01 Shipping AI Features That Don’t Break in Production

Shipping AI Features

That Don’t Break in Production

Lessons from building Android apps
with AI in enterprise and indie environments.

Andrii Veremiienko
Android Developer

02 Two Worlds of Delivery

Two Worlds of Delivery

Over the last 6 months I had the chance to work in two very different environments:

Enterprise
Revolut

A large fintech product used by millions. Complex architecture, strict standards, high reliability bar.

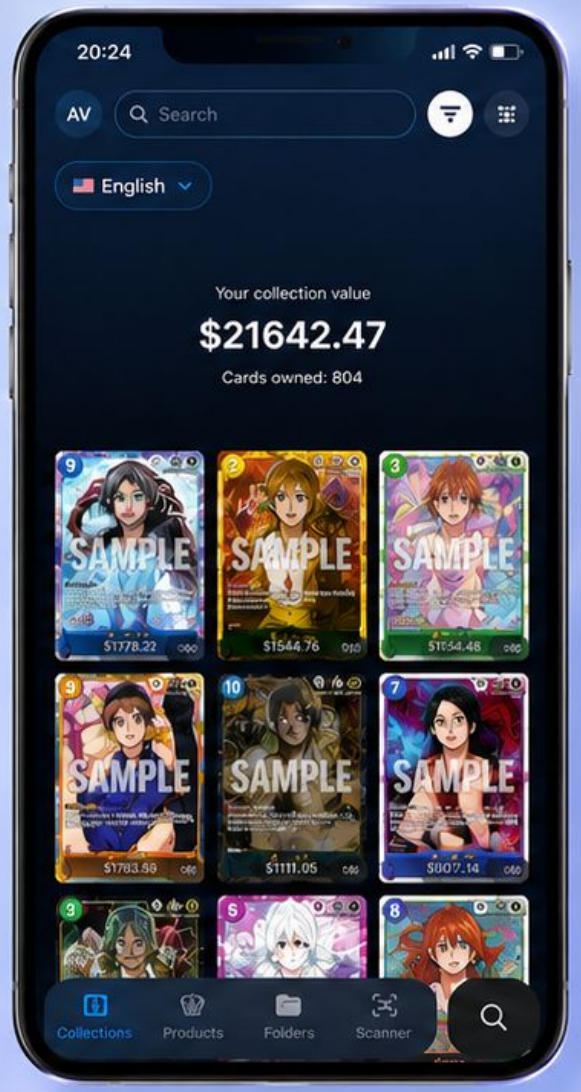
Indie
Logia & Padel Coach

My own apps built from scratch. Fast iterations, small team (aka me 😅), freedom to experiment.

Same goal everywhere:
Ship the best possible product in less time.

Revolut
Enterprise Fintech

Logia
Indie Collection App



What AI Actually Changed

AI changed the economics of development

6 months → 2 months

Speed
Ideas ship faster with AI-powered development.

UI
AI accelerates UI iteration and polish.

Testing
AI generates tests and catches edge cases early.

Review
AI helps reviewers focus on what matters most.

The First Mistake: Giving AI Too Much Freedom

> *The more experience I gained, the less freedom I gave the model.*

Build it
Ship something that works

Use MVVM
Structure your codebase

Reuse components
Build once, use everywhere

Follow architecture
Keep the big picture in place

Explain decisions
Make the why as clear as the how

05 Enterprise Problems

Enterprise Problems

The hidden costs of how software gets built at scale.

