FDE：企业AI落地实战手册

The FDE Handbook: A Practical Playbook for Forward-Deployed Engineers

许良 著

0to1.site

2026 年 9 月 · v1.0

开源仓库：github.com/RichardXu001/the-fde-handbook

自序

一、2026 年，每个人都在问同一个问题

2026 年，FDE——Forward Deployed Engineer，中文通常译为"前线部署工程师"，也被称为"前沿部署工程师"——突然无处不在：咨询巨头把它做成主力服务线，模型公司亲自下场组建交付组织，招聘网站上同名岗位每周都在新增。

热闹之下，我反复听到的是同一批问题，而且提问的人身份越来越具体：

- 工程师问：这和外包、和驻场咨询有什么区别？我会不会被点名转岗？
- 技术负责人问：招了AI工程师还是落不了地，我是不是该招个 FDE？
- 甲方问：供应商说"效果很好"，我怎么知道什么时候真的可以上线？
- 创业者问：这门生意的账，到底应该怎么算？
- 怀疑者问（而且问得有道理）：这是不是咨询换了个名字？

这些问题背后，是同一件更根本的事：**AI 从 Demo 到客户价值之间的那段路，到底要修哪些桥、谁来修。**这本书回答的就是这件事。它不回答"FDE 是不是风口"——那是主观问题；它回答的是"FDE 是什么、怎么干、账怎么算、人怎么办"——那是实际的操作问题。

二、为什么我来写这本书

我自己是个AI领域的创业者，也做过一些企业AI改造的项目，一开始也是边摸索边做。

写 FDE 的文章已经很多，但是系统化介绍 FDE 的书很少；而且这个话题上，很多内容经过一轮轮转述早已变形——每一环都在简化、外推，传到最后，真实度已经像复印纸复印十次一样模糊了。

我于是把自己的项目经历、身边朋友的经历，加上社区的讨论、公开的资料，还有一些优秀的开源项目，一起写成了这本书——每一条引用来源，都在正文和附录中标了出来。

本书案例来自三种渠道：**本人的交付与观察经历；我身边亲历者的讲述（均已征得同意并进行脱敏）；公开材料**（包括访谈、播客、社区讨论、公司公告、开源项目等，详见附录 C）。

三、这本书怎么读

全书六部，每章回答且只回答一个问题：

1. **第一部·角色与边界**：FDE 到底是什么，像谁，不是谁，会不会是一阵风，会不会被 AI 取代。
2. **第二部·从发现到生产**：进入客户现场之后，怎么选问题、怎么做最小可行部署、怎么回答"什么时候上线"。

3. **第三部·工程底座**：数据权限、Agent 约束、评测、自证、成本——从 PoC 到生产隔着的八堵墙，逐堵翻。
4. **第四部·采用与移交**：系统上线了为什么没人用，为什么是一把手工程，怎么把系统交给客户自己运行，中国为什么是另一套打法。
5. **第五部·商业与产品化**：这门生意的大账怎么算、怎么收费、定制量为什么必须下降、现场经验怎么变成产品。
6. **第六部·人与组织**：FDE 的一天、能力模型、转型、建团队、单干，以及这个角色的终局。

每一章首先写给谁，都标在章首的简报里——工程师、技术负责人、甲方、创业者、怀疑者，都能在目录里找到为自己排的问题链。

不赶时间的读者从第 1 章顺序读；赶时间的读者可以从第二部或第三部切入——工程章自成体系；只想做决策的读者，可以直接去第五部看账，再回头补第一部。

四、写在前面

这本书想说的，其实就是一句话：**做无法规模化的事情，规模化地做。**前半句是 FDE 的日常——每个客户现场都是一次性的、不可复制的苦工；后半句是这门手艺的价值所在——把苦工里可复制的部分沉淀为产品，让下一个客户走得更快。

FDE的名字也许会变，但责任不会。如果你读完这本书只记住一件事，我希望是这一件。

最后说句实在话：这本书能给你的，最多是把我踩过的坑提前标出来，帮你少走几步弯路；真正的判断力，还是得你自己到客户现场里，一次次磕碰中锻炼出来。书读得再细，也替代不了你自己上场试一次。

欢迎关注我的公众号【业界良新】，也欢迎来我的个人网站逛逛 https://0to1.site，会持续分享我的看法和心得。

是为序。

许良

2026.9上海虹桥

前言 · AI 项目到底卡在哪里?

> A complex system that works is invariably found to have evolved from a simple system that worked.
> 一个能跑起来的复杂系统，几乎总是从一个能跑起来的简单系统演化来的。
> ——约翰·加尔（John Gall），《系统论》（Systemantics）

2026 年，几乎每一场行业峰会都在讲同一件事：AI 是这一轮商业竞争的胜负手。资本追着"智能体"概念跑，芯片巨头的市值一次次刷新纪录，创业公司的故事里只要有"AI"两个字，就好讲得多——这一片火热，是真的。

但把目光转到企业的生产系统里，画风则完全不一样。MIT NANDA 的一份报告给出一个数字：95% 的组织，从生成式 AI 身上拿不到任何可测量的损益影响［1］。翻译成大白话：显卡买了，token 也在源源不断地烧，账上却看不到一分钱回报。

数字背后是具体的人、具体的沮丧——2021 年，一位干过部署工作的工程师在 Hacker News 上说："它让我成了一个更差的工程师。"［2］五年后，一位中国的 AI 交付工程师换了个说法，意思没变："四版迭代下来，最大的帮助是给领导提供情绪价值。"［3］

Demo 的时候，人人都说这次是真惊艳；可一到真正落地，为什么全卡住了？到底卡在哪儿了？同一个问题，不同的人会给出不同的答案。我们接下来要做的，就是把这些答案摆在一起，看看它们说的，会不会其实是同一件事。

一、落地卡点真实存在

AI 落地的卡点不是感觉，是有数字支撑的事实。

MIT NANDA 实验室 2025 年发布的《GenAI Divide》报告给出了三个数字［1］：

- **95% 的组织，没有从 AI 获得可测量的损益影响**——报告原话是"无可测 P&L 影响"（P&L 即损益，指说不清对盈亏的影响），不等于"95% 的项目失败"，两个说法经常被混用，但不是一回事；
- 仅约 **5%** 的定制企业 AI 工具真正进入了生产环境；
- 在采购策略上，外购（约 67%）远多于自建（约 33%）——但"买"也没有解决问题：买了，照

[1] MIT Project NANDA《The GenAI Divide: State of AI in Business 2025》
[2] Every engineer should do a stint in consulting（Hacker News 讨论帖）
[3] AI 职觉：别再卷 Agent 了（小宇宙播客）

样停在 Demo。

第三个数字特别值得停一下：多数企业的第一反应是"买"，而报告显示买来之后卡点依旧。这说明缺的不是 AI 工具的供给，是**把工具装进业务的能力**。报告本身的归因也指向这个方向：卡点不在模型能力，在学习缺口（组织还没学会怎么用）与集成。

二、第一堵墙：集成

那么"装进业务"到底卡在哪里？一线的答案比想象中要土。

绝大多数"AI 用不起来"的企业，先撞上的根本不是模型墙，是**二十年来没打通的系统墙**：数据躺在七八个互不相认的系统里，流程横跨三套工具，接口是十年前离职的人写的。AI 并没有修复这堵墙，Agent 也不会自动修复它（第 9 章展开）[4]。

