ENGINEERING DISCIPLINE FOR THE AI ERA

Spec-Driven and Test-Driven Development, with GitHub Spec-Kit and Specky.

How specifications and tests turn AI coding agents from unpredictable assistants into reliable engineering partners.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
90 to 120 minutes

DATE
2026-06-10

AGENDA

Eleven chapters, one thesis: intent first, code last.

01 The problem: vibe coding and intent debt

02 Spec-Driven Development fundamentals

03 Test-Driven Development fundamentals

04 Where SDD meets TDD

05 GitHub Spec-Kit in depth

06 Specky in depth

07 The end-to-end workflow

08 Quality gates and traceability

09 Anti-patterns and failure modes

10 Results from the field

11 Getting started: an adoption roadmap

CHAPTER 01 · THE PROBLEM

Vibe coding does not scale.

AI coding agents are powerful, but prompting them without a written specification produces software whose intent lives nowhere. This chapter names the failure patterns and the debt they create.

THREE FAILURE PATTERNS

Why "prompt and pray" fails on anything bigger than a script.

PATTERN 01 · LOST INTENT

The "why" exists only in a chat history.

Requirements are scattered across prompts. Six months later nobody can say what the system was supposed to do, only what it happens to do.

PATTERN 02 · DRIFT PER ITERATION

Every regeneration moves the target.

Without a stable spec, each "fix this" prompt re-interprets the goal. The agent optimizes the last message, not the product.

PATTERN 03 · UNVERIFIABLE OUTPUT

Code that looks right is not code that is right.

Plausible output passes review by humans who skim. Without tests derived from requirements, correctness is a feeling, not a fact.

THE COST

Intent debt

Technical debt is code you owe. Intent debt is understanding you owe: every behavior shipped without a written, testable requirement is a decision your future team must reverse-engineer from the code itself.

Symptoms: onboarding measured in months, "do not touch" modules, rewrites that lose features nobody documented, and AI agents that cannot help because the repository carries no machine-readable intent.

TWO WAYS OF WORKING

Same agent, same model. The discipline is the difference.

VIBE CODING

Prompt, accept, hope

- Intent lives in ephemeral chat threads
- Tests written after the fact, if at all
- Each change re-negotiates the goal
- Review checks style, not requirements
- Agent context rebuilt by hand every session

SPEC PLUS TEST DRIVEN

Specify, verify, implement

- Intent versioned in the repository as specs
- Tests derived from acceptance criteria, written first
- Changes amend the spec, then the code
- Review checks spec compliance and test coverage
- Agents load specs and constitution as durable context

CHAPTER 02 · FOUNDATIONS

Spec-Driven Development.

SDD inverts the usual relationship between specification and code: the spec is the primary artifact, executable and versioned, and code becomes its generated, verifiable expression.

DEFINITION

In SDD, the specification is the source of truth. Code is a projection of it.

A spec in SDD is not a Word document that dies after kickoff. It is a structured, versioned artifact that states what the system must do, why, and how success is verified. Humans and AI agents both read it, both are accountable to it, and every implementation decision traces back to a line in it.

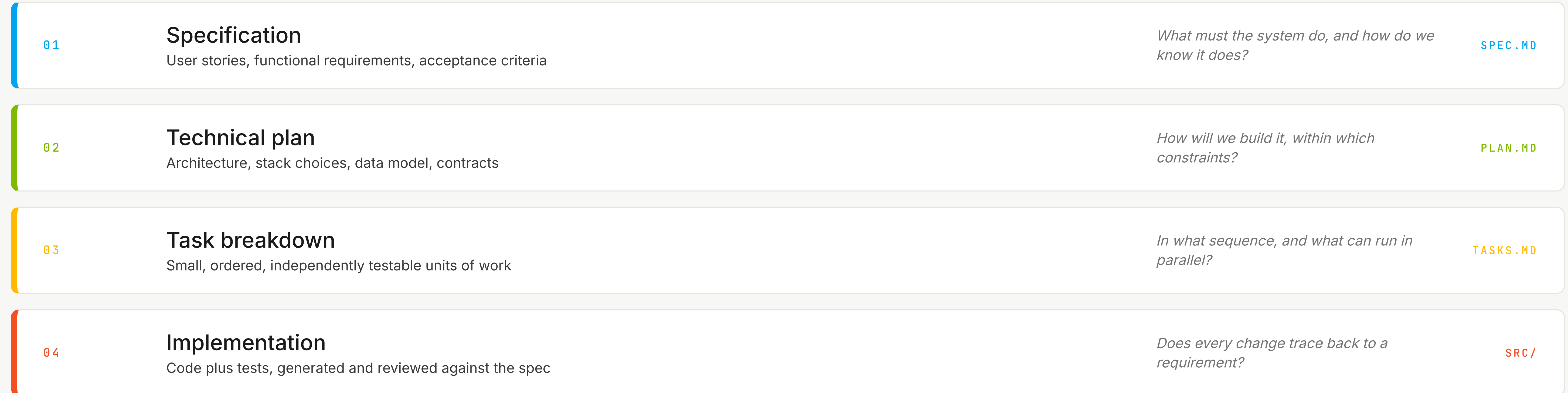
01 Written before code, refined continuously, never abandoned.

