kaizen SOFTWORKS

Greenfield projects *SDLC playbook*

How PMs and Engineering can build with AI agents at full speed, without slowing each other down.

Authors: Martin Mari and Martin Mato

EXECUTIVE SUMMARY

Agentic AI is no longer just an engineering tool. PMs build working software to validate hypotheses with real customers. Engineering teams use it to build and deploy at scale. Both work with agents, today, on the same team.

But they need opposite things from the same environment. The **PM needs to move fast**, break things, throw the work away and start again. **Engineering needs correctness, security**, and the guarantee that nothing breaks. It's the same dial, turned in opposite directions: tune the environment for the PM and engineering loses its guarantees; tune it for engineering and the PM is back to waiting on every iteration.

How do both work at full speed without stepping on each other?

One cycle for each, each in its own environment: exploration, where the PM validates what to build, and construction, where engineering builds it to last. A single artifact crosses between them: a spec with a test plan. The prototype stays; the spec travels. Exploration owns the what. Engineering owns the how.

This playbook is for tech leads designing how their team codes with agents, and product leaders already vibe-coding who want to propose a working model to engineering. We assume you've used agentic coding tools and know your own SDLC well enough to know where it hurts.

Agentic development gives each role of the product trio more autonomy

The PM can now validate a product hypothesis with working software, answering 'Does *this solve the problem?*', without pulling engineering into every exploration cycle.

The UX designer shapes what that software looks and feels like through a living design system, answering 'Does *it feel right?*', without waiting on implementation to see the result.

The engineer focuses on building correctly and at scale, answering 'Does *this work reliably?*', without being the bottleneck for every product decision upstream.

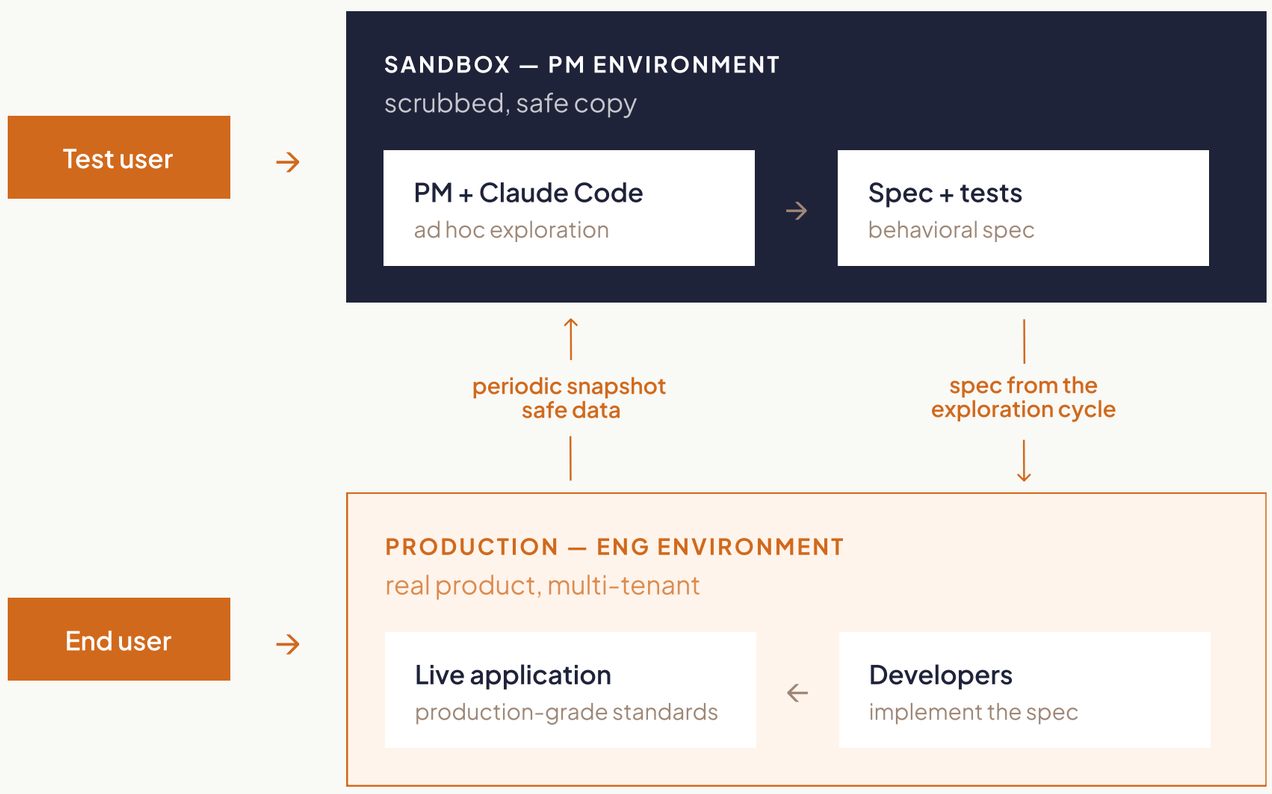
Each role moves faster. None does the other's job.

