4-PAGE TACTICAL GUIDE

The Anti-PMO Playbook: How to Move Software from Idea to Delivery Using Everyday Language

A tactical guide for flat tech teams, fast-growing startups, and agencies who want to ship code without corporate red tape.

THE MANIFESTO

When did building great software require a dictionary of project management acronyms? Somewhere along the way, we let complex ticketing systems, steering committees, and endless alignment meetings stand between a business need and a finished line of code. If your organization doesn't have a massive PMO to police your backlog, your software intake quickly turns into total chaos. This playbook gives you the exact framework to structure your requests using plain text, keeping your backlog clean and your developers focused entirely on building.

```
idea -> everyday language -> structured scope
no acronyms, no steering deck, no translator
intake stays off the sprint board until it is ready
```

Rule #1: Separate the Chaos from the Backlog

Letting non-technical managers drop unvalidated ideas directly onto an engineering delivery board kills developer velocity. Keep a distinct Intake Layer completely separate from the Sprint Backlog. Chaos belongs in intake. The board only receives work that already survived a yes.

X THE BROKEN PATH

- **Email / Slack**
- v
- **Raw Jira ticket**
- v
- **Confused developers**
- v
- **Wasted engineering hours**

OK THE INTAKE LAYER

- **Plain English input**
- v
- **Automated guided validation**
- v
- **Structured technical scope**
- v
- **Clean developer execution**

The 4 Questions That Eliminate Bad Feature Requests

Give your team an immediate, copy-and-paste tool. Replace a generic text box with four precise, everyday questions that extract clear requirements without forcing anyone to use project jargon.

01

The Problem

What is currently broken or frustrating for the user? (Do not suggest a software solution yet).

02

The Audience

Who exactly is experiencing this issue, and how often?

03

The Target Value

What does success look like once this issue is solved?

04

The Constraints

Are there any strict legal, technical, or timing boundaries we must avoid hitting?

Symmetric Sizing: Sizing Both Sides of the Work

Traditional engineering estimates always fall short: they completely ignore the business rollout effort. Size a software change by looking at both the code complexity and the process training / operational updates side-by-side. A yes that only funds the build is a calculated delay.

```
code_side     = build + hookups + data move + go-live
business_side = training + rollout + support + notices
size          = (code_side, business_side)  // both required
```