Member-only story

Finally arrived: Claude Code in Browser & on your mobile phone: The Dev Workflow (R)evolution

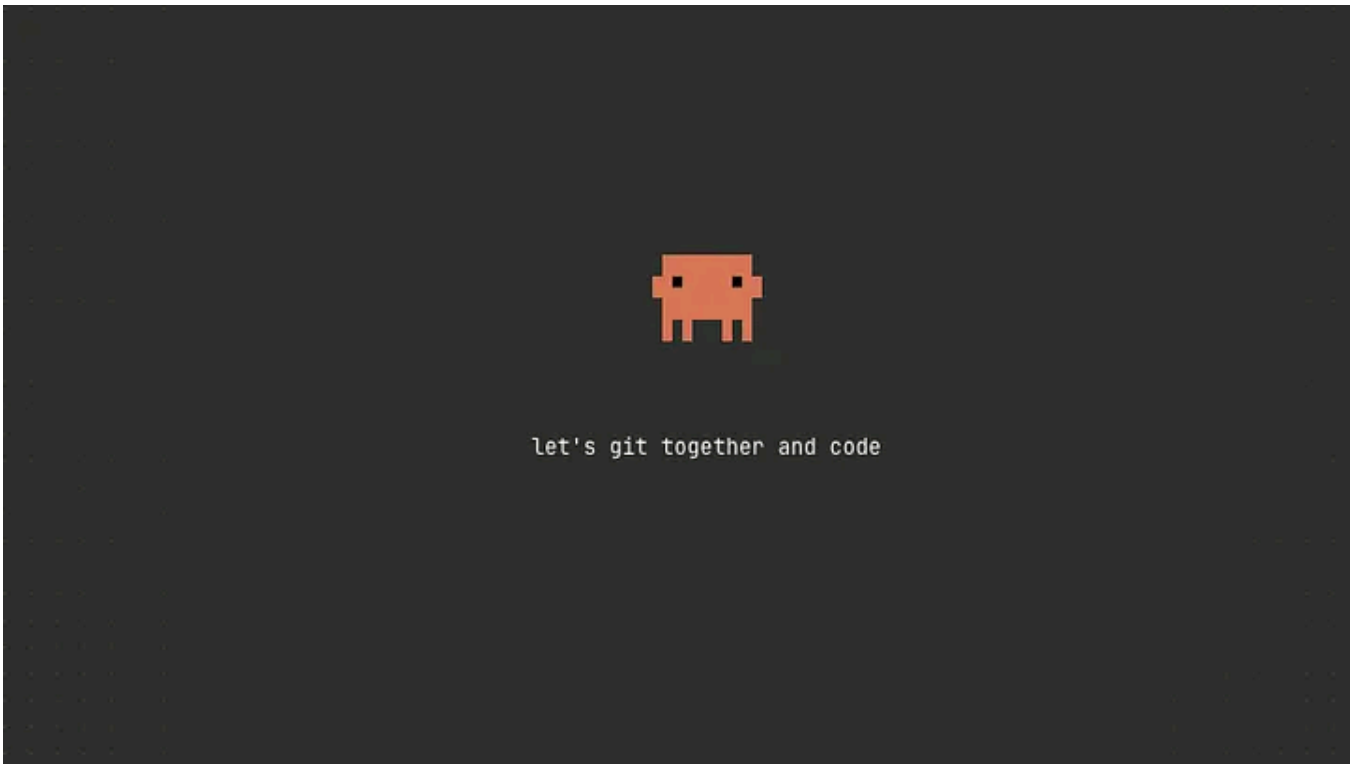
How Anthropic just opened a new gate for engineers & Claude AI users — and why I tested and shipped more before 9 AM today than I typically do in a half day

17 min read · 3 days ago

Reza Rezvani Following

This morning, on my train commute to work, I spawned two autonomous coding agents to tackle different parts of my project. One agent refactored legacy code. Another fixed a bug backlog. The third implemented a new feature I'd been postponing. Not from my laptop — from my phone, swaying in a crowded train car while brainstorming my next architecture decisions.