模型能力这几年翻了几个数量级，这堵墙一寸都没动过。它不歧视任何模型——你换再强的模型，墙还是那堵墙。

三、第二堵墙：权限

第二堵墙更隐蔽：Agent 撬不动企业里的权限体系。

Agent 只拥有其背后用户的原有权限，卡住了也不会像真人同事那样靠人情与变通绕过权限墙[4]——所以 Agent 落地到企业里，常常死在第一步：它连需要的数据都拿不到（第 10 章展开）。

集成与权限，这两堵墙有个共同点：**它们都不是技术问题，是"没有人对打通它们负责"的问题。**模型厂商不管，软件供应商不管，客户的 IT 部门各有各的边界——卡点就在这些"三不管"的缝隙里。

四、Demo 之后的一周

以下示例由多方真实教训综合改写：人物与公司虚构，用于展示落地卡点的形状。

一套智能客服 Agent，演示当天表现完美：问答流畅、结论准确、大屏漂亮。接入真实工单系统的第一周——

第一天，权限墙：工单数据在三个系统里，Agent 的账号只申请到了其中一个的只读权限。等审批。

第三天，工具误操作：权限打通了，Agent 在一次批量操作里把测试数据写进了正式字段。回滚用了一下午。

[4] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

第五天，效果退化不可测：换了版本之后，客服主管说"感觉变差了"，工程师说"我看不出差别"——没有人有基线，没有人能证明任何东西。

第二周，业务部门不用：一线客服还是用旧工单界面，"新系统点进去还要多一步"。系统绿着，用的人数为零。

每一堵墙，本书后面都有一章专门处理：权限（第 10 章）、工具边界（第 11 章）、效果可证（第 12、13 章）、没人用（第 15 章）。这里先让读者看见它们排在一起的样子——**Demo 与生产之间，隔的从来不是"再聪明一点"，是一串各有姓名的墙。**

五、质量的卡点有具体的深度

墙之外，还有一个更细的卡点可以量化：质量。

X上（原推特）有保险从业者公开过一组数字：内部智能体首次上线的真实准确率约 70%，合同承诺超过 98%［5］。从 70 到 98 的这 28 个百分点，就是实打实的"交付能力"——它不是靠调 prompt 调出来的，是一整套评测与回归的循环垫出来的（完整展开见第 7 章与第 12 章）。

还有一道更慢的差距：模型能力一年内从一代跳到下一代，落地速度却远远落后［6］。这符合许多人的直观感受：AI 能力狂奔，世界感觉平淡，就像自动驾驶的技术曲线早已陡峭、街上的车流却照旧。这道"采用差距"与 MIT 的数据互证：卡点不在能力，在采用与集成。本书第 15 章与第 28 章专门讨论。

六、主角出现

把上面的线索合起来：

- 卡点存在且可测量；
- 卡点的名字叫集成、权限、评测、组织激励——全都不是模型问题；
- 这些墙的共同特征是**没有直接负责人**：没有一个人对"上线、可用、可维护"负责到底。

于是可以给出本章的核心观点：**模型能力是通用商品，把模型装进具体业务的能力是稀缺品**——FDE（Forward Deployed Engineer，本书统一称"前线部署工程师"；招聘语境也常称"前沿部署工程师"）这个角色在 2026 年的爆发，正是市场对这一卡点的定价。

它不是新发明。Palantir 在二十多年前就用"前线部署工程师"的组织形态跑通过这套打法：把工程师派到客户现场，对真实运行的结果负责。2026 年，市场重新需要它：咨询巨头把它做成服务线，模型公司亲自下场组建交付组织（第 4 章的时间线会逐日摆开）。

[5] Inside an Applied AI company（Pace 长文）

[6] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

接下来的六部二十八章，就是把"修哪些桥、谁来修"这句话拆开：它是什么（第一部·角色与边界）→怎么交付（第二部·从发现到生产）→工程墙（第三部·工程底座）→采用与移交（第四部）→生意（第五部·商业与产品化）→人与组织（第六部）。

开篇那句加尔定律，就是这条路线图的另一种说法：能跑起来的复杂系统，从来不是设计出来的，是从一个简单可用的系统一步步演化来的——而"演化"这两个字背后，必须有人一步步负责走完。

第一张清单，从最基本的问题开始：FDE 到底是什么——他对什么结果负责？

第一部分 · FDE 到底是什么

本部共五章，回答一个总问题：**这个突然爆发的角色，究竟是什么。**

它是什么（第 1 章，抛开头衔看责任结构）；它像谁、不是谁（第 2 章，三个可检验的行为判据）；它是不是外包换皮、一阵风（第 3 章，把泡沫和真实具体拆开）；为什么偏偏是 2026年突然爆发（第 4 章，一条可核验的时间线，而不是情绪）；它会不会被 AI 取代（第 5 章，哪些任务正在被吃掉、哪一层吃不掉）。

五章是一条问题链：**定义 → 辨异 → 质疑 → 时机 → 终极质疑**。每一章都为下一章制造问题——第 1 章立起"责任结构"这个定义，第 2 章立刻用它去切割相邻角色，第 3 章让怀疑者检验这个定义是不是新瓶装旧酒，第 4 章追问如果不是泡沫为什么偏偏现在爆发，第 5 章则问出最狠的一个：如果 AI 把它吃掉了呢。

本部的第一读者是**想看清FDE到底指什么的人**：准备入场的工程师、要写 JD 或读 JD 的管理者、以及所有被概念轰炸到怀疑的人。工程细节不讲——那是第三部分的事；怎么赚钱一部也不讲——那是第五部的事。本部只做一件事：把"FDE"这三个字母，变成一个可以拿在手里检验的东西。

第 1 章 · FDE 到底是什么——他对什么结果负责?

> 伟大的代价是责任。
> The price of greatness is responsibility.
> ——温斯顿·丘吉尔，1943 年哈佛演讲

2026 年，有三个人的名片都写着 Forward Deployed Engineer（FDE，通常称"前线部署工程师"，或者"前沿部署工程师"），工作却相差很大。

第一个人每天把约 75% 的时间用在模型优化和工程代码上。第二个人同时服务十个客户，会议和开发各占一半，随时要处理线上问题。第三个人坐在客户办公室收集需求，当场用 AI 做出原型，确认后再交给后端团队。

同一个头衔，三种工作方式。

所以，判断一个岗位是不是 FDE，不能只看名片。要看他对什么结果负责，负责到什么程度。

一、先承认：头衔帮不上忙

用某家公司的招聘页来定义 FDE，得不到完整答案。

同叫 FDE，工作内容和要求可能完全不同。Baseten 把模型优化和基础设施工程列为核心工作［1］。一家欧洲公司 2026 年的远程 FDE 职位要求 Python、SQL 和 PoC 能力［2］。国内 Linux DO 社区的一位大厂 FDE 则描述自己的日常：去客户现场收集需求，用 AI 做出原型，确认后交给后端团队［3］。三家公司使用同一个头衔，实际做的却是三种不同的工作。

反过来，有些岗位不叫 FDE，却承担着完整的 FDE 工作。公司可能称它为"解决方案工程师""交付工程师""AI 应用架构师"，也可能只是客户成功团队里负责写代码的人。

