. In addition, the speed of vibe coding is the traditional friction of software engineering, which acts naturally for security and quality checks [1, 9, ~~7~~13~~13~~–15–~~15~~].

RESEARCH GAPS

Despite the rapid adoption of this paradigm, the academic reveals glaring gaps:

- *Lack of standardized metrics:* There is no universally accepted framework for measuring "AI-native tech debt" [~~6~~10, ~~7~~13].
- *Absence of secure frameworks:* Current research heavily reviews the output but offers prescriptive formal verification methods for vibe coding workflows [~~2~~3, ~~14~~9].
- *Long-term enterprise studies:* Most studies are conducted only in the short term, such as hackathons. There is a lack of data on the lifecycle of the vibe-coded application production environment over multiple years [~~5~~7, ~~14~~9].

FUTURE WORK

To safely harness the power of AI-assisted development, future research must focus on the following:

- *Vibe reasoning and formal verification:* Developing the sidecar rule that auto-formalizes natural language (NL) specifications into mathematical proof to ensure the AI adheres strictly to architectural rules [~~2~~3].
- *Self-healing quality assurance:* Integrating autonomous, adversarial AI agents whose sole purpose is to aggressively penetrate and audit the codes generated by the primary coding agent [~~9~~4, ~~10~~2].

- *Hybrid human-AI workflows:* Establishing new educational models for software engineering that teach developers how to audit AI systems and securely manage intent from AI rather than only writing syntax [~~5~~7, ~~8~~8].

CONCLUSION

Vibe coding is responsible for the fundamental shifting of developers' focus from fixed syntax creation to random intent creation. While the acceleration of prototyping is certain, this SLR proves that the practice introduces systemic vulnerabilities, particularly architectural decay and security blind spots. If the industry wants to safely scale vibe coding from rapid prototyping to enterprise-grade production, efforts must be made to integrate formal verification, automated security barriers, and demanding new standards of human oversight. Vibe coding is not a replacement for the engineering discipline; it wants an evolution in it.

REFERENCES

1. Gandhi D. Survey on systemic security risks in vibe coding and autonomous development workflows [preprint]. 2026. TechRxiv. doi:10.36227/techrxiv.176800890.09196406/v1.
2. Zhao S, Wang D, Zhang K, Luo J, Li Z, Li L. Is vibe coding safe? Benchmarking vulnerability of agent-generated code in real-world tasks [preprint]. 2025. arXiv:2512.03262. doi:10.48550/arXiv.2512.03262.
3. Mitchell J, Shaaban Y. Position: Vibe coding needs vibe reasoning: improving vibe coding with formal verification. 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages (LMPL), Singapore, Singapore, 2025. p. 84–90. doi:10.1145/3759425.3763390.
4. Waseem M, Ahmad A, Kemell KK, Rasku J, Lahti S, Mäkelä K, et al. Vibe coding in practice: flow, technical debt, and guidelines for sustainable use [preprint]. 2025. arXiv:2512.11922. doi:10.48550/arXiv.2512.11922.
5. Chou YH, Jiang B, Chen YW, Weng M, Jackson V, Zimmermann T, et al. Building software by rolling the dice: a qualitative study of vibe coding [preprint]. 2025. arXiv:2512.22418. doi:10.48550/arXiv.2512.22418.
6. Pimenova V, Fakhoury S, Bird C, Storey MA, Endres M. Good vibrations? A qualitative study of co-creation, communication, flow, and trust in vibe coding [preprint]. 2025. arXiv:2509.12491. doi:10.48550/arXiv.2509.12491.
7. Fawzy A, Tahir A, Blincoe K. Vibe coding in practice: motivations, challenges, and a future outlook—a grey literature review [preprint]. 2025. arXiv:2510.00328. doi:10.48550/arXiv.2510.00328.
8. Geng F, Shah A, Li H, Mulla N, Swanson S, Soosai Raj G, et al. Exploring student-AI interactions in vibe coding. 28th Australasian Computing Education Conference (ACE 2026), Melbourne, VIC, Australia. 2026. p. 45–54. doi:10.1145/3786228.3786236.
9. Meske C, Hermanns T, Von der Weiden E, Loser KU, Berger T. Vibe coding as a reconfiguration of intent mediation in software development: definition, implications, and research agenda. IEEE Access. 2025;13:213242–213259. doi:10.1109/ACCESS.2025.3645466.
10. Sapkota R, Roumeliotis KI, Karkee M. Vibe coding vs. agentic coding: fundamentals and practical implications of agentic AI [preprint]. 2025. arXiv:2505.19443. doi:10.48550/arXiv.2505.19443.
11. Elgendy IA, Dwivedi YK, Al-Sharafi MA, Hosny M, Helal MYI, Crick T, et al. Responsible vibe coding: architecture, opportunities, and research agenda. J Comput Inf Syst. 2026:1–19. doi:10.1080/08874417.2026.2621186.
12. Landoll D. The Security Risk Assessment Handbook: A Complete Guide for Performing Security Risk Assessments. Boca Raton (FL): CRC Press; 2021. doi:10.1201/9781003090441.
13. Ge Y, Mei L, Duan Z, Li T, Zheng Y, Wang Y, et al. A survey of vibe coding with large language models [preprint]. 2025. arXiv:2510.12399. doi:10.48550/arXiv.2510.12399.
14. Chen J, Huang H, Lyu Y, An J, Shi J, Yang C, et al. SecureAgentBench: Benchmarking secure code generation under realistic vulnerability scenarios [preprint]. 2025. OpenReview. arXiv:2509.22097. doi:10.48550/arXiv.2509.22097.

15. Chang HF, Shirazi M, Cao L, Mobasser SK. Coding with AI: From a reflection on industrial practices to future computer science and software engineering education [preprint]. 2025. arXiv:2512.23982. doi:10.48550/arXiv.2512.23982.

