UDC 004.8:004.413

THE IMPLEMETNATION OF VIBE CODING IN SOFTWARE ENGINEERING

Kungurtsev A.V.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Goncharova I.V. – Lecturer at the Department of Foreign Languages

Annotation. Vibe coding is an emerging AI-driven development technique characterized by a more intuitive, "by feel" approach, using natural language prompts for LLMs and involving minimal manual line-by-line programming, applied to create and improve software models. In order to understand this groundbreaking paradigm shift, in this article we examine the main points of practical implementation, effects on the labor market, and the benefits and risks of the phenomenon in question. The main objective is to dig into queries of maintainability, security, long-term technical efficiency and speed and accessibility.

Keywords: Vibe Coding, Large Language Models, Natural Language Programming.

Introduction. For decades, software engineering has been all about taking things to the next level of Annotationion. We moved from machine code to assembly, then to high-level languages like C, C++ and Java. Now we have frameworks that take care of the basics of architecture build-up and make it primitively Annotation by delivering the developer from manually configuring the files and CLI (Command Line Interface) commands. Today, we are witnessing the latest and perhaps most radical leap: «Vibe Coding.» Originated within developer circles and popularized by prominent figures in the field of computer science like Andrej Karpathy, the term «Vibe Coding» refers to the methodology where the developer functions more as a supervisor and a manager of intent than as a writer of syntax. The concept implies that the developer «vibes», so gets along and is in harmony with a LLM such as Claude, Chat GPT, Gemini, etc., describing desired outcomes in natural language and continuously makes adjustments, instead of meticulously crafting and applying functions manually. This shift represents a fundamental change in the relationship between human logic and machine execution, as the ratio of involvement changes towards the machine (shown in Figure 1).

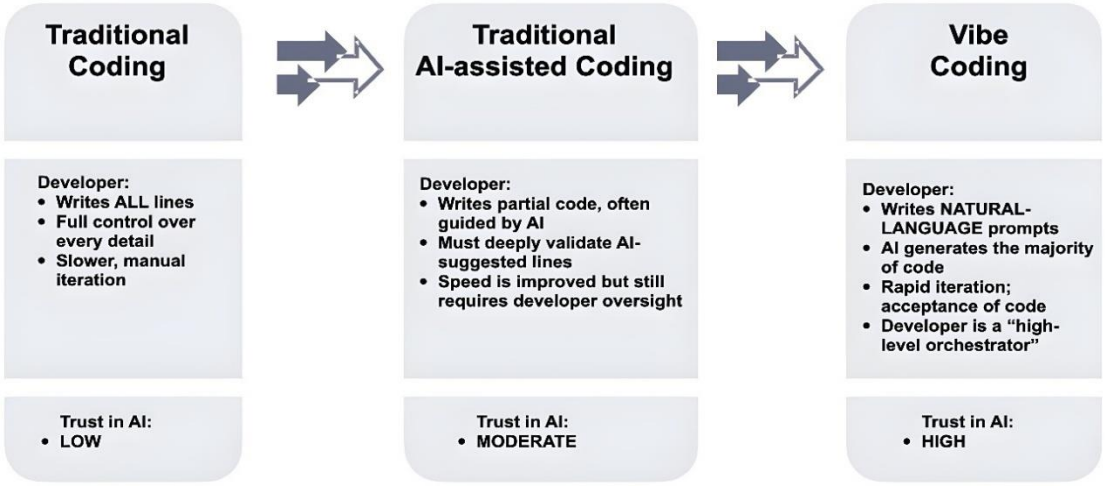

Figure 1 – Representation of a paradigm shift in coding

Main part. At its core, vibe coding is the realization of Natural Language Programming (NLP) powered by Large Language Models (LLMs). According to Google Cloud and IBM, the gist of this phenomenon is the shift from thinking about how to do something and focus on what exactly needs to be done. Traditional coding requires the developer to translate an algorithm into a rigid, syntactically correct sequence of commands. Vibe coding bypasses this translation layer. The developer provides a high-level «vibe» (a description of a feature, a UI layout, or a chain of reasoning) and the AI generates the underlying foundation of code, logic, and multi-file integration.