这不只是命名混乱。FDE 先在具体公司里形成，再被不同组织按自己的方式使用，岗位形态自然会变化。但它们承担的核心责任，往往很接近。

[1] Baseten：我做前线部署工程师学到了什么（Het Trivedi）
[2] 欧洲远程 FDE 职位说明（@afahmy_dev 转帖）
[3] 有在做 FDE 这个工作的佬吗（linux.do 帖）

二、一个浓缩的定义

要理解为什么会有 FDE，可以先看鲍勃·麦格鲁（Bob McGrew）的说法。他曾是 Palantir FDE 模式的亲历者，后来担任 OpenAI 首席研究官。他在 Y Combinator 的播客中把 FDE 概括为：**为单个客户交付许多能力**。传统软件工程师则是**为许多客户做单一能力**［4］。

产品工程师把一个能力做成通用产品，交给许多用户。FDE 面对一家具体客户，要在它的系统、权限、流程和协作关系中，把几项能力接上去、跑起来，并让客户持续使用。麦格鲁描述的流程包括：进入现场、快速做原型、持续迭代、接入生产环境。这些工作都围绕客户现场展开。

在 Palantir，相关人员又分成 Echo团队 和 Delta团队。Echo 是业务前线，由驻场分析师和客户经理组成，负责判断客户该解决什么问题；Delta 是技术前线，也就是狭义的部署工程师，负责把方案做成能运行的系统［4］ ［5］。国内企业往往不把这两类工作分给两个小队，而是让同一个人或一个小组同时承担［5］。

这套分工不按职级、技术栈或是否懂机器学习来划分。它只区分一件事：谁负责判断该做什么，谁负责把它做出来。

最后补一句定义层面的澄清：FDE 既指角色，也指建制（组织形式）。它的完整形态不是一个人，而是一支客户现场的小队——Echo/Delta 与第 26 章的中国三人组，是同一支小队的不同编法；单人 FDE，是把建制裁剪后压进一个人。本书多数章节以个人视角展开，但那个"个人"的背后，始终站着一支可以拆开、也可以合并的团队。

三、FDE 的时间都花在哪了

职位说明未必反映真实工作，日程表往往更可靠。Baseten 的 FDE 赫特·特里维迪（Het Trivedi）曾公开拆解时间分配：约 75% 用于软件工程和模型优化，15% 用于技术咨询，10% 用于客户关系［1］。

这组数字能澄清两种常见误解。FDE 不是主要靠沟通成交的"会写代码的售前"，因为工程工作占了大头。FDE 也不等同于被派去写代码的外包工程师，因为他还要花时间理解问题、与客户协作。

这只是单个公司的岗位设计，不能当作行业平均［1］。但它说明了一个要点：FDE 以工程工作为主，同时必须在客户关系中完成工程工作。后文会不断看到这一点：一个人承担什么责任，时间就会花到哪里。

这只是美国样本。中国 FDE 可能投入更多时间梳理业务问题、维护客户关系。有从业者称，自己约

[1] Baseten：我做前线部署工程师学到了什么（Het Trivedi）

[4] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[5] 腾讯研究院《FDE 模式行业观察与实践》

60% 的时间都在做这些事，技术占的反而不是大头——这笔中国时间账的完整展开见第 18 章。

四、责任四分法

国内开发者社区里，一位 FDE 曾把工作过程概括为："深入业务→流程优化方案与技术架构→开发→培训/灰度/调整业务流程→反馈/监控/修复/优化成本" [6] 。

这段过程可以拆成 FDE 需要承担的四类责任。

一个完整的 FDE，要同时对四件事负责。

1. **问题发现**——判断客户的哪个场景值得做。不是客户说什么就做什么，而是替客户把"模糊的愿望"筛成"值得解决的问题"。（第 6 章展开）
2. **生产工程**——让它安全、稳定、可维护地跑在客户的真实环境里，而不是停在演示环境。（第三部展开）
3. **业务采用**——让真实用户真的在用，系统上线不等于有人使用。（第 15 章展开）
4. **产品回流**——把现场沉淀的经验带回产品，让下一个客户更快。（第 22 章展开）

回看这段表述："深入业务"属于问题发现，"开发"属于生产工程，"培训/灰度"属于业务采用，"流程优化沉淀"属于产品回流。它与四类责任基本一致（见图 1–1）。

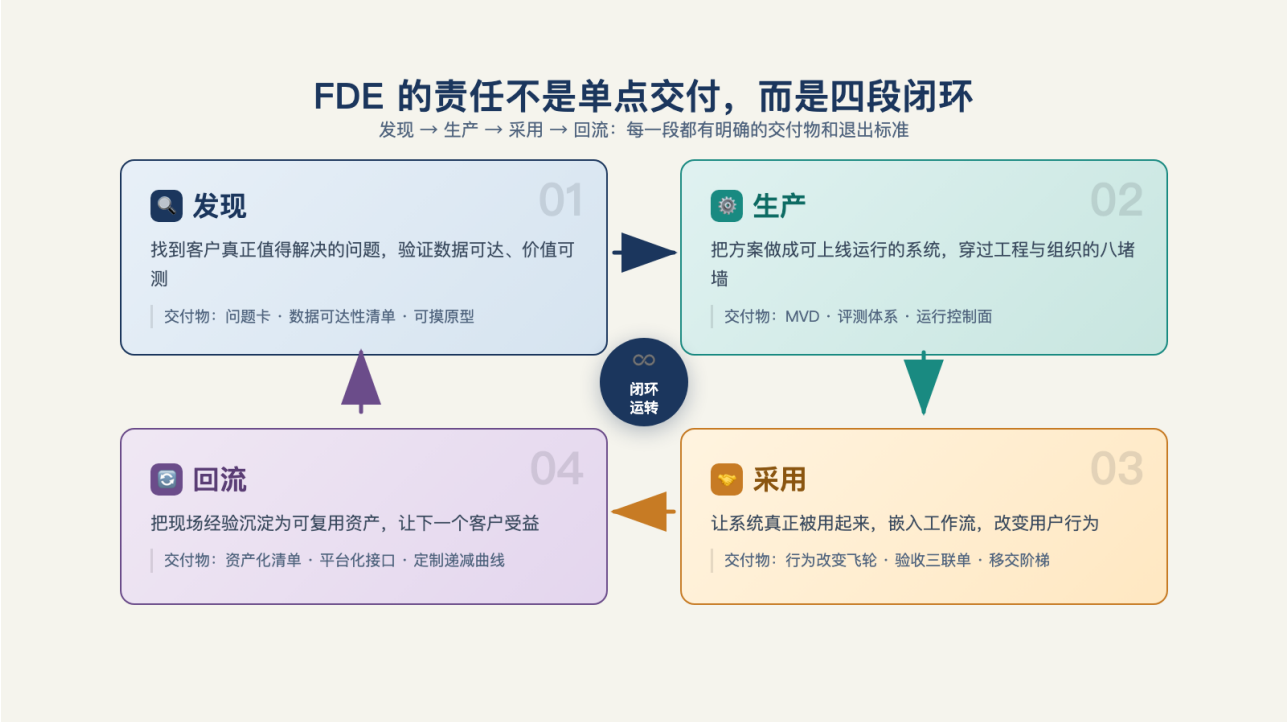


图 1–1：FDE 的责任不是单点交付，而是发现、生产、采用、回流四段闭环。

[6] linux.do 帖 #2580046

同时承担四类责任，才是完整形态的 FDE。只承担其中一两类，也可能是 FDE 工作的一部分。这并不意味着高低之分。多数岗位本来就只承担一个部分，关键是知道自己承担的是哪一部分。

