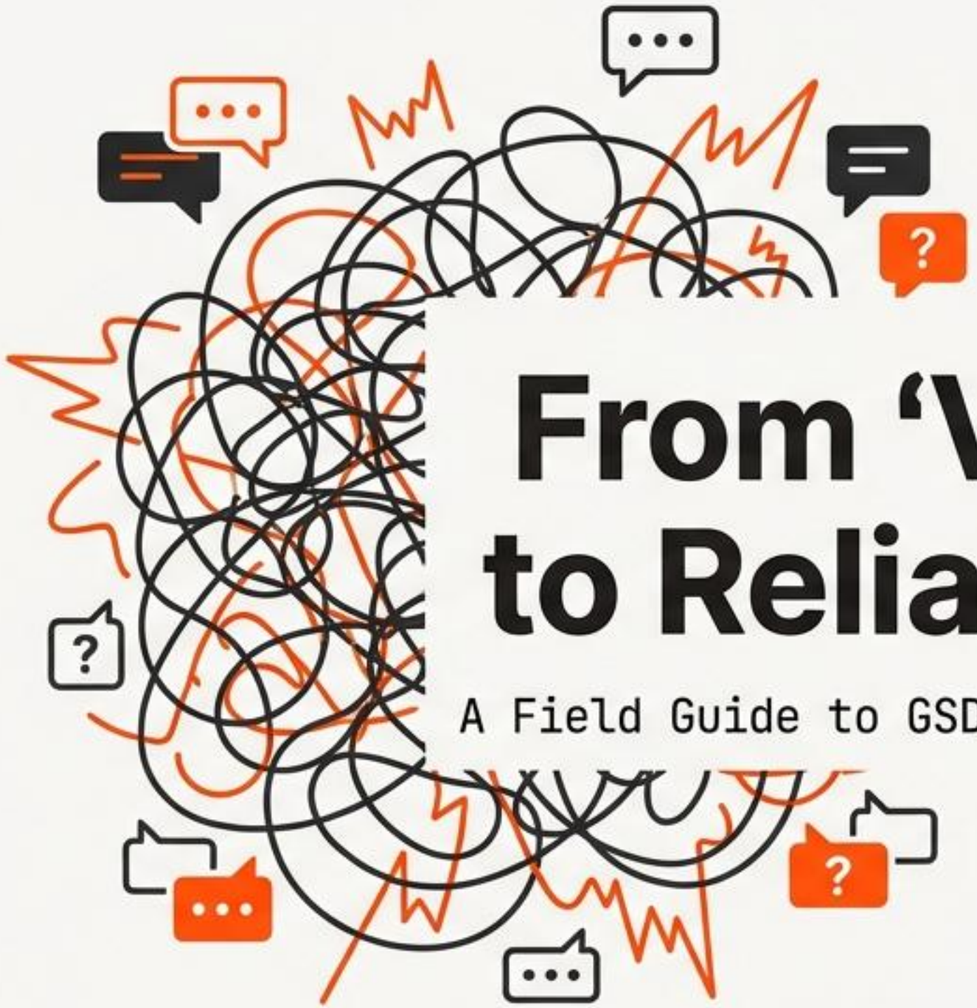
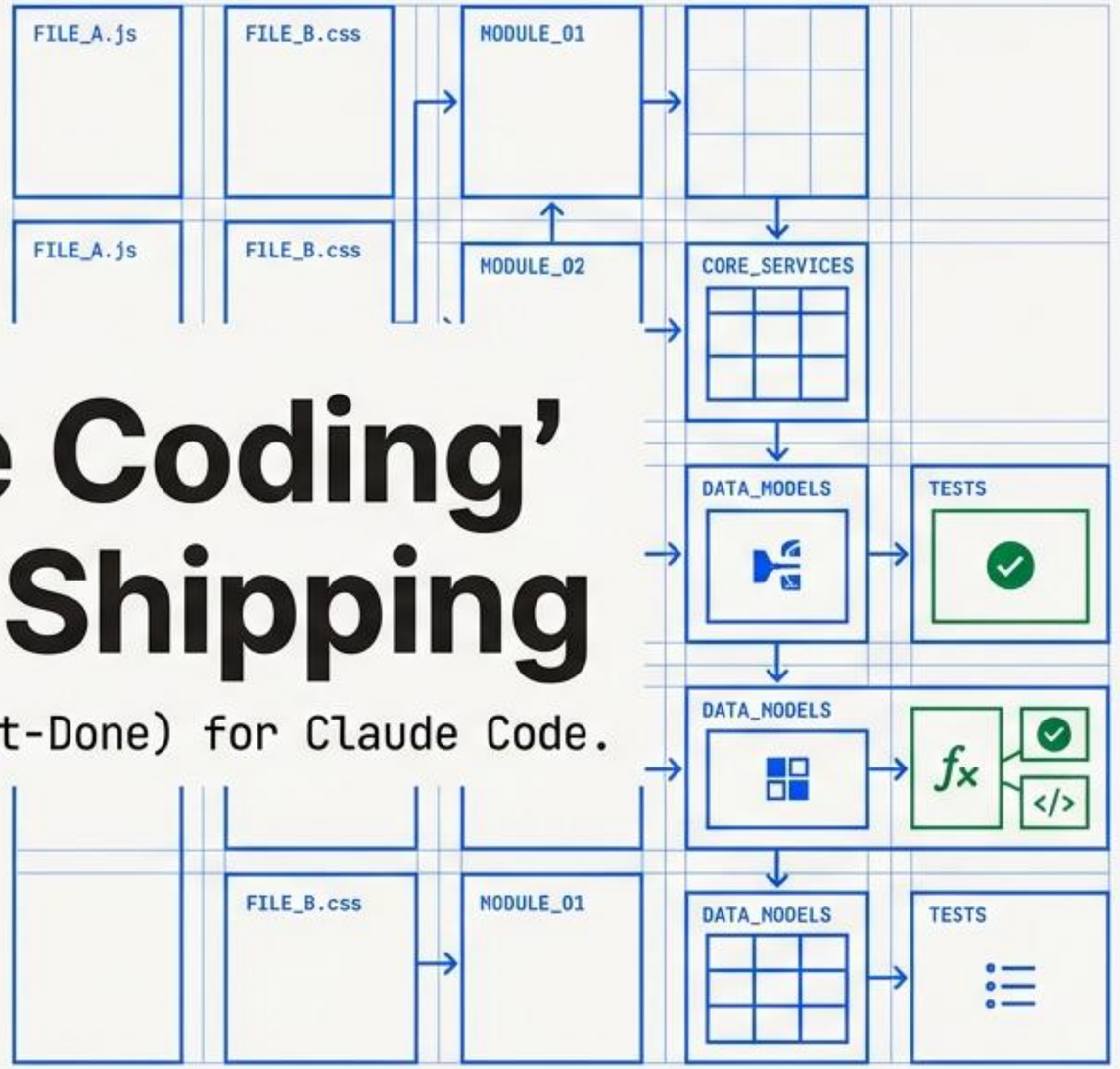
BEFORE