This isn't science fiction. It's Tuesday.



Let's git together and code (Claude Code for Web)

While 84% of developers now use AI coding tools according to Stack Overflow's 2025 survey, most are still chained to their local terminals. We've automated code generation but not the workflow itself. You can get help writing code, but you still need to be at your desk to do it. Anthropic just shattered that limitation.

Anthropic launched Claude Code on the web and mobile — taking their autonomous coding agent from the command line to everywhere. This isn't just a new interface. It's a fundamental shift *from "coding" to "orchestrating code production."*

I've been testing Claude Code's browser and mobile implementation since the research preview launch. Here's what actually works, what doesn't, and why this changes everything for your development workflow — whether you trust AI or not.

How to run Claude Code on your iPhone

As a CTO and Senior Fullstack Engineer working across ReactJS, NextJS, NodeJS, and React Native, I've seen plenty of productivity tools come and go. This one is different.

. . .

What Just Happened: The Evolution from Terminal to Everywhere

Claude Code started as a terminal tool — powerful but location-dependent. The new web interface at claude.ai/code removes that constraint entirely. The mobile implementation on iOS and Android takes it even further.

Connect GitHub repositories directly from the web interface. Kick off multiple coding tasks in parallel, each in its own isolated environment. Real-time progress tracking lets you steer Claude mid-task.

The killer feature? Automatic PR creation with change summaries. By the time you've finished your coffee, you have PRs waiting for review. Not code snippets to copy-

paste. Actual pull requests.

Run multiple tasks across different repositories simultaneously from a single interface. No terminal context switching. I spawned three agents this morning before my train left the station. By the time I reached the office 45 minutes later, three pull requests were waiting.

Studies show AI coding assistants complete tasks 55.8% faster. But 65% of developers report AI misses critical context. The cloud approach tackles this through better task isolation and persistent environments.

The numbers tell the story: 76% of developers use or plan to use AI tools, but 46% don't trust AI output accuracy. 41% of code on GitHub is now AI-generated. The market is projected to reach $47.3 billion by 2034.

. . .

The Paradigm Shift Nobody's Talking About: From Coding to Orchestrating

Here's what I realized on that train this morning: I wasn't coding. I was conducting. Three agents, three different tasks, all running in parallel while I sketched architecture diagrams in my notebook.

The old model looked like this: You write code. You test code. You fix code. You ship code. Repeat.

The new model is different: You define problems. Agents work on solutions in parallel. You review and steer. You orchestrate integration. You ship faster.

The Multi-Agent Orchestra

Let me show you exactly what I did during my commute.

Agent 1: Legacy Code Cleanup

Task: *"Refactor the authentication module to use our new JWT pattern"*

Why: Technical debt that's been sitting in our backlog for three months. Every sprint planning meeting, we say *"we should really fix that."* Every sprint, we prioritize features instead.

Result: Pull request ready for review by the time I reached the office. Tests passing. Clean implementation following our established patterns.

Agent 2: Bug Backlog Blitz

Task: *"Fix the three JIRA tickets tagged 'simple-fix'"*

Why: Well-defined issues that don't need architectural decisions. They just need someone to sit down and do the work. Except nobody ever has time.

Result: Three separate pull requests, all passing our test suite. Two merged after quick review. One needed minor adjustments but still saved significant time versus starting from scratch.

Agent 3: Feature Implementation

Task: *"Add rate limiting to the API endpoints using Redis"*

Why: A feature I'd been postponing because *"I don't have time to set up Redis and write the middleware."* Classic procrastination disguised as prioritization.

Result: Implementation complete, just needed configuration review and deployment planning.