02 Precise enough that an agent can plan and implement from it.

03 Testable by construction: every requirement maps to acceptance criteria.

THE ARTIFACT CHAIN

Four artifacts, each derived from the previous one.



CORE PRINCIPLES

Four principles that make SDD work in practice.

PRINCIPLE 01

Intent before mechanism

What and why precede how

- Specs state outcomes, not implementations
- Technology choices live in the plan, not the spec
- Ambiguity is flagged, never silently resolved

PRINCIPLE 02

Executable specifications

Specs drive generation

- Structured enough for agents to act on
- Acceptance criteria become test cases
- The spec is input, not documentation output

PRINCIPLE 03

Continuous refinement

Specs evolve with the product

- Change requests amend the spec first
- Spec and code reviewed together in PRs
- Stale specs are treated as build breaks

PRINCIPLE 04

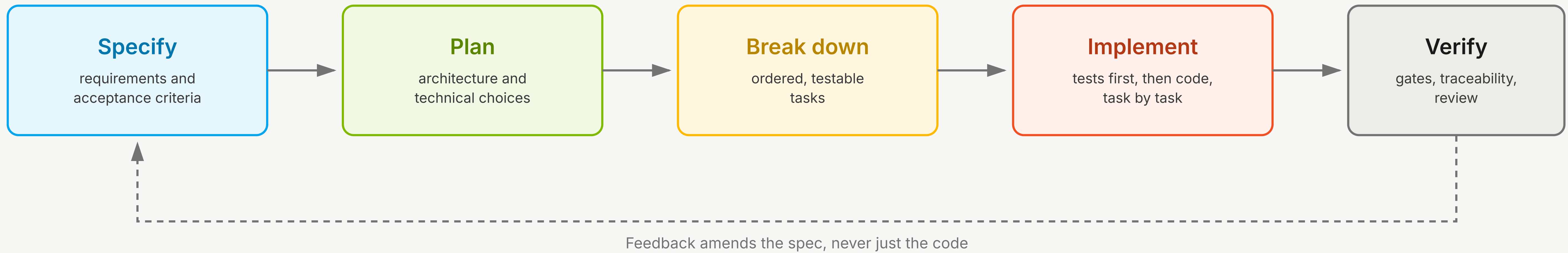
Constitution as law

Non-negotiables in one file

- Project-wide principles every artifact obeys
- Security, quality, and style baselines
- Checked at every phase gate

LIFECYCLE

The SDD loop: every change flows through the spec.



QUALITY BAR

What separates a usable spec from a wish list.

01 **Testable.** Every requirement has acceptance criteria a machine can check. "Fast" becomes "p95 under 200 ms at 1,000 RPS".

02 **Unambiguous.** EARS notation forces structure: "WHEN a user submits invalid credentials, THE SYSTEM SHALL return 401 within 100 ms".

03 **Scoped.** Explicit non-goals prevent agents and humans from gold-plating beyond the requirement.

04 **Marked where uncertain.** Open questions carry a [NEEDS CLARIFICATION] tag instead of a silent guess.

05 **Traceable.** Stable requirement IDs (FR-001, NFR-003) that tests, tasks, and commits reference.

CHAPTER 03 · FOUNDATIONS

Test-Driven Development.

TDD is the discipline of writing a failing test before the code that makes it pass. Twenty years of practice, newly essential: tests are how humans verify what agents produce.

DEFINITION

Red, green, refactor: the smallest loop in software engineering.

Write a test that fails because the behavior does not exist yet. Write the minimum code that makes it pass. Improve the design while the tests stay green. Repeat in cycles of minutes, not days. The test suite becomes a living, executable specification of behavior.

