From Ephemeral Code to Permanent Intelligence

A Developer's Guide to the Spec-Kit Plus Framework

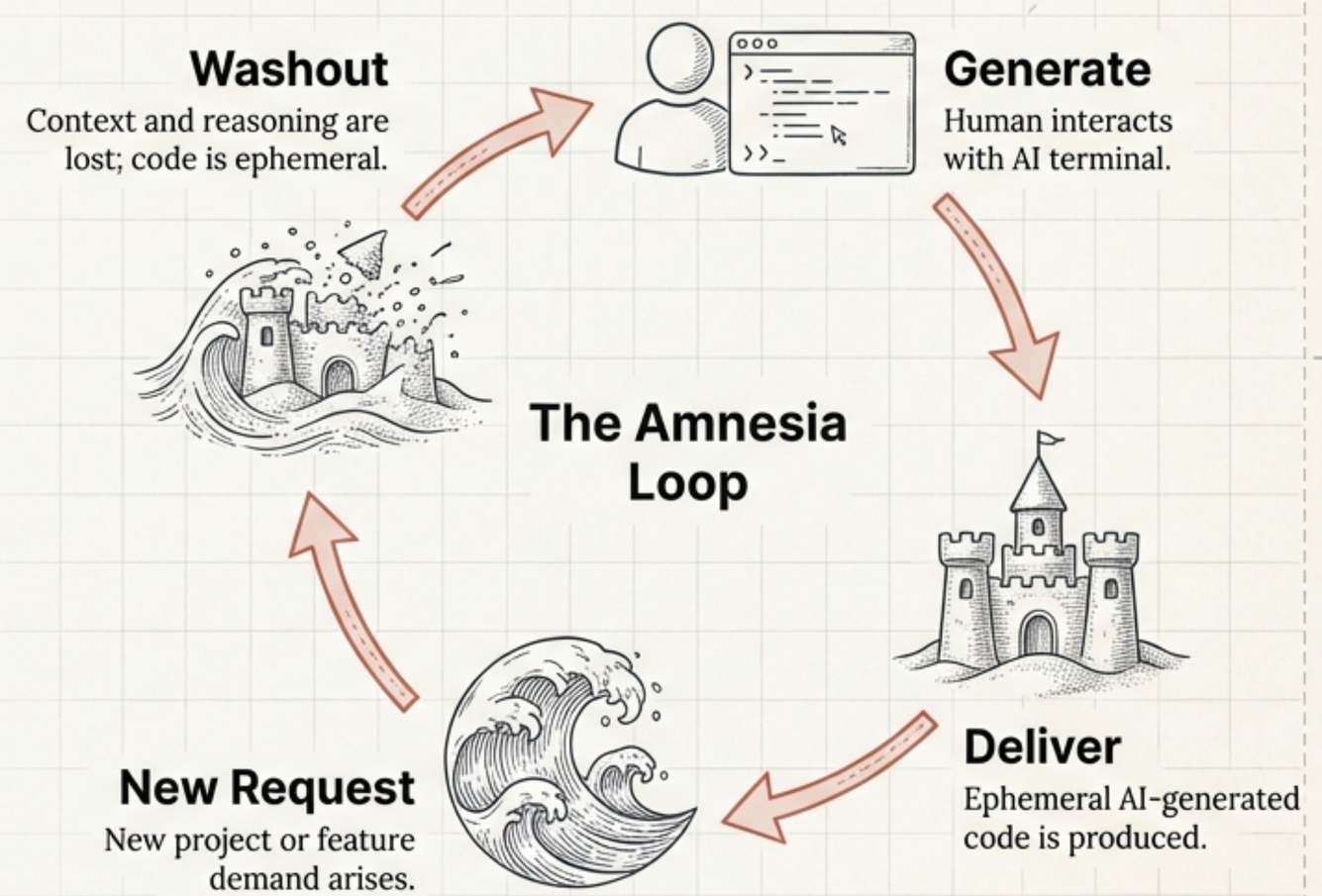
AI assistants deliver code, but the reasoning vanishes.

Every feature you build generates two outputs: working code and the intelligence behind it (the decisions, trade-offs, and successful prompts).

Today, we discard the intelligence and keep the code, which is often ephemeral and quickly rewritten.

This means every new project, and even every new feature, starts from a state of amnesia. We lose the "why," forcing us to solve the same problems repeatedly.