This is not an exaggeration of filling in the blanks. It is a back and forth conversation. Vibe coding is based on the models' ability to understand the situation when the plan is not clear and composed enough.

Today the implementation of vibe coding lies within integrated development environments (IDEs), that provide a suitable UI and all the necessary tools. Examples include Google AI Studio and Gemini-powered tools that generate full apps and support one-click deployment, IDE assistants that insert and test code, and agent frameworks that orchestrate multi-step tasks. Tool choice depends on one's requirements and goals, such as multi-file coherence, long-running agentic tasks, certain platform support, speed, cost and quality.

In the labor market influence of vibe coding is a subject of intense debate. The new approach lowers the barrier to entry for building software, enabling non-programmers and domain experts to create applications, which can accelerate innovation by reducing production costs and time-to-market for small projects. On the other hand, the generated code can also be complicated, duplicated, or poorly structured, increasing long-term maintenance costs, as studies report higher rates of logical errors and vulnerabilities in AI-written code, and real incidents show that agents can sometimes “hallucinate” or fabricate data.

In addition, the «junior developer paradox» is becoming more evident and problematic. If AI handles all the entry-level coding tasks, junior developers may find fewer opportunities to learn the fundamental syntax and debugging skills required for senior-level architectural oversight. When a developer «vibes» their way into building a complex system without understanding the underlying logic, they become unable to fix the system when their methods become unreliable, for instance when AI generates «spaghetti code» that functions in the short term but is impossible to refactor manually later. This creates a gap in skill levels, which can lead to crisis in the industry.

The cause of the emergence of vibe coding is the massive expansion of LLM context windows and reasoning capabilities; the effect is a decoupling of «coding» from «programming». In the future, we can expect that there will be a high-demand niche for those who build and optimize the LLMs and underlying systems and a huge number of vibe-coding engineers who assemble products through high-level management. However, the future outlook provided by gray literature reviews suggests a looming «maintenance crisis.» As billions of lines of “vibed” code enter production, the industry will eventually require a new generation of AI tools specifically designed to deconstruct and clean up the unoptimized outputs of today’s generative agents.

Conclusion. The implementation of vibe coding in software engineering is an inevitable evolution, representing the ultimate Annotationion of logic. Giving freedom to human creativity, it democratizes application development by focusing on architecture and letting a machine do the more monotonous part of the work. However, software engineering is more than just making things work; it is about making things last. Practical implementation should treat vibe coding as a toolchain component for ideation but requiring testing, code review, and refactoring before production use. Organizations that combine agentic automation with disciplined engineering basics can benefit while lowering the documented risks to quality and security. The most reasonable path is not to abandon engineering basics but to adapt it for a world where natural language and autonomous agents are first-class development inputs.

References

1. *Vibe Coding Explained: Tools and Guides [Electronic resource]. Mode of access: https://cloud.google.com/discover/what-is-vibe-coding. Date of access: 06.03.2026.*

2. *Vibe coding [Electronic resource]. Mode of access: https://en.wikipedia.org/wiki/Vibe_coding. Date of access: 06.03.2026.*

3. *Vibe coding: programming through conversation with artificial intelligence [Electronic resource]. Mode of access: https://arxiv.org/abs/2506.23253. Date of access: 06.03.2026.*

4. *A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions [Electronic resource]. Mode of access: . Date of access: 06.03.2026.*

5. Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook – a Grey Literature Review [Electronic resource]. Mode of access: https://www.researchgate.net/publication/396094422_Vibe_Coding_in_Practice_Motivations_Challenges_and_a_Future_Outlook_-_a_Grey_Literature_Review. Date of access: 06.03.2026.

6. The Evidence Against Vibe Coding: What Research Reveals About AI Code Quality [Electronic resource]. Mode of access: https://www.softwareseni.com/the-evidence-against-vibe-coding-what-research-reveals-about-ai-code-quality/. Date of access: 06.03.2026.