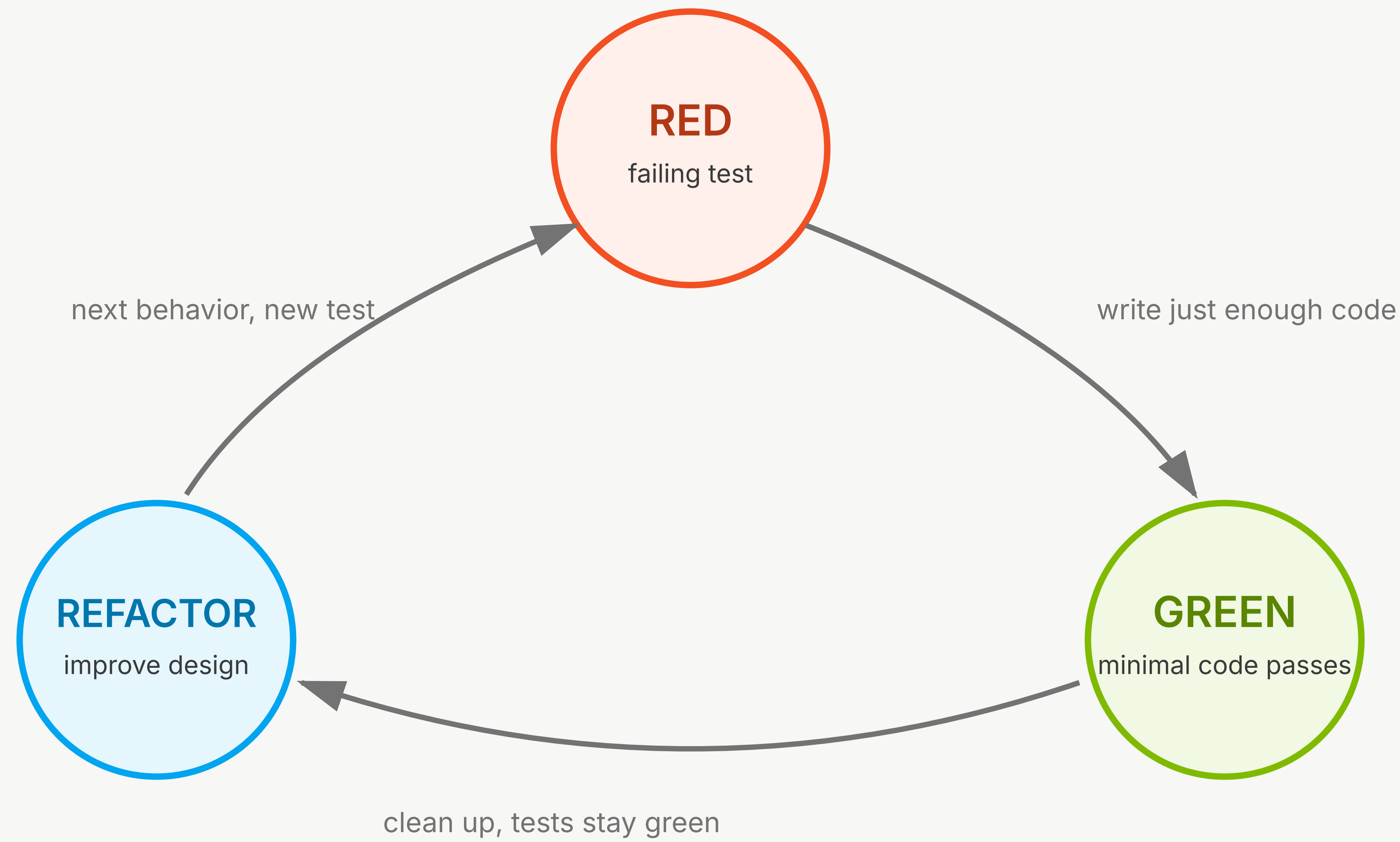
01 **Red.** The failing test proves the test itself works and the behavior is missing.

02 **Green.** The simplest passing implementation, resisting speculative generality.

03 **Refactor.** Design improves under the protection of a green suite.

THE CYCLE

Minutes per loop. The speed is the point.



THE THREE LAWS OF TDD

Robert C. Martin's formulation, still the sharpest.

01	You may not write production code until you have written a failing unit test.
02	You may not write more of a unit test than is sufficient to fail, and not compiling counts as failing.
03	You may not write more production code than is sufficient to pass the currently failing test.

Applied to agents, the laws become a contract: the agent must show the failing test before it is allowed to write the implementation. Spec-Kit and Specky both encode this ordering into their task templates.

WHY IT PAYS

What a test-first suite buys you, especially with agents in the loop.

DESIGN PRESSURE

Hard-to-test code is hard-to-use code.

Writing the test first forces small interfaces, injected dependencies, and low coupling before the implementation calcifies.

REGRESSION SHIELD

Refactoring without fear.

A green suite makes aggressive cleanup safe, for humans and for agents asked to "modernize this module".

EXECUTABLE DOCUMENTATION

Tests state behavior precisely.

Each test names a behavior and proves it. New team members and agents read the suite as ground truth.

AGENT VERIFICATION

The only scalable review of AI output.

Humans cannot line-review thousands of generated lines per day. A requirement-derived suite can.

CHAPTER 04 · CONVERGENCE

IV

Where SDD meets TDD.

Two disciplines, one chain of trust: the spec defines what correct means, the tests enforce it, and the code earns its way in by passing. Together they close the loop that vibe coding leaves open.

