VOLUME 02

Agent-native 软件工程

从“让 AI 写代码”到“让 Agent 在边界内执行系统计划”

真正的分界线不是会不会 Prompt，而是能不能把模糊目标压成边界清晰、可验证、可回滚的工程任务。

银白认知体系 · 2026.08

基于真实项目、长期 Agent 协作与个人经验的阶段性整理

目录

01 为什么 Vibe Coding 只是起点

AI 已经能很快生成页面、API、数据库和客户端代码，所以“把功能写出来”正在变得越来越便宜。但真实软件的成本大量藏在功能之后：状态、失败恢复、版本兼容、发布、日志、用户环境、升级和维护。

因此，Vibe Coding 可以是极好的原型方式，却不能自动等于软件工程。真正的工程从一句“然后呢？”开始。

02 人和 Agent 的职责重分配

人负责	Agent 负责
定义目标与价值	搜索、实现、样板代码
架构与模块边界	跨文件修改与重构
权限与风险上限	执行测试、静态检查
验收标准	生成文档、整理日志
最终判断与责任	大规模重复执行

“AI 不是方向盘，AI 是发动机；Agent 不是负责人，Agent 是执行层。”

03 功能目标如何倒推研发计划

成熟的研发计划不必一开始就“预测完整未来”。更有效的方式是从功能目标倒推：用户到底想得到什么结果？这个结果依赖哪些模块？每个模块的失败模式是什么？最小闭环是什么？如何验证？

于是计划从愿景逐步压缩成可执行步骤。

```
我要实现一个功能
  ↓
用户可感知结果是什么?
  ↓
涉及哪些模块 / 平台 / 状态?
  ↓
哪些地方最可能失败?
  ↓
最小可验证闭环是什么?
  ↓
Agent 任务拆分 + 边界
  ↓
实现 → 测试 → 证据 → 合并
```

04 边界：先规定不能做什么

Agent 执行能力越强，越要先写清边界。很多灾难不是因为模型不会写代码，而是因为任务描述只有“做什么”，没有“哪些东西绝对不能碰”。

高质量任务往往同时包含：目标、允许修改范围、禁止修改范围、兼容约束、回滚要求和验收证据。

> **边界模板**
> 允许：修改 A/B 模块；禁止：改协议格式、删除兼容代码、触碰生产配置；必须：保留旧接口、运行指定测试、给出变更清单；失败：停止并报告，不自行扩大范围。

05 Git：让犯错成本可控

Git 的价值不只是“保存代码”，而是降低探索成本。只要当前状态被可靠记录，很多高风险尝试就从“不能动”变成“可以试，错了回滚”。

这也是 Agent-native 开发可以大胆放权的重要前提：不是相信 Agent 永不出错，而是把失败限制在可逆范围。

06 验证：完成声明不等于完成

Agent 说“已完成”只是一个文本输出。工程完成必须有证据：编译通过、测试通过、实机表现、日志、哈希、文件数、截图、性能指标或可复现实验。

越复杂的系统，越不能把自述当事实。验收应该尽量独立于实现者。

“证据优先于叙述，验证优先于相信。”

07 状态机、日志与可观测性

复杂系统最怕“看起来连接了，实际上没进入业务状态”。状态机的价值是把隐含状态显式化；日志与指标的价值，是让系统在失败时能够解释自己。

对 Agent 来说，可观测性还具有第二层作用：没有日志时，它只能猜；有结构化日志时，它才能沿证据定位。

工程对象	解决的问题
状态机	状态是否有唯一语义、迁移是否合法
日志	发生了什么、先后顺序是什么

指标	性能和稳定性有没有变化
测试	改动有没有破坏已知行为
Known-Good 基线	出问题时能不能退回稳定状态

08 复杂度预算：别把小工具造成航母

工程能力变强以后，一个新的风险是过度设计。状态机、插件、抽象层、DI、协议层都不是“高级就应该用”，它们只应该在问题复杂度需要时出现。

真正成熟的工程师既能压住大型系统，也能面对小工具时克制地说：生命周期就这样，做到这里够了。

“架构的目的不是展示架构，而是控制复杂度。”

09 跨端系统为什么指数变难

单端应用的错误大多发生在一个平台内部；跨端系统增加了一个更麻烦的空间：A 端认为正常、B 端表现异常，根因可能在链路、协议、时序、权限或状态同步。

复杂度不是简单的 Win + Android，而是 Win、Android，以及它们之间所有可能发生的事情。

```
单端：UI → 业务 → 数据 → OS

跨端：A 端状态 → 协议 → Transport → 网络/USB/蓝牙 → B 端状态
      ↘ 版本 / 权限 / 时序 / 重连 / 兼容 / 用户操作 ↙
```

10 插件化：主程序不依赖附加能力

当一个底座已经解决 UI、设备连接、日志、更新、权限和基础框架，新需求就不必每次新建独立软件。插件应该是纯附加：主程序不依赖插件，插件单向依赖主程序提供的能力。

理想结果是：插件崩溃、未安装、版本旧，都不能拖死核心连接和主流程。

11 Agent 安全：只读、最小权限、可回退

1. 诊断默认只读：先看日志、进程、文件、配置，再决定是否修改。
2. 任何破坏性动作先解释影响面，并提供备份/回滚路径。
3. 明确禁止递归删除、覆盖系统目录、修改无关服务或扩大任务范围。

4. 高风险写操作尽量拆成“检查 - 预览 - 执行 - 验证”四步。
5. 权限只给完成任务所需的最小集合。

12 工程闭环：从代码到用户

代码完成只是产品生命周期的一环。真正闭环还包括构建、测试、打包、安装、升级、回滚、文档、官网、支持和真实用户环境。

AI 让“写”变便宜以后，人的价值会更集中在：做什么、为什么做、怎么组织、怎么证明做对了、出了事故怎么收回来。

```
需求 → 架构 → 实现 → 测试 → 打包 → 发布 → 用户 → 反馈 → 维护 → 下一版
```