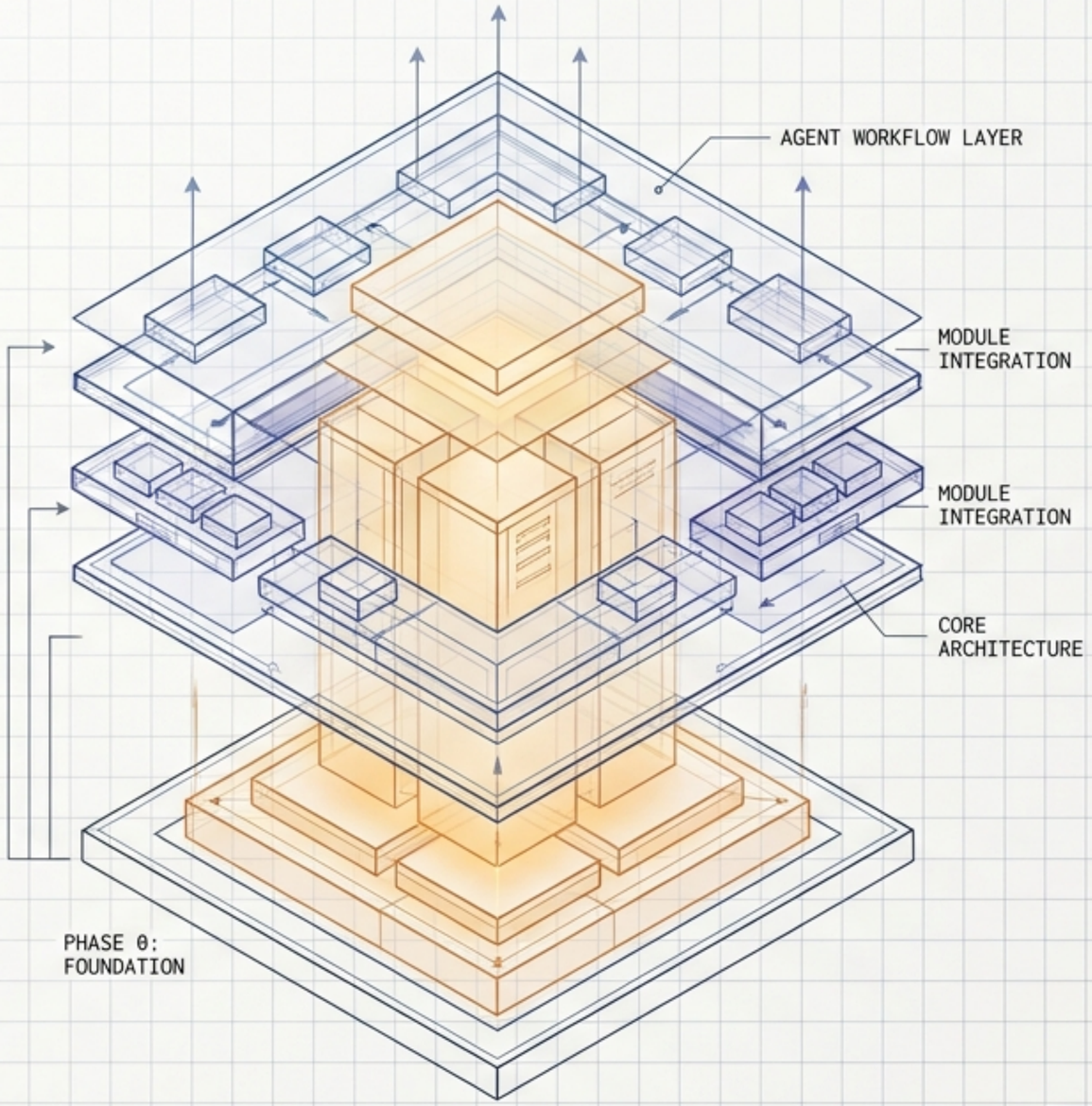
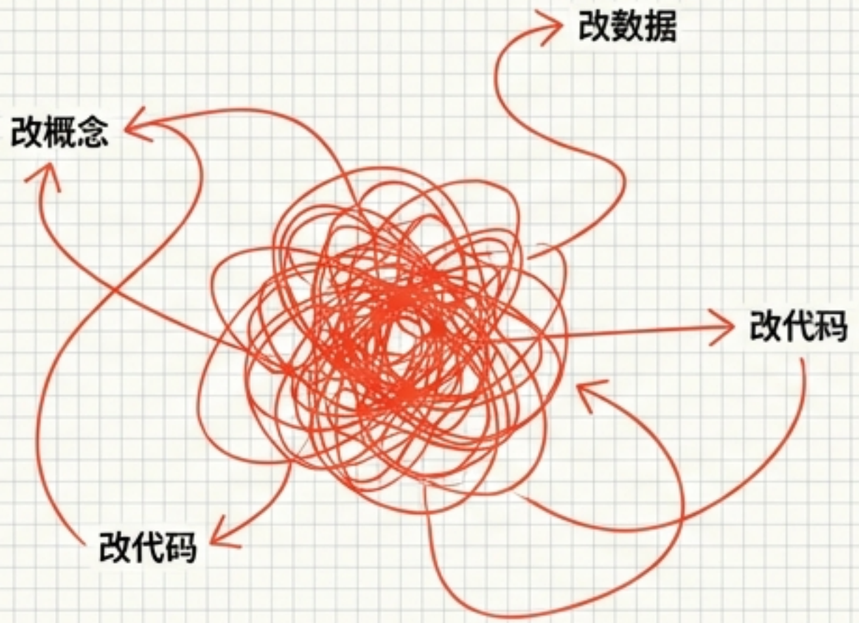
代码之前的一切

