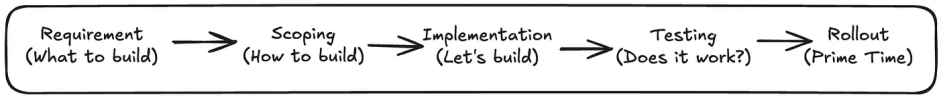
AI Assisted Coding: Quicker Code Doesn't Mean Higher Velocity


Anindya Chakraborty Follow 6 min read · Nov 9, 2025

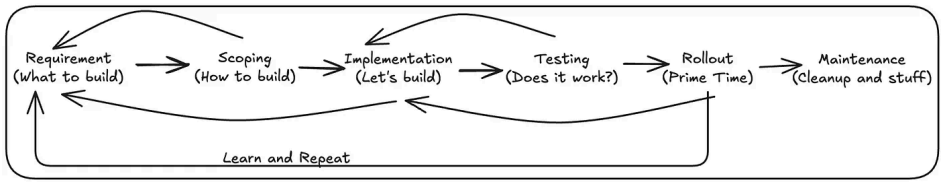
Engineers report coding 3–5x faster with AI assistants, yet many teams find their sprint velocities haven't budged. The bottleneck has simply moved — and understanding where it went is critical for getting real productivity gains.

How is Software Built?



A simplistic view of making software products

This is a very simplistic view of making a feature or a new product, reality is always much more messy. Most of these steps don't happen such neatly and linearly (for good reasons). And you keep doing this in repeat to make more and more complex pieces of products.



A more realistic view of software development life cycle

Let's take a more deeper look into implementation since that's where we see most popular AI assistants focusing.

The implementation phase (simplistic view)

You write code and tests, you test them and iterate on the code and when you are happy with the outcome, you send them for peer reviews. In this pipeline your bottlenecks are primarily writing code and tests. Peer reviews and local testing can also become bottlenecks given the criticality of the code paths, team culture and size of the change (10 lines of a change is much easier to review than 200 lines of a change) and development environment support.

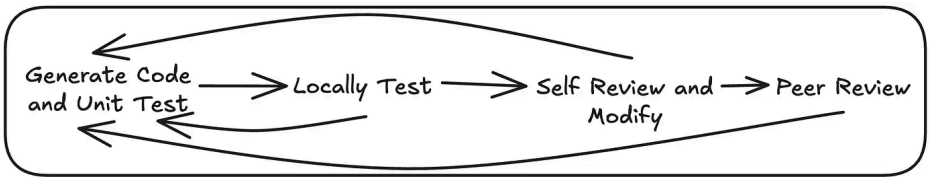
> *Peer review is a dual purpose role — they help you ensure quality, both short and long term. And they help spread the knowledge about a change — you are not alone anymore, someone else understands the change.*

What is (Gen) AI Changing?

In the past couple of years there have been a deluge of AI tools across this workflow with most success and adoption in the implementation phase. If you don't believe me, think of the 3 most known AI assistants in this workflow — Claude Code, Cursor and Github Copilot. They all are focused primarily on the implementation part of the workflow.

What makes this different from previous automation tools? Unlike code generators or scaffolding tools that handled boilerplate, modern AI assistants generate complete, context-aware implementations — entire functions, classes, or features. The scope and sophistication of what they produce fundamentally changes the developer's role from writer to reviewer-editor.

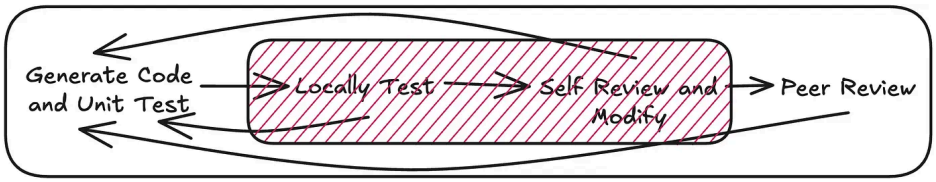
With AI assistants, most of the code is written by AI, in minutes, but the engineer "implementing" is still the one responsible for the code being written. This changes the implementation dynamics drastically.