But **that capability introduces a structural tension**. When exploration and construction share the same environment, they compete. Exploration needs speed, freedom to break things, and zero friction. Construction needs correctness, security controls, and architectural guardrails.These goals don't naturally coexist.

One environment can't do both well. Tuned for exploration, it loses the guarantees construction needs. Tuned for construction, exploration crawls. Every iteration stuck waiting on engineering.

So how do you let each one run at full speed?

We propose a separate environment for each cycle.



02 THE TWO CYCLES

Exploration and construction need different conditions to succeed

	EXPLORATION	CONSTRUCTION
QUESTION	What should we build?	How do we build it correctly?
OWNED BY	PM + UX	Engineering
OPTIMIZED FOR	Speed, autonomy, zero friction	Correctness, security, maintainability
	Preview Environment: sanitized data, controlled blast radius	Production repo + CI/CD pipeline
OUTPUT	Spec + Test Plan	Deployed feature

The bridge between them is an explicit handoff artifact: a spec and test plan that carries intent without carrying implementation risk. Exploration owns the discovery. Construction owns the execution. The handoff is the contract between them. Neither stream waits on the other.

03 THE HANDOFF ARTIFACT

The handoff carries intent, not implementation risk.

The handoff artifact is a **spec and test plan**. It carries the intent of the exploration cycle into construction without carrying implementation risk or architectural assumptions. The spec has three parts:

01 **A behavioral description:** what the feature does from the user's perspective, including edge cases and failure modes.

02 **A test plan:** concrete acceptance criteria that define when the feature is done.

03 **A prototype reference:** a link to the working build in the Preview Environment, which engineering can use as a visual and functional reference.

What the spec leaves out: architectural decisions, implementation approaches, and the prototype code itself. Those belong to the construction cycle.

The spec owns the *what*. Engineering owns the *how*.

This is where ownership transfers. The PM closes the exploration cycle by writing the spec. Engineering opens the construction cycle by reading it.

04 CONWAY'S LAW FOR AGENTS

In the agentic era, boundaries are infrastructure.

We know from Conway's Law and Team Topologies that architecture mirrors communication structure. That principle now extends to agents. The boundaries you define shape what gets built: what context agents can see, what tools they can reach, what they output, which review gates they pass through.

The Preview Environment and the Production Repo are not just environments. They are **agent interfaces**. Designing those boundaries on purpose is the infrastructure work of the agentic era.

If agents see too little, they guess. If they reach the wrong tools, they create risk faster. If they work without constraints, they reproduce inconsistency.

05 SPRINT 0

Build the system that lets agents build well.

> KEY INSIGHT
>
> The best prompt is the code you already have. An agent working on a well-structured, well-documented codebase will consistently produce better output than one starting from a bad boilerplate or an empty repo.

Two things are worth the human investment in Sprint 0.