五、同一周里的三个 FDE

以下是一个经过脱敏的示例：用于演示四分法的用法。

P，模型平台型 FDE。 上午在排查客户模型推理的延迟毛刺，下午给客户的算法团队讲解批处理接口的取舍。他的责任重心在生产工程——发现问题靠平台同事，推动使用靠客户自己的团队，经验回流靠平台路线图。按四分法，他主要落在第二格。

M，多客户并行型 FDE。 十个活跃客户，每周五十小时，会议和构建对半。上午在 A 客户的联调会上拍板一个数据接口方案，中午处理 B 客户的生产告警，晚上给 C 客户写周报。四类责任他都要管，但每一类都分不到足够的时间——他的风险不在能力，而在精力被十个客户摊薄。

W，驻场原型型 FDE。 在客户办公楼里，上午跟业务部门过需求，中午用 AI 工具把确认下来的流程拼成可点击的原型，下午把原型移交公司的后端团队去实现生产版本。发现和原型归他，生产和回流在别处——按四分法，他只覆盖第一格的一部分。

三个人都叫 FDE。用四分法来看，他们承担的责任并不相同。

哪个才算"真的"FDE？三者都可以这样称呼。更有用的问题是：他们分别处在哪个位置。只有先看清位置，后面讨论考核、转型和商业模式才有基础。

六、最后

本章可以总结成三句话：

- FDE 不是固定工种，而是一种**责任结构**：对客户业务中真正运行、可维护、可衡量的结果负责。
- 判断时不看名片，先看四件事：**是否写生产代码、现场经验是否回到产品、是否对采用负责、对结果负责到什么程度**。
- 同一头衔下的岗位千差万别是常态；四分法量出的是位置，不是高下。

标准定好了，但一个新的问题立刻冒出来：这个责任结构，和售前、咨询、外包、客户成功这些"老邻居"的边界到底在哪里？这是第 2 章要讨论的事。

第 2 章 · FDE 和售前、咨询、外包、客户成功的区别，评判标准到底是什么？

> 名不正，则言不顺；言不顺，则事不成。
> ——《论语·子路》

"FDE 和售前、咨询有什么区别？"这个问题常被问起，答案却并不一致。有人说 FDE 更懂业务，有人说它更偏工程。

先看两个岗位。它们都要进入企业客户，部署一套能跑通业务的 AI 系统；都要写代码、接系统、盯效果。A 公司称这个岗位为"解决方案工程师"：他写演示代码，成交后离场，按人天报销差旅。B 公司把近似的工作称为"FDE"：代码要进生产环境，现场经验要回到团队，报价还和业务结果相连。

只看每天做什么，两份岗位描述几乎一样。再看合同与考核，两者承担的责任却不同。

这种情况很常见。不同公司的 FDE 本来就可能是不同角色；同一个人换一份合同，即使工作内容不变，角色也可能从 FDE 变成解决方案工程师。

既然名字不能定义角色，应该看什么？

一、岗位职责对比

最直觉的办法，是按"写不写代码、面向谁、什么时候离场"把相邻角色列一张表：

角色	写不写代码	面向谁	什么时候离场
软件工程师（SWE）	写，产品代码	自己公司的产品	持续迭代，不离场
售前/解决方案工程师（SE）	写，演示代码	潜在客户	成交即离场
咨询	基本不写，产出方案与规划文档	客户业务方	里程碑验收后离场
外包/驻场开发	写，客户指定的代码	客户项目	合同结束离场
客户成功（CSM）	基本不写	已成交客户	持续服务，不离场

售前也可能写生产代码，咨询也可能驻场一年，外包团队也可能很懂业务。表格列的是角色通常做什么，而例外恰恰出在"通常"上。一旦例外变多，它就不能再用来判断，只能提醒人们不要想当然（见图 2–1）。

图 2–1：同样接触客户，角色的差异在责任终点与经验是否回流。

为什么这些例外如此常见？

Cresta 的 AI Agent FDE 负责人钟钱杰说过："任何你能教会客户的，就不是 FDE。" [1] 他用传统 SAP 实施作对照。SAP 的规则明确，可以写进手册；客户学会后，内部团队能继续维护。这类工作可以完整交接，比较接近咨询或实施。

他所说的 AI Agent 不是这样。即使客户有数百名工程师，也未必能把幻觉率和响应延迟稳定控制住。问题不在于客户缺少资料，而在于判断会随模型和场景变化，无法一次写进手册。遇到这种情况，就需要有人持续在现场处理。

这句话揭示了表格的隐含前提：它假定"是否写代码""面向谁"等工作内容稳定不变。但工作复杂到难以交接时，角色边界会随项目变化。今天看似是解决方案工程师的工作，明天可能需要长期驻场；一项能完整交接的工作，无论叫什么，最后都会变成一次性的知识转移。

工作内容会变，因此不能只用内容判断角色。本章改看三项可观察的标准：代码是否进入生产环境、现场经验是否回到产品、收费按投入还是按结果计算。

二、评判标准一：代码的去向

第一问：**这个人写的代码，最后运行在哪里？**

[1] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

售前写演示代码，是为了促成成交。演示结束后，这段代码通常也就结束使命。FDE 写生产代码，是为了让系统持续运行。代码要通过客户的审查，进入部署流程，并接受客户的运维要求。

同样一行代码，去向不同，责任也不同。演示代码出错，可能失去一笔订单；生产代码出错，可能影响客户的业务。

AI 编程工具扩大了 FDE 能直接改动的范围。过去，一个 FDE 很难同时处理多个客户项目的不同代码库。现在，只要他说得清想要的结果是什么，就可以借助 AI 修改 TypeScript、Swift 或数据库相关代码［1］。这使他不仅能把代码交付给客户，也更容易把现场发现的问题和可行解法落回公司的产品线［1］。

三、评判标准二：经验的流向

第二问：**项目结束后，现场学到的经验去了哪里？**

经验可能有三种去处，也对应三种角色。

第一种是回到产品。FDE 把定制中发现的需求、遇到的问题和可复用组件带回产品团队，变成下一位客户可以直接使用的能力。这是 FDE 应有的流向。

第二种是进入报告。经验写成结项文档和案例库，为下一次提案提供素材。这更接近项目制咨询。

第三种是留在人脑中。人员离开后，经验也随之消失。这是外包项目常见的情况。

长期研究这类商业结构的从业者巴克利克·帕特尔（Bhaulik Patel）曾提醒：不回流的 FDE 组织，本质上是 **"藏在软件公司内部的高成本专业服务组织"**［2］，不管名片上写什么。

用这条标准看国内常见的一种形态：FDE 到客户处收集需求，现场做原型，确认后交给后端团队。经验停在交接文档里，因此按第二条标准，它更接近**原型外包**，而不是完整 FDE［3］。这不代表它低级。在特定采购结构下，这可能是合理分工（第 18 章会展开）。标准的作用是帮助定位，不是给角色贴褒贬标签。