COMPLEMENTARY BY DESIGN

SDD and TDD answer different questions. Neither replaces the other.

SDD ANSWERS

What and why

- Captures intent at product and feature level
- Defines scope, constraints, and non-goals
- Aligns stakeholders before code exists
- Gives agents durable, versioned context
- Granularity: feature, days

TDD ANSWERS

Does it, provably

- Verifies behavior at unit and integration level
- Drives modular, testable design
- Catches regressions on every change
- Gives agents a binary pass/fail signal
- Granularity: behavior, minutes

THE BRIDGE

Acceptance criteria are the translation layer between spec and test.

SPEC.MD · REQUIREMENT FR-007

```
## FR-007 Account lockout
WHEN a user fails authentication
  5 times within 15 minutes,
THE SYSTEM SHALL lock the account
  for 30 minutes
AND SHALL notify the user by email.

Acceptance criteria:
- 5th failure inside window locks
- 4 failures do not lock
- lock expires after 30 minutes
- email job enqueued exactly once
```

TEST_LOCKOUT.PY · DERIVED FIRST, BEFORE ANY CODE

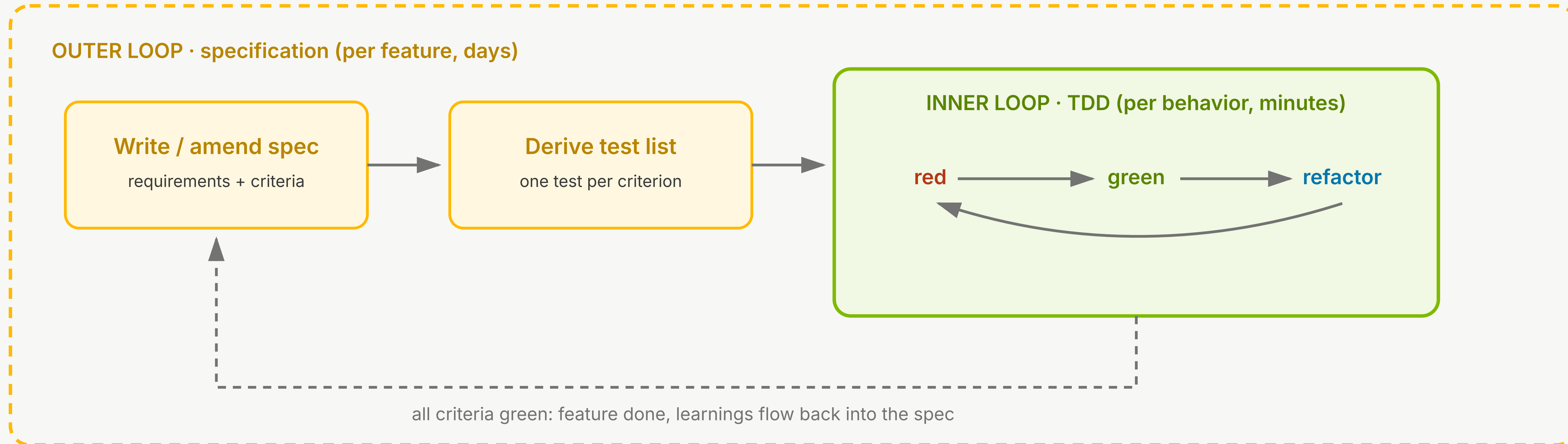
```python
def test_fifth_failure_locks_account():
    # FR-007 / AC-1
    fail_login(user, times=5, window="15m")
    assert account(user).locked

def test_four_failures_do_not_lock():
    # FR-007 / AC-2
    fail_login(user, times=4, window="15m")
    assert not account(user).locked

def test_lock_expires_after_30_minutes():
    # FR-007 / AC-3
    ...
```

THE DOUBLE LOOP

An outer spec loop wrapping an inner test loop.



CHAPTER 05 · TOOLING

GitHub Spec-Kit.

An open-source toolkit from GitHub that operationalizes SDD: a CLI, structured templates, and slash commands that walk any coding agent through specify, plan, tasks, and implement.

GITHUB.COM/GITHUB/SPEC-KIT

Spec-Kit turns SDD from a philosophy into a repeatable workflow.

Install the Specify CLI, initialize a project, and the kit scaffolds a constitution, spec templates, and agent commands. From then on, every feature follows the same phased path, and every artifact lands in a predictable place in the repository, reviewable like code.

01 Open source, agent-agnostic, MIT licensed.

02 Works inside the agent you already use, via slash commands.

03 Artifacts are plain Markdown in git: diffable, reviewable, durable.

ARCHITECTURE

Three layers: CLI, templates, and your agent.