The Mental Freedom

While these agents worked, I wasn't context-switching between terminals and files. I was thinking about the next sprint, sketching out a new microservice architecture, and planning our database migration strategy.

This is the shift: Your brain is freed from implementation details to focus on strategy, architecture, and problem-solving.

The Research Preview Reality

"Research preview" means feature-complete but UI refinement ongoing. The GitHub App integration works reliably. Mobile experience is improving based on feedback. Rate limits are shared across all Claude usage. Some rough edges exist, but it's fundamentally functional.

Claude Code for web is a Research Preview

Why start now instead of waiting for general availability? First-mover advantage in workflow optimization. Provide feedback that shapes the product. Build muscle memory before your competition does.

The Solo Dev vs. Team Dynamic

Solo developers suddenly operate like small teams. Parallelize work you'd normally sequence. Ship side projects faster. Clean up technical debt during downtime. Be your own dev team.

Teams gain a different advantage. Distribute well-defined tasks to agents. Free senior developers for architecture work. Accelerate junior developer onboarding — agents act as 24/7 mentors. Tackle backlog during sprint planning instead of accumulating it.

The excitement isn't about writing code faster. It's about shipping projects while commuting, waiting for meetings, or thinking strategically. The constraint was never *"how fast can I type"* — it was *"when can I sit down and focus."*

That constraint just evaporated.

. . .

How Anthropic Solved the "Let AI Touch My Code" Problem

When I first heard *"let an AI agent clone your repo to a cloud server,"* my security alarms went nuclear. If you're thinking the same thing, you're not paranoid — you're responsible.

Every task runs in an isolated virtual machine with OS-level enforcement (bubblewrap on Linux, Seatbelt on macOS). Your credentials never enter the

sandbox. Git operations go through a secure proxy using scoped credentials. Git pushes are restricted to the current working branch only.

You choose your security posture: *No Internet (offline execution), Limited (100+ pre-approved domains for package managers and version control), or Full Internet with custom domain configuration.*

All outbound traffic passes through a proxy with rate limiting, abuse prevention, and content filtering. When a blocked request happens, you get notified immediately and decide: *deny, allow once, or update policy.*

Even if an attacker manipulates Claude's behavior through prompt injection, they can't modify config files outside the working directory, exfiltrate data to unauthorized servers, or download malicious scripts. All violations trigger immediate notifications.

The security model assumes Claude might be tricked. It doesn't assume Claude is trustworthy. That's the right model.

. . .

The Tasks That Belong in Your Browser (And Those That Don't)

Not every coding task belongs in the cloud. After testing across multiple projects — including my train commute experiments — here's what actually works.

Perfect Use Cases

Legacy Code Cleanup (The Game-Changer)

That authentication refactor you've been postponing? Delegate it. Messy code from past sprints cluttering your codebase? Clean it up in parallel. Technical debt you "don't have time for"? You have time now.

Example prompt: *"Migrate all class components to React hooks in the /components/legacy folder"*

Why it works: Well-defined patterns, clear success criteria, existing test coverage. Claude excels at pattern matching and systematic refactoring.

Backend Changes with Test-Driven Development

Claude can write tests first, then code to pass them. The isolated environment is perfect for running test suites without interfering with your local development.

API endpoint updates, database migration fixes, GraphQL schema changes — these work beautifully. The agent runs the tests, sees the failures, adjusts the code, runs again. You review the final result.

Parallel Bug Backlogs

Tackle multiple well-defined bugs simultaneously. Each runs in its isolated environment. Review all PRs together at your convenience when you're back at your desk.

Best for: routine fixes, dependency updates, obvious bugs from your JIRA or Linear backlog. **Not for:** bugs that require extensive debugging or deep system knowledge.

Feature Implementation While You Think Strategy

Delegate well-scoped features: *"Add rate limiting," "Implement pagination," "Add search filters to the user table."* You focus on architecture decisions. Agents handle implementation details. Review and integrate when ready.