为 Claude Code 注入系统级智能体工作流



AI DEVELOPMENT PROCESS: THE VIBE CODING TRAP VS. STRUCTURED ENGINEERING

THE TRAP: "VIBE CODING" CHAOS

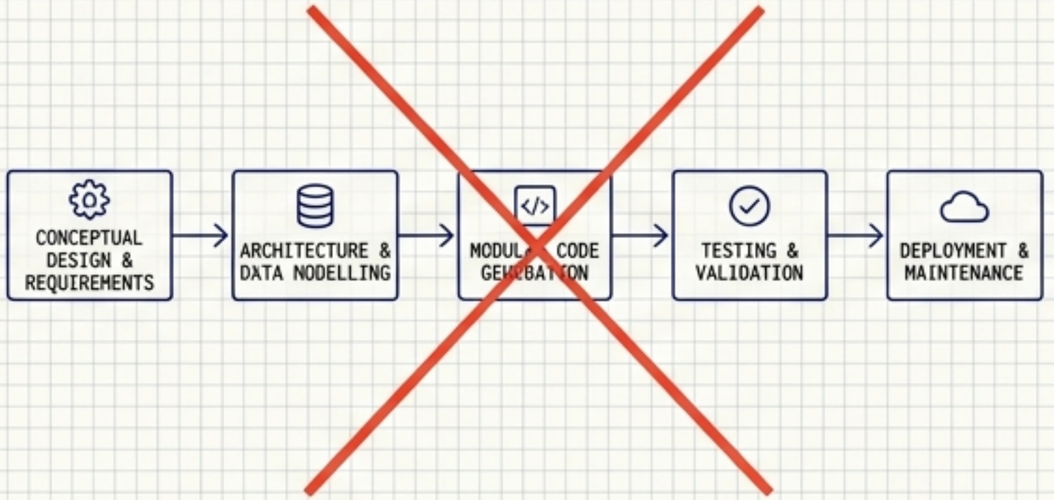


“Vibe Coding”的开发泥潭

基于模糊想法直接让 AI 写代码，最终必然陷入无尽的推翻重来。

- 想法随时变动
- 上下文逐渐碎裂
- 局部修补导致全局崩塌

THE LOST IDEAL: STRUCTURED ENGINEERING

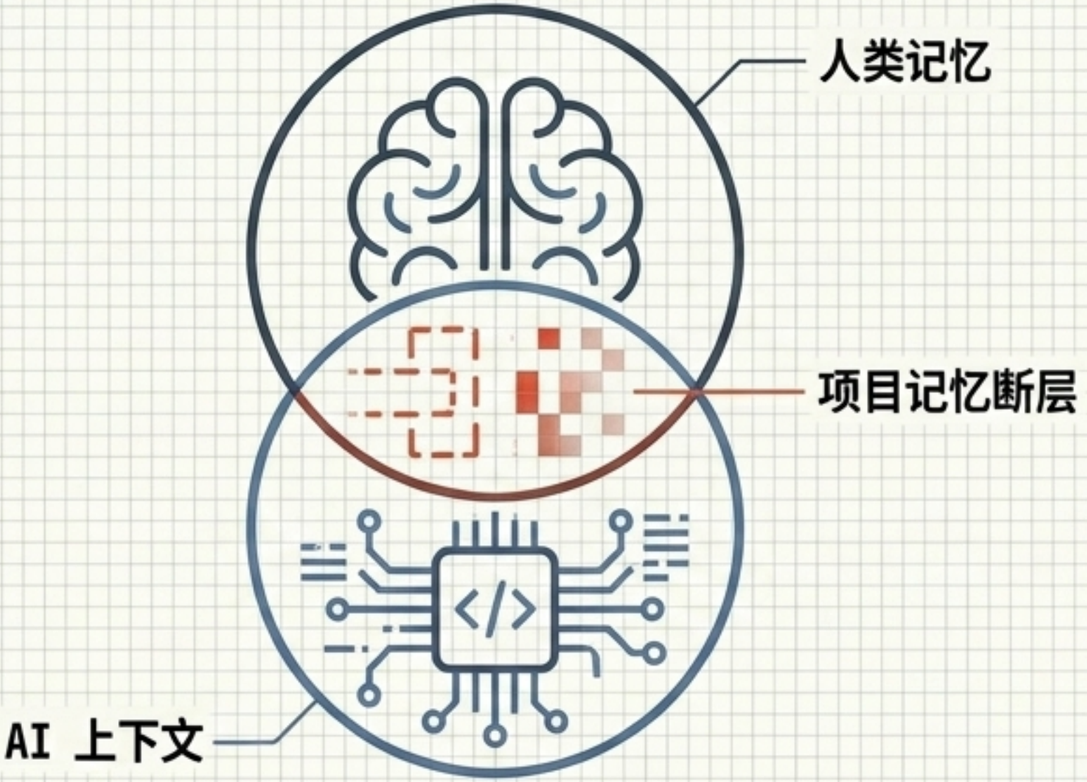


LOST STRUCTURE

Without a solid engineering foundation, the organized path is inaccessible.