#	Layer	Details	Role	Tag
01	**Specify CLI**	uvx specify init, project scaffolding, agent detection	*Bootstraps the structure once per project*	PYTHON
02	**Templates and scripts**	spec, plan, and tasks templates, constitution, phase-gate checks	*Encodes the method so quality does not depend on memory*	.SPECIFY/
03	**Your coding agent**	GitHub Copilot, Claude Code, Gemini CLI, Cursor, and others	*Executes each phase through slash commands*	/SPECIFY

THE PHASES

Four commands, four artifacts, four gates.

/SPECIFY

Specification

From idea to requirements

- User stories and functional requirements
- Acceptance criteria per requirement
- [NEEDS CLARIFICATION] for every gap

/PLAN

Technical plan

From requirements to design

- Stack, architecture, data model
- Contracts and API surfaces
- Checked against the constitution

/TASKS

Task breakdown

From design to work units

- Small, ordered, testable tasks
- [P] markers for parallelizable work
- Tests listed before implementation

/IMPLEMENT

Implementation

From tasks to verified code

- Executes task by task, test first
- Stops at gates and failures
- Output traceable to requirement IDs

PHASE 1 IN PRACTICE

/specify: describe the what and why, get a structured spec.

AGENT SESSION · FEATURE 003-TEAM-DASHBOARD

```
> /specify Build a team dashboard where managers see open tasks per member,
  filter by sprint, and export the view to CSV. Managers only; members see
  their own tasks. No real-time updates needed in v1.

Created branch: 003-team-dashboard
Created spec:   specs/003-team-dashboard/spec.md

## User stories          3 stories, 11 functional requirements
## Acceptance criteria  per requirement, EARS formatted
## Non-goals            real-time sync, member-level analytics
## Open questions       2 × [NEEDS CLARIFICATION]
  - CSV export: include archived tasks?
  - "manager" defined by org chart or project role?
```

The command creates a branch and a numbered spec folder. Open questions block the next phase until a human answers them.

PHASE 2 IN PRACTICE

/plan: technical choices, constrained by the constitution.

```
● AGENT SESSION · FEATURE 003-TEAM-DASHBOARD

> /plan Use the existing React + FastAPI stack. Postgres for storage.
  Reuse the auth middleware. CSV generation server-side.

Reading constitution.md ... 9 articles loaded
Created plan:      specs/003-team-dashboard/plan.md
Created contracts: specs/003-team-dashboard/contracts/dashboard-api.yaml
Created data model: specs/003-team-dashboard/data-model.md

Constitution check
  ✓ Art. III  test-first mandate acknowledged in plan
  ✓ Art. VII  no new framework without justification
  ✗ Art. V    observability: missing structured logging section
Plan gate: 1 violation to resolve before /tasks
```

The plan inherits every requirement ID from the spec. Constitution violations are surfaced as gate failures, not buried in prose.

PHASES 3 AND 4 IN PRACTICE

/tasks orders the work, /implement executes it test-first.

AGENT SESSION · FEATURE 003-TEAM-DASHBOARD

```
> /tasks
Created tasks: specs/003-team-dashboard/tasks.md · 18 tasks
  T001  setup: migration for dashboard_views table
  T002 [P] contract test: GET /api/teams/:id/dashboard  (FR-001)
  T003 [P] contract test: GET .../export.csv            (FR-009)
  T004  unit tests: sprint filter logic                  (FR-004)
  T005  implement: dashboard query service  depends: T002, T004
  ...

> /implement
T002 ... wrote test, ran suite: FAIL (expected, endpoint missing)
T005 ... implemented service, ran suite: PASS 14/14
Gate: coverage on changed lines 96%, traceability 18/18 tasks linked
```

Tests are tasks of their own and always precede the implementation tasks they verify. The agent must show red before green.

THE CONSTITUTION

Project law: principles that every spec, plan, and task must obey.

01 Written once at init, amended rarely and deliberately.

02 Checked automatically at the plan gate and the implement gate.

03 Typical articles: test-first mandate, simplicity, observability, security baselines, dependency policy.

04 Gives agents stable values that survive across sessions and models.

MEMORY/CONSTITUTION.MD · EXCERPT

```
# Article III · Test-first (non-negotiable)
All implementation tasks MUST be preceded
by failing tests derived from acceptance
criteria. Red before green, no exceptions.

# Article VII · Simplicity
Start with the simplest design that meets
the spec. New frameworks require written
justification in plan.md.

NEVER merge with failing gates.
NEVER resolve a [NEEDS CLARIFICATION]
  by guessing.
```

AGENT SUPPORT

One method, many agents. Spec-Kit is deliberately agent-agnostic.

FIRST-PARTY

GitHub Copilot

Slash commands in VS Code and GitHub Copilot coding agent. Tightest integration with the GitHub flow: branches, PRs, checks.