> **Correct code is not always acceptable code.**

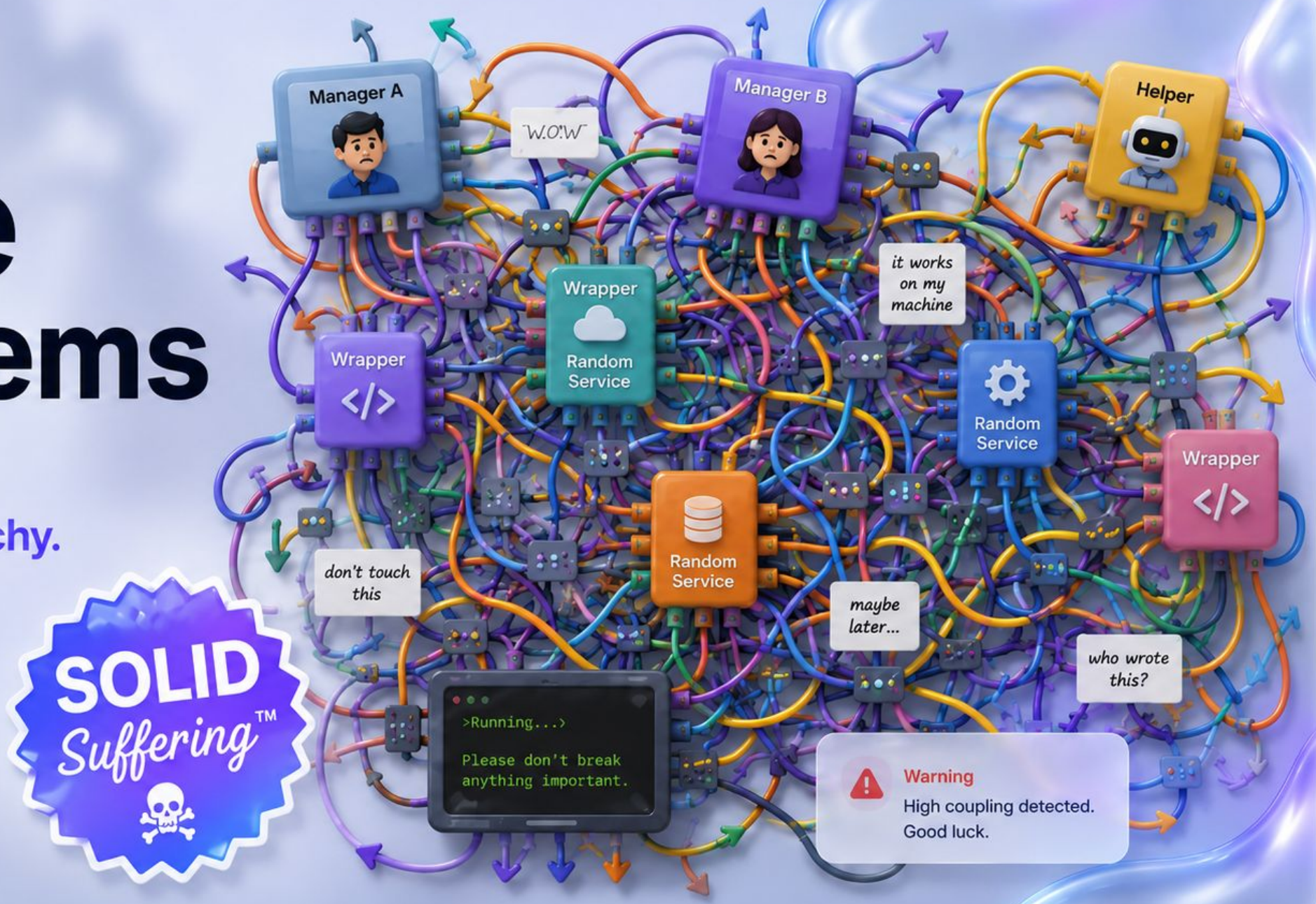
06 Indie Problems

Indie Problems

No architecture means code anarchy.

Problem

When everything talks to everything, nothing works the way you expect.

