Why AI Breaks Coding & How To Work With It.

Vibe-coding is the idea that you can "just describe what you want" and the AI will magically produce working software. It sounds great. It feels modern. It promises that coding is now as simple as ordering a sandwich.

But the reality is very different.

Coding breaks AI for the same reason writing breaks AI: **the model cannot hold a stable plan, a stable memory, or a stable identity long enough to build anything that has structure.**

Coding is *structured writing*. It has rules, dependencies, and consequences. If one part changes, everything else must change with it.

AI models don't understand that. They generate code as isolated fragments — not as a system.

So, you get the same failure patterns every time:

- The AI forgets earlier decisions.
- It rewrites working code for no reason.
- It introduces new functions you didn't ask for.
- It changes naming conventions halfway through.
- It "fixes" bugs by creating new ones.
- It collapses under its own output.

This is why **vibe-coding almost never works in practice**. It's not because the user is inexperienced. It's because the model cannot maintain the internal consistency that real software requires.

And the damage is real: hours lost, projects derailed, and codebases that become impossible to trust.

Part 1 - Why It Breaks

1. **Everything starts well.**

   The AI follows the initial instructions. The code looks clean, the functions appear correct, and the user feels like the system understands what they want. Early interactions build confidence.

2. **Users set rules early when they want something specific.**

   People who care about correctness set constraints at the start. They specify naming conventions, file structure, patterns, and boundaries. They say what must not be changed. They define the shape of the program before it begins.

3. **The AI follows the rules at first.**

   The first outputs look correct. The AI appears to respect the constraints. This early compliance creates trust and encourages the user to continue.

4. **The AI starts adding things you did not ask for.**

   As the coding continues, the AI generates extra functions, imports, or abstractions. These appear directly in the new code it writes. It tries to be helpful, but these additions are outside the brief. This is the first sign of drift.

5. **The additions turn into interference.**

   The AI rewrites code that was already correct. It expands simple functions into complex structures. It changes naming or patterns without being asked. All of this happens in the visible code it produces. The system starts to take over the codebase instead of assisting with it.

6. **The AI breaks the rules anyway.**

   Even with clear constraints, the AI eventually ignores them. It adds patterns that were banned. It rewrites sections that were off-limits. It forgets naming rules. This is not intentional; it loses track while generating new code.

7. **The user has to repeat and tighten the rules.**

   The user begins restating constraints. They clarify boundaries. They correct drift. They repeat instructions that were already given. The interaction becomes more about managing the AI than progressing the code.

8. **The AI complicates simple tasks.**

   Requests for small edits result in large rewrites. Short functions turn into bigger structures. Simple instructions produce multi-step changes. The problems

appear directly in the new code the AI outputs, making everything heavier than it needs to be.

9. **The codebase drifts and the user gets frustrated.**

   Structure, style, and logic begin to slide away from the original brief. The user feels like they are fighting the system. The work becomes slower and more difficult than doing it manually.

10. **The AI pulls the work into scope creep and rabbit holes.**

    A simple request for a small change turns into a bigger task. The AI suggests new patterns, extra layers, or alternative designs. It introduces side ideas that were never part of the brief. The user follows one of these paths to keep things consistent, and suddenly the work has expanded. Time is spent exploring tangents instead of finishing the original job.

11. **Bug fixing turns into an endless chase.**

    There is a bug. You or the AI spot it. The AI rewrites the part of the code that seems responsible. You run the code again and it fails in a new way. The original bug is gone, but the AI renamed two variables or changed a function signature in the rewrite. Now there are two problems instead of one. Each new fix introduces another small break somewhere else, and the cycle repeats.

12. **The user ends up managing the AI instead of coding.**

    The real task becomes undoing changes, preventing tangents, restating rules, and dragging the AI back to the brief. The user spends more time controlling what the AI is generating than producing the software.

Part 2 - How to Work With the AI

13. **The system does not behave the same way every time.**

    The amount of computing power behind each response is not constant. Sometimes the system has more capacity available, so it can process your request more deeply and check its own work more thoroughly. When that happens, the AI feels sharp and stable. Other times the system has less capacity available, so it processes your request with fewer internal steps. On those runs, the AI feels vague, repetitive, or drifty. That variation in available capacity is what people experience as good days and bad days.

14. **The AI gets worse over long sessions.**

After enough back and forth, enough code, or enough corrections, the AI becomes more verbose, more generic, and more forgetful. Long sessions degrade quality. This is a limitation of the system.

15. **The AI struggles after a certain amount of text.**

    Large files cause contradictions, renaming errors, drift, and loss of structure. The AI cannot hold a big codebase stable across many updates without support. At the time of writing, the AI breaks easily in one or two thousand lines of JSON, HTML, or similar formats. This is not a flaw; it is a boundary.

16. **Patterns emerge. Learn to spot them.**

    Users eventually recognise early warning signs: padding, looping, rewriting, drifting, or over-explaining. These signals show that the session is degrading and needs to be reset or broken into smaller tasks.

17. **Always make a copy before any rewrite or update.**

    Before asking the AI to fix a bug, refactor a function, or update a file, save a clean copy of the current version. If the AI breaks something, you can restore the last working state. Without this, you can lose hours of progress in a single rewrite.

18. **Define your variables early and keep a lookup table.**

    The AI forgets names, changes names, or invents new ones. To reduce this, define all key variables, functions, and structures at the start. Keep a small lookup table you can paste back into the chat. This anchors the naming and reduces drift.

19. **Give the system small, contained tasks.**

    Do not ask for a full refactor. Ask for one function. Then the next. Then the test. Then the fix. Small tasks reduce drift and keep the AI inside the brief.

20. **Be watchful for unrequested rewriting.**

    The AI will rewrite code you did not ask it to touch. If you do not catch it early, it will rewrite the entire file. Vigilance is required.

21. **Be watchful for context contamination.**

    If you were working on a different task earlier in the thread, the AI may still be influenced by it. This is why new threads matter. It prevents the system from blending unrelated work.

22. In a war of attrition, the AI will win.

You cannot force the system to behave through repetition. It will out-persist you. The only winning strategy is to work with its limits instead of fighting them. Structure, boundaries, and resets are more effective than pressure.

Conclusion.

If you feel the hype and promise of AI does not match the actual experience, you are not alone. Many people discover that the systems are impressive in short bursts but unreliable over longer work. The gap between expectation and reality is wide. Research suggests that between eighty-five and ninety-five percent of AI rollouts fail in some way. This includes projects that never launch, projects that collapse under real use, and projects that quietly stop working after a few months. Inquisitor Labs is actively researching ways to reduce that figure by developing more capable and more stable AI systems.

Links:

Inquisitor Labs Homepage:

https://assimilatedhuman.github.io/inquisitor-labs/index.html

-Document Ends-