CLI AGENTS

Claude Code, Gemini CLI, Codex CLI

Commands installed as agent-native prompts at init. The same four phases, the same artifacts, in the terminal.

IDE AGENTS

Cursor, Windsurf, Qwen, and more

specify init --ai <agent> generates the right command format per tool. Teams can mix agents on one repo because the artifacts are shared.

The artifacts, not the agent, carry the project. Switching agents mid-project costs nothing because the spec, plan, and tasks are plain Markdown in git.

CHAPTER 06 · TOOLING

Specky.

A spec-driven orchestrator that adds interactive discovery, EARS-notation requirements, quality gates with traceability matrices, and dual-runtime support for GitHub Copilot and Claude Code.

SPECKY · OPEN SOURCE

Specky turns natural language into production-grade specs through guided discovery.

Where Spec-Kit gives you the phased skeleton, Specky leans into the conversation: it interviews you about the feature, scans existing codebases, drafts EARS-notation requirements, and will not let a phase close until its quality gate passes. It installs as agent commands for both GitHub Copilot and Claude Code from a single source.

01 Interactive discovery: the agent asks before it assumes.

02 Brownfield aware: auto-scans the repo to ground specs in reality.

03 Gates with evidence: traceability matrix generated, not promised.

CAPABILITIES

Four capabilities that define the Specky workflow.

CAPABILITY 01

Guided discovery

Interview before artifact

- Structured questions on scope, users, constraints
- Edge cases surfaced before they become bugs
- Answers recorded into the spec, not lost in chat

CAPABILITY 02

EARS requirements

Structured, testable language

- WHEN / WHILE / WHERE / IF templates
- Every SHALL maps to a verification
- Ambiguity becomes syntactically visible

CAPABILITY 03

Design and diagrams

Architecture made visible

- Mermaid architecture and sequence diagrams
- Data models and API contracts in the design doc
- Pre-implementation review gate for humans

CAPABILITY 04

Sequenced execution

Tasks with gates

- [P] parallel markers, dependency ordering
- Quality gates per phase, evidence required
- Requirement-to-test traceability matrix

CHOOSING A TOOL

Spec-Kit and Specky: same philosophy, different center of gravity.

GITHUB SPEC-KIT

The standard skeleton

- Origin: GitHub open source
- Focus: phased workflow and templates
- Requirements style: structured Markdown
- Breadth: a dozen-plus supported agents
- Best when: adopting SDD as a team standard on greenfield features

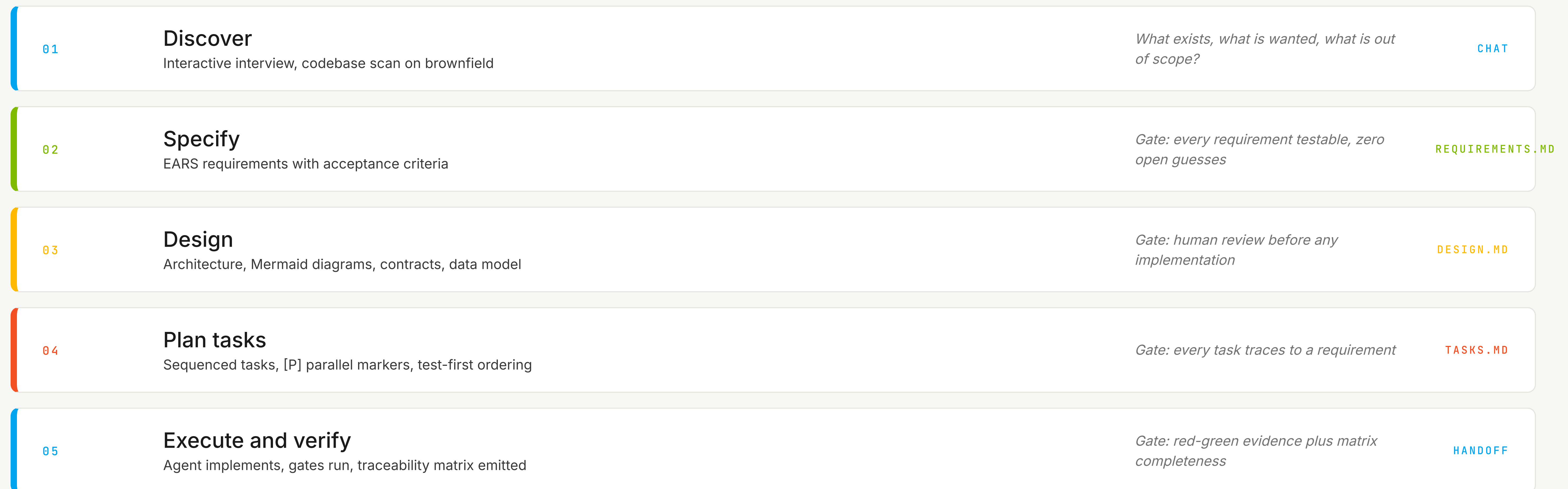
SPECKY

The opinionated orchestrator

- Origin: open source, Kiro-inspired
- Focus: discovery depth and quality gates
- Requirements style: EARS notation
- Depth: GitHub Copilot and Claude Code, one source
- Best when: brownfield work, strict traceability, regulated or audit-heavy contexts

WORKFLOW

From conversation to gated delivery in five moves.