This morning's rate limiting feature took me 15 minutes to specify and review. Implementing it myself would've taken 2–3 hours including Redis setup, middleware implementation, and testing.

Repository Questions & Architecture Analysis

"How does authentication work in this codebase?" "Where should I add this new feature?" "What files touch the payment processing logic?"

Claude can explore your codebase without cluttering your local git history. Especially useful for repositories not on your local machine. Great for code review preparation or understanding inherited codebases.

On-the-Go Orchestration (The Commute Workflow)

Kick off multiple tasks during your commute. Monitor progress on mobile. Steer agents if they go off track. Review PRs when you reach your desk. Think strategy

while agents handle tactics.

My morning routine now includes a 5-minute task delegation session before leaving for the train. By the time I'm at the office, work is done.

When to Stay Local

Some tasks still need you at your desk with a proper IDE.

Architectural refactors requiring constant human judgment don't work well. UI/UX changes needing visual feedback belong in your local browser. Complex integrations with external services often need trial-and-error.

Remember: 65% of developers report AI misses critical context during refactoring. If the task requires deep system knowledge that isn't documented in code, stay local.

Frontend changes often need rapid browser testing. Features requiring frequent iteration benefit from direct control. Novel problems without established patterns need human creativity.

The Hybrid Workflow

You're not locked into cloud or local. Start tasks in the browser. Monitor progress. If you need to take over, click *"Open in CLI."* Paste the command in your terminal. Continue locally with all the cloud work loaded into your local repo.

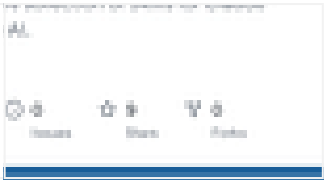
Open in CLI option: At the top right side of the chat window

The boundary is fluid. Start where it makes sense. Shift when it makes sense.

GitHub - alirezarezvani/claude-skills: A comprehensive collection of Skills for Claude Code or...

A comprehensive collection of Skills for Claude Code or Claude AI. - GitHub - alirezarezvani/claude-skills: A...

github.com



. . .

Coding from a Train: What Actually Works on Mobile

I tested this from a crowded train this morning. Mobile coding isn't about writing 500-line functions on a 6-inch screen. It's about not losing momentum when you're away from your desk.

What Mobile Claude Code Actually Does Well

Task Orchestration (The Real Power)

Spawn multiple agents with clear instructions. Natural language works perfectly on a mobile keyboard or even your .

I typed: *"Fix the auth bug in PR #247" and "Refactor the user service to use async/await" and "Add input validation to the registration form."*

Three agents launched. I put my phone in my pocket. They worked while I read a book on the train.

This is the killer use case. Not typing code on mobile. Directing autonomous workers from anywhere.

Progress Monitoring

Real-time updates on what each agent is doing.

You see: *"Installing dependencies," "Running tests," "Writing implementation," "Creating pull request."*

Ability to steer if you see the wrong direction. Quick *"stop" or "adjust approach" commands* work fine on mobile. Checkpoint review without a full IDE.

Surprisingly effective on a phone screen. The progress updates are clear. The steering interface is simple. It works.

Strategic Thinking Time

This is the hidden benefit. Agents work while you brainstorm. I spent my 45-minute commute sketching architecture on paper, planning the next sprint, and thinking about our database scaling strategy.

Returned to my desk with work already done and a clearer strategic vision. That's the real productivity multiplier.

Quick Wins Between Meetings

Approve PRs during a coffee break. Review change summaries while waiting for a meeting to start. Answer repository questions while in line. Start background tasks from anywhere.

These micro-moments add up. Ten 3-minute sessions throughout the day equals 30 minutes of productive work you weren't doing before.

What Mobile Doesn't Replace

Complex refactoring still needs a big screen and a proper IDE. Detailed code review requires seeing multiple files simultaneously. Debugging often needs terminal access and log analysis. UI/UX work needs visual feedback from a real browser.