- Clear Objectives
- Stable Context
- Modular Integrity

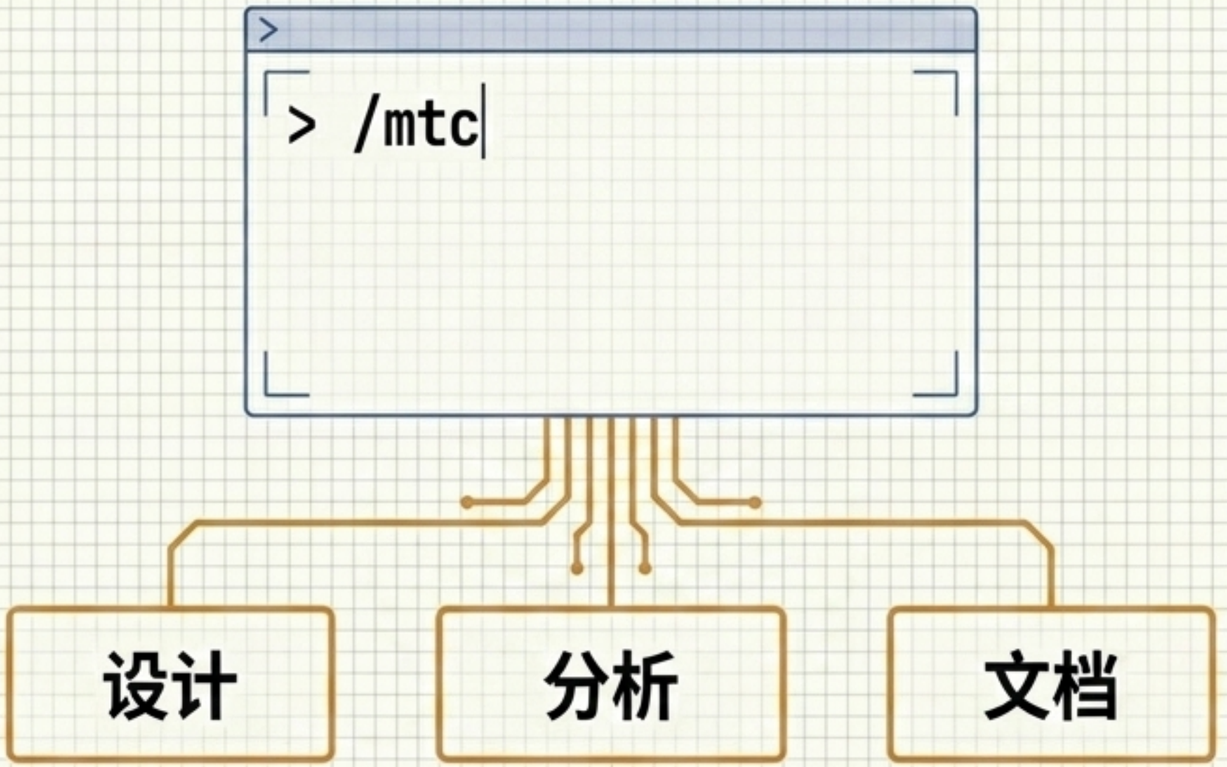
PROJECT: AI-DEV-FLOW | DRAWN BY: VIBE_CODING_ANALYSIS
REV: A.01 | DATE: OCT 26, 2024 | SCALE: NTS



“不是因为 AI 忘性大，是因为你忘性大。”