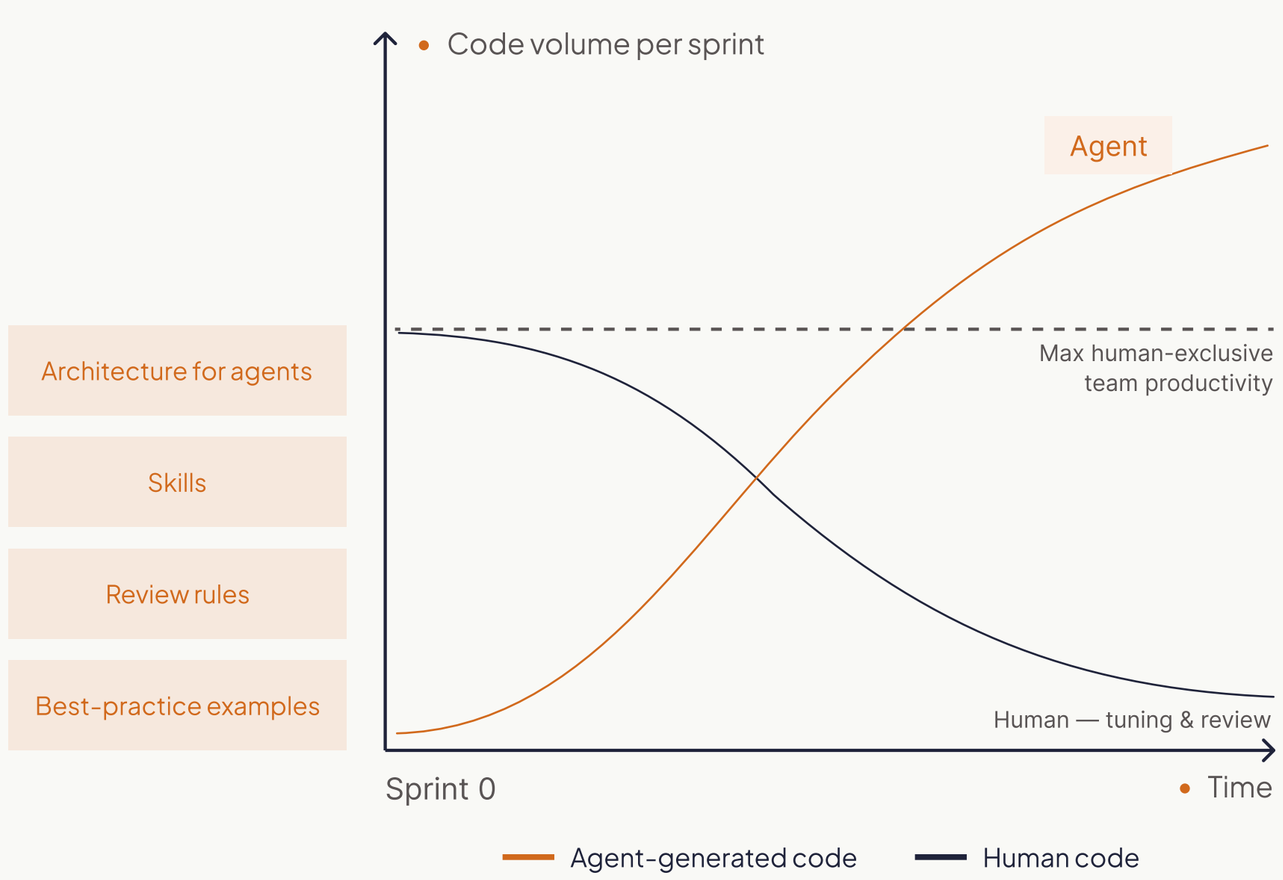
- **Enabling constraints. What guides the agent before it acts:**
  - Architecture decisions, captured as lightweight ADRs.
  - An AGENTS.md: the tool-agnostic contract every coding agent reads before touching the repo (CLAUDE.md if using Claude Code).
  - Skills and prompt templates for recurring workflows.
  - An /examples folder with reference implementations that set the quality bar.
- **The verification layer. What catches problems after the agent acts:**
  - Review rules (architecture, security, token discipline).
  - Automated tests and lint gates.
  - A sign-off process before anything reaches production.

The verification layer is not just quality control. It is how a leader earns the right to extend the agents more autonomy, which is why **building it is not a task you delegate**. Once the safety net catches bad output more reliably than manual review does, engineers can accelerate with confidence.

06 HUMAN-AGENT INTERVENTION IN THE CONSTRUCTION CYCLE

Human effort moves from repetitive building to system tuning.

The traditional model has human code growing linearly: every new feature costs the same human effort. The agentic model inverts this. High human investment upfront in Sprint 0, then the agent takes on progressively more of the implementation work while humans shift to architecture, verification, and system tuning.



07 THE PREVIEW ENVIRONMENT

Product exploration moves fast, but doesn't break things.

The Preview Environment is the operational home of the exploration cycle. It's where PMs branch freely, build working software, and share it externally for customer validation, without touching the production codebase or production data. Three structural properties define what it needs to be.