AFTER



From ‘Vibe Coding’ to Reliable Shipping

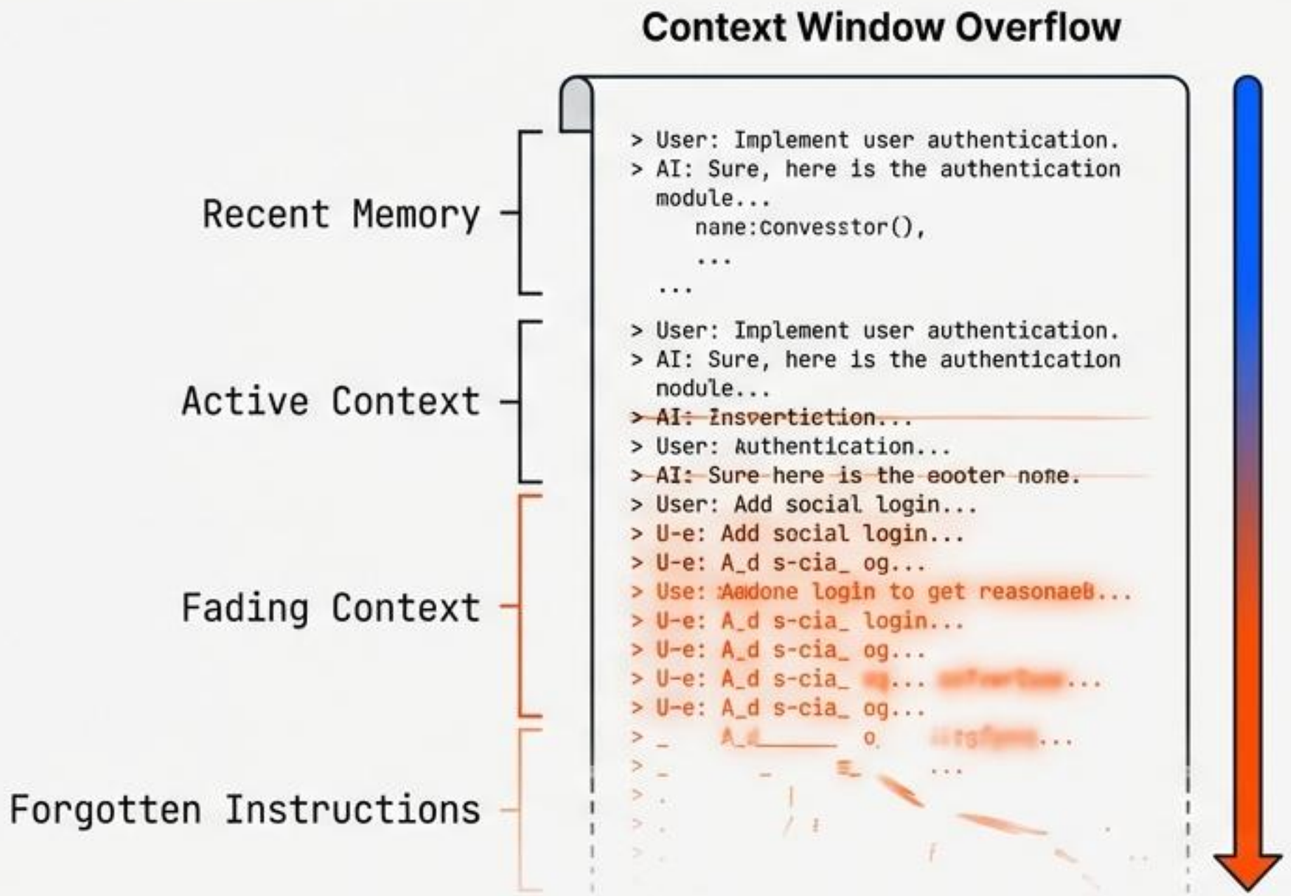
A Field Guide to GSD (Get-Shit-Done) for Claude Code.