三天后回顾代码时，机制与概念脱节——你无法分辨是最初概念没想清楚，还是 AI 在实现时发生了偏移。

PSOJECT: AI-DEYORY-GAP | BANN 87: VIBE_CODING_ANALYSIS
REV: A.01 | OATE: OCT 26, 2024 | SEALE: RTS



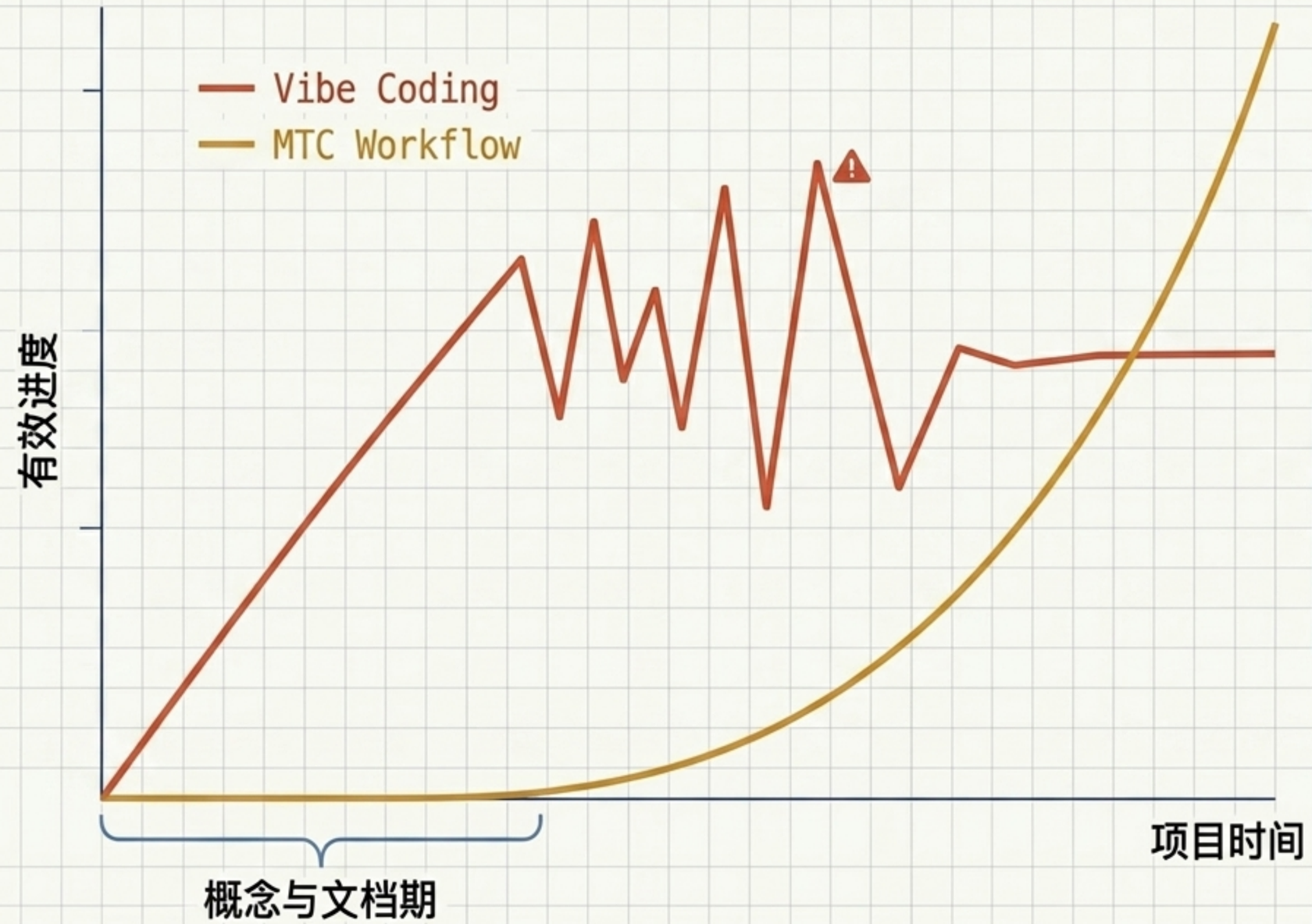
引入 MTC: More Than Coding

MTC 不是代码模式的低配版，它是「代码之前的一切」。
通过在 Claude Code 中构建 /mtc skill，将 AI 从“代码打字机”升级为“虚拟开发团队”。

开发范式对决

维度	Vibe Coding（随性编程）	MTC Workflow（系统化智能体）
启动速度	极快（1 分钟开写）	较慢（需 30–60 分钟前期规划）
返工率	极高（每次迭代都在推翻重写）	极低（在写代码前解决概念冲突）
上下文管理	依赖 AI 记忆和短期对话窗口	固化为 Markdown 文档作为硬输入
AI 角色	盲目的“代码打字机”	具备全局视角的“虚拟开发团队”