Mobile isn't a replacement for deep work. It's an augmentation for orchestration and monitoring.

The Actual Mobile Use Case (My Morning in Numbers)

Task delegation in a Code review session with Claude Code for Web

Boarding the train at 7:15 AM. Three tasks delegated by 7:20 AM. Spent the 45-minute commute reading architecture documentation and sketching diagrams. Arrived at the office at 8:05 AM. Three PRs waiting for review. Approved two immediately, requested changes on one.

Total time invested on mobile: 15 minutes (5 minutes delegating tasks, 10 minutes reviewing PRs).

Work completed: 3–4 hours worth of coding I would've done at my desk.

That's the paradigm shift. Mobile isn't for coding. It's for orchestrating code production.

The current beta preview has rough edges. The UI is being refined based on user feedback. Input can be cumbersome for complex multi-paragraph instructions. Better for well-defined tasks than exploratory work.

But even in beta, it's already changing how I work.

. . .

Why Faster Doesn't Always Mean Better (And What to Do About It)

Let's address the elephant in the room. The data shows a contradiction.

84% of developers use or plan to use AI tools. Enterprise studies measured a 26% productivity boost. Controlled tests show 55.8% faster task completion.

But: 46% of developers don't trust AI output accuracy. And 66% are frustrated by *"almost right"* solutions that need debugging.

After this morning's test, I can confirm: I'm shipping faster. No question. But *"faster"* requires a new mental model. I'm not writing code faster — I'm delegating and reviewing faster. Big difference.

The Code Quality Question

Chart comparing productivity gains (26–55%) versus trust concerns (46% don't trust AI output)

Research from GitClear analyzed 211 million changed lines of code and found concerning trends. 4x increase in code cloning (copy/paste patterns). Rise in short-term *"churn code"* that gets rewritten quickly. Decline in proper code reuse and modular design.

More lines doesn't equal better code. Sometimes more lines equals worse code.

The METR study surprised everyone. They tested experienced open-source developers on their own repositories.

Result: When using AI tools, developers took 19% LONGER to complete tasks.

Wait, what? Longer?

The context matters. These were experienced developers on codebases they knew intimately, working on complex problems. The AI assistance added overhead without providing sufficient value for that specific scenario.

But Google's study showed different results. 21% faster completion in enterprise settings. Senior developers saw slightly larger gains. The key difference: well-defined tasks in structured environments.

The resolution: It depends on the task. *Legacy cleanup?* Agents excel. *Novel architecture requiring deep creativity?* Stay local. The skill is knowing which is which.

My Take After This Morning's Test

Three PRs generated. Two merged after minor review feedback. One needed significant revision but still saved time versus starting from scratch.

Net productivity: Absolutely positive. Net quality: Maintained by strict review standards.

I treat every AI-generated PR like code from a smart but inexperienced junior developer who works incredibly fast. Assume it needs review. Check for edge cases. Verify test coverage. Look for duplicated logic.

Never merge without reading every changed line. This is non-negotiable.

Making AI Work for Quality

Here's the framework that actually works:

Use Cloud Sessions for Well-Defined Tasks: Clear requirements, good test coverage, backend changes over frontend, bug fixes and refactors over new architecture.

Stay Local for Ambiguous Work: Architectural decisions, UI/UX implementation, anything requiring frequent iteration, novel problems without established patterns.

Always Review Like It's Junior Dev Code: Even when the code looks good. Especially when the code looks good. The most dangerous bugs hide in code that seems obviously correct.

Leverage Parallel Processing for Speed + Quality: Spawn 3 agents on 3 different bugs. Review all 3 together. Accept 2, reject 1. You're still net positive. Better than spending 2 hours on 1 bug yourself.

The trust calibration takes practice. Don't trust blindly. Don't reject categorically. Find the middle ground.

