API Design

Lessons on Usability, Agility, and Pragmatism

An exploration of practical API design principles through the lens of *Software Engineering Radio*'s interview with **Wesley Beary**, Heroku veteran and Ruby ecosystem contributor.

SE Radio 679: Wesley Beary on API Design

- Marcos Rodríguez
- Lucas Uña
- Adrián Sánchez
- Joona Pikkarainen

What Is (Really) an API?

The Broad Reality

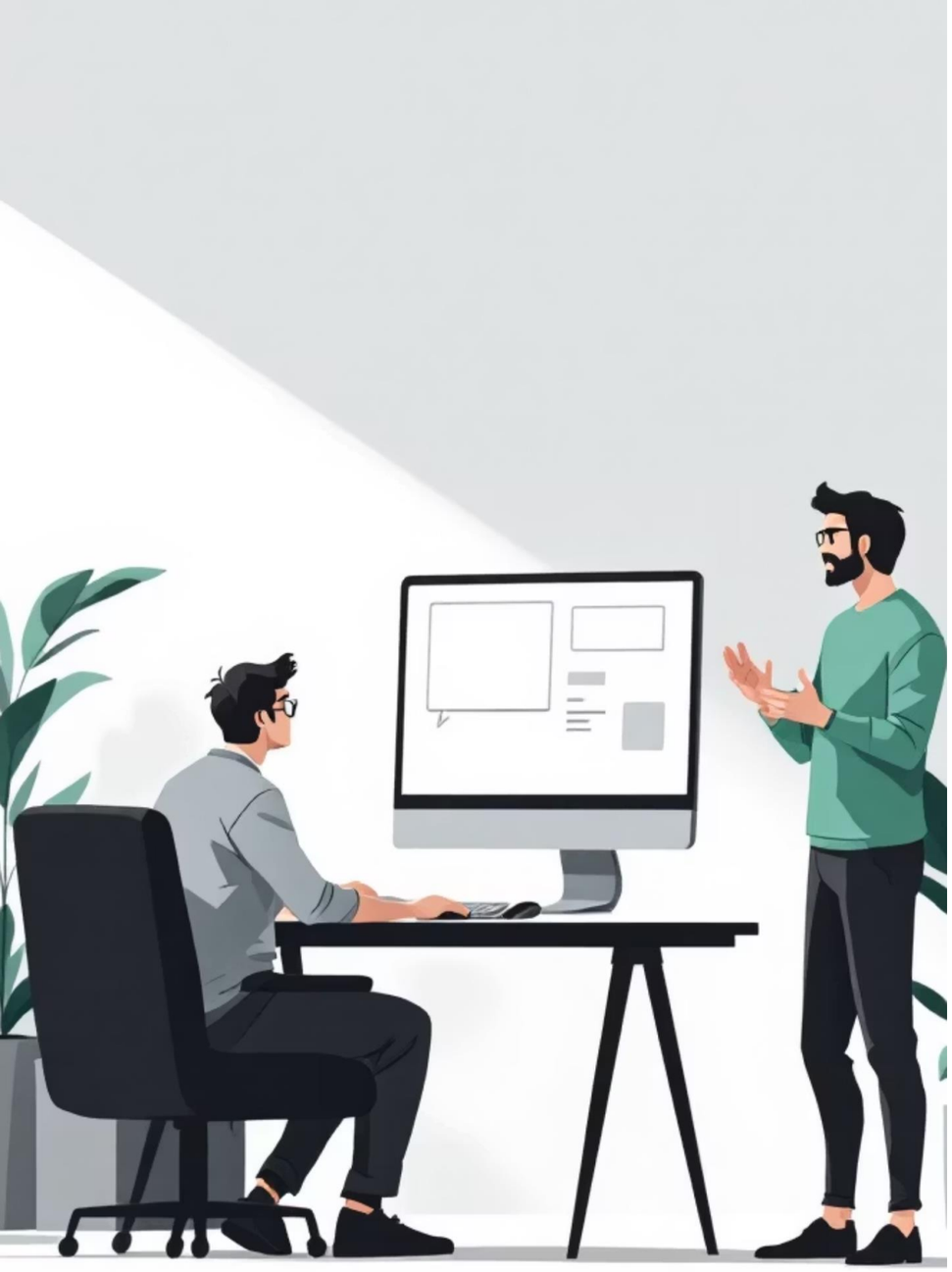
APIs aren't just HTTP endpoints. They live in software libraries, command-line interfaces, and anywhere we expose functionality to external users.