Implementation with AI coding assistants

The New Bottlenecks

> *The bottleneck shifts from writing code and unit tests to understanding code, local testing, and self/peer reviews.*



The new bottlenecks

As AI generates more and more code and unit tests, the bottleneck shifts from "writing" code and unit tests to local testing, and self/peer reviews. Remember the role of "code review" — both holding quality bar and spreading knowledge. Given the human "implementer" is still the one responsible for the code generated, they now have to spend as much, if not more, time as the peer reviewer reading and understanding the code being generated.

Compounding this challenge, in most cases, given the quality limitations of today's AI coding agents, especially related to generating maintainable code, this potentially also means the engineer needs to change the generated code either by hand or by further prompting.

AI Coding Agents: Code Quality

Any fool can write code that a computer can understand. Good programmers write code that humans can understand. —...

medium.com

Understanding AI-generated code is fundamentally different from reviewing code you wrote yourself. When you write code, you've already reasoned through the logic, considered edge cases, and built a mental model. With AI-generated code, you're reverse-engineering someone else's (or something else's) solution — which can take considerably longer than writing it yourself would have taken.

The Knowledge Transfer Problem gets worse, not better. In traditional code review, the author deeply understands their changes and the reviewer learns from them. With AI-generated code, both author and reviewer are simultaneously learning what the code does, effectively doubling the cognitive load and time required.

Self and Peer Review

> *10 lines of code = 10 review feedback. 500 lines of code = LGTM*

A long time before CI/CD was a popular thing, 2 decades ago I worked on a software that was shipped every 6 months. Even back then, our team had a code review policy of sending back any "large" change. When humans wrote all the code, this was easy to follow since we tend to build things in smaller logical chunks most of the time instead of building one large feature in one go.

Even with AI agents, you can go "generate just this one method", then another prompt for "add usages of this" etc. If you go to that granularity, then it's just a bit more work with the great AI autocompletes to write your own code and you lose out on most of the potential of generating code faster.

The challenge is that review time doesn't scale linearly — it scales exponentially with change size. A 500-line review doesn't take 5x longer than a 100-line review; it can take 10–15x longer as reviewers must hold increasingly complex context in their heads and trace interactions across many changes.

Full disclosure: recognizing this problem led us to build an agent that helps you split the large changes AI agents generate, group them into logical chunks to make it easier to review/send it for peer reviews. Additionally it provides granular annotation of each change to make it easier to understand the changes (beyond the summaries your AI coding agent of choice generated). The agent is open source and you get to provide the models you prefer (from Claude and OpenAI to open source models like Qwen Coder).

GitHub - armchr/armchr: Armchr is a set of tools for AI coding agents.

Armchr is a set of tools for AI coding agents. Contribute to armchr/armchr development by creating an account on...

github.com

What This Means for Teams?

To actually achieve higher velocity with AI coding assistants, teams need to adapt their workflows:

1. Prompt for smaller, logical changes: Rather than asking AI to "implement the entire feature," break requests into reviewable chunks — single functions, focused refactors, or specific behaviors.

2. Invest in review tooling: Tools that help understand, annotate, and split AI-generated changes become as critical as the generation tools themselves.

3. Treat AI code like external contributions: Apply the same scrutiny you'd give to code from a junior developer or external contributor — assume nothing about correctness or maintainability.

4. Pair programming with AI: Have one developer prompt the AI while another reviews in real-time, catching issues before they compound.

The next generation of AI coding tools must prioritize reviewability and understandability, not just generation speed. Until they do, faster code generation will continue to create slower code review — leaving overall velocity unchanged.

Ai Coding | Ai Coding Agent | Ai Coding Assistant | Software Development

Written by Anindya Chakraborty

34 followers · 33 following

Follow

Lucky to be onboard multiple Titanics and Rocket Ships over 20+ years.

No responses yet

Write a response

What are your thoughts?

More from Anindya Chakraborty

Anindya Chakraborty

AI Coding Agents: Code Quality

Any fool can write code that a computer can understand. Good programmers write code...