01 **Sanitized seed.** The environment runs on a periodic, deterministic snapshot of production data, scrubbed of sensitive information. Not symbolic data, not mocks, a real-enough copy to make exploration meaningful. When a PM shares a preview link with a real customer, that customer sees nothing they shouldn't see.

02 **Privileged operations blocked.** There is no direct connection to the production database. This is an infrastructure constraint, not a policy. PMs can build, break, and iterate at speed because the environment is designed to make that safe, not because everyone is being careful.

03 **Contained blast radius.** Changes in the Preview Environment stay there. Sharing a preview link with a stakeholder or customer is deliberate and safe. Propagation to production is not possible from here.

UX shapes the design system that feeds exploration.

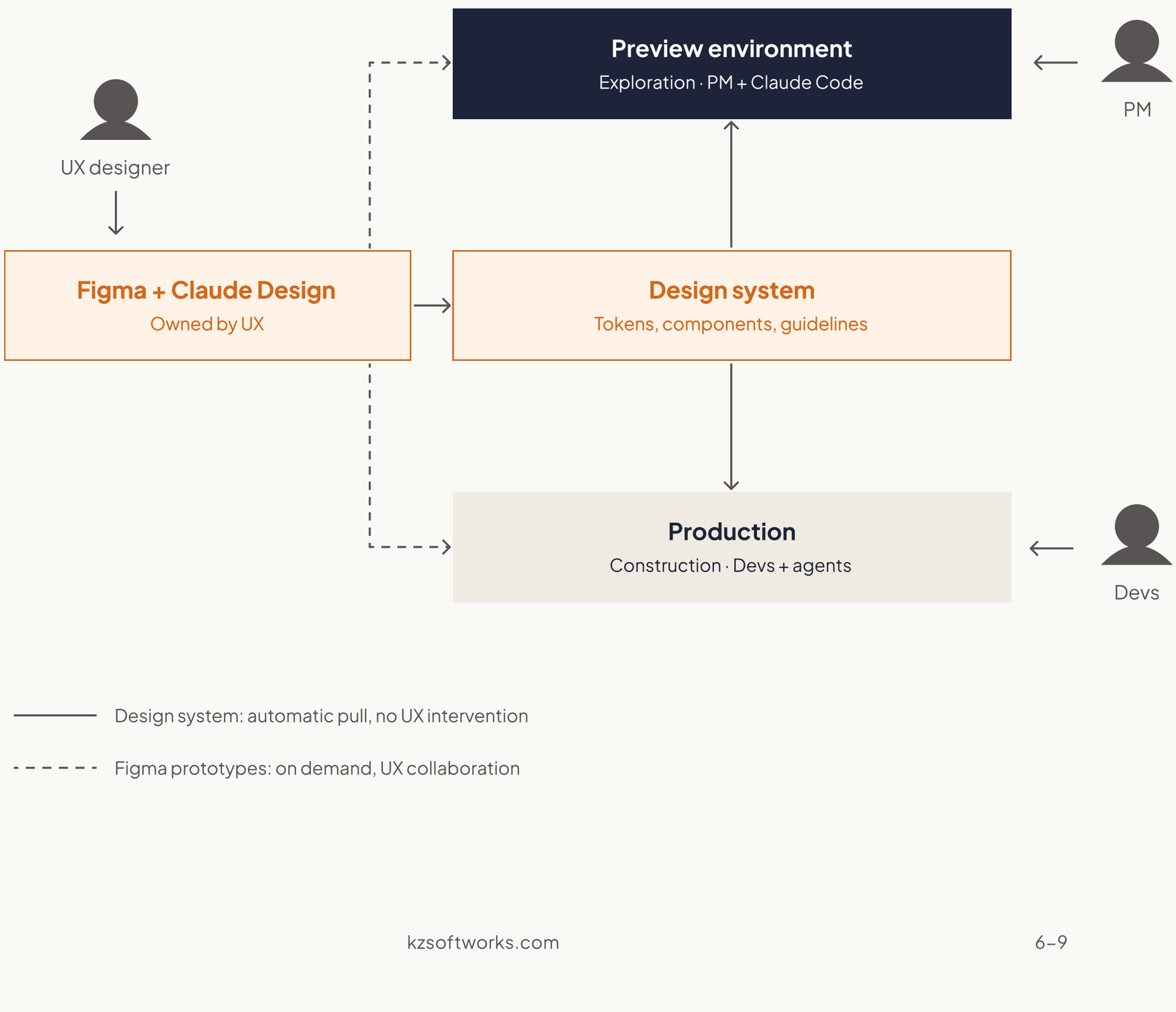
The UX workflow is designed around a single principle: the designer is never the bottleneck between the PM and a working prototype. Designer input is critical for both humans and agents. Key tools include Figma and Claude Design.