At its core: The interface that lets users tell your system what work they want done.

The Real Meaning

> **A.P.I.** = *Assumptions Probably Incorrect*

This humble acknowledgment drives everything about pragmatic API design.



Design Doesn't Survive Contact with Users

The Abstract Design Problem

Trying to solve everything in your head before coding usually misses drastic edge cases and real-world usage patterns.

Rapid Prototyping

Build lightweight interfaces and "use them with rage"—test them as a real, frustrated user would experience them.

Blind Playtesting

Give documentation to users, watch them struggle in silence, and resist answering questions to avoid biasing the test.

The Danger of "Pure CRUD"

The Overloaded UPDATE Trap

Trying to make one update method do everything based on "magic parameters" creates incomprehensible code that frustrates users.

The Action Pattern Solution

Separate reading and writing state from executing processes. Don't just update status to "restarting"—call a restart action.

Proposed namespace:

`/servers/:id/actions/restart`

The Principle of Least Surprise

Knowledge Transfer

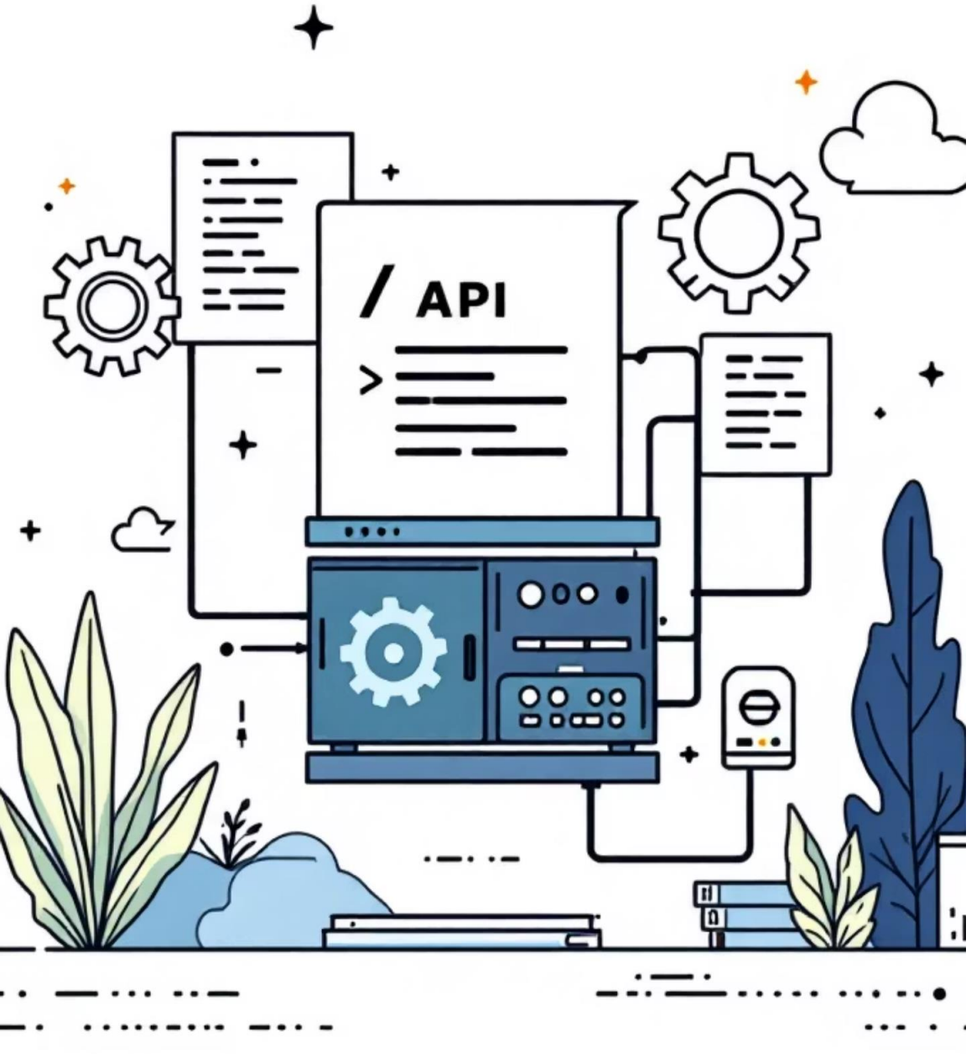
If a user learns one corner of your API, they should instinctively guess the rest without documentation.