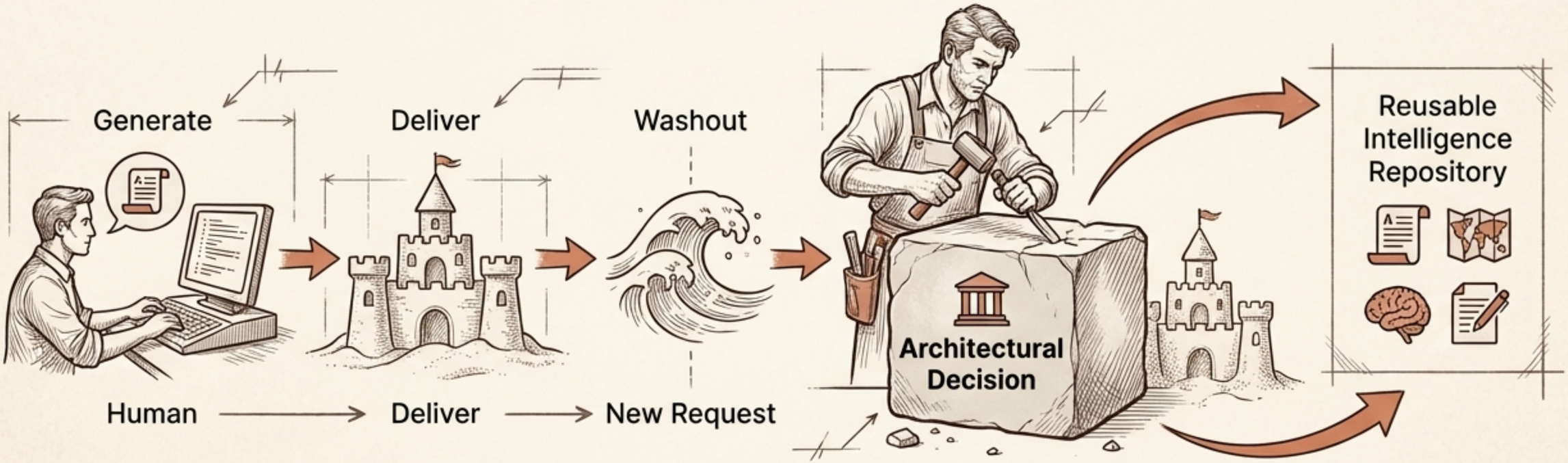
> We are building with sand, not stone.



Spec-Kit Plus is a framework to capture intelligence, not just code.

Spec-Kit Plus is a **Specification-Driven Development** with **Reusable Intelligence** (SDD-RI) framework. It provides a methodology and structure to systematically capture the "why" behind your code as permanent, reusable assets.

- It provides: Templates for specifications and plans, slash commands to orchestrate AI subagents, and a directory structure that separates permanent intelligence from ephemeral code.
- **Critical distinction:** It is NOT an AI service. It is a framework that works WITH your existing AI tools (Claude Code, Gemini CLI, etc.).



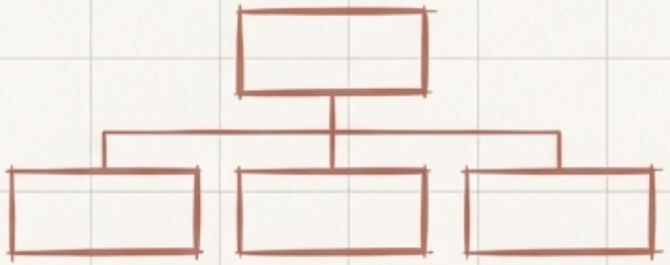
The framework is built on two architectures of reusable intelligence.

Horizontal Intelligence

Captures reasoning and decisions *across time*. It's the institutional memory that persists from one project to the next, preventing repeated mistakes.

- Architectural Decision Records (ADRs)
- Prompt History Records (PHRs)

Vertical Intelligence



Distributes work through a hierarchy of specialized AI subagents. It's the embedded expertise that can be delegated to solve specific problems consistently.

- Skills & Subagents

Horizontal Intelligence turns decisions into a compounding knowledge base.

