

Vibe Coding: Building Real Things with AI

Middle School Vibe Coding — Grades 6–8

Tech for Everyone

CHAPTER 1

The New Way to Build

When AI Can Write the Code, Your Ideas and Judgment Matter More

A few years ago, turning an idea into working software meant months of memorizing syntax — every semicolon, every bracket, every keyword had to be exactly right or nothing ran at all. Today you can describe what you want in plain English and watch an AI build it in seconds. This is called vibe coding — and this chapter is about what it really is, how it works, and why it makes *your* thinking more valuable, not less. The tools have changed dramatically, but the person steering them matters more than ever, and by the end of this chapter you'll understand exactly why.

1.1 — What Vibe Coding Actually Is

vibe coding means building software by *describing* what you want in everyday language and letting an AI generate the code for you. Instead of writing every line yourself, you write a request, read what the AI produces, run it to see what it does, and guide it toward what you actually meant. The name comes from the idea that you describe the *vibe* — the feel and behavior — of what you want, and the AI handles the exact wording of the code.

Here's a concrete picture. Imagine you want a program that greets a user by name. The old way, you'd have to know that in a language like Python you write something like `print("Hello, " + name)` — and if you forgot a quotation mark, the whole thing would fail. The vibe coding way, you type: *"Ask the user for their name, then greet them by it."* The AI writes the exact syntax for you. You still have to check that it did the right thing, but you didn't have to memorize the punctuation.

To be clear about what's new here: people have used computers to help write code for years, with autocomplete and error-checkers. What's different now is the *scale* — you can describe an entire feature in a sentence and get working code back, so the skill shifts from producing the code to directing and checking it. The term "vibe coding" was coined in 2025 and spread fast, because it captures a real shift: the hardest part of building is moving from *what you type* to *what you ask for and how you judge the answer*. For decades, the bottleneck in software was translation — turning a clear human idea into stiff, unforgiving computer syntax. AI removes most of that translation step. But notice what it does **not** remove: you still have to know what you want,

explain it clearly, and recognize whether the result is any good. Vibe coding doesn't erase the thinking — it moves the thinking to a new place, closer to design and judgment and further from typing.

That's why the *way* you do it matters so much. Done badly, vibe coding is just clicking "accept" on whatever the AI spits out and hoping it works — which is how people ship apps full of bugs they never noticed. Done well, it's a skilled loop of describing what you want, testing what you get, spotting what's wrong, and refining your request until the result is genuinely good. Think of two students given the same AI: one accepts the first answer and moves on; the other reads it, tries to break it, and improves it three times. They finish with wildly different projects, even though they used the exact same tool. This book teaches the "done well" version — the habits that make you the reason the project turns out great.

Try It

Reverse-Engineer an App You Use

Open an app you use every day and pick one small feature — the "like" button, a search bar, a streak counter. In one or two sentences, describe *exactly* what it does, as if you were asking an AI to build it. Now notice everything you left out: what it looks like, what happens when it's tapped twice, what if there's no internet. That gap between "what I said" and "what it really does" is the whole skill of vibe coding, and spotting it is the first step to closing it.

1.2 — How the AI Writes Code

Your AI helper is powered by a large language model (LLM) — a program trained on enormous amounts of text and code from the internet, books, and public software. When you ask for something, it doesn't "look up" the answer in a filing cabinet of correct programs. Instead it **predicts**, piece by piece, the most likely next word or symbol that fits your request, based on the patterns it saw during training. In a real sense, it is an extremely sophisticated autocomplete — the same idea as your phone suggesting the next word in a text, but trained on so much code that

its guesses are often good enough to run.

