VOLUME 05

墨堤案例研究

一个“手机声音推到电脑”的需求，如何在不到两个月里长成跨设备系统

这不是项目宣传册，而是一份方法论案例：真实复杂度如何暴露、Agent 如何让高起点项目变得可行、认知又如何被项目反向重构。

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 起点：我只是觉得现有软件不好用

2026 年 6 月 10 日，最初计划并不是做“设备互联协议”。问题非常具体：市面上的手机音频推流到电脑方案不好用，所以想做一个更顺手的工具。7 月 1 日正式开发。

如果只看需求，它似乎很简单：手机采集声音，传到电脑，电脑播放。但真正开始实现以后，隐藏在一句话下面的系统问题逐层暴露。

“一开始我只是想解决一个具体痛点，系统方向是在把问题挖深以后自己长出来的。”

02 为什么第一个大型项目起点异常高

普通个人开发者第一个项目往往是单端工具：一个 Windows 应用、一个 Android App、一个网站。墨堤从一开始就需要至少两个操作系统、两个客户端、一条实时数据链路和真实设备环境。

这意味着成长不是先学局部再看系统，而是先被系统整体复杂度狠狠干一遍，再回头补局部。

单端大型项目	跨端互联项目
主要复杂在单平台纵向深度	同时存在平台深度与系统横向耦合
状态大多在一个进程/设备	状态需要跨设备同步
错误归属相对明确	现象在 B，根因可能在 A 或链路
发布一个客户端	双端版本、兼容、链路共同演进

03 从音频管线到跨端系统

最初链路包含采集、PCM、编码、网络、Jitter Buffer、解码和播放。任何一段不稳定，用户看到的都可能只是“没声音”。

当系统真正要给人用时，新的问题马上出现：权限、后台保活、虚拟麦克风、设备发现、断线重连、热切换、安装与更新。于是“音频功能”很快变成“跨端产品工程”。

```
Capture → PCM → Frame → Opus → Packet → Transport
                                    ↓
                        Jitter / Loss / Delay
                                    ↓
                          Decode → Playback
```

04 四链路逼出来的抽象

LAN、Wi-Fi Direct、Bluetooth、USB 的性质差异巨大。如果每条链路都直接绑业务逻辑，项目会迅速变成四套不可维护的实现。

于是一个核心抽象被逼出来：应用描述“我要传什么”，协议与 Transport 负责“当前怎么传”，Adapter 把平台与介质差异隔离在边界之外。

```
Application
   ↓  数据意图
Profile / Protocol Core
   ↓  统一传输语义
Transport Interface
   ↓
LAN | P2P | Bluetooth | USB Adapter
```

05 协议不是包头，而是复杂度下沉

V2 的 15 字节包头只是最小骨架：Magic、Version、Type、LinkType、Seq、Len 让数据可识别、可版本化、可排序、可解析。真正的长期价值不是“字段多”，而是把跨链路差异、可靠性、排队、优先级、参数和回退尽量下沉到底层。

这也是从“工具”转向“协议/平台”的关键：第三方上层不应该理解每种链路的所有细节。

06 状态机、重连与真实失败

原型阶段最容易出现“UI 显示 Connected，但实际没有 Streaming”这样的假状态。随着链路增加，Idle、Searching、Connecting、Connected、Streaming、Reconnecting、Error 等状态必须有统一语义。

状态机不是为了写得漂亮，而是因为系统已经复杂到不能再靠几个 Boolean 猜当前发生了什么。

07 为什么跨端复杂度是超线性的

复杂度不是简单的“Windows 一份 + Android 一份”。双端每新增一个状态、链路或版本，都会增加组合。真实 Bug 经常跨层：系统权限影响采集，采集影响编码，网络抖动影响缓冲，缓冲又影响用户感知。

因此，工作量的增长更多来自交互项，而不是代码行数。

08 Agent 为什么让这条路线变得可行

前 AI 时代，个人同时承担 Windows、Android、音频、网络、官网、发布，最容易死在执行成本：查 API、写样板、跨栈切换和低级调试就足以耗尽时间。

Agent 把大量执行工作压缩以后，人的时间可以更早投入架构、边界、验收和系统演进。项目复杂度开始与“个人手写代码吞吐量”部分解耦。

过去的主要瓶颈	Agent 加持后的变化
查资料与找入口耗时	快速定位文档、解释错误
跨栈样板实现重复	批量生成与重构
陌生领域进入成本高	先进入，再在反馈中补知识
个人吞吐量限制项目上限	上限更取决于系统理解和验收能力

09 从项目倒推知识体系

项目推进方式和学习方式后来变成同一个结构：先提出功能目标，再让 Agent 帮助倒推研发计划；执行时暴露知识缺口，再回头补理论。

网络、OS、音频、状态机、Git、发布、本地化、嵌入式等知识因此不是孤立章节，而是被挂到同一张系统图上。

“从学习技术，变成理解系统；从记知识点，变成知道知识应该挂在哪一层。”

10 从产品到平台：插件与底座

当互联底层、UI、日志、更新、设备框架都已经存在，新需求不一定值得再做独立软件。更自然的方向是把小功能做成纯附加插件：要 UI 用现有 UI，要单端功能挂现有框架，要跨端直接消费底层能力。

但依赖方向必须保持：Core 不依赖插件，插件单向依赖 Core；插件失败不能影响主程序核心进程。

11 工程方法沉淀

1. 先跑最小闭环，再扩展，而不是先把愿景全部实现。
2. 先建立 Known-Good 基线，再让自动化逐步接管。
3. 先观测，再控制；先 Shadow Mode，再扩大权限。
4. 每一次抽象都要有真实重复问题作为理由。
5. 把失败模式写进架构：超时、重连、回退、日志、版本。

6. 用真实第一方应用持续压力测试底层抽象。

12 两个月真正制造了什么

到 2026 年 8 月底，最值得记录的并不只是一个越来越大的仓库。更重要的是，一套新的成长循环已经形成：真实项目 - Agent 杠杆 - 高复杂度问题 - 快速学习 - 认知升级 - 更大的项目。

墨堤作为第一个真实大型项目，最大的副产品是把“系统性认知”提前压缩到了一个极短时间窗口里。即使未来具体产品路线继续变化，这套工程方法、学习方式和长期记录仍然会留下来。

“项目不只在制造软件，也在制造能够理解更大系统的人。”