Architectural Decision Records (ADRs)

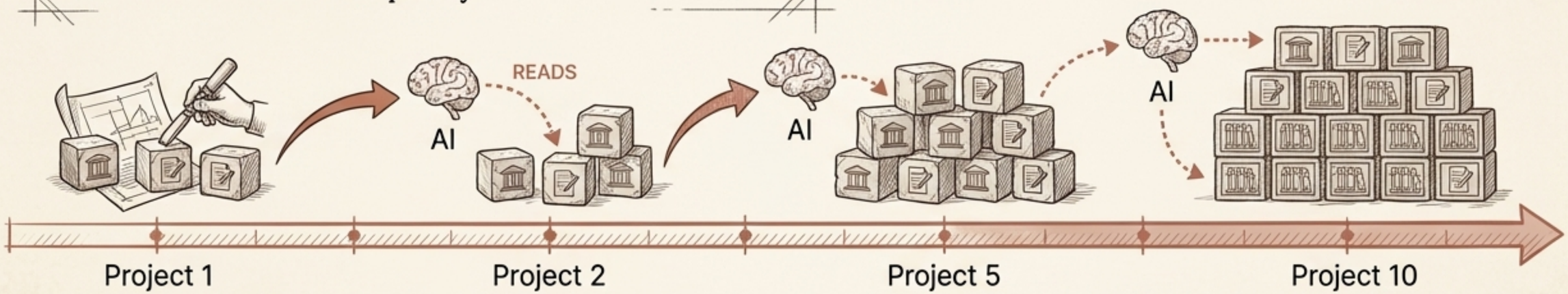
Document the "WHY" behind significant decisions.

- **Bad documentation:** "Used JWT for authentication."
- Good ADR: "Chose JWT over sessions because: (1) stateless auth needed for microservices, (2) mobile clients benefit from token refresh. Tradeoff accepted: token revocation complexity."

Prompt History Records (PHRs)

Automatically log AI collaboration sessions to capture what works and what fails.

How it works: System-generated records of `/sp.*` command executions, ensuring successful patterns are retained.



This isn't just documentation. AI agents read ADRs for context and learn from PHRs to avoid past mistakes.

Vertical Intelligence delegates work to specialized AI subagents.

Instead of using one generic AI, you delegate tasks to specialists. Each subagent is designed with the Persona + Questions + Principles (P+Q+P) pattern to activate reasoning, not just prediction.

Persona

"You are a requirements analyst who obsesses over edge cases before implementation."
(NOT "*You are a helpful assistant.*")

Questions

"What inputs can break this? What assumptions are hidden? What's the simplest test?"
(NOT "*Is this good?*")

Principles

"Every data input must document boundary conditions (zero, negative, overflow)."
(NOT "*Make it good.*")