四、评判标准三：报价方式与责任

第三问：**这个岗位的报价方式，按投入算还是按结果算？**

按人天计费，客户购买的是时间；这段时间产出多少，由客户自己判断。按结果收费，客户购买的是某

[1] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

[2] @deployengineer（Bhaulik Patel）系列长文

[3] 有在做 FDE 这个工作的佬吗（linux.do 帖）

个结果的改善；供应商也要承担一部分结果没有改善的风险。

两种计费方式没有高下，却会引导不同的行为。按人天收费的组织会关心人员是否被充分使用。按结果收费的团队会更关心问题是否真的解决。一位 AI 交付合伙人说得很直接：按日计费，增效的上行收益与自己无关［4］。

报价方式是三项标准中最容易核实的一项，因为它写在合同和发票上。岗位可以宣称重视客户成功，但收费方式会说明它实际承担了什么责任。

五、同一个项目的三种不同方式

以下示例，仅用于展示三条标准的组合用法。

一家零售企业要上"智能补货 Agent"。同一件事，三种合同结构，长出三种角色——

- **结构一：按人天外包。** 乙方派两名工程师驻场六个月，按天计费。工程师写的代码进了生产，但经验流向个人；效果没人担保，工时就是交付物。三条标准：①生产代码 √ ②不回流 × ③按投入 √——这是外包，驻场而已。
- **结构二：固定范围交付＋回流条款。** 同一批工程师，合同写明"定制过程中产生的可复用组件归乙方并回产品线"。经验开始回流，但效果责任仍在验收节点终止。三条标准：①√ ②√ ③混合——这是带产品责任的交付，已站在 FDE 门槛上。
- **结构三：结果分成。** 不达基线改善不收费，改善部分按比例分成。乙方主动砍掉伪需求、只做能测量的事。三条标准：①√ ②√ ③按结果 √——这是完整责任结构的 FDE。

这个演示说明，改变角色的不是工程师本人，也不是技术本身，而是合同结构和经验的去向。同一个工程师在三种安排里，承担的是三种责任。

六、三问定位卡

把三条标准收成一张可以自查、也可以在面试时反问雇主的卡片：

> **三问定位卡**
> 1. 这个岗位写的代码，上不上生产？（问工程负责人：上一个项目的代码现在还跑着吗？谁在维护？）
> 2. 项目结束后，现场经验去哪？（问面试官：上一个项目的定制组件，进产品线了吗？有没有"回流评审"这种机制？）
> 3. 我的考核指标是工时还是结果？（问 HR 与业务负责人：这个岗位背什么指标？采用率？留

[4] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

存？还是人天利用率？）

三问没有标准答案，关键看组织能不能答清楚。一个组织能明确回答三问，说明它的责任安排至少是清楚的，无论岗位是否叫 FDE。三个问题都说不清，头衔再新也无法说明责任由谁承担。

七、现实是骨感的

必须说实话：是否符合三条标准是"理想型"，不会非黑即白。

现实中的多数岗位都是混合体。有人七成时间写生产代码，三成时间支持售前；多数组织也只能做到部分经验回流。

因此，这些标准应该用来定位和规划。先看清自己或目标岗位在三条标准上处于哪里，再判断想往哪个方向调整，以及需要付出什么。

标准已经有了，它能把位置定出来。但位置定出来之后，新的问题跟着来了：这套责任结构，到底是新东西，还是老东西换了名字？

第 3 章 · FDE 是不是外包换皮、一阵风?

> 已有的事，后必再有；已行的事，后必再行。日光之下，并无新事。
> ——《圣经·传道书》1:9

一位从事 FDE 咨询的从业者直接承认："FDE 是新瓶装旧酒。"他把这门生意拆成"技术咨询顾问，加上 SaaS 驻场工程师"，"噱头里有真有假。"[1]

从业者自己这样说，本章似乎可以结束了。但这句话只说对了一半。本章要具体区分：哪些表现只是泡沫，哪些需求不会随着概念热度消失。先正面回应怀疑，才能把问题说清楚。

一、四层标准：项目结束后留下了什么

判断一个团队是不是 FDE，不必先看岗位名称。先看项目结束后留下了什么[2]。这个标准可以分成四层：

- **只给客户留下一个系统**：这是外包。系统交付后，合作结束。
- **带回一些经验，但下次无法复用**：这是项目制交付。经验变成结项文档，下一次提案时可以参考。
- **把经验做成 Skill、模板或产品能力**：这是 FDE。现场经验被整理成可复用的内容。
- **这些沉淀能明显降低下一位客户的成本**：这是可以规模化的 FDE。团队开始从过去的项目中持续受益。

这四层标准与第 2 章的第二项标准"经验的流向"一致，但多问了一步：经验回去以后，下一位客户是否因此做得更快。第 21 章会讨论定制如何递减，第 22 章会讨论经验如何回到产品；这里先不展开（见图 3–1）。

[1] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[2] 腾讯研究院《FDE 模式行业观察与实践》

图 3–1：判断 FDE 不是看是否驻场，而是项目结束后哪几层能力留下来。

开发者社区里，"FDE 是不是咨询换皮"的讨论不断出现。一位从业者说得更尖锐：不少"FDE 培训"是在"把 AI 的信息差，卖给有闲钱又焦虑的小老板" [3] 。

从业者承认它有旧的一面，媒体和培训市场又不断包装新概念。泡沫究竟在哪里?

二、泡沫的三种具体表现

把常见现象分开看，泡沫主要有三种表现：

第一，头衔膨胀。 把原有的实施岗、售前岗改名为 FDE，是最容易跟随热点的做法。最常见的情况，是把"会用 AI 编程工具写代码"包装成 FDE。中文开发者社区的一位用户说："不单纯在 cc 里 vibe coding 就能叫 FDE，这只是解决开发的问题。" [4]

第二，培训收割。 课程把概念名词和工具操作卖给焦虑的转型者，卖点是"再不转就晚了"。这并非 FDE 独有，每个新头衔都容易出现类似培训。FDE 的特殊之处在于，它的核心是承担真实交付责任；这种责任不能仅靠课程学会，必须在真实场景中练出来（第 25 章会展开）。

第三，组织内耗。 有些公司跟风设岗，却没有经验回流机制，也不按结果考核。FDE 最后只成了昂贵的驻场人力。有一些从业者提到的"藏在软件公司内部的高成本专业服务组织"，说的就是这种情况

[3] 有在做 FDE 这个工作的佬吗（linux.do 帖）

[4] linux.do 帖 #2580046

[5] @deployengineer（Bhaulik Patel）系列长文

[5]。

这三种情况有一个共同点：它们都处在三项评判标准里弱的那一档——不写生产代码、不回流、按人天计费。泡沫不在"FDE"这个词，而在于人们如何使用这个词。

三、真实的那一半

再看另一半。判断一个需求是泡沫还是长期存在的需求，不能只看热度，要看支撑它的条件会不会消失。

"Prompt 工程师"可以作对照。ChatGPT 带动生成式 AI 普及后，这个头衔一度很热，随后很快降温。原因在于提示词技巧依赖模型当时的弱点；模型每进步一代，其中一部分技巧就会失去价值。Prompt 工程是一种会被工具进步改变的技能。

FDE 面对的不是一项可以单独练成的技能，而是一段持续的合作关系：客户有真实的生产环境，供应商要把系统交付进去。在这种关系中，总要有人负责接入系统、让它运行，并让客户持续使用。模型变强，也不会自动摆平客户的权限、流程和利益分配。Agent 越深入业务，系统集成反而可能越复杂[6]。