Inconsistency Friction

Why does Git use `delete` for some things and `remove` for others? Users wonder if there's a hidden architectural difference.

Descriptive Errors

Don't just say "File not found." Say *which* file is missing and why—it matters for debugging.



OpenAPI and Living Documentation

Visual Generation

Automatic HTML documentation for humans that stays in sync with your implementation.

Contract Testing

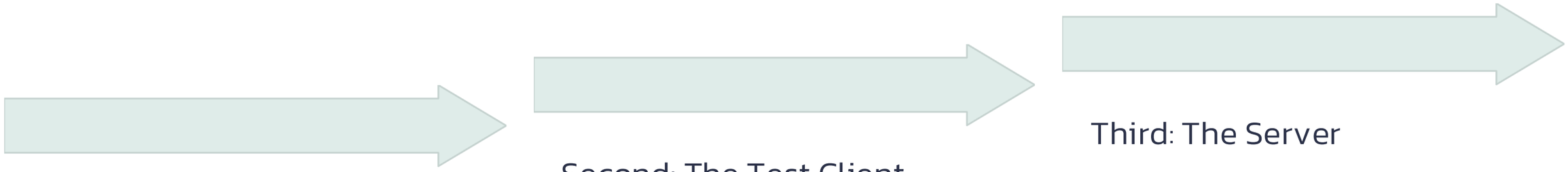
Tests fail automatically if the server response doesn't match the design specification.

Mock Servers

Spin up a fake server instantly to start coding the frontend immediately.

The Ultimate API Workflow

What's the cheapest and fastest way to avoid costly mistakes?



First: The Spec (OpenAPI)

It's plain text—the cheapest thing to change or throw away. Define the contract before implementation.

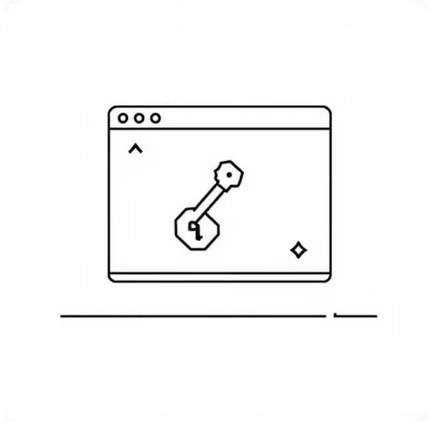
Second: The Test Client (CLI/SDK)

It immediately reveals fields you forgot to include and catches design flaws early.

Third: The Server

Only start backend implementation once the client contract is stable and validated.

Learning from Masters



Git

Analyse command naming inconsistencies to understand how terminology affects learnability.

GitHub CLI

Study modern tool design patterns that prioritise user intuition over technical purity.

OpenSSL

Learn from interfaces that frustrate users—understanding pain points prevents repeating mistakes.

Key Takeaways

Assumptions Are Usually Wrong

Design doesn't survive contact with real users—prototype and test early.

Separate Actions from State

Use explicit action endpoints instead of overloaded CRUD operations.

Consistency Matters

The Principle of Least Surprise reduces cognitive load and accelerates learning.

Automate Documentation

OpenAPI keeps docs, tests, and mocks in sync automatically.

The Human Factor

> *"Successful APIs aren't necessarily the most technically complex, but those that respect the human developer's time, intuition, and cognitive load."*

Great API design isn't about theoretical perfection—it's about **pragmatic usability** that makes developers feel smart rather than frustrated.

¿Preguntas?

Gracias por su atención.