Turiba University

(Jyoti)

（ Friberly Coding ）

Professional Bachelors Degree

Computer System

Author
Jyoti

Thesis Advisor
Jānis Pērkas

What is Fiberly Coding ?

It appears that "FRiBeRLy CODiNG" is likely a stylized or misspelled version of Vibe Coding (or Vibecoding), which is a modern concept in software development driven by Artificial Intelligence (AI).

Vibe Coding building software by giving natural language instructions (prompts) to a Large Language Model (LLM), which then generates the code

Vibe Coding: **A New Paradigm in Software Development**

I. Introduction: What is Vibe Coding?

- **Definition:** Vibe Coding is a method of software development where the user relies heavily on Generative AI (specifically, Large Language Models or LLMs) to write the code based on natural language prompts.

- **The Shift**: It fundamentally shifts the developer's role from writing syntax to directing, testing, and refining the AI's output.

- **Core Principle**: The user focuses on what the software should do (the intent or "vibe"), not how to write the technical code

.

- # Advantages and Use Cases

Vibe Coding lowers the barrier to entry and dramatically speeds up development.

A. Key Advantages

1) Rapid Prototyping: Quickly generate a Minimum Viable Product (MVP) or functional demo in hours, not weeks.
2) Accessibility: Allows non-coders (entrepreneurs, designers) to build working software without deep technical expertise.
3) Automation: Automates the creation of boilerplate, low-level, or repetitive code, freeing up expert developers.

## B. Major Use Cases

1. Proof-of-Concept: Creating initial prototypes to quickly test a business idea.
2. Task Automation: Building simple, personalized scripts or tools for individual needs (e.g., a data sorter).
3. Learning: Helping new developers or students understand coding concepts by seeing instant, working examples.

• Challenges and Best Practices

1. Code Quality & Security: AI-generated code may contain bugs, be less optimized, or have security vulnerabilities if not thoroughly reviewed.

2. Lack of Accountability: If the user accepts code without understanding it, ownership and responsibility for flaws become blurred.

3. Complexity Limit: LLMs may struggle with highly complex, novel, or specialized architectural problems.
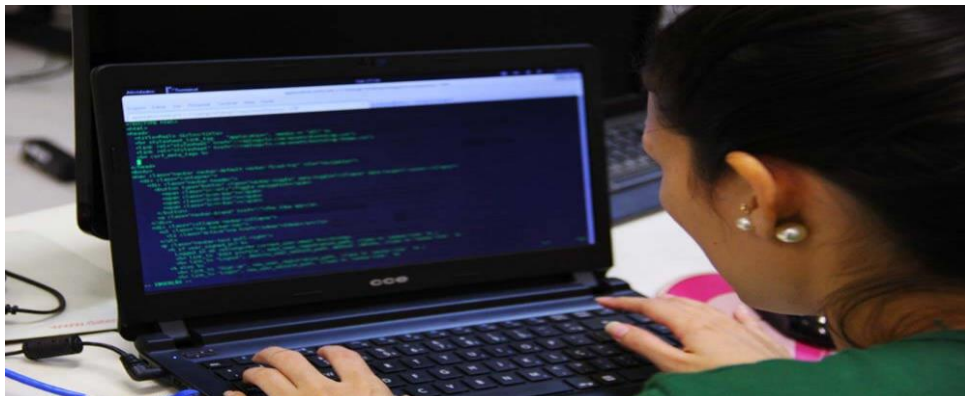
B. Best Practices

1. Test Everything: Never deploy AI-generated code without rigorous human testing and validation.

2. Use for Drafts: Leverage AI for first drafts and low-level tasks, but have a developer own the final architecture and critical logic.

3. Specific Prompts: Provide clear, precise, and explicit instructions to the LLM to get accurate results.