财务上还有一条旁证。一位在 Palantir 工作多年的工程师回忆，公司 2023 年的毛利率约为 80%，传统咨询巨头埃森哲约为 32%[7]。如果 FDE 只是换了名字的驻场外包，利润结构应更接近咨询公司；而它呈现出的特征更接近软件公司。差别来自哪里？来自现场经验被反复整理进产品。这正是四层标准的第四层，第 22 章会解释其中的机制。

四、结论：对一半，错一半

本章的结论分两半，各自成立：

- **需求是真的。**把 AI 安全接入客户生产环境，是长期存在的需求。生产环境复杂，组织之间也需要建立信任；这些条件短期内不会消失。
- **角色未必都叫 FDE。**FDE、能力升级后的咨询团队，或客户自己的内部团队，都可能承担这项工作。市场会改变头衔，概念热度会下降，但问题还在。

至于这个概念还能流行多久，哪些工作会被 AI 替代，头衔本身会不会消失，分别留给第 5 章和第 28 章回答。

这里还剩一个关于时间的问题：如果它不是一阵风，为什么偏偏在 2026 年刮成这样？第 4 章把 2026

[6] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

[7] Reflections on Palantir（Nabeel Qureshi）

年上半年发生的事展开。

第 4 章 · 为什么 FDE 在 2026 年突然火起来?

> 人们总是高估未来两年的变化，而低估未来十年的变革。
> ——比尔·盖茨《未来之路》（1995）

2026 年 5 月，两个星期一之间发生了两件事。5 月 4 日，Anthropic 宣布与黑石、H&F、高盛组建交付合资公司［1］；5 月 11 日，OpenAI 成立控股子公司 DeployCo，据报初始资金逾 40 亿美元、首日即有约 150 名 FDE 到岗［2］。

这件事热度很大。让我们先用一条**事件时间线**，看清 2026 年的 FDE 到底为什么火。

一、时间线：2003 到 2026

时间	事件
2003	Palantir 成立；2000 年代中期 FDE 实践成形
约 2007	早期员工首次进驻客户保密环境，驻场交付形态成形
2026–02	微软发布 Agent Factory 白皮书；富士通在投资人简报中披露 AI 驱动开发投入
2026–03–18	埃森哲×微软联合推出 FDE 实践服务，官方口径"数千名 AI 工程师"直接驻客户
2026–05–04	Anthropic 组建交付合资公司（黑石/H&F/高盛参与）
2026–05–11	OpenAI 成立 DeployCo（控股子公司；据报逾 40 亿美元、收购约 150 名 FDE）

把这张事件表拉成两条曲线看得更清楚：集成与采用的复杂度从 ERP 时代就存在、从未真正消失，AI 模型能力只是 2023 年之后才被画出来的新变量（见图 4–1）。

[1] Anthropic 与黑石、H&F、高盛组建企业 AI 服务合资公司

[2] Forward deployed engineer is AI's hottest job as OpenAI and Google race to hire

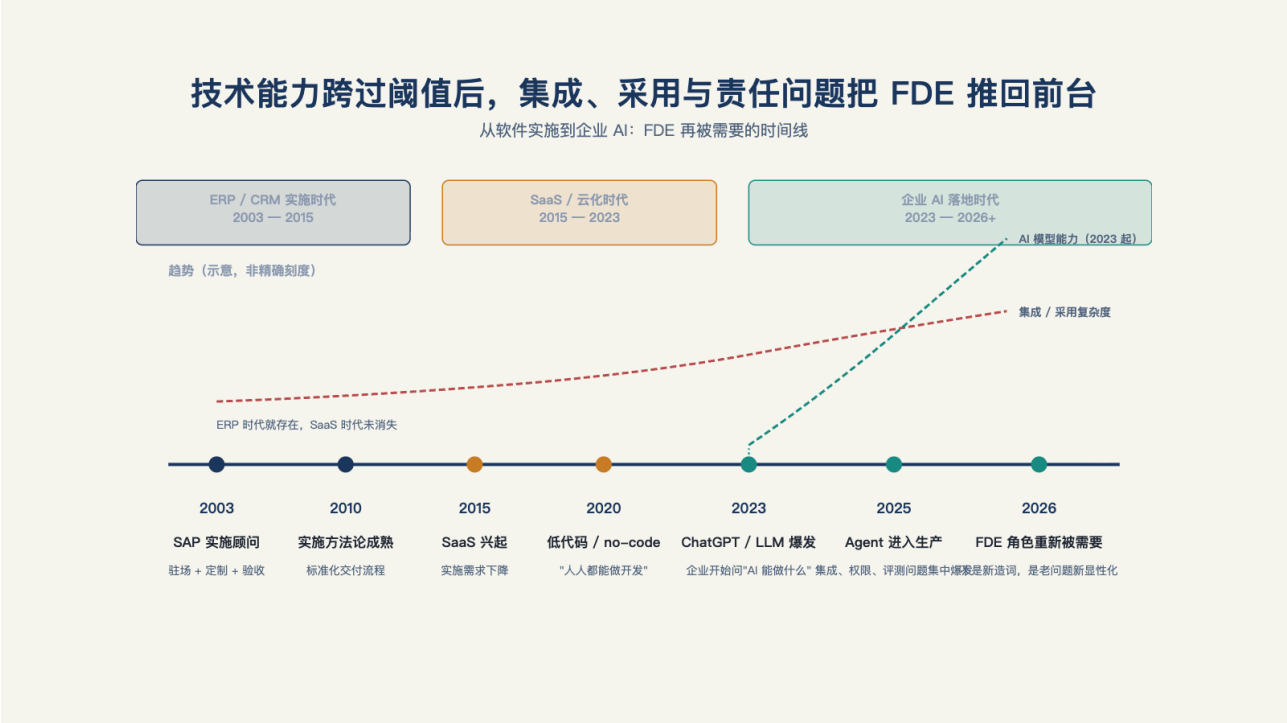


图 4–1：技术能力跨过阈值后，集成、采用与责任问题把 FDE 推回前台。

二、时间线解读

把时间线里的原始事实连起来看，能读出层层加码的解读。

这几层解读指向同一件事：咨询巨头、模型公司、云厂商、FDE 概念的原创者，这几类原本并无交集的组织，几乎在同一个窗口期，各自摸到了同一个市场痛点、找到了同一个切入点——这块地带已经大到没有谁能继续忽略。

第一层：咨询巨头入场，FDE 从战术变成采购品类。 当埃森哲把它做成主力服务线［3］，"前线部署"就不再是某家产品公司的独门打法，而成了企业采购目录里的一个标准条目——甲方可以像买标准品一样，走预算、走招标、走验收把它买进来。品类化抬高了门槛：后续进入者的起点也提高了。

第二层：模型公司下场，是"只卖 API 是远远不够的"。 能力最强的模型公司亲自组建交付组织，等于公开承认：光有模型能力，还是实现不了客户价值。交付是模型商业化的下半场——这恰好是前言那个"卡点存在"论点的企业级版本。

第三层：云厂商同样下场，逻辑是"算力要落进生产环境才变成收入"。 云计算厂商卖的是算力，但客户买算力的前提，是能把算力真正用进生产系统——一堆闲置的 GPU 配额换不来续费。这个动机与模型公司如出一辙：只卖底层能力不够，得有人把能力接进客户的实际业务，付费才能持续。