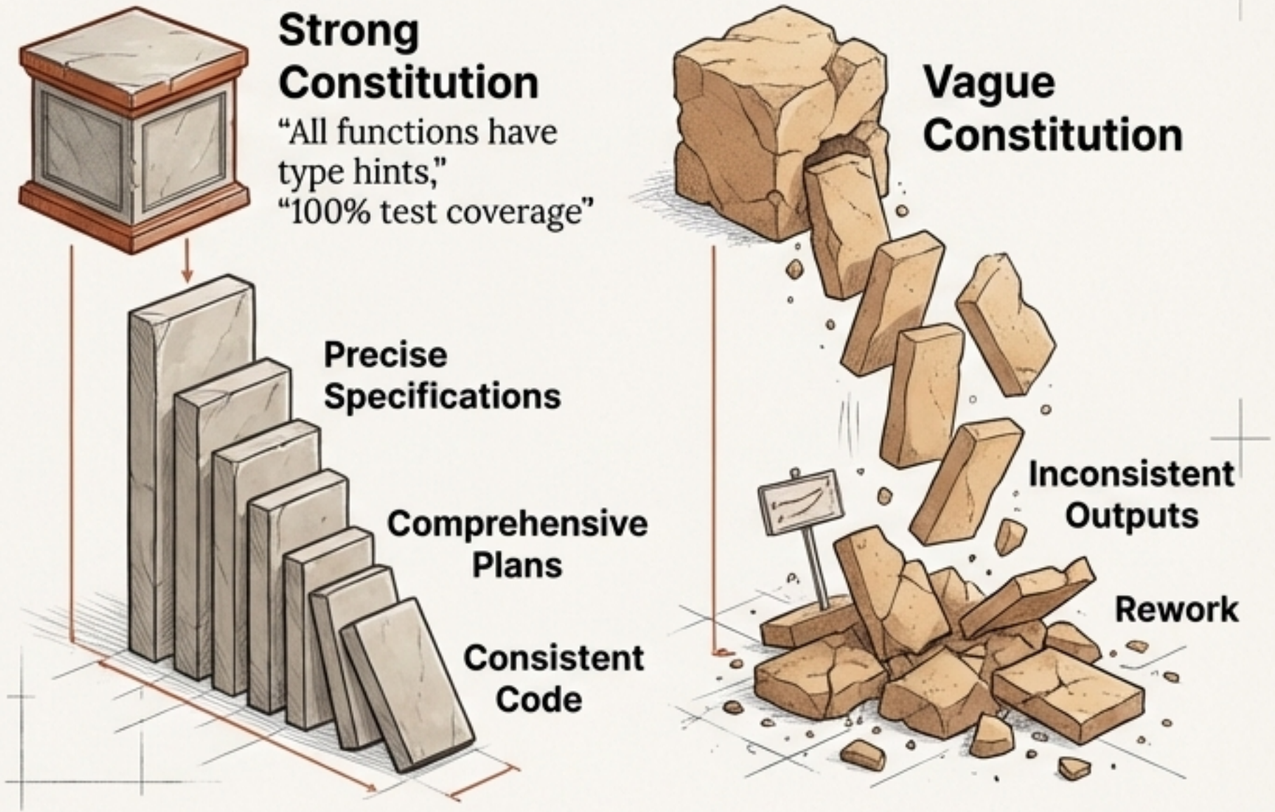
Key Insight

The **Specification Subagent** works for any feature needing a spec (auth, payments, uploads). You don't rebuild this expertise; you reuse it.

A strong Constitution is the source of all downstream quality.

The Constitution is a document created once per project that defines immutable standards for all features. It's the rulebook you and your AI must follow.

- **Constitution (Global Rules):** Code quality, testing requirements, error handling patterns.
- **Specification (Feature Rules):** User stories, acceptance criteria for one feature.



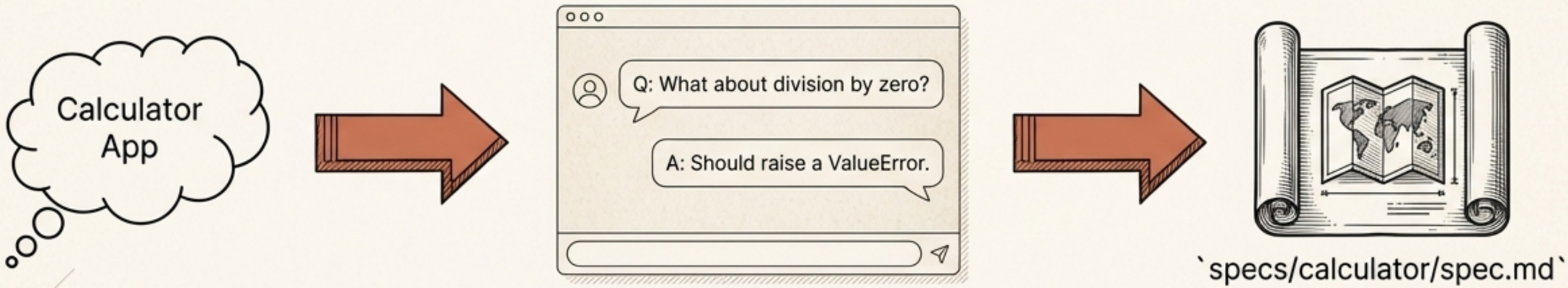
Expert Insight

"In AI-native development, the Constitution isn't bureaucracy—it's leverage. The 30 minutes you invest here saves hours of rework later."

The workflow begins by defining "what" success looks like, before writing a single line of code.

Using our "calculator project" example, the journey starts not with code, but with conversation. The Specify and Clarify phases translate an idea into a complete, testable specification that the AI can build from.

1. **Evals-First Conversation**: An informal human-AI chat to define business success criteria. ("What does a 'good' calculator do? How does it handle division by zero?")
2. **/sp.specify**: Formalizes the conversation into `spec.md`, defining scope, user stories, and SMART acceptance criteria.
3. **/sp.clarify**: An AI-driven "devil's advocate" check that probes the spec for ambiguities, missing assumptions, and edge cases.