Context-Engineering & Spec-Driven Development Workflow.

@KRUNKOSAURUS

The Problem: Context Rot & Hallucination

Why the "chat until it's done" approach breaks down at scale.



Context Window Overflow

The AI forgets early instructions as the conversation lengthens, leading to **degraded quality**.

Ambiguity

Without explicit specs, the AI chooses **"reasonable defaults" you didn't actually want.**

Lack of Roadmap

Generating code without **acceptance criteria leads to endless loops, not finished products.**

Insight: "Reliably building" is fundamentally different from "merely generating".

The Solution: GSD Defined

GSD is a context-engineering and spec-driven development workflow implemented as a Claude Code plugin.

Old Way	New Way (GSD)
Ephemeral Chat Memory. Fragile, temporary state that **vanishes** when the window closes.	**Persistent File Memory.** **Concrete, permanent state** stored in .md files.

The Engine: 4 Pillars of Stability

1. File-Based Memory

`PROJECT.md` and `STATE.md` persist the vision, beating chat history limitations.

2. Atomic XML Plans

Plans include name, files, action, and verification steps. Reduces ambiguity to zero.

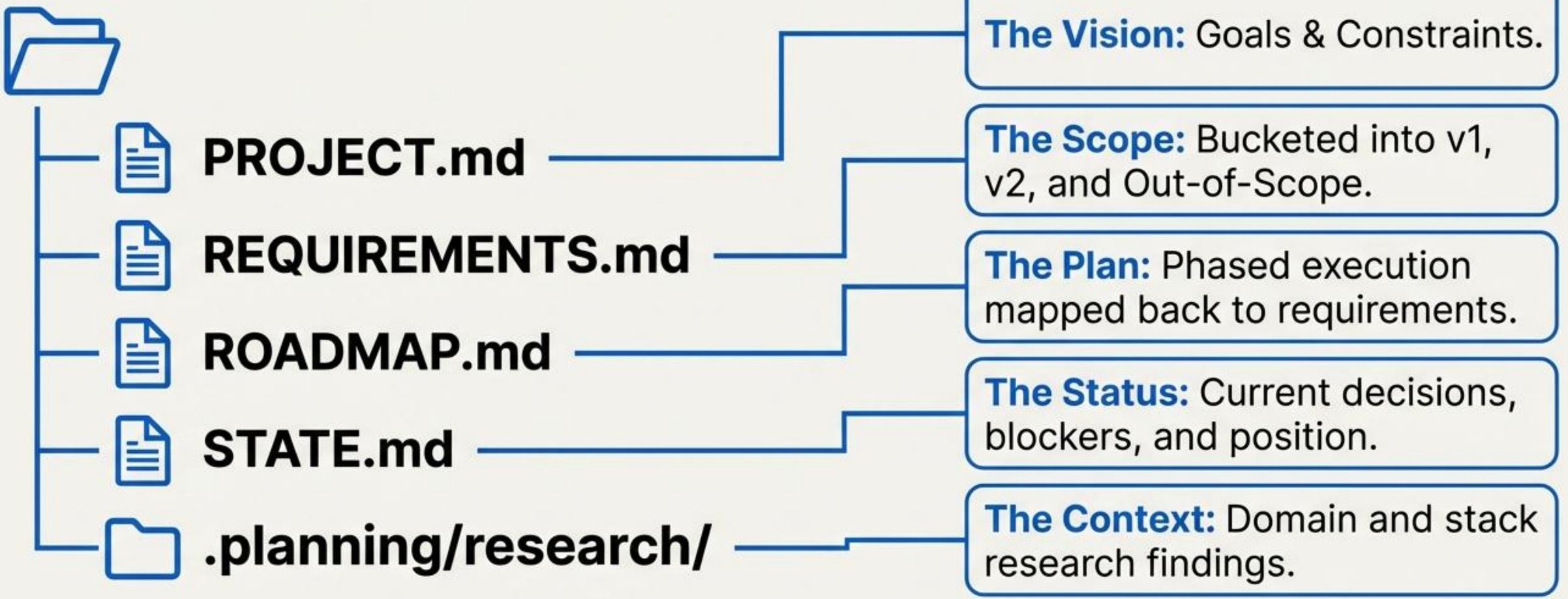
3. Sub-Agents & Parallelism

A thin coordinator spawns specialized agents (research, planning, execution) to keep the main context clean.

4. Atomic Commits

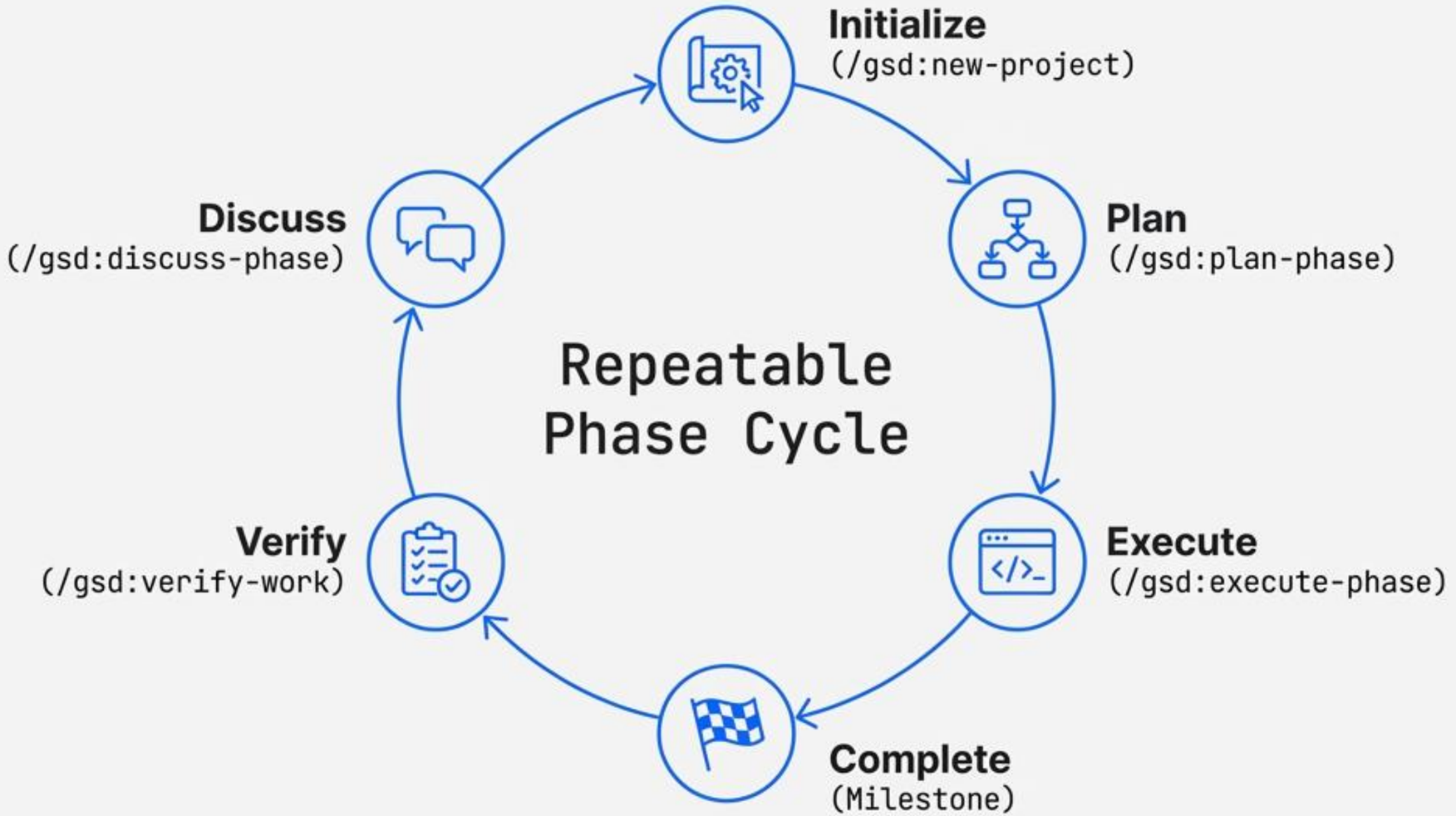
Each plan is its own auditable unit, making git bisect trivial and changes easily revertible.