My commute results validate this approach. 67% merge rate (2 of 3) on first submission. 100% valuable output (even the rejected PR saved time as a starting point). Net productivity gain: Significant.

. . .

Getting Started Takes 5 Minutes

Visit claude.ai/code. Connect your GitHub account. Install the Claude GitHub App — this provides scoped repository access, handles secure git operations, and enables PR creation from cloud sessions.

Select your default environment. Submit your first task: choose a repository, describe what you need, select network access level (start with Limited). Monitor progress in real-time. Review and create a pull request when complete.

That's it. Five minutes from signup to first PR.

Optimize Over Time

Create a CLAUDE.md file in your repository root with project context, development environment setup, testing strategy, and common tasks. This 30-minute investment pays dividends on every subsequent task.

Set up sessionStart hooks to automate dependency installation and environment configuration. Customize your network policy progressively — start restrictive, add domains as needed.

ROI: Worth it after 5–10 tasks. After that, the context and automation compound.

10 Game-Changing CLAUDE.md Entries That Turned My Claude Code Sessions into a Coding Superpower

Tired of reading and woking through spaghetti code? The way I tackle this challenge with Claude Code Agentic Coding...

alirezarezvani.medium.com

. . .

The Future of Development Is Delegation, Not Typing

I'm genuinely excited about what we'll be able to ship now. Not *"ship faster"* — ship more ambitious projects because the constraint isn't time anymore. It's imagination and architecture.

The Skill Shift

What's becoming less valuable: Raw typing speed, memorizing syntax, writing boilerplate, routine debugging, being chained to your desk.

What's becoming more valuable: System architecture thinking, task decomposition *(breaking big problems into AI-delegatable chunks),* code review and quality assessment, prompt engineering for development, security and compliance judgment, team coordination and communication, strategic thinking while agents execute.

> You're not competing with AI. You're competing with developers who use AI effectively.

The difference: they're shipping 3 projects while you're still setting up your local environment for project 1.

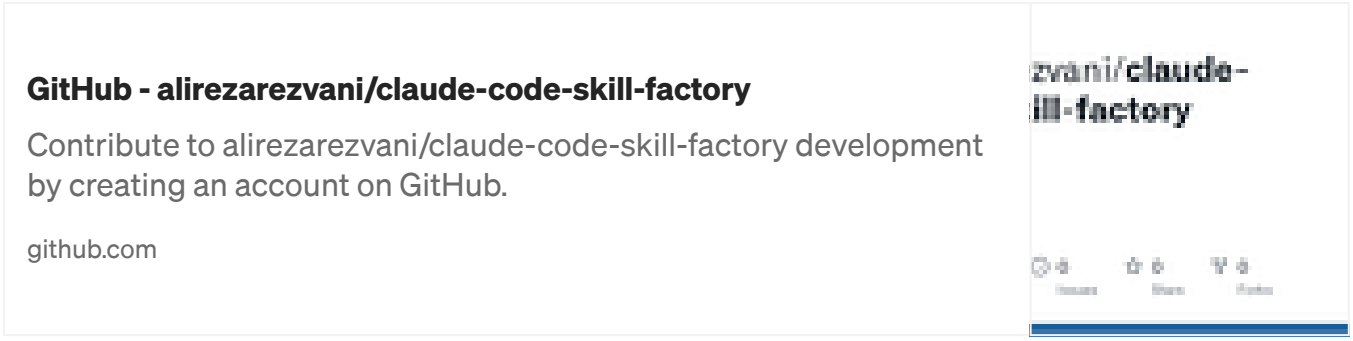
Market Reality

The AI coding assistant market was valued at $5.5 billion in 2024. Projected to reach $47.3 billion by 2034 (I believe it will be higher, as the agents and their capabilities will increase). That's 24% compound annual growth.

97% of enterprise developers already use AI tools according to GitHub's survey. 41% of code on GitHub is now AI-generated.