The design system, maintained by UX, feeds the Preview Environment directly. Agents operating in the environment pull design tokens and component conventions automatically. PMs work within the established visual language without managing it themselves. This way, agents can pick up most of the guidance, so PMs don’t have to handle branding and common system patterns on every iteration.

When needed, designers can also provide full-screen prototypes and collaborate with PMs during ideation. Once ready, they can deliver a high-fidelity prototype so PMs can build the feature using the design as direct input.
If designer input isn’t needed, PMs can develop using Claude Code, and the design system will be pulled automatically.

When PMs propose UI changes during exploration, designers can sync those changes from the codebase back into the design system using the Claude Design agent.

The right level of design involvement scales to what each feature needs. The goal is continuous flow for both streams.



UX DESIGN IS NOT A BOTTLENECK BETWEEN PM AND FUNCTIONAL PROTOTYPE.

2 shared workspaces:
· Figma + Claude Design, owned by UX designers.
· Exploration environment, owned by PMs.

09 THE CONSTRUCTION CYCLE

In construction, the spec is the source of truth.

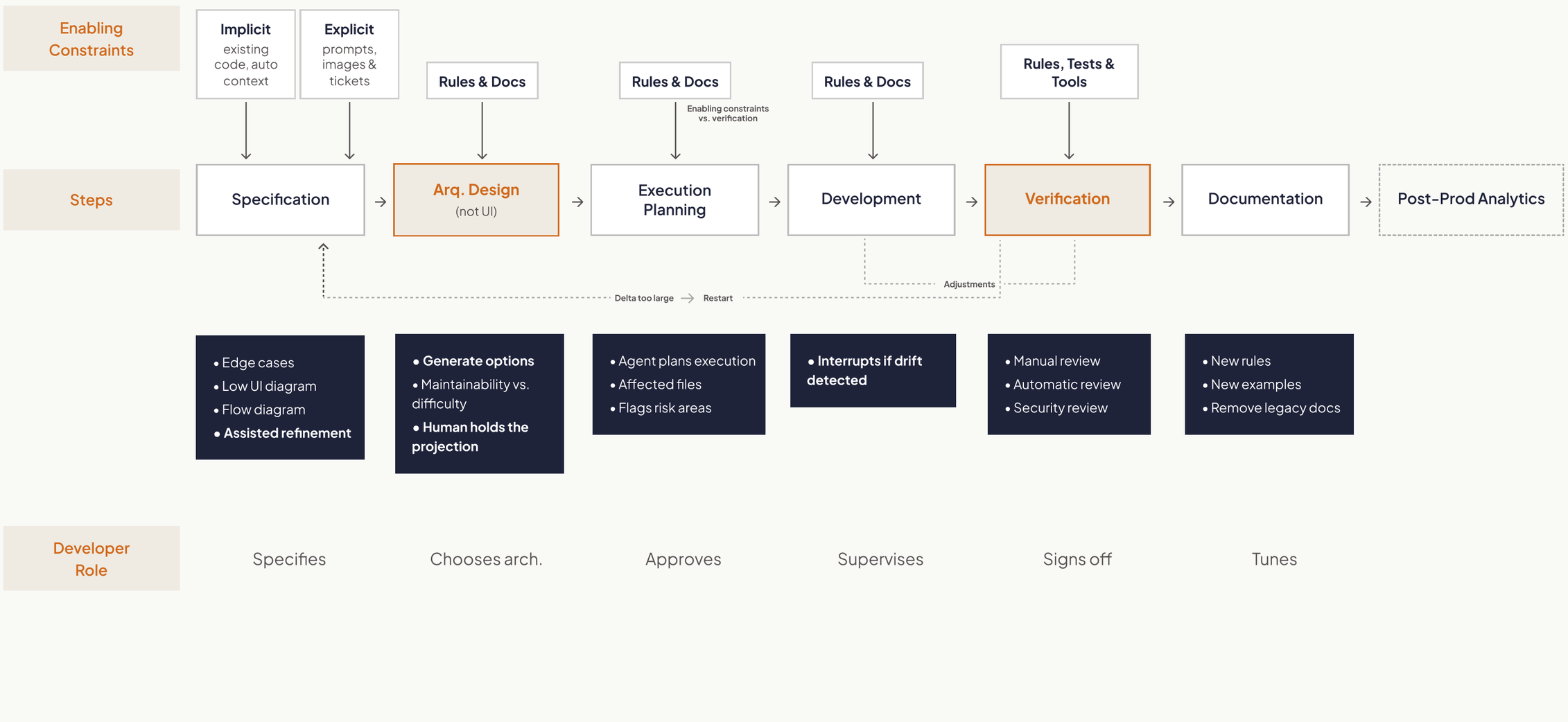
Once the spec and test plan arrive from exploration, the construction cycle follows a defined sequence. It's spec-driven development: the spec is the source of truth. The agent builds toward it. The developer makes sure nothing diverges from it. At each step, enabling constraints guide the agent, and the developer's role shifts from builder to decision-maker and supervisor.