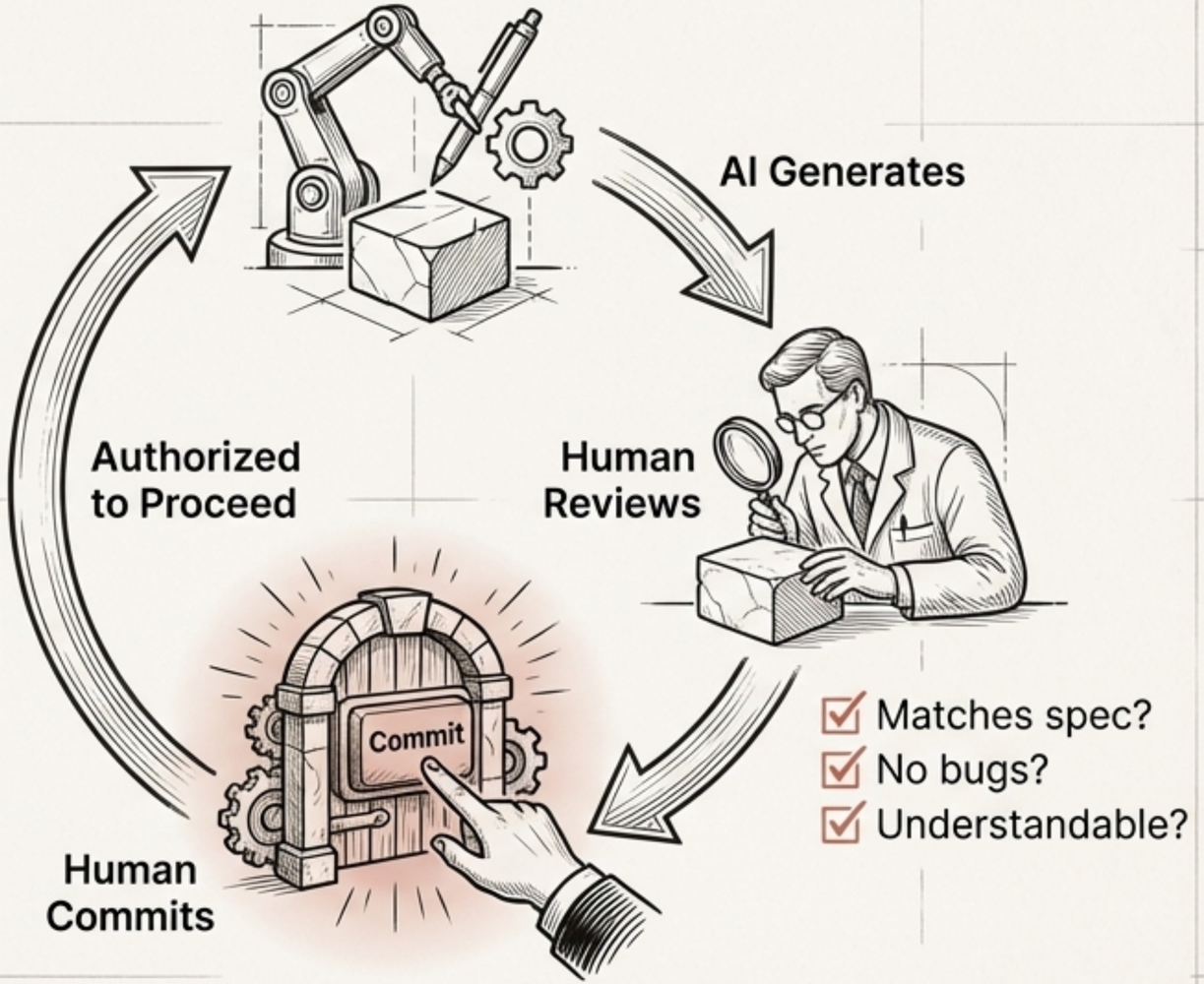


🎓 **Expert Insight:** "In AI-native development, your ability to write a clear specification is more valuable than your ability to write code. The spec IS your code."

The Checkpoint Pattern puts you in control of the workflow.

After the spec is clear, the `/sp.plan` and `/sp.tasks` commands create the implementation strategy.

This process is governed by the Checkpoint Pattern, which prevents large, un-reviewable blocks of AI work and keeps you in command.