This explains both why it feels magical and why it sometimes fails. Because it has seen millions of examples of, say, a "guess the number" game, it can produce one instantly. But because it's predicting rather than reasoning about *your* exact situation, that prediction is powerful but not guaranteed. It has two consequences you must remember:

- The AI can be **confidently wrong**. It can produce code that looks right, uses real commands, runs without an error message, and still does the wrong thing. For example, ask for a program that calculates an average and it might add the numbers correctly but forget to divide — the code runs fine, but every answer is wrong. It won't warn you; it'll present the mistake with total confidence.
- The AI has **no idea what you actually meant** — only what you typed. If you ask for "a game," it can't know you pictured a two-player game with a scoreboard. It fills every gap with a guess, and its guess is whatever was *most common* in its training, not what's in your head.

One more reason it slips up: the AI's training has a cutoff date, so it may not know about the newest version of a tool, and it can blend patterns from thousands of slightly different examples into something that *looks* standard but isn't quite right for your exact setup. That's not a sign it's broken — it's just what prediction does.

Here's a useful way to picture it: the AI is like a brilliant new intern who has read every programming book ever written but has never met you, can't see your screen unless you describe it, and will never admit "I'm not sure." That's a genuinely valuable helper — but only if you supply the goal and check the work. Understanding this is exactly what separates someone who *uses* AI from someone AI uses. You are working with a fast, well-read, occasionally mistaken assistant — never an all-knowing oracle.

Key Idea

An LLM predicts likely code; it does not understand your goal or guarantee correctness. Your job is to supply the goal and verify the result.



Career Takeaway

"Can Define the Problem" Is the Job Now

As AI takes over the typing, employers increasingly want people who can do what AI can't: figure out *what* to build, describe it clearly, and judge whether the result is actually good. Product designers, engineers, analysts, and founders all live in that space. The good news is you can practice it right now — every time you turn a fuzzy idea into a precise request and then check the result, you're rehearsing one of the most valuable skills in the modern workforce.

1.3 — Your New Role: Engineer and Designer

For a long time, being "good at coding" mostly meant being good at *writing* code — memorizing commands and typing them without mistakes. When AI can generate that code for you in seconds, the memorizing-and-typing part stops being the valuable part. The skills that matter now are the ones AI *can't* do for you:

- **Deciding what to build** and why it matters. The AI will never wake up and decide your school needs a lunch-line timer app. That vision comes from you.
- **Describing the problem** precisely enough to get a good result. The clearer your request, the better the code — and writing clear requests is a real skill you'll practice all book long.
- **Judging quality** — is this actually good, correct, and safe? The AI can't tell you its own answer is wrong, so someone has to, and that someone is you.
- **Debugging** — finding and fixing the flaw the AI introduced. When something breaks, "make it work" isn't a plan; figuring out *what* broke is the job.

Notice that none of these are typing. They're *thinking* skills: design, communication, and judgment. This is why the title of this chapter says your role shifts from typist to engineer and designer. A software engineer has always spent far more time deciding what to build and whether it works than actually typing — and now that's true for you from day one.

Picture two students building the same weather app. The first types "make a weather app," accepts whatever comes back, and turns it in. The second decides it should show the temperature *and* whether to bring a jacket, notices the AI forgot the jacket advice, tests what happens on a freezing day, and asks for a fix. Both used the same AI. Only the second one actually engineered anything — and it shows in the result. The difference wasn't typing speed; it was deciding, describing, judging, and debugging.

Think about how movies get made. The most important person on set isn't the one holding the camera; it's the director, who decides what the scene should be, judges each take, and calls for another when it isn't right. AI is your camera — fast, capable, and completely dependent on someone telling it what to shoot and whether the shot was any good. In the AI era, the person who can define a problem and evaluate a solution is worth far more than the person who can only produce syntax. That's genuinely good news for you: the skills that now matter most — curiosity, clear thinking, and good judgment — are ones you can grow, not ones you're stuck with. You are stepping into the director's chair: the engineer who plans the work, judges it, and directs the redo — not the bystander who watches it happen and hopes for the best.



Current Events

"Vibe Coding" Went From Slang to Everywhere in Months

The phrase "vibe coding" was coined in early 2025 and, within a year, showed up in classrooms, startups, and major software companies. Tools built specifically for schools — with teacher controls and privacy protections — appeared quickly, precisely because raw consumer chatbots weren't safe or legal for student use. The speed of that shift is a lesson in itself: the tools will keep changing, so the durable skill is learning *how to work with AI*, not memorizing one app.