CHAPTER 07 · PRACTICE

VII

The end-to-end workflow.

Greenfield and brownfield follow the same spine with different first moves. This chapter walks a feature from intent to merged pull request.

GREENFIELD

New project: constitution first, then features in numbered folders.

01 **Day 0.** specify init, write the constitution with the team: test-first, simplicity, security and observability baselines.

02 **Per feature.** /specify, answer clarifications, stakeholder sign-off on the spec before any planning.

03 **Design.** /plan against the constitution; contracts and data model reviewed like code.

04 **Build.** /tasks then /implement; tests precede code, [P] tasks fan out to parallel agent sessions.

05 **Merge.** PR contains spec delta, code, tests, and the gate report side by side.

BROWNFIELD

Existing codebase: specify the seam, not the whole system.

01 **Scan.** Let the tool index the codebase: stack, patterns, test layout, integration points. The spec must respect what exists.

02 **Characterize.** Before changing legacy behavior, write characterization tests that pin current behavior down.

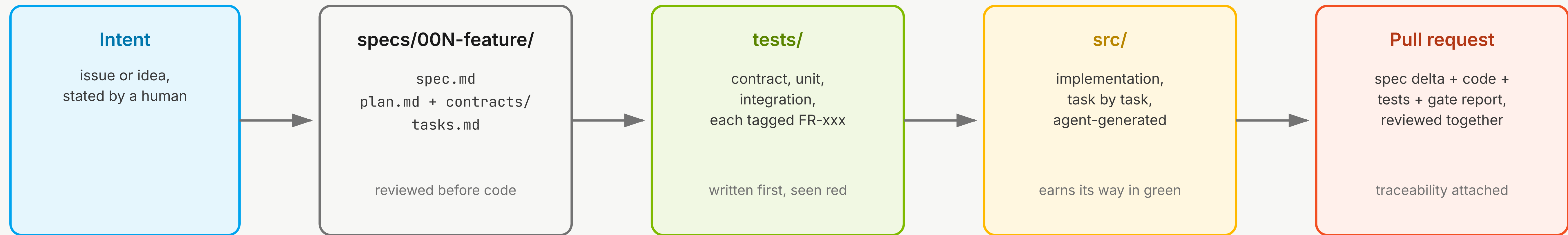
03 **Specify the delta.** The spec covers the change and its blast radius, with explicit compatibility requirements.

04 **Migrate incrementally.** Tasks sized so each lands behind a green suite; no big-bang rewrites.

05 **Pay down intent debt.** Each touched module leaves with a spec it never had.

ONE FEATURE, END TO END

What actually lands in the pull request.



The reviewer's question changes from "does this code look fine" to "does this code satisfy this spec", a question with an answer.

CHAPTER 08 · ASSURANCE

VIII

Quality gates and traceability.

A gate is a checkpoint that blocks progress until evidence exists. Traceability is the evidence: every requirement linked to the tests that verify it and the code that satisfies it.

THE GATES

Four gates between an idea and a merge.

Gate	Name	Criteria	Blocks
G1	**Spec gate** After /specify	*All requirements testable, zero unresolved clarifications*	BLOCKS /PLAN
G2	**Plan gate** After /plan	*Constitution articles satisfied, contracts cover every FR*	BLOCKS /TASKS
G3	**Task gate** After /tasks	*Every task linked to a requirement, tests ordered before code*	BLOCKS /IMPLEMENT
G4	**Merge gate** Before PR merge	*Suite green, coverage threshold met, traceability matrix complete*	BLOCKS MERGE

TRACEABILITY

The matrix: requirement, test, code, status. No orphans in either direction.

TRACEABILITY.MD · GENERATED AT THE MERGE GATE

Requirement	Tests	Implementation	Status
FR-001	test_dashboard_contract.py::3 cases	api/dashboard.py	PASS
FR-004	test_sprint_filter.py::5 cases	services/filters.py	PASS
FR-007	test_lockout.py::4 cases	auth/lockout.py	PASS
FR-009	test_csv_export.py::3 cases	services/export.py	PASS
NFR-002	test_perf_dashboard.py::p95_under_200	(query index, migration 14)	PASS
FR-011	MISSING	views/archive.py	GAP

Two failure smells the matrix exposes instantly: requirements with no test (unverified intent) and code with no requirement (unrequested behavior). Both block the merge gate.