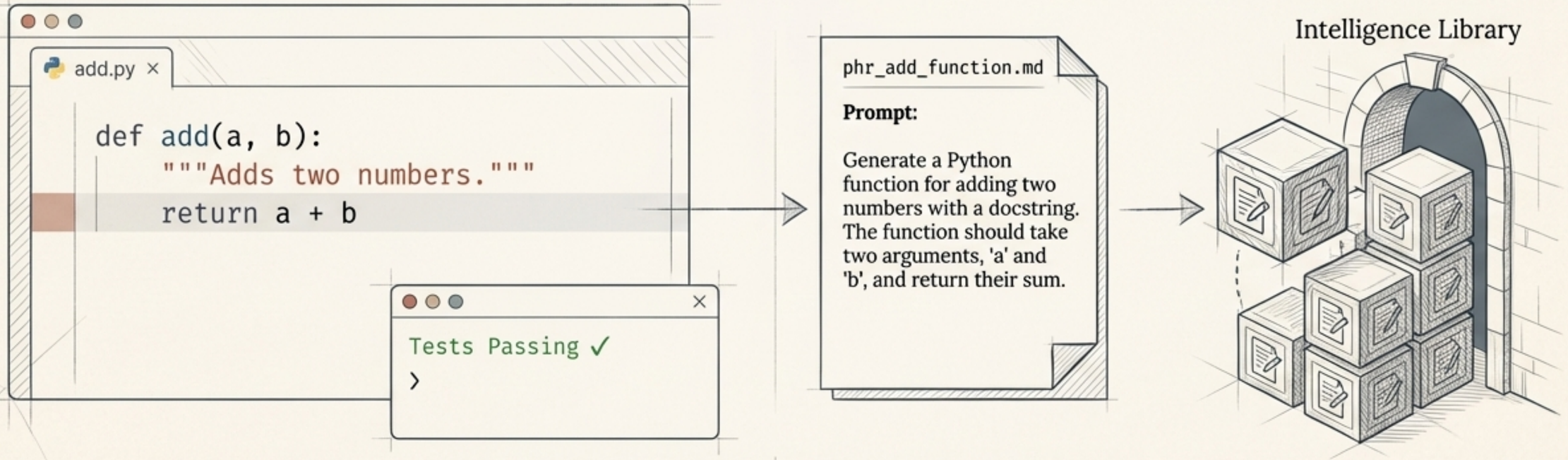


Expert Insight: "This pattern transforms risk management. Catching bugs in 200 lines of code at at a checkpoint is 100x cheaper than catching them in 5000 lines later."

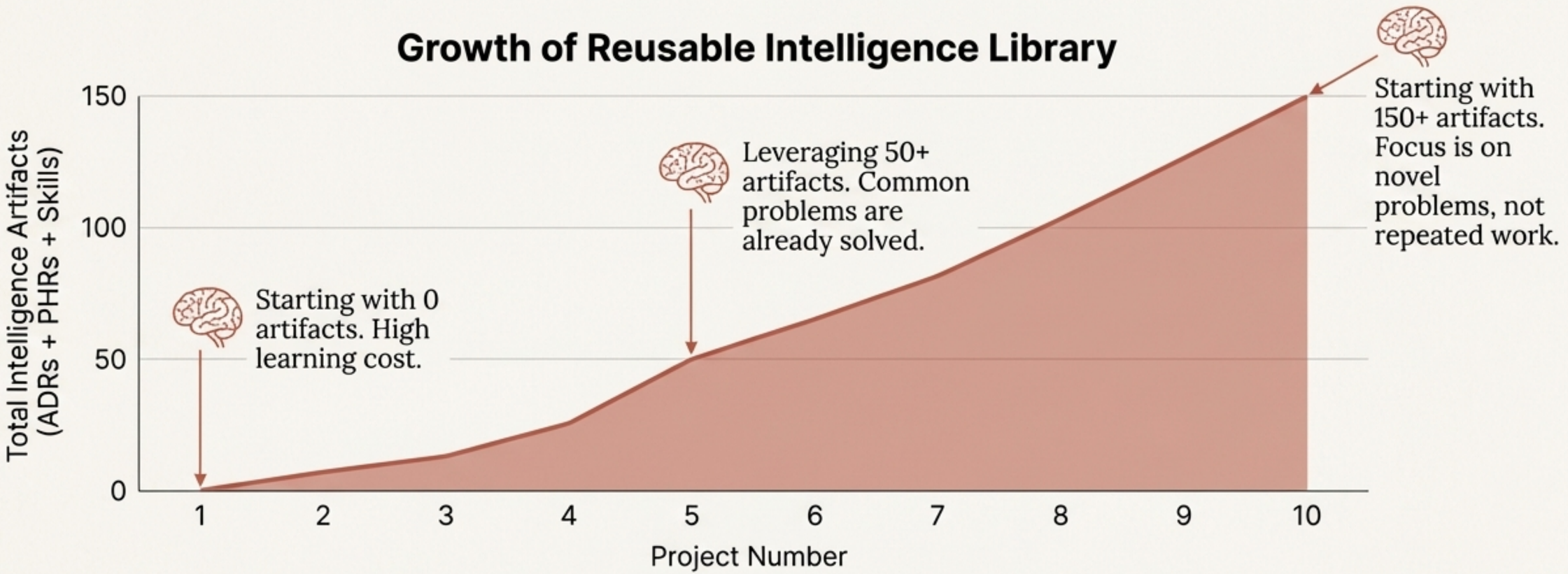
Implementation generates code while automatically capturing successful patterns.