The Brain: Artifacts You Must Love



Key Insight: **Each task runs with only the context it needs from these files.**

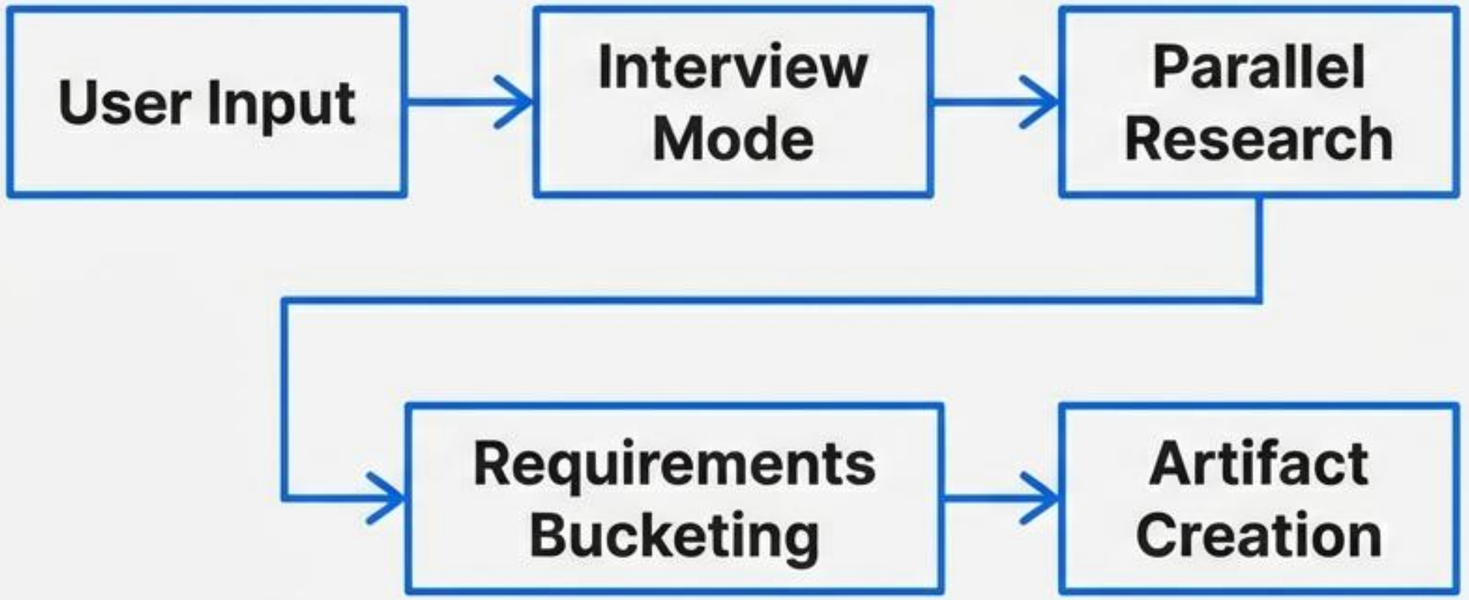
The Core Workflow Loop



A repeatable machine that turns ideas into shipped code.

Step 1: Initialize

`/gsd:new-project`



Output

Pro-Tip:
Be ruthless about scope and constraints. The better your “v1 vs. v2” boundary, the more the system shines.

