# **Vibe Coding**

Vibe Coding is a relatively new way of building software engineering products which is heavily reliant on using agentic technologies or LLMs and rely more on the user’s intuition and interaction instead of formal design and planning. The user/developer will not make detailed specs about the design of the application but will rather feed in short prompts in natural language to let the LLM generate a workable application. It eventually becomes a rinse and repeat process which you are adjusting, testing and refining the product using subsequent prompts. The main idea is to cut out time spent in building frameworks and stay in a flow state of converting ideas into working pieces.

## **Comparison between Karpathy & Willison**

Andrej Karpathy describes vibe coding as a creative shift where instead of focusing on syntax, developers focus on intent. He/She tells the LLM what the application is supposed to do, and it gets you part way there. Even though it is not perfect, your job is cut in half because the LLM will generate half of the working code on itself. Additional changes can be implemented with subsequent prompts and you can try and fine tune your application simply by interacting with the agent.

Simon Willison, on the other hand, suggests that vibe coding is completely relying on AI-generated code without scrutiny and this is something that is very risky in a high stakes production environment. In his recent blog, he suggests vibe coding is more suitable for low-stakes projects & prototypes where rapid development is prioritized over code quality and maintainability. He also emphasizes the difference between vibe coding and AI-assisted programming but does go on to say that vibe coding is great at lowering the barrier to entry for new developers. The difference in both of their approaches is that of suspicion. Karpathy is very optimistic about using vibe coding as a technique while Simon understands and recognises the fact that vibe coding has potential but that does not mean that you can just run with it.

**Tool Comparison**

|  | Key Features | Use Case Fit (Ethics) | Ease of use | Integration Depth |
| --- | --- | --- | --- | --- |
| Copilot | Inline suggestions, test writing, quick code scaffolds | Inline suggestions, test writing, quick code scaffolds | Moderate (IDE setup needed) | High (context-sensitive completions) |
| Replit AI Agent | AI chat, in-browser IDE, one-click hosting and UI preview | Ideal for quick prototyping of interactive stories | Very easy (no install) | Very high (real-time + multimodal) |
| Cursor + Sonnet | Completions plus natural language chat in a VS Code wrapper | Strong for refining story branches and dialogue flows | Easy (streamlined setup) | High (semantic + conversational) |

## **Case Study: NicolasZu’s Flight Simulator**

In *The Vibe Coding Bible*, developer NicolasZu creates a flight simulator by simply prompting Copilot with ideas like “simulate plane pitch angle” or “add a cockpit display.” Instead of scripting every piece in advance, he starts with an idea and lets the AI take a first pass. Every feature that was added is built using a short input followed by testing the output, tweaking and repeating.

The main idea is not just focus on having a plan beforehand and sticking to it but rather let the structure take form as he built the app. The feedback given over multiple iterations resulted in a working, interactive flight sim complete with weather systems and dashboard visuals built through short exchanges with AI. He stresses on the fact the quality of work done using this approach is heavily dependent on how the developer guides the AI. There is a requirement of skillful prompting and communicating with AI effectively.