[3] 埃森哲推出微软前线部署工程实践

[4] Palantir Foundry「AI FDE」产品文档

第四层：FDE 概念的原创者，开始把它做成 AI 产品。 Palantir 的"AI FDE"智能体——自然语言操作、闭环执行、遵循用户现有权限［4］——意味着 FDE 概念开始自我迭代：真人 FDE 的执行工作，正在被装进产品里。

这几层叠起来，回答上面的问题：2026 年的密集下场不是舆论事件，而是一连串动作——市场正在把"交付能力"当作独立于"模型能力"的稀缺品来定价。

三、通用技术的生产率红利，为什么总是来得更晚

上面的解读回答了"谁在入场"；但一个扎心的问题绕不开：整个社会的资本开支已经这么大，为什么整体生产率数字还没有明显跳升？这种质疑不能只靠"AI 很重要"来反驳，放进更长的技术史里，反而看得更清楚。

经济史学者 Paul David 记录过工厂电气化的滞后：电力从 1880 年代进入美国工厂，到生产率明显释放，中间隔了大约四十年［5］。原因不是工厂不会用电，而是早期工厂只是把蒸汽机原地换成电动机，却保留了天轴、传动带、多层厂房这套围绕蒸汽机设计的旧结构。电的真正优势——分布式供能、工序独立调度、更灵活的厂房布局——要等到厂房本身按电力重新设计之后，才被释放出来。据该文献转引的历史统计，1899 年电动机在美国制造业驱动功率里的占比还不到 5%，1919 年前后就已升至半数以上，到 1929 年达到四分之三左右——设备换装本身并不慢，慢的是厂房与工序按电力重新设计［5］。

个人电脑经历过同一个阶段。经济学家罗伯特·索洛 1987 年那句被反复引用的书评——"计算机时代无处不在，唯独不在生产率统计里"——说的不是电脑没用，而是当时的企业把电脑当成更贵的打字机：在电脑上打字、打印、装订，再交给人归档，流程本身没有变［6］。

不同技术不能机械类比，但这段技术史给出一个可核验的提醒：通用技术往往先被采纳、再被塞进旧流程，最后才在组织被重新设计之后，显现为生产率的跃升。

今天 AI 容易被误读，正是因为模型能力的进步按月发生，看得见；组织重构的代价——部门边界、预算、权限、绩效怎么重新划——按季度甚至按年发生，看不见。

生产率数字暂时没有跳升，证明不了 AI 没有价值，而恰恰说明组织重构还没追上模型能力——这恰好呼应上一节的解读：各家不约而同选在这一年往"交付"上砸真金白银，正是在补组织重构这一课。

四、降温收尾

[5] The Dynamo and the Computer: An Historical Perspective on the Modern Productivity Paradox（Paul A. David）
[6] We'd Better Watch Out（书评）（Robert M. Solow）

一批结构不同的组织——咨询巨头、模型公司、云厂商——在同一个时间窗口里，对"交付"做了真金白银的结构性投入。这类投入为什么必要，技术史上电力、个人电脑走过的路已经给出过先例。

岗位会不会长期存在、概念会不会退潮、今天入场的人会不会获益，是另一个问题——时间线证明的是投入，不是回报，回报要看第五部那本账，以及第 28 章那个终局问题。

时间线本身还留着一个伏笔，藏在第四层解读里：概念的原创者已经把 FDE 做成了 AI 产品。发明这个角色的人，正在用 AI 替代它的执行部分——这个角色还剩下什么，是本章事实摆完之后绕不开的下一个问题，第 5 章正面回答它。

第 5 章 · FDE 会被 AI 取代吗?

> 计算机永远无法被追责，因此计算机永远不应该做管理决策。
> ——IBM，1979 年内部培训材料

把 FDE 这个概念带进世界的公司 Palantir，2026 年亲手把它做成了 AI 产品：Palantir 的"AI FDE"智能体，用自然语言操作 Foundry、自动执行、闭环运行，且所有操作遵循用户现有权限 [1] 。概念的原创者，率先把自己的执行工作 AI 化了。

这听起来像"FDE 已死"的讣告，其实是本章的第一份证据——它精确演示了 AI 吃掉的是什么（任务），吃不掉的是什么（责任）。把这两样东西分开，是本章讨论的前提。

FDE 之所以存在，本来就是因为它站在**技术能力**、**业务价值**、**成本可接受**三者的交叉地带——只有"AI 能做到""客户愿意为此付钱""投入产出划算"三个条件同时成立，才需要一个人去把它接进生产环境。模型能力每往前一步、推理成本每降一格，这块交叉地带就扩大一圈，AI 能独立覆盖的业务范围也跟着变大。理解这一点，才看得清"AI 会不会取代 FDE"这个问题问的到底是什么——不是这个角色会不会消失，是这块地带的边界，往哪个方向移动。

[1] Palantir Foundry「AI FDE」产品文档

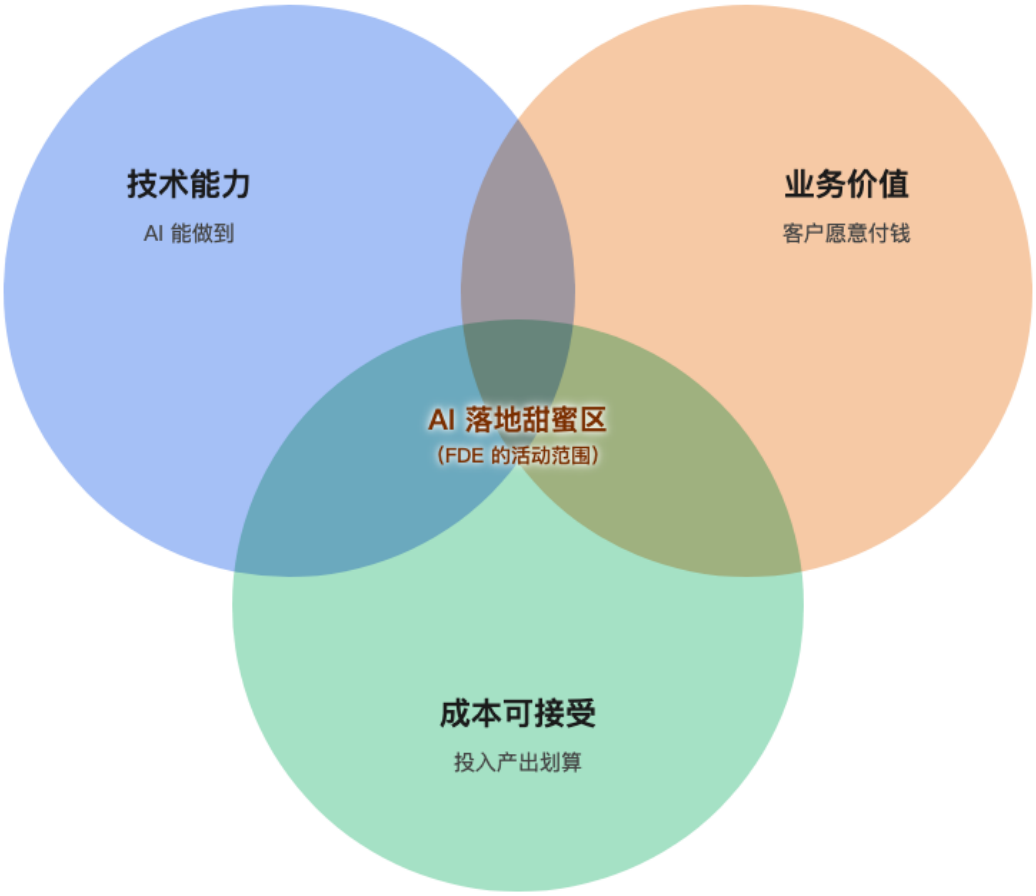


图 5–1：技术能力、业务价值、成本可接受三个圆两两重叠不难，难的是三者同时重叠——这块交叉区域，就是 FDE 的活动范围。

一、AI 会取代 FDE 吗?