Two principles govern this cycle.

Constraints and verification are complementary, not redundant. Constraints guide before. Verification catches after. Both are necessary.

Drift interruption. If the agent's output diverges from the spec or the architecture, the developer stops the cycle and resets, rather than patching over it. The spec is the source of truth at every moment.

The construction cycle is architecture-agnostic. The spec describes behavior, not implementation. Whether the team builds with a monorepo, microfrontends, serverless, or any other pattern, the cycle is the same: a spec arrives, a deployed feature leaves. Engineering keeps full ownership of those architectural decisions.



10 WHEN TO APPLY THIS MODEL

Greenfield is where the model compounds fastest.

This model was designed for greenfield agentic projects. That's where its return is highest and its trade-offs are most manageable. On a greenfield project, **Sprint 0 is not remediation. It's design**. The team sets the architecture, conventions, and verification layer from day one, before the agent has seen a single line of inconsistent code. The codebase becomes the context the team intended, not the context that accumulated. The ramp-up curve is steepest, and the ceiling is highest.

On a **brownfield project, the dynamics are different.** The existing codebase is already the context, and that context carries the decisions, inconsistencies, and patterns of years of development. Sprint 0 is harder to run, because you're retrofitting constraints onto a system that was never built to receive them. The gains are real, but more marginal and slower to compound.

Every sprint that runs without a well-structured context for agents, a working verification layer, and clear architectural decisions is a sprint that trains the agent on context you didn't intend to provide.

Agent output can exceed the human ceiling. That's the net productivity gain. But the agent is only as good as the context it works in, which is why Sprint 0 is the most important investment in any agentic project.

11 TAKEAWAY

Agentic development is a new way of working.

Adopting AI in software development is not about choosing the right coding assistant. The deeper question is how the team reorganizes work around new capabilities. The teams that get the most out of AI will be the ones that separate exploration from production, define explicit hand-offs, preserve engineering ownership, and treat agent boundaries as part of the architecture.

AI can accelerate software development, but only when you redesign the system around it. Do that, and the gains don't show up just once. They *compound*, sprint after sprint.

ABOUT THE AUTHORS



Martin Mari

IT Innovation & Organizational Design
at Kaizen Softworks

Claude Certified Architect



Martin Mato

Principal Software Engineer
at Kaizen Softworks

ABOUT KAIZEN SOFTWORKS

We help teams build for the agentic era, *responsibly*.

12 YEARS
IN BUSINESS

Building software that lasts beyond the first release.

7 YEARS
AVG. PARTNERSHIP LENGTH

Clients stay because the work keeps up with their growth.

5 YEARS
AVG. TEAM TENURE

The people who learned your business watch it grow.

Start with a conversation. Leave with clarity.

AI is rewriting what software can do. We help you turn it into smarter products, faster workflows, and teams that keep getting better.

Start a conversation

(https://www.kzsoftworks.com/get-in-touch)

kaizen SOFTWORKS

kzsoftworks.com · hello@kzsoftworks.com

© 2026 Kaizen Softworks