Think: If the specific tool you learn on is replaced in two years, what part of what you learned still matters?

1.4 — The Tools You'll Use — and Their Guardrails

You will not vibe code in a random chatbot. Consumer AI tools like ChatGPT, Claude, and Gemini have **age limits** (generally 13+ or 18+) and **privacy rules** that make them off-limits for classroom use with students. This isn't your school being overly strict — it's the law. Companies that make these tools set those age limits on purpose, because U.S. laws called COPPA and FERPA put strict rules on collecting data from young people and student records. When you type into a normal chatbot, your messages can be stored and used to train future AI. COPPA and FERPA exist so that can't happen to students without permission.

So instead you'll work inside **school-approved, teacher-governed tools** built specifically for classrooms — for example, **CodeHS Vibe Coding**. In a tool like that, a teacher can turn the AI on or off, review every message you send it, and the AI is set up to politely refuse anything that isn't about coding, so it stays on task. Your data is protected under a school agreement rather than tossed into a public system. Other classroom tools, like **MIT App Inventor**, let you build real, working phone apps with AI features built right in, all inside an environment made for students.

It helps to see these guardrails the way you'd see rules at a real workshop:

- **Age and privacy rules** keep *you* protected, so your personal information isn't collected or sold.
- **Teacher visibility** means help is close when you're stuck, and the tool can't be misused without anyone noticing.
- **On-topic AI** keeps the tool focused on building, which actually helps you learn faster.

Here's what this looks like in practice. In a teacher-governed tool, if you asked the AI to help with your history homework or to chat about the weekend, it would gently steer you back to coding — because it's built to do one job well. And because your teacher can see the conversation, the tool doubles as a safety net: if you get badly stuck or something goes wrong, help already knows what happened. Compare that to a public chatbot, where your messages disappear into a company's servers with no teacher, no protection, and no guarantee about how your words get used later.

These guardrails aren't there to slow you down — a woodshop has guards on the saws so people

can build ambitious things *safely*, not to stop them from building. The guardrails are exactly what make it possible to learn on real, powerful tools without putting your privacy at risk. And understanding *why* they exist — not just following them — is a big part of becoming a responsible creator instead of just a user. The builders companies most want to hire are the ones who take safety and privacy seriously, so this habit is worth starting now.

Key Idea

Classroom vibe coding happens in teacher-governed tools, not consumer chatbots, because laws like COPPA and FERPA protect students' data and privacy. The guardrails make real tools safe to learn on.

SEL Connection

It's normal to feel two opposite things about vibe coding at once: excited that you can build so fast, and uneasy that the AI is "doing the hard part." Here's a reframe — the AI is doing the *tedious* part, not the *hard* part. The hard part is still yours: deciding what's worth building, noticing when something's off, and caring enough to make it good. Those are human strengths. Letting AI handle the busywork doesn't make you less capable; it frees you to focus on the judgment and creativity that actually make a project yours.

Key Concepts Checklist

- ☐ I can define vibe coding and explain the "done well" version
- ☐ I can explain how an LLM generates code by predicting patterns
- ☐ I understand that AI can be confidently wrong and only knows what I typed
- ☐ I can describe how my role shifts to designing, describing, judging, and debugging
- ☐ I can explain why consumer chatbots are limited for classroom use
- ☐ I know what teacher-governed tools like CodeHS Vibe Coding do
- ☐ I can name why guardrails like COPPA and FERPA exist
- ☐ I can compare the AI to a well-read intern who has never met me
- ☐ I can explain why "thinking" skills matter more than typing in the AI era

CHAPTER 1 · ACTIVITY

Activity: From Idea to Build Plan

The New Way to Build

Great vibe coding starts before you touch a keyboard.