PROJECT: AI-DEV-GAP	DRAWN BY: VIBE_CODING_ANALYSIS	
REV: A.03	DATE: OCT 26, 2024	SCALE: NTS



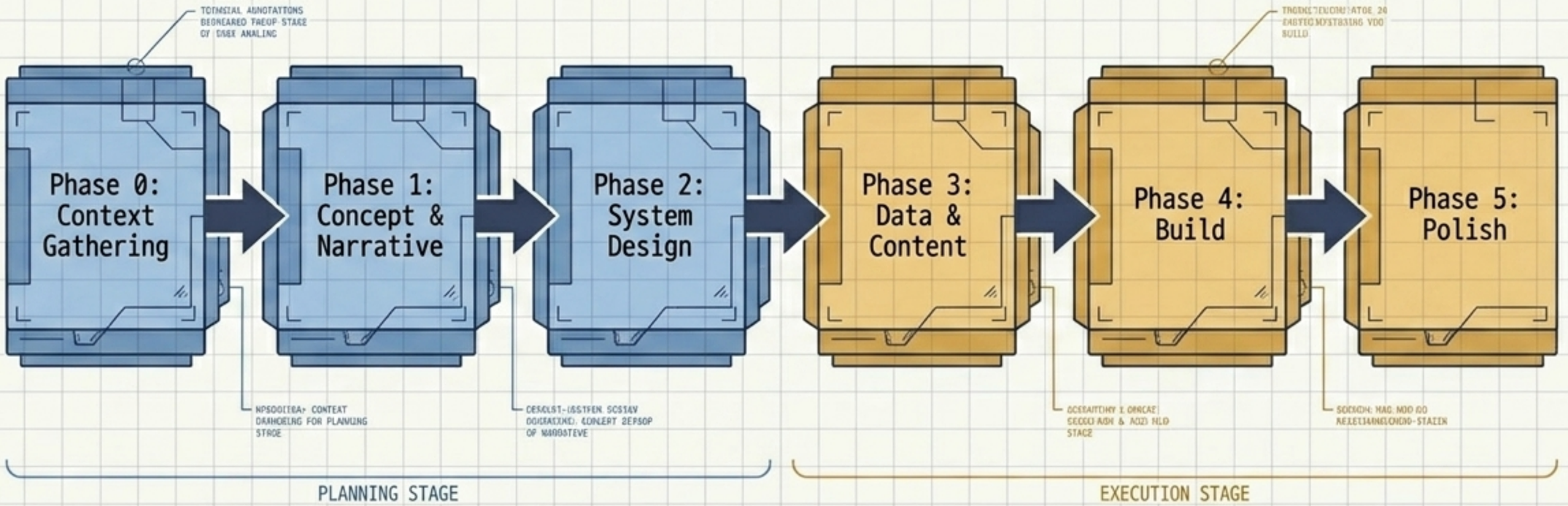
> 10 分钟的概念工作，省下 1 小时的返工

- Inside Job 游戏开发实测：
  - 前期投入：40 分钟出概念与系统文档
  - 后期收益：对比直接开发，节省 2-3 小时返工时间

PROJECT: AI-DEV-GAP	DRAWN BY: MTC_WORKFLOW_ANALYSIS	
REV: A.01	DATE: OCT 26, 2024	SCALE: NTS

MTC 的 6 阶段工程管线