CHAPTER 09 · CAUTION

Anti-patterns.

Both disciplines fail in predictable ways. Naming the failure modes is cheaper than living them.

SDD ANTI-PATTERNS

Five ways spec-driven work goes wrong.

01 **The frozen spec.** Written once, never amended; code drifts and the spec becomes fiction. Treat spec updates as part of every change.

02 **Waterfall cosplay.** Specifying the entire system for months before any code. SDD specs are per-feature and days deep, not project-wide tomes.

03 **Implementation leakage.** Specs that dictate libraries and table names. The what contaminated by the how loses its power to outlive the stack.

04 **Silent assumption.** The agent (or the author) resolves ambiguity by guessing instead of tagging [NEEDS CLARIFICATION].

05 **Gate theater.** Gates exist but are skipped under deadline pressure. A gate that can be waved through is documentation, not a gate.

TDD ANTI-PATTERNS

Five ways test-first work goes wrong, doubly so with agents.

01 **Test-after rationalization.** Code first, then tests that mirror the implementation. They pass by construction and verify nothing.

02 **Mock everything.** Suites that test the mocks. Integration behavior, the thing that breaks in production, stays unverified.

03 **Coverage worship.** 95% line coverage with assertion-free tests. Coverage measures execution, not verification.

04 **Agent gaming the suite.** An agent told "make tests pass" may weaken the tests. Tests derive from the spec, and spec-side criteria are the agent's read-only ground truth.

05 **Brittle coupling.** Tests bound to private internals shatter on every refactor and teach teams to delete them. Test behavior at stable interfaces.

CHAPTER 10 · EVIDENCE

Results from the field.

What changes, measurably, when teams move from prompt-driven to spec-and-test-driven AI development.

WHAT TEAMS REPORT

The pattern across early adopters is consistent in direction.

REWORK

Fewer regenerate-from-scratch cycles.

Clarifications happen at spec time, where a wrong answer costs a sentence, not at code time, where it costs a sprint.

REVIEW

PR review shifts from style to substance.

With gates checking tests and traceability, humans spend review time on design judgment, the part machines are worst at.

ONBOARDING

Specs become the fastest path into a codebase.

New engineers and new agent sessions load the same artifacts. Context rebuilding stops being a per-person tax.

HONESTY

The method exposes weak requirements early.

Teams discover that many "agent failures" were specification failures. The discipline relocates the problem to where it can be fixed.

Treat any specific percentage with care: published numbers vary by team, codebase, and baseline. The directional pattern above is what recurs.

CHAPTER 11 · ADOPTION

XI

Getting started.

You do not adopt SDD plus TDD in one reorg. You adopt it one feature at a time, with a roadmap that survives contact with deadlines.

ROADMAP

Crawl, walk, run: ninety days to a working practice.

#	Phase	Activities	Scope
01	**Crawl · weeks 1 to 3** One pilot feature, one team, one tool	*Init Spec-Kit or Specky, write the constitution, run one feature end to end*	1 FEATURE
02	**Walk · weeks 4 to 8** Default for new features on the pilot team	*Wire gates into CI, require the traceability matrix in PRs, refine templates from retro feedback*	1 TEAM
03	**Run · weeks 9 to 13** Scale out, brownfield in	*Second and third teams onboard with the proven constitution; characterization-test the first legacy seams*	ORG

The only metric that matters in week 3: did the pilot team voluntarily start their second feature with /specify.

TAKEAWAYS AND REFERENCES

If you remember five things from these fifty slides.

01 Vibe coding creates intent debt; specs are how intent survives.

02 SDD answers what and why; TDD proves does it. You need both loops.

03 Acceptance criteria are the bridge: every criterion becomes a test, written first.

04 Spec-Kit gives the standard skeleton; Specky adds discovery depth and gated traceability.

05 Adopt one feature at a time. The constitution and the gates do the scaling.

PRIMARY REFERENCES

`github.com/github/spec-kit`
Spec-Kit repository, Specify CLI, templates

`Specky · open source`
Specky repository, EARS workflow, dual runtime

Kent Beck, *Test-Driven Development: By Example*

Robert C. Martin, the three laws of TDD

EARS: Easy Approach to Requirements Syntax, Mavin et al.

GitHub blog: Spec-driven development with AI

THANK YOU

Specify first. Then let the agents build.

The teams winning with AI coding agents are not the ones with the best prompts. They are the ones whose intent is written down, testable, and versioned next to the code.

CONTACT

Paula Silva

Software Global Black Belt

paulasilva@microsoft.com

THIS DECK

SDD + TDD with Spec-Kit and Specky

v1.0.0 · Published 2026-06-10

NEXT STEP

Pick one feature this week

Run it through /specify, /plan, /tasks, /implement