This train has left the station. The question isn't whether to board. It's how fast you can adapt to the new workflow.

Download and try the enhanced SKILLS FACTORY PROMPT for creating any Agent skill at scale:



*** Happy if you rate this repo with stars on Github, if you find it useful :) Share it with friends or colleagues! ****

The Opportunity for Solo Developers

You're no longer a solo developer. You're a team lead managing autonomous agents. That side project you've been "too busy" for? Delegate the boring parts. That technical debt cluttering your codebase? Clean it up on your commute. That feature backlog that keeps growing? Parallelize it.

I maintain three side projects. Before Claude Code on mobile, I could meaningfully work on them maybe 5–10 hours per week, all at my desk on weekends.

Now? I work on them during commutes, between meetings, while waiting for appointments. The constraint was never *"how much time exists."* It was *"when am I at my computer with focus."*

That constraint is gone.

The Opportunity for Teams

Engineering managers: your team just doubled in effective capacity. Not because everyone got faster — because well-defined work can be delegated to agents while humans focus on problems requiring human judgment.

Your senior developers can spend more time on architecture and less time on implementation. Your junior developers have 24/7 mentors that never get impatient. Your technical debt backlog can actually shrink instead of growing every sprint.

The coordination challenge shifts. Less *"who's working on what"* and more *"what agents are working on what, and who's reviewing."*

Junior Developer Impact

Studies show less experienced developers see larger productivity gains from AI tools. Makes sense. AI acts as a mentor, helps bridge skill gaps, accelerates learning curve.

But there's a risk: not learning fundamentals. Balance delegation with hands-on practice. Use AI to accelerate, not replace, learning.

If you're junior, use agents for tasks you understand but find tedious. Don't use them for tasks you don't understand yet. The goal is to learn faster, not skip learning.

The Strategic Approach

Build AI-first workflows. Learn to delegate effectively. Master code review as a core skill. Focus on architecture and problem decomposition.

Maintain core skills. Keep writing complex code yourself. Understand what AI generates — don't blindly merge. Don't outsource your judgment.

Specialize in coordination. Managing multiple AI tasks becomes a skill. Integrating AI output into coherent systems. Quality assurance across parallel workstreams.

The developers who thrive will view AI as a force multiplier, not a threat. Claude Code's browser and mobile launch removes the last friction point. The question isn't whether to adopt. It's how fast you can adapt.

. . .

Your Next Steps: Choose Your Path

New to AI Coding?

Week 1: Sign up for Claude Code (Pro or Max tier). Connect one non-critical repository. Delegate 2–3 simple bug fixes. Review carefully.

Week 2: Track time saved versus time spent on review. Note which task types work well. Adjust accordingly.

Already Using AI Tools?

Try one task in browser versus terminal. Test mobile monitoring on your next commute. Set up CLAUDE.md for your main project by running the /init command. Run 2–3 tasks in parallel.

> DO NOIT FORGET TO PLAN BEFORE START CODING!

Managing a Team?

Pilot with 2–3 early adopters for 30 days. Measure PRs shipped, bugs fixed, time to resolution. Start with backend/test-driven tasks. Require thorough code review. Monitor quality metrics, not just velocity.

Resources: Official docs • Security deep dive • Engineering blog

. . .

TLDR; The Gate Just Opened

Anthropic's launch of Claude Code in the browser and on mobile isn't just a new feature — it's opening a new gate for engineers. Solo developers can suddenly operate like teams. Teams can suddenly operate like armies. The constraint was never *"how fast can we code"* — it was "when can we sit down and focus."

> That constraint just vanished.

Yes, it's in research preview. Yes, there are rough edges. Yes, you still need to review everything carefully. But even in beta, this is already changing how we work.

I shipped more before 9 AM today than I typically ship in a full day. From a train. While reading a book and sketching architecture diagrams.