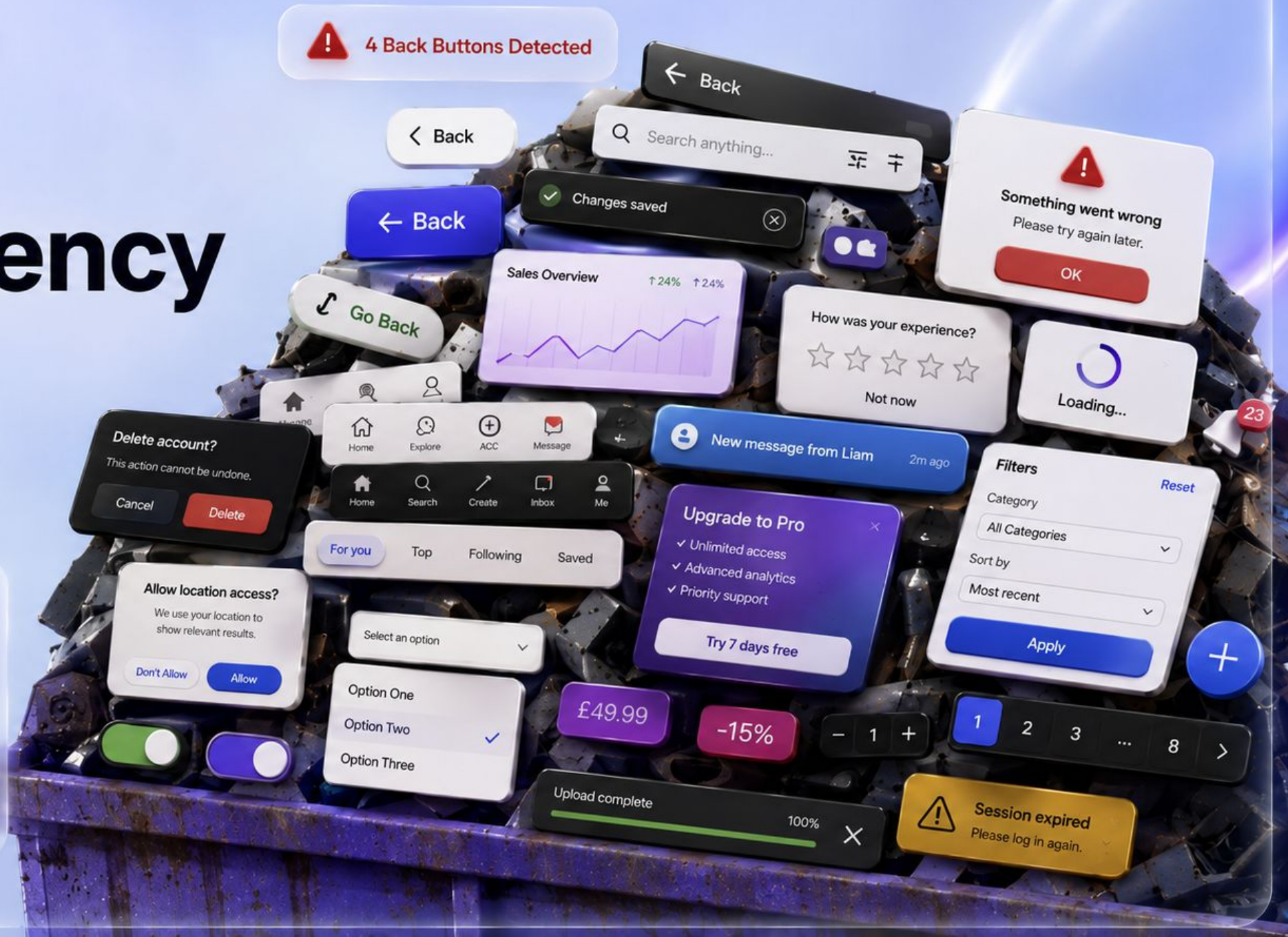


07 The UI Consistency Trap

The UI Consistency Trap

When every team builds their own UI, the product becomes a patchwork of patterns.

> **Every screen looked good.**
> **Together they looked random.**



08 The Hidden Problem: Understanding

The Hidden Problem: Understanding

> “Ownership degrades faster than code quality.”

Code works → You don’t fully understand it → Debugging becomes expensive

Andrii Veremiienko
Android Developer

09 Solution #1

Solution #1:

Build Context Before Implementation

"Don't ask AI to code.
Ask AI to understand."

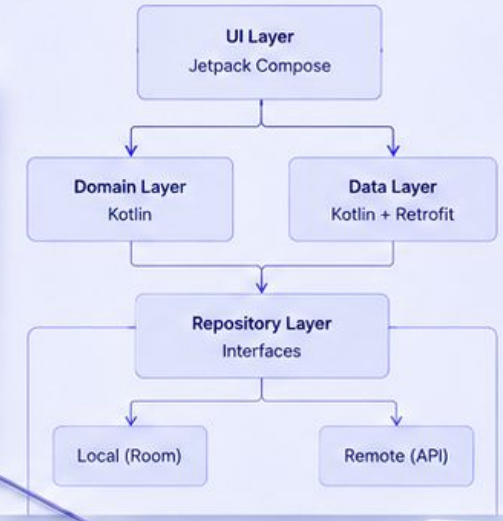
Why?
Understand the system, not just generate code.

Similar Code Found

- OrderRepository.kt — 87% match
- SyncManager.kt — 82% match
- UserDataSource.kt — 78% match

+12 more results

Architecture Overview



Pattern Summary

- Repository pattern with single source of truth
- Flow for async data streams
- Result wrapper for errors
- DI via Hilt modules

Confidence: 92%

Example: OrderRepository.kt

Kotlin

```
class OrderRepository(private constructor(
  private val remote: OrderApi,
  private val local: OrderDao
):
  fun getOrders(): Flow<Result<List<Order>>> =
    flow {
      val localData = local.getOrders()
      emit(Result.Success(localData))
      try {
        val remoteData = remote.fetchOrders()
        local.insertAll(remoteData)
        emit(Result.Success(remoteData))
      } catch (e: Exception) {
        emit(Result.Error(e))
      }
    }
}
```

Notes

- Orders are cached locally and synced in background
- API can fail – show stale data with indicator
- Paging used for large lists
- Auth required for requests

Last updated: Today

Focus on purpose, relationships and rules.