The `/sp.implement` command orchestrates code generation, task by task, following the checkpoint pattern. Your role shifts from author to validator.

- **The 5-Step Validation Protocol:** A systematic process for reviewing AI-generated code: 1. Read & Understand, 2. Check Against Spec, 3. Run Tests, 4. Manual Test, 5. Approve & Commit.
- **Automatic Intelligence Capture:** During this process, Prompt History Records (PHRs) 📝 are automatically created in the `history/prompts/` directory, documenting the successful interactions.



Your intelligence library compounds, making every subsequent project dramatically faster.

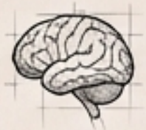


- **Project 1:** Create 3 ADRs + 10 PHRs.
- **Project 2:** Start with 13 artifacts, create 11 new ones.
- **Project 10:** Start with the accumulated intelligence from 9 prior projects. You rarely repeat mistakes.

You evolve from executing workflows to designing your own intelligence.

The true power of Spec-Kit Plus comes from encoding your own recurring patterns as reusable intelligence. Use this framework to decide when to create a new Skill or Subagent.

Pattern	Frequency (>=3 projects?)	Complexity (>=5 decisions?)	Org Value?	Encode?
Specification Review	✓	✓	✓	**YES**
Edge Case ID	✓	✓	✓	**YES**
Git Branch Creation	✓	✗	✗	**NO**



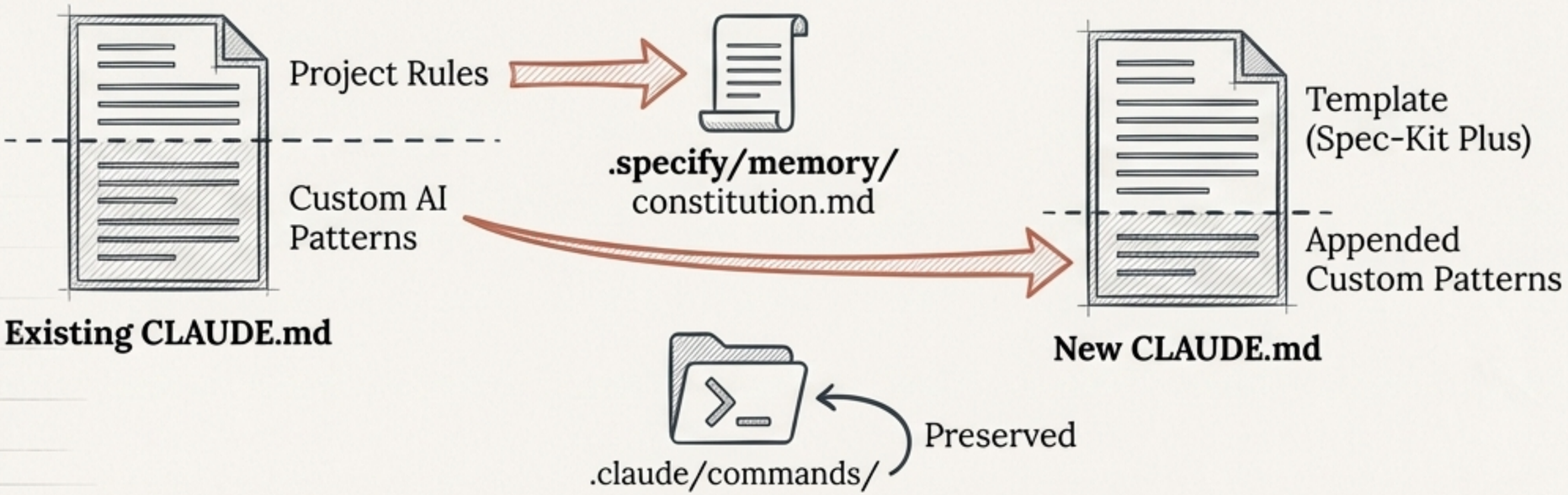
You use the same **Persona + Questions + Principles (P+Q+P)** pattern you learned about earlier to build your own custom tools, transforming tacit knowledge into explicit, shareable assets 🧠.

Bring intelligence capture to your existing codebase with a safe, proven workflow.

You can add Spec-Kit Plus to an existing project using `specifyplus init --here`.

Critical Warning: This command is EXPERIMENTAL and **will overwrite your existing CLAUDE.md file**. Do not run it without following the safety protocol.

The Merge Strategy



A step-by-step safety protocol for brownfield adoption.