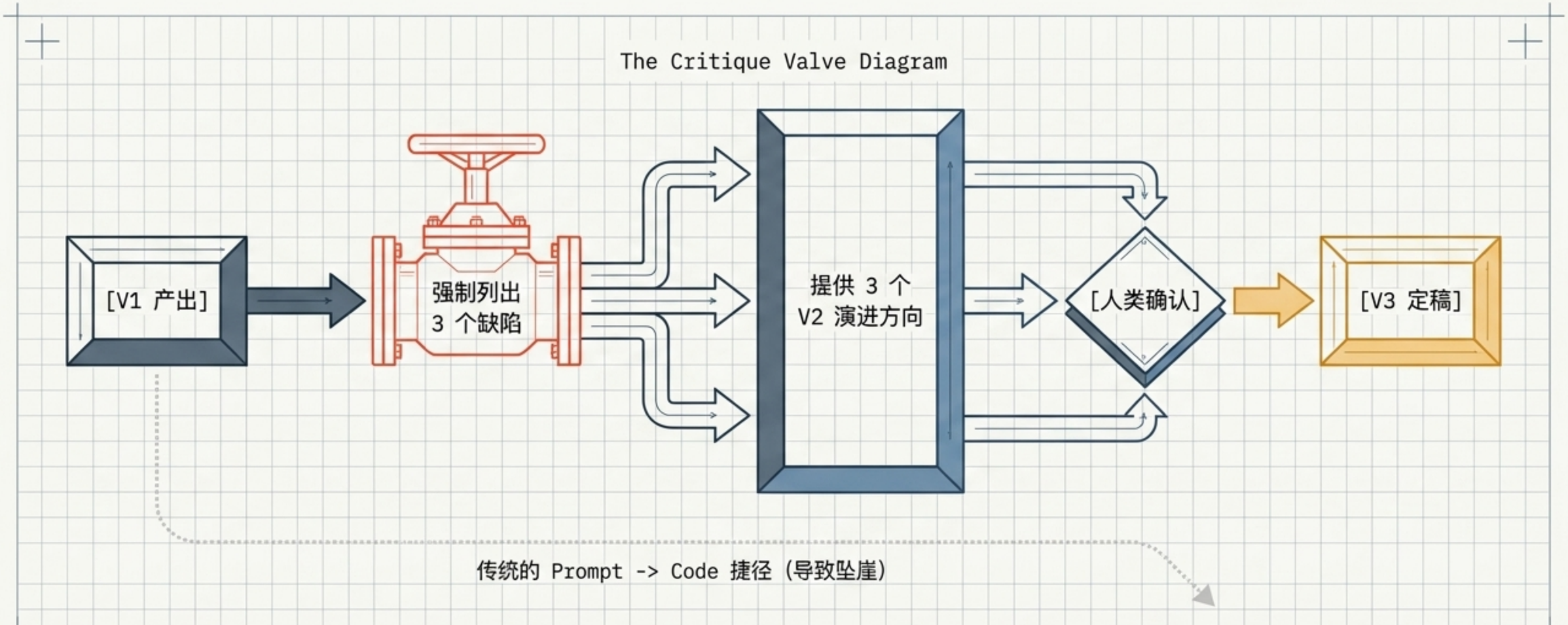
强制顺序，文档驱动，拒绝跳步。



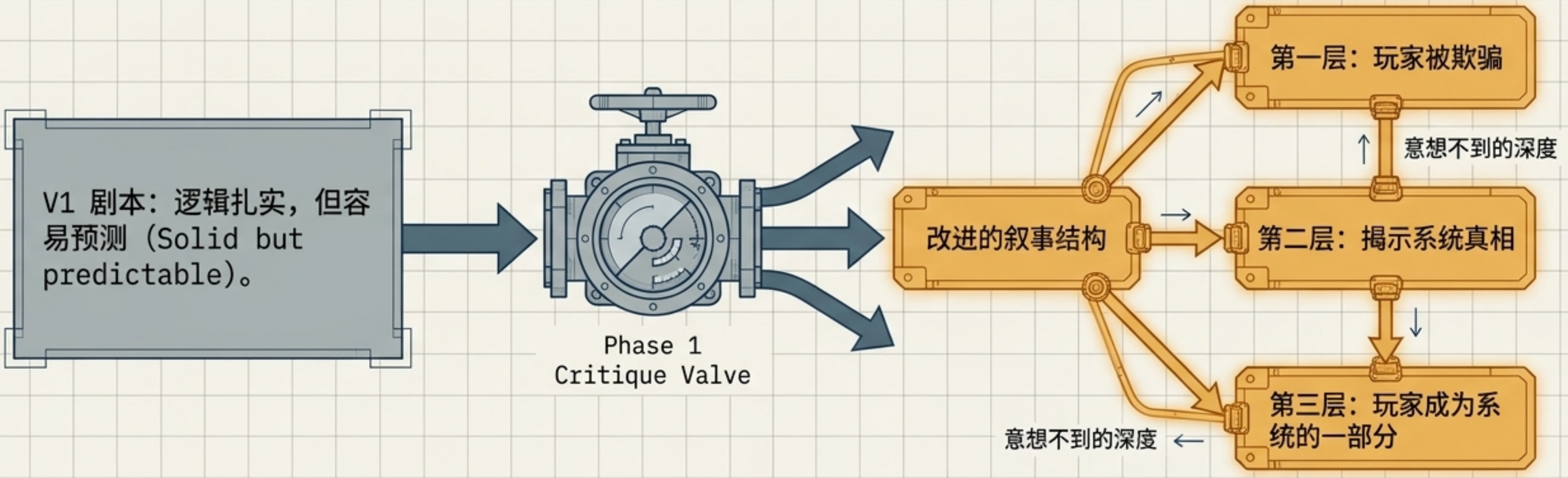
PROJECT: AI-DEV-GAP	DRAWN BY: MTC_WORKFLOW_ANALYSIS	
REV: A.04	DATE: OCT 26, 2024	SCALE: NTS

Phase 1: 强制自我批评机制

如果让 AI 一边写代码一边反思概念，它不会反思，它只会继续写代码。必须在独立阶段强制执行。



机制战胜 AI 的本能



以独立游戏《Inside Job》为例：
经过 Phase 1 强制批评后：AI 将缺陷拆解，提供改进方向，最终重构出意想不到的三层叙事结构。
自我批评不是 AI 天生会做的，是工作流强迫它做的。