Implementation Plan

1. Add new field to Order model
2. Update local + remote data sources
3. Update repository + mapping
4. Add UI state + loading indicator
5. Add tests for happy + error path

Small steps.
Clear intent.
Easy to review.

01 **Search similar code** — Find patterns and examples across your codebase.

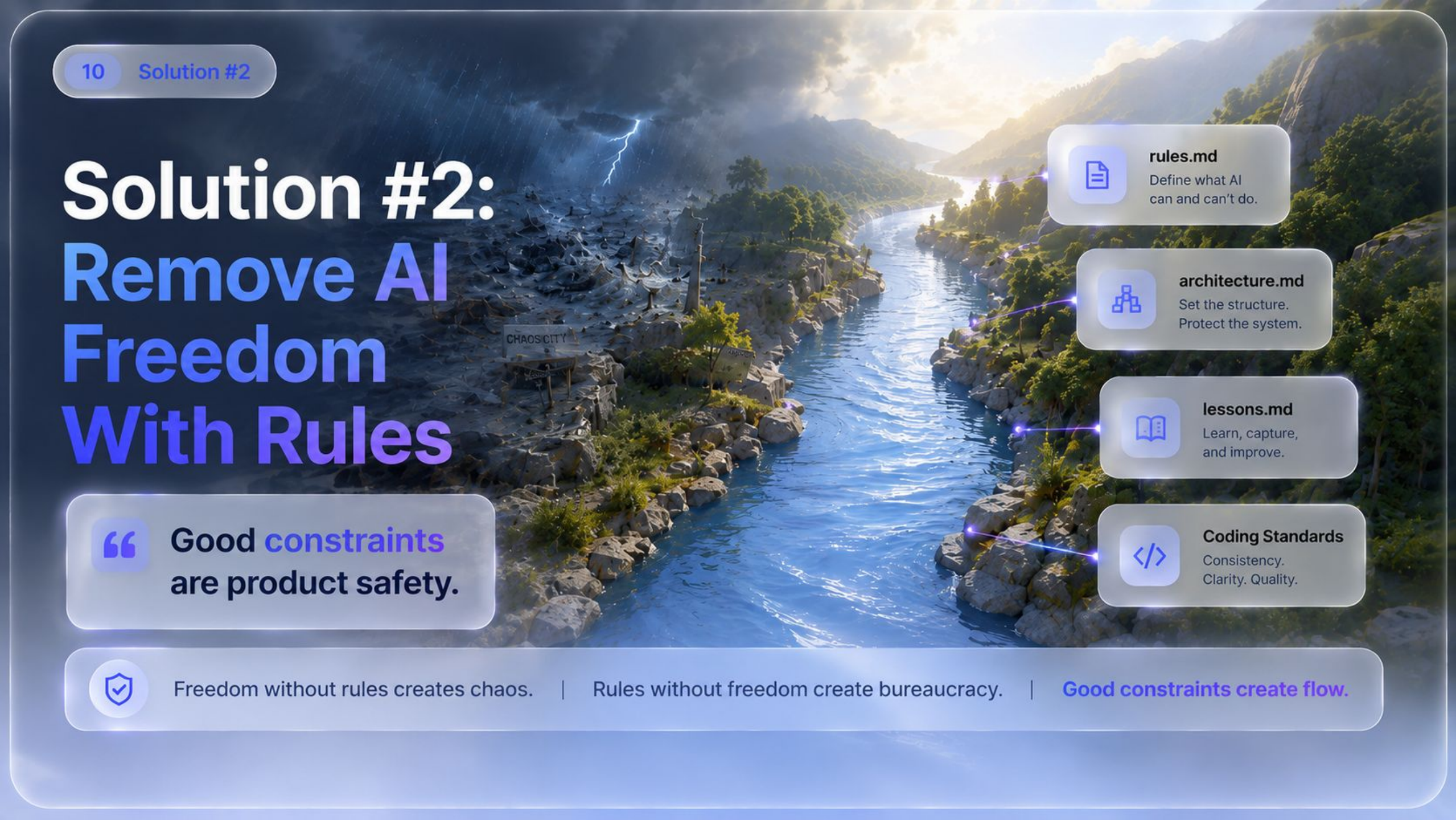
02 **Summarize pattern** — AI summarizes how it works, decisions, and best practices.

03 **Write notes** — Capture key rules, constraints, and edge cases.

04 **Create plan** — Break the task down into clear, testable steps.

05 **Implement small diff** — Make minimal changes with confidence and easy rollback.