Step 2: Discuss Phase

```
/gsd:discuss-phase [N]
```

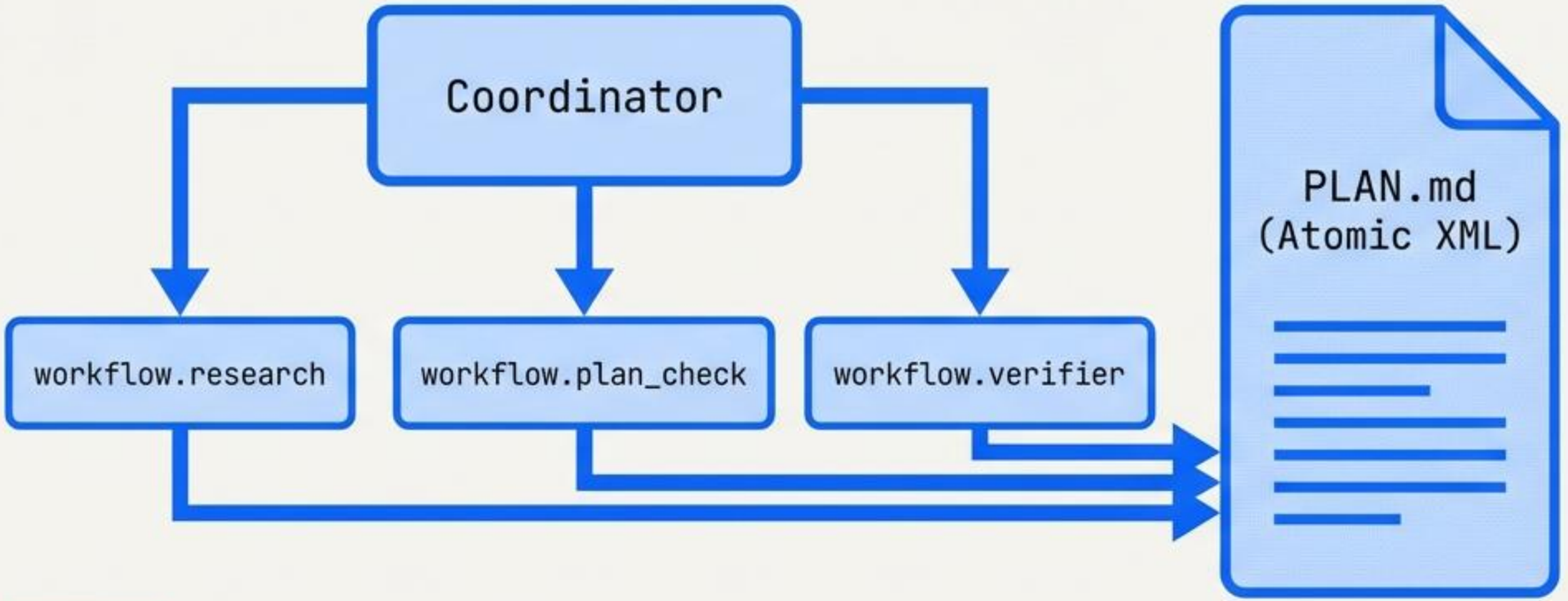
The Goal: Convert one-sentence ideas into real implementation preferences.

Output: The {phase}-CONTEXT.md file—the "don't guess" anchor.

Pro-Tip: How to Rock It: Explicitly answer the gray areas.
✅ UX behavior (empty states, errors)
✅ API contracts (response shape, pagination)
✅ "What matters most" vs. "Nice-to-have"

Step 3: Plan Phase

`/gsd:plan-phase [N]`



The Action: Produces 2–3 atomic plans (small enough to execute in clean contexts).

This is writing the SPEC, not the CODE.

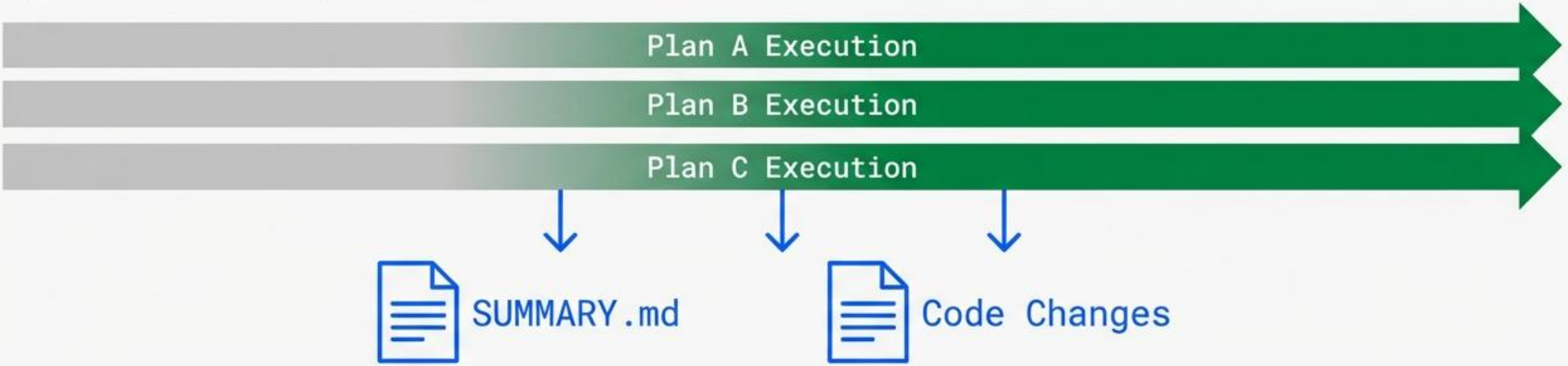
Pro-Tip

How to Rock It

Treat plans like mini Pull Request descriptions. If something is underspecified, fix it here, not during execution.