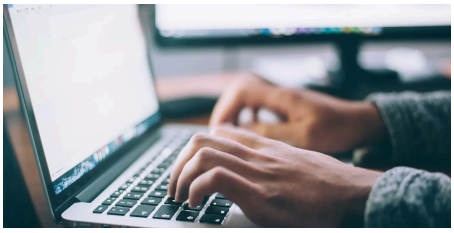
Sep 4 5

Anindya Chakraborty

A Beginner’s Guide to AI Assisted Coding—Part II

A comprehensive guide for software engineers venturing into using AI coding...

Nov 6



Anindya Chakraborty

A Beginner’s Guide to AI Assisted Coding

A comprehensive guide for software engineers venturing into using AI coding...

Sep 26 4

Anindya Chakraborty

Prompt Splitting: Improving LLM Accuracy with Tool-Based Tasks

In my experience working with large language models (LLMs) like ChatGPT or Gemini, I’ve...

Aug 26, 2024 8

See all from Anindya Chakraborty

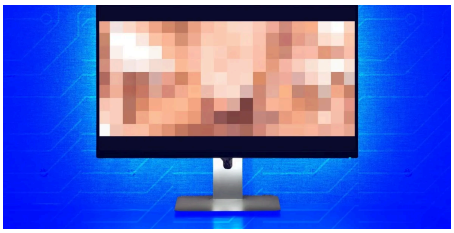
Recommended from Medium

In Write A Catalyst by Adarsh Gupta

7 Websites that I Visit Daily as a 9–5 Software Engineer

Every day, every single day.

Nov 9 708 11

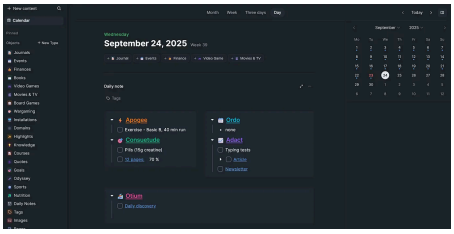


In Entrepreneurship Handbook by Joe Procopio

AI Porn Is Here—and We’re All Pretending It’ll Be Fine

Sam Altman has decided that five years is enough of a head start for the porn...

Oct 31 3.6K 142

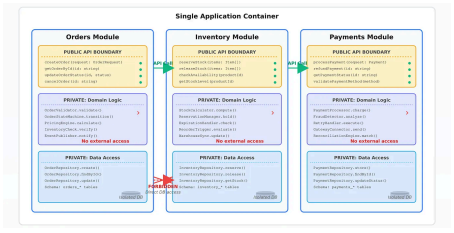


Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.

Sep 23 8.2K 283

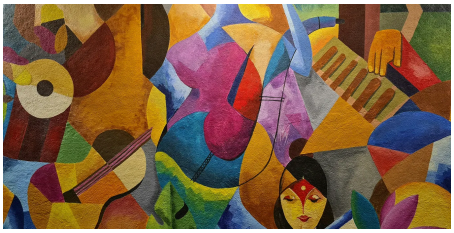


The Atomic Architect

Architecture Patterns That Actually Scale In 2025: The Only...

Last month, a company I advised shut down.

Nov 8 1.3K 40

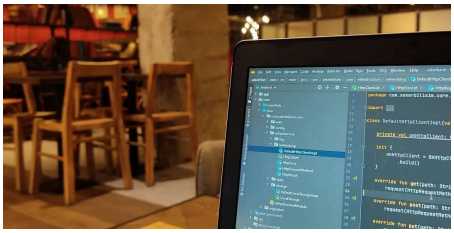


Praveen Govindaraj

Context Engineering 2.0: The Context of Context Engineering

Looks like a complicated title, yes same to me as well

6d ago



In Become Bet... by Alparslan Selcuk Develioglu (...

Vibe Coding for 10 Years Experienced Software Developer

A veteran developer’s first encounter with vibe coding, who even said “it got too easy”...

Nov 1 92 3

See more recommendations