Part 1: Pick a Build

Choose a small project you'd genuinely use — a tip calculator, a flashcard quiz, a reaction-time game. Write one sentence describing what it does.

Part 2: Predict the AI's Gaps

List **three things the AI would get wrong or leave out** if you only gave it that one sentence. (What colors? What happens when the user makes a mistake? What device is it on?) These gaps are exactly what you'll learn to fill in Chapter 2.

Part 3: Human vs. AI

Make a two-column chart: "What the AI does for this project" and "What *I* must do." Which column holds the more important work — and why?

CHAPTER 1

Chapter Review

The New Way to Build

Chapter Summary

- **Vibe coding** is building software by describing what you want in natural language and letting an AI generate the code, then testing and refining it. Done well, it's a skilled loop — not just accepting whatever the AI produces.
- AI code helpers run on **large language models (LLMs)** that **predict** likely code from patterns; they do not look up guaranteed answers and can be **confidently wrong**.
- The AI only knows **what you typed**, not what you meant — so vague requests produce vague results.
- Your role shifts from typist to **software engineer** and designer: deciding what to build, describing it precisely, judging quality, and debugging. These are thinking skills, not typing skills.
- Consumer chatbots (ChatGPT, Claude, Gemini) have **age limits** and **privacy rules**, so classroom vibe coding uses **teacher-governed tools** like CodeHS Vibe Coding or MIT App Inventor.
- Laws like **COPPA** (under-13 privacy) and **FERPA** (student records) are why those guardrails exist.

Multiple Choice

1. Which best describes "vibe coding" done well?

- A) Accepting whatever code the AI generates without reading it
- B) Describing what you want, then testing and refining the AI's code in a loop
- C) Memorizing programming syntax before you start
- D) Coding only with colorful blocks

2. How does a large language model produce code?

- A) It searches a database for the exact correct answer
- B) It copies code from the teacher's computer
- C) It predicts likely code, piece by piece, from patterns it learned
- D) It asks another human to write it

3. Why must you always check AI-generated code?

- A) The AI refuses to run code
- B) The AI can be confidently wrong and only knows what you typed, not what you meant
- C) Checking makes the code run faster
- D) The AI never makes mistakes, so checking is just tradition

4. In the AI era, which skill becomes *more* valuable?

- A) Typing code quickly from memory
- B) Spelling every keyword perfectly
- C) Defining the problem clearly and judging whether a solution is good
- D) Avoiding computers entirely

5. Why do students use teacher-governed tools instead of consumer chatbots?

- A) Consumer chatbots are always more expensive
- B) Consumer chatbots have age limits and privacy rules that make them off-limits for

classroom student use

- C) Teacher tools are the only ones that can write code
- D) The law requires all coding to happen on paper

6. What do COPPA and FERPA protect?

- A) The speed of the school's internet
- B) The AI company's profits
- C) Children's online privacy (under 13) and the privacy of student education records
- D) The copyright on video games

Short Answer

1. Explain the difference between vibe coding "done badly" and "done well." Why does the difference matter?
2. An LLM is described as a "confidently wrong" predictor. What does that mean, and how should it change the way you use AI-generated code?
3. Your role shifts from typist to engineer/designer. Name two things in that new role that the AI *cannot* do for you, and explain why each one matters.

Key Terms

Term	Definition
Vibe coding	Building software by describing what you want in natural language to an AI, then testing and refining it
Code	The precise instructions, in a programming language, that tell a computer what to do
Large language model (LLM)	An AI trained on huge amounts of text and code that generates responses by predicting likely next words
Predicting	Making a best guess about what comes next based on patterns, rather than knowing the answer for certain
Software engineer	A person who designs, plans, and directs how software gets built and checks that it actually works, rather than only writing the code
Teacher-governed tool	A classroom AI tool where a teacher can control access and review student use
COPPA	A U.S. law protecting the online privacy of children under 13
FERPA	A U.S. law protecting the privacy of student education records

Answer Key

Multiple choice: 1. B • 2. C • 3. B • 4. C • 5. B • 6. C