Step 4: Execute Phase

`/gsd:execute-phase [N]`



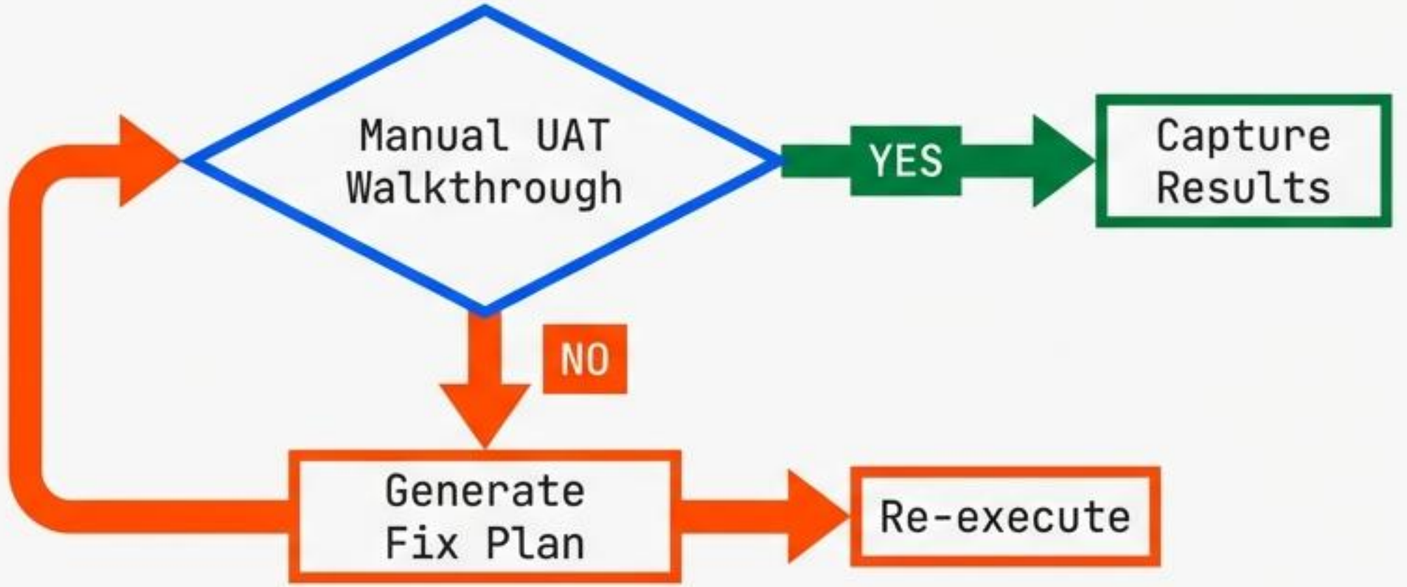
Pro-Tip

How to Rock It: Keep the repo ready to change.

- Tests must be runnable
- Linting wired up
- Dev server starts
- Insight: GSD can only verify what your project can actually run.