This workflow uses git isolation and redundant backups to guarantee zero data loss. Follow these steps precisely.

1 Isolate

```
git checkout -b feat/skp-adoption
```

(Protects your main branch.)

2 Backup

```
cp CLAUDE.md CLAUDE.md.backup
```

(Creates a manual fallback.)

3 Commit

```
git commit -am "chore: backup
before speckit init"
```

(Creates a historical recovery point.)

4 Initialize

```
specifyplus init --here
```

(Runs the experimental command.)

5 Merge

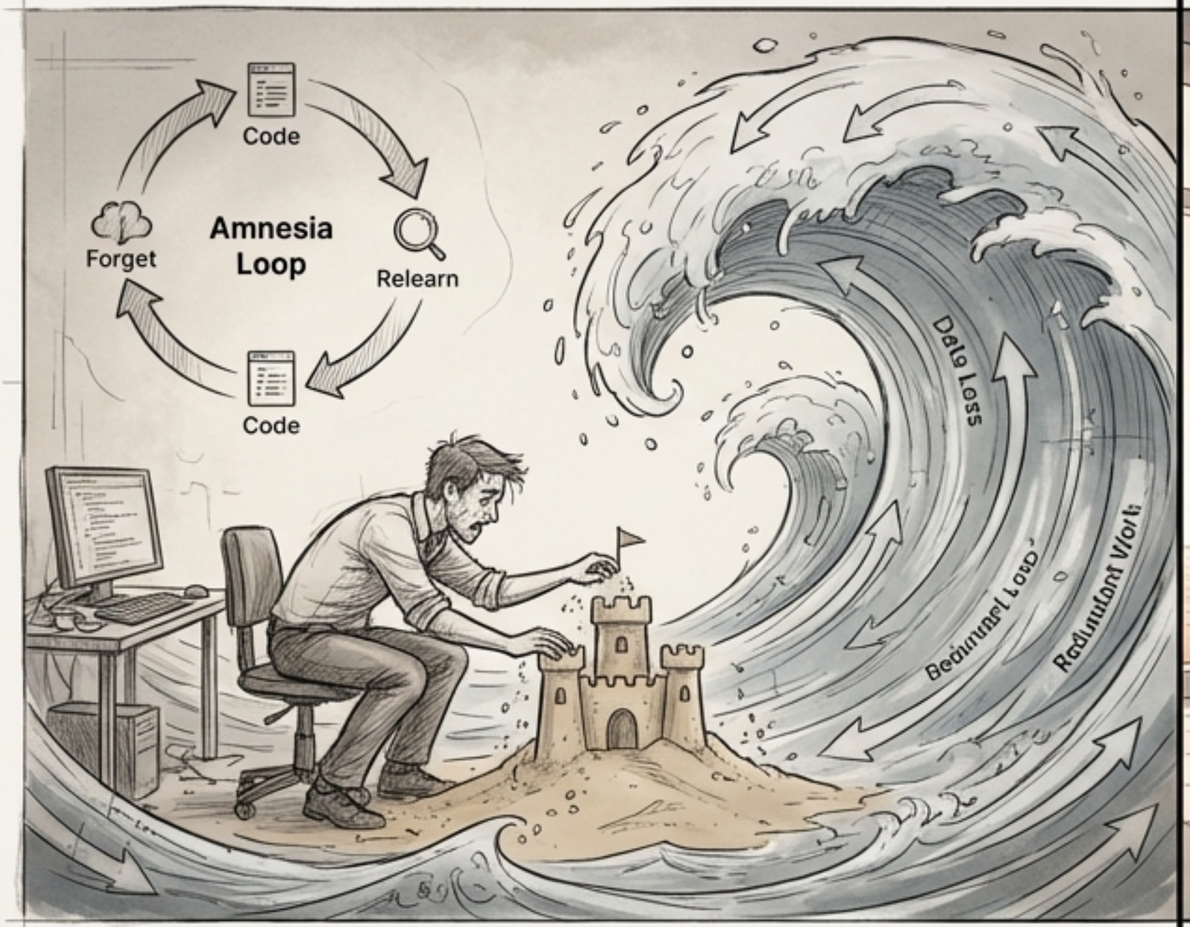
Manually edit `constitution.md` and `CLAUDE.md` using your backup file.

6 Validate & Commit

Test the setup and commit the new, merged configuration.

Stop being a code generator. Become an intelligence architect.

Ephemeral Code



Permanent Intelligence

In AI-native development, the units of reuse are specifications, agent architectures, and skills.
The value you create is permanent, compounding with every project you build.