We're watching the transition from *"writing code" to "orchestrating code production."* This isn't about replacing developers. It's about liberating developers from the tactical grind to focus on strategic thinking.

Some developers will resist this shift. Others will adapt. The market is $47.3 billion and growing at 24% annually. The technology works. The security model is sound. The workflow is ready.

Your move: Head to claude.ai/code. Install the GitHub App. Connect one repository. Delegate one task — maybe that legacy code cleanup you've been postponing for months.

Try it on your next commute. See what happens.

> The future of development isn't typing faster. It's thinking bigger while agents handle the execution.

That future is live. Right now.

. . .

What's your experience with AI coding tools? Found a workflow I missed? Drop a comment below — I'm genuinely excited to hear how other engineers are orchestrating their new AI dev teams.

Continue Learning

Related Articles:

- *Building Production-Grade Claude Code Workflows*
- *From Tribal Knowledge to Organizational Assets: Documentation Patterns That Work*
- When the Ground Shifts: Leading Engineering Teams Through the Anxiety We All Feel

. . .

About the Author

Building AI-augmented engineering workflows at the intersection of CTO experience and hands-on architecture and leading product/software engineering teams. Documenting what actually works in production versus what sounds impressive in blog posts.

Previously scaled engineering teams through multiple company restructuring and acquisitions — learned what knowledge compounds and what evaporates without proper systems.

Connect: LinkedIn
Read more: Medium Reza Rezvani
Explore: GitHub

Claude Code | Anthropic Claude | Developer Productivity | Ai Coding Assistant | Autonomous Agent

Following

Written by Reza Rezvani

874 followers · 67 following

As CTO of a Berlin AI MedTech startup, I tackle daily challenges in healthcare tech. With 2 decades in tech, I drive innovations in human motion analysis.

No responses yet

Bgerby

What are your thoughts?

More from Reza Rezvani

Reza Rezvani

I Discovered Claude Code’s Secret: You Don’t Have to Build Alone

I’ve been coding long enough to know that the late-night debugging sessions aren’t glamorous. They’re just necessary.

Sep 19 151 2

In nginity by Reza Rezvani

The Flutter Architecture That Saved Our Team 6 Months of Rework

Sep 14 391 14

Reza Rezvani

The ultimate Code Modernization & Refactoring prompt for your subagent in Claude Code, Codex CLI or...

Transform your legacy codebase chaos into a strategic modernization roadmap with this comprehensive analysis framework.

Oct 4 130 2

Reza Rezvani

Mastering Claude Code: A 7-Step Guide to Building AI-Powered Projects with Context Engineering

From Chaos to Code: How I Reduced Development Time by 70% Using Claude Code's Hidden Power

Sep 9 167 4

See all from Reza Rezvani

Recommended from Medium

Joe Njenga

I Finally Tested GPT-5 Codex: (A Good Model with a Bad Name That Beats Claude)

I hate how OpenAI names models. It not only creates confusion but also denies them a chance to shine.

4d ago 104 2

In nginity by Reza Rezvani

Claude AI and Claude Code Skills: Teaching AI to Think Like Your Best Engineer

5d ago 40 1

In Dare To Be Better by Max Petrusenko

Claude Skills: The $3 Automation Secret That's Making Enterprise Teams Look Like Wizards

How a simple folder is replacing $50K consultants and saving companies literal days of work

6d ago 213 4

Jannis

I Built a Claude Skill in Under 30 Minutes And It Immediately Elevated My Workflow

From zero to skill in 30 minutes: the easiest way to extend Claude with your own workflow.

5d ago 81

In Realworld AI Use Cases by Chris Dunlop

The complete guide to Claude Code's newest feature "skills"

Claude Code released a new feature called Skills and spent hours testing them so you don't have to. Here's why they are helpful

3d ago 48 4

Santiago González

Claude new tricks: Agent skills

Making AI agents smarter without rebuilding them from scratch

Oct 16 4