PROJECT: AI-OEV-GAP		REV: A.04
DRAWN BY: NTC_WORKFLOW_ANALYSIS	DATE: OCT 26, 2024	SCALE: NTS

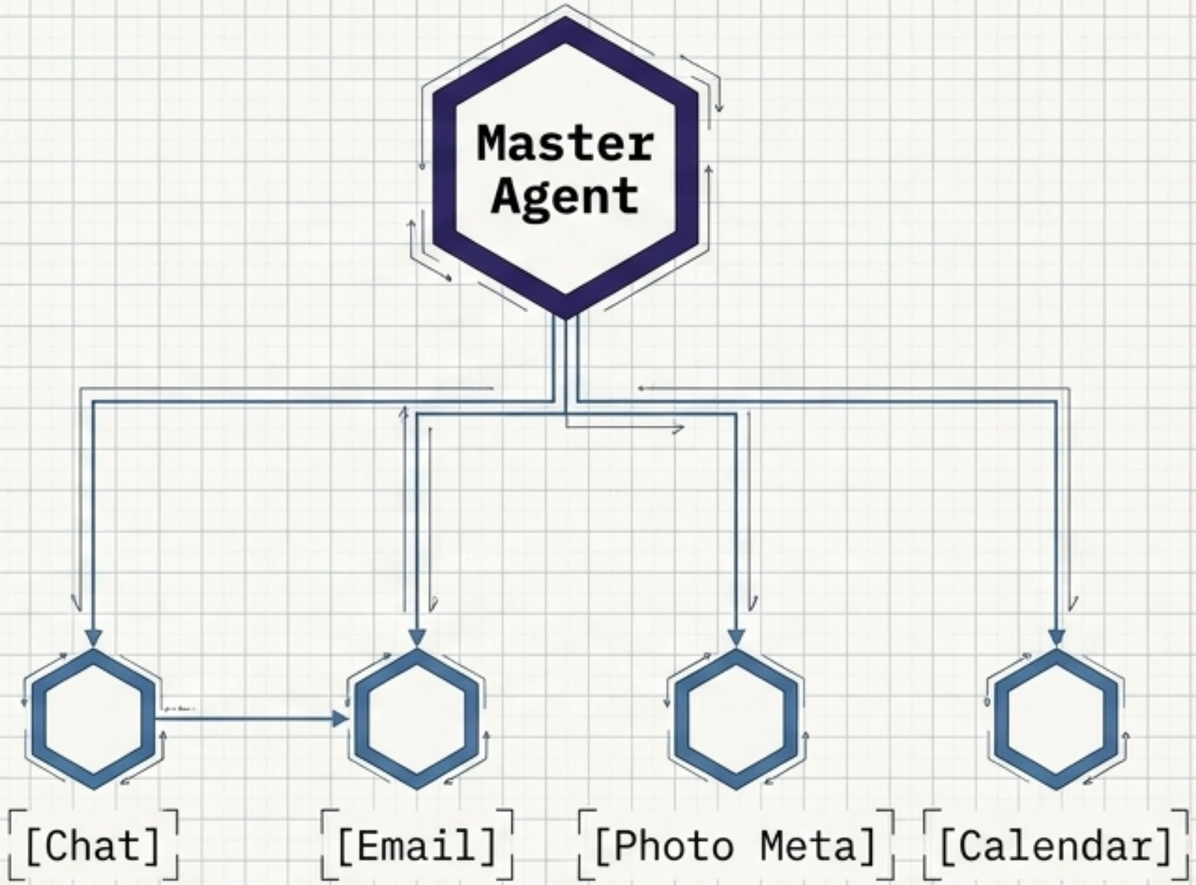
Phase 2：系统设计与文档继承

确立机制、解锁链与信息架构。

上一阶段的输出，是下一阶段的唯一硬输入。每一个决策都被固化在 docs/ 文件夹中，形成项目的长期记忆护城河。

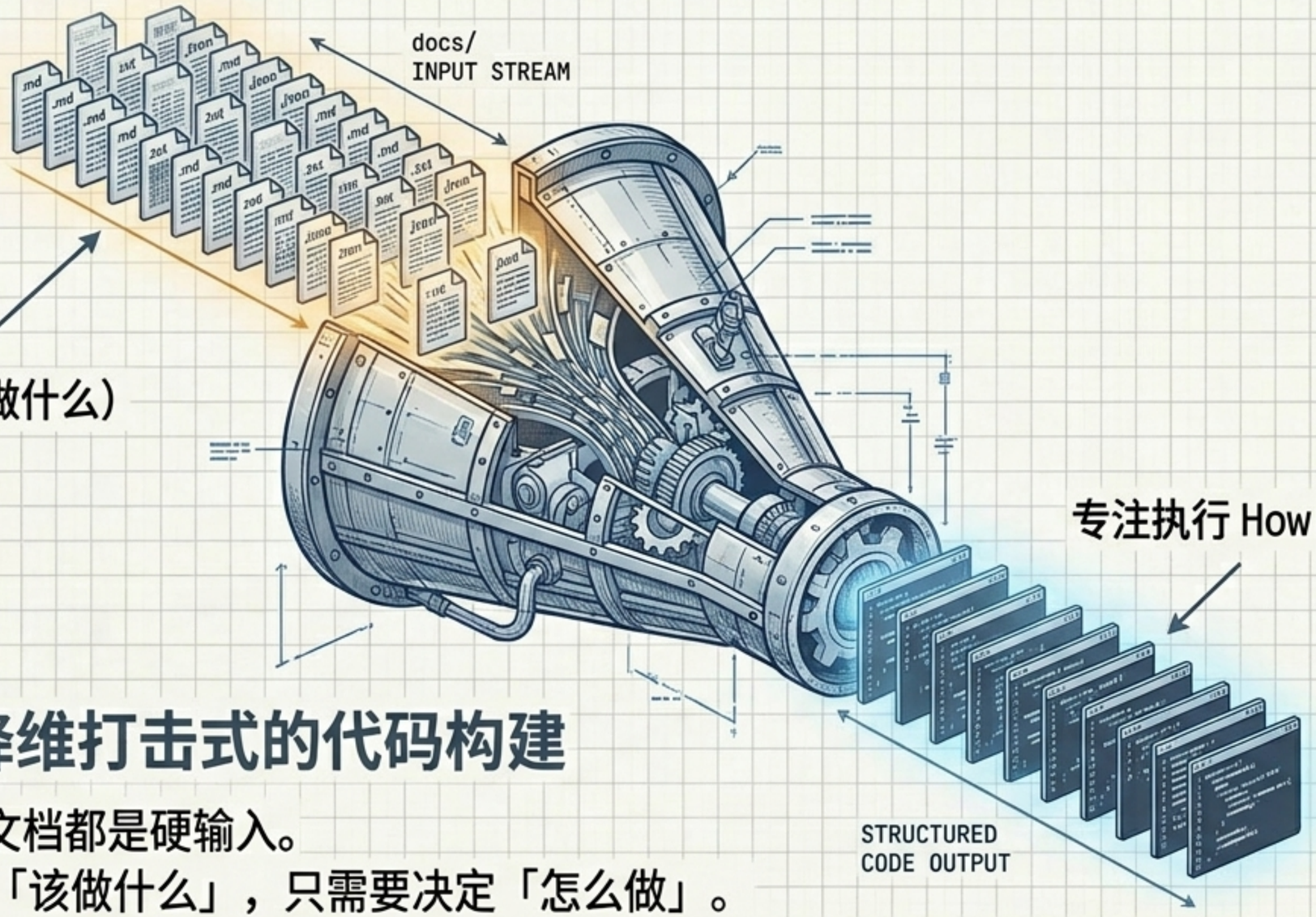
Phase 3 & 4: Subagent 多线程并行加速

- 进入 Mock 数据生成与代码构建阶段，任务互不依赖。
- 一键 Spawn（衍生）多个子智能体，专注度高，极速并行处理。



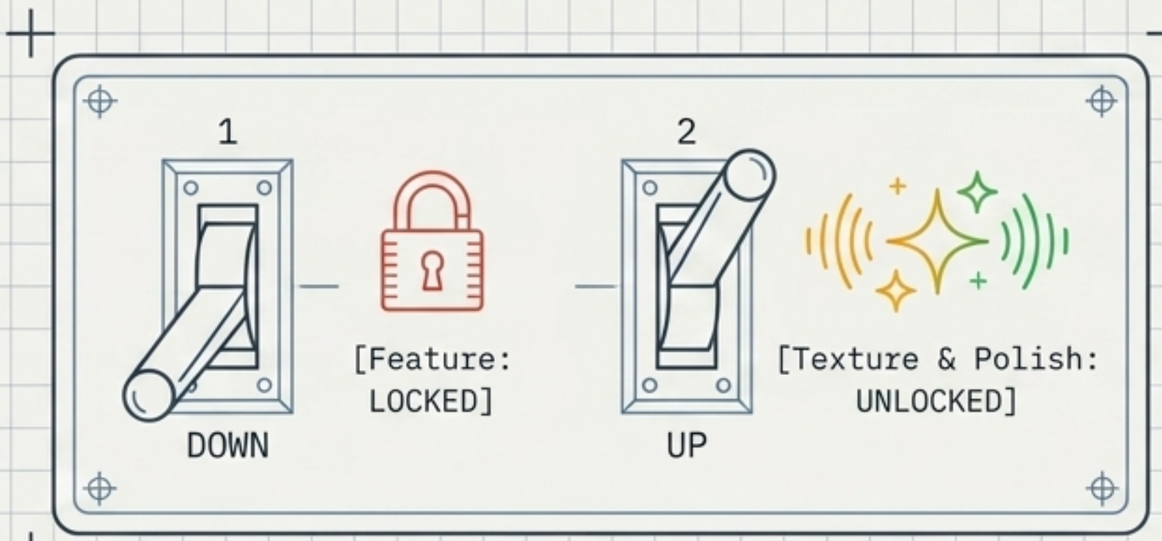
并发调度矩阵

阶段（Phase）	任务类型	调度策略	解释
Phase 1 **（概念）**	剧本、核心机制迭代	单线程 （Single Thread）	需要全局视角，拆开会导致逻辑碎裂
Phase 3 **（数据）**	生成海量对齐的 Mock 数据	多线程并行 （Spawn Subagents）	任务独立，4 个智能体并行极速生成
Phase 4 **（代码）**	独立前端组件开发	主从架构 （Master + Subagents）	主 Agent 控架构，子 Agent 并行写组件



Phase 4: 降维打击式的代码构建

此时，前面所有文档都是硬输入。
AI 不再需要决定「该做什么」，只需要决定「怎么做」。
在这个阶段，它不是在局部上下文中盲人摸象，而是在完整设计的指导下构建系统。



Phase 5: 质感打磨

只加质感，不加功能。

专注于音效、动画与彩蛋的注入，为系统级的坚实架构披上完美的外衣。

限制，孕育出专家级的创造力

赋予 AI 最大的自由，往往得到平庸且混乱的结果。

给 AI 设定最严格的强制流程 (拒绝跳步、文档硬输入、必须自我批评)，
它才能展现出真正的架构师能力。

MTC 不仅是一个 /mtc skill，
它是人机协同构建复杂系统的未来蓝图。