Step 5: Verify & Audit

`/gsd:verify-work [N]`



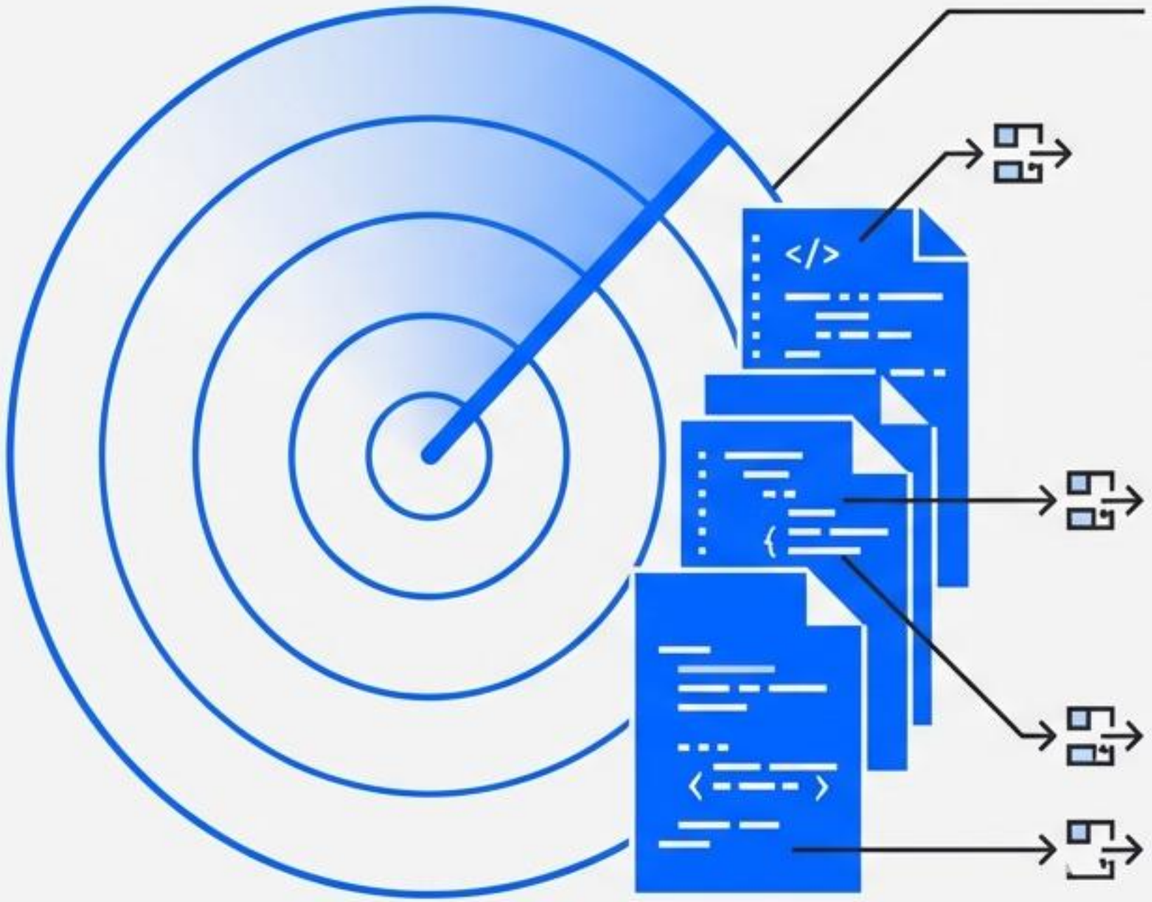
The Action: You prove the phase met the promise.

Pro-Tip: How to Rock It

Test like a Product Manager. Check the 'happy path' AND the 'sharp edges' (errors, boundaries, weird inputs).

Working with Brownfield Projects

/gsd:map-codebase

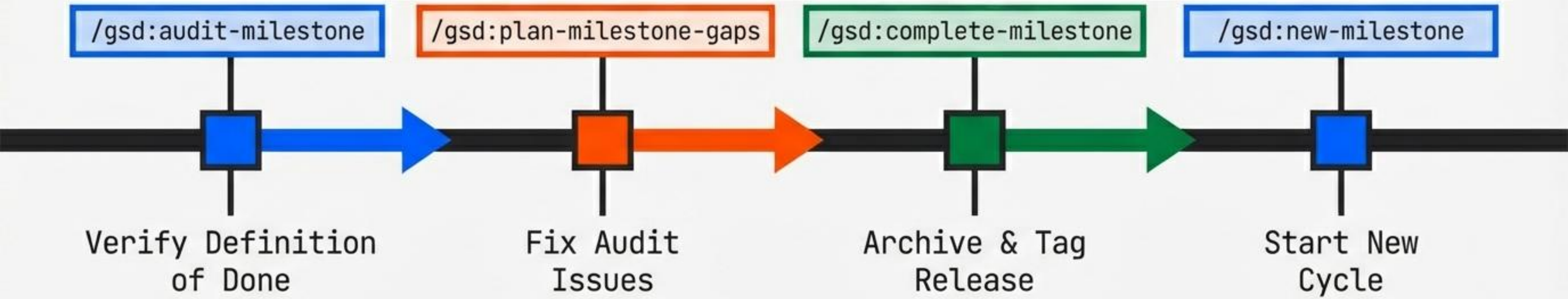


Mechanism:
Spawns
parallel agents
to learn stack
conventions and
constraints BEFORE
trying to write
new code.

Benefit:

Ensures the
workflow adapts to
the existing repo
rather than
ignoring it.

Milestone Management



Iterative shipping with a clean history.

Command Reference

Core Workflow

`/gsd:new-project`

`/gsd:discuss-phase`

`/gsd:plan-phase`

`/gsd:execute-phase`

`/gsd:verify-work`

Utilities & Navigation

`/gsd:progress` (Status check)

`/gsd:debug` (Structured debugging)

`/gsd:add-todo` / `:check-todos`

`/gsd:settings` (Configure agents)

`/gsd:quick` (Ad-hoc tasks)