Solution #2: Remove AI Freedom With Rules



rules.md
Define what AI can and can't do.

architecture.md
Set the structure. Protect the system.

lessons.md
Learn, capture, and improve.

Coding Standards
Consistency. Clarity. Quality.

> Good constraints are product safety.

Freedom without rules creates chaos. | Rules without freedom create bureaucracy. | **Good constraints create flow.**

Solution #3: Make Tests Useful

Useful, not decorative.

Good AI Tests

Protect behavior and outcomes.

Behavior

Validate what the feature does, not how it's implemented.

Edge Cases

Cover unusual inputs, failures, and boundaries.

Regression

Prevent old bugs from returning as you iterate.

VS

Decorative Tests

Look good, add little value.

Implementation Noise

Tests tied to internal details that change constantly.

Weak Assertions

Vague checks that always pass but catch nothing real.

Over-Mocking

Mocks everywhere, testing mocks—not behavior.

Solution #4: Build Core UI Before Features

If design decisions are already made, AI stops inventing new ones.

Design first. Iterate faster.

A strong design system gives AI the structure to build consistently.

What AI Still Cannot Do For You

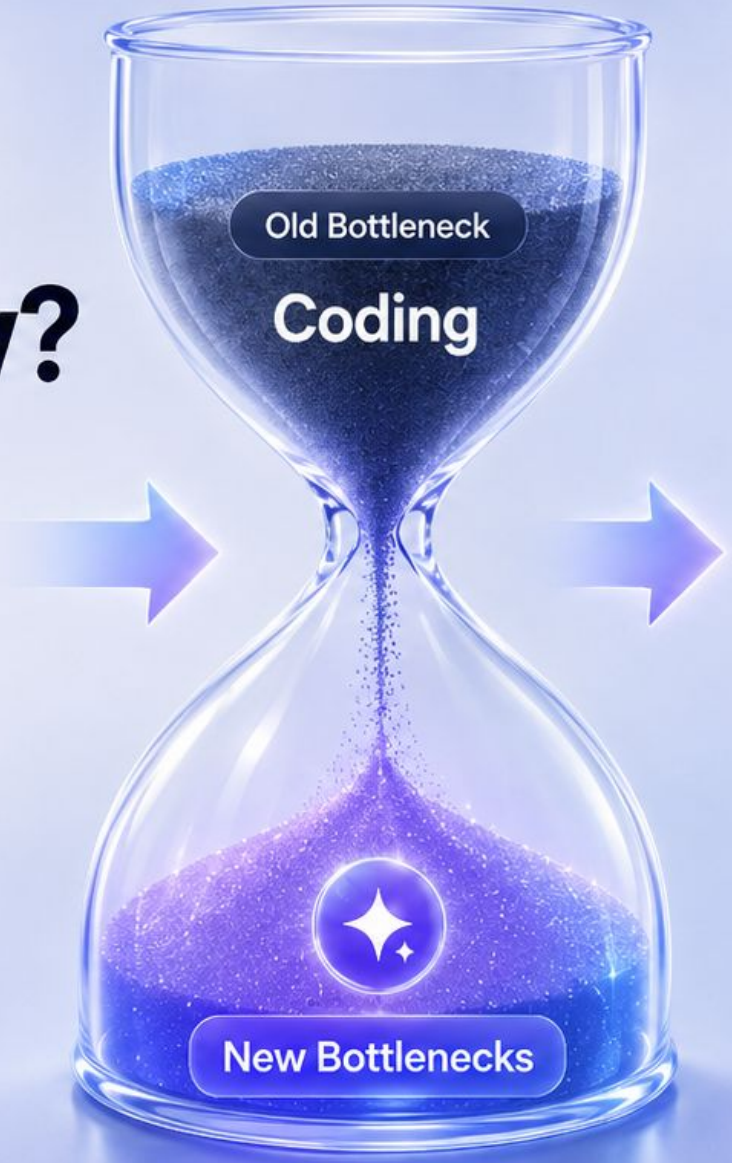
AI is a powerful enabler —
but great products are still shaped
by human judgment, intuition,
and context.

> **AI can implement.
> It cannot own
> product vision.**

What Do We Have Now?

The bottleneck has moved.
AI accelerates execution—
but new constraints emerge.

> **AI is a multiplier, not a replacement.**



Understanding
Deep domain insight and problem framing.

Architecture
Designing scalable, secure, maintainable systems.

Validation
Ensuring quality, safety, and real-world reliability.

Ownership
Accountability, decisions, and long-term impact.

> The more experience I gained,
> the less freedom I gave the model.

> AI made me responsible
> for more code than ever before.

Andrii Veremiienko
Android Developer