AI 取代的从来不是岗位，是**任务**；而一个岗位是不是被取代，取决于它剩下的任务还够不够撑起一份责任。或者换一种问法：FDE 的工作里，哪些任务正在被吃掉，哪些吃不掉?

把 FDE 的工作分解成任务清单，逐层标注，得到三档：

档	任务	状态
正在被 AI 化	写集成代码、样板对接、生成初版文档、演示材料、常规数据处理	大量被吃掉
部分被 AI 化	评测设计、失败判定、方案比选、数据契约起草	AI 出草稿，人定稿
难以被 AI 化	责任承担、客户信任的建立与维护、权限担保、验收谈判、对采用负责	基本未动

第一档的清单还在变长，而且速度在加快——这不是在说威胁，是在说杠杆。真正的分界线在第三档，值得逐条问一句"为什么"。

二、责任层为什么难被替代

第一，B 端采购的本质是风险转移。 客户买外部团队的东西，买的不只是能力，是"出了事有人负责"。模型不能签合同，不能被追责，不能持有权限——这些不是技术限制，是法律与组织结构的限制。一个不能被告的主体，接不住"对结果负责"这个契约。

第二，权限是人的属性，不是模型的属性。 一线从业者反复观察到的现实是：Agent 只拥有其背后用户的原有权限，当进度卡在权限问题上，也不会像真人同事那样"找谁谁谁帮忙"绕过去，所以常常死在企业落地的第一步（具体将在第 10 章展开）［2］。权限体系信任的是"人＋身份"，AI 要继承这份信任，只能通过人。

第三，判断力是人留在决策环上的前提。 一位从业者的说法扎得准：服务商赚的是"审美判断的钱、搭建工程体系的钱"；他还补了一个观察——当人的能力不如 AI 时，人无法辨别 AI 输出的好坏，只能被 AI 压制，成为"AI 傀儡"［3］。反过来说，人要留在决策环上，前提就是判断力跑赢自己手里的工具。

第四，有些信息只能靠人去现场采。 硅谷一位从业者给过一个论据：他白天与客户开会、吃饭，把 AI 拿不到的隐性信息和默契捞回来；晚上把这些信息转化成 AI 的上下文；AI 能给出多套方案和一张利弊对比表，但最终选哪一套、并为这个选择承担后果，还得是人——"人是 AI 的触手"［4］。

四条合起来，正是题记那句话背后的完整逻辑：计算机永远无法被追责，所以责任结构必须落在一个能被追责的人身上。AI 越来越擅长**把事情做对**（执行层），但"**做对的事情**"（判断与担保层）这份责任，仍然需要一个可以被信任、被追责、持有权限的人来扛（见图 5–2）。

[2] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

[3] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[4] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

图 5–2：工具先替代可标准化动作，FDE 的价值随现场责任与产品化能力上移。

三、FDE 的三种做法

举个例子，看看不同阶段里 FDE 怎么工作——场景为演示性质，不对应具体公司的真实操作。

客户说："把这季度的销售数据补齐，下周一要用。"

十年前的 FDE（人）：飞到客户现场，找 IT 谈数据权限，写脚本从三个系统里抽数、清洗、对账，用 Excel 交给客户——一周。学到的是这家客户的数据长相与流程习惯，都在他脑子里。

如果完全交给 AI（技术上限，按 Palantir 文档描述）：收到自然语言指令，AI 理解意图后生成 Foundry 操作，按用户现有权限执行数据管道，全程留痕，异常时暂停等待澄清——同样的事，当天。它不知道也不需要知道这家客户的组织历史；它继承了用户的权限边界，也继承了权限的所有限制。

今天的 FDE（人＋AI）：人把指令拆成 AI 可执行的部分，盯住异常与审批节点；执行的部分，交给 AI 去放大。人的一天从"写脚本"变成"定义问题、审核动作、处理例外"——正是第二、三档任务。

对比这三种做法，结论落在人身上：组合形态里，人的单位产出被 AI 进一步放大了，但放大的前提是**人站在判断与担保层**。

站在执行层的人，会越来越没有位置。

四、结构的演进

先看两个尺度。

往小了看：第 1 章那位同时挂着十个客户的工程师，靠工具杠杆同时照看十个客户——这已经是"人＋AI"组合的雏形。下一步是 Agent 杠杆：人只做判断、担保、客户关系，Agent 把执行放大。他的"十"会变成更大的数字，但他的身价从来不押在这个数字上，押在他站的那一层。

往大了看：麦格鲁有个说法很妙——今天的 AI 初创公司，其实就是基础模型公司的 FDE：模型公司造能力，AI 公司把它装进具体行业 [5] 。这个结构会一层层往上递归：能力和应用之间永远有落差，只是落差不断被推到更高的地方——"把能力送到最后一公里"这件事不会消失，只会换个楼层重新出现。

那终局呢？不妨设想一下：如果到了那一天，模型已经发展到极限，不再进步、成本也降无可降，这块地带的扩张就会停下来。

那么，FDE 大概率会分流成两条路线：一条沉淀成行业 SaaS，卖标准化软件；一条沉淀成专业咨询，卖标准化服务。今天的 FDE 两头都不是——正因为地带还在扩张、标准化的时机还没到，它才必须留在中间，在技术和产品暂时够不到的地方打前站。这块中间地带能撑多久，就看模型还能进化多久。

五、最后

接下来两句话说给两种人听。

已经在做 FDE 的人：别把身家押在执行层——它正在被吃，不是风险预告，是进行时。把重心放到判断和担保层：评测的眼光、验收的设计、信任的经营、责任的胆量。怎么练，将在第 24 章里深入讨论。

准备入场的人：好消息是门槛在往下掉，坏消息是"会用工具"越来越不值钱。想清楚一件事：你要进来，是因为"我愿意担责任"，不是因为"我会用这些新玩具"。

但知道是什么和会不会做，是两回事——下一章开始进现场：客户说完"我们想要 AI"之后，第一件要命的事，是从一堆模糊的许愿清单里，挖出那个真正值得解决的问题。

[5] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

第二部分 · 问题是怎么被解决的

> 本部分共三章，回答一个总问题：**从客户说出"我们想要 AI"，到一套系统真的在生产环境里跑起来，中间这段路怎么走。**

三章是一条流水线，上一章的产出是下一章的输入：先找对问题（第 6 章，发现不是头脑风暴，是筛选——把客户模糊的愿望清单先撑大、再过五道关，收成一个值得做的切入点；选错问题，后面做得越好浪费越彻底）；问题定了，做出第一个"真"的版本（第 7 章，最小可行部署（MVD，Minimum Viable Deployment）——真实数据、真实用户、真实基线，测出真实差距，才是这一步真正交付的成果）；差距测出来了，才答得上客户必然会问的那句话（第 8 章，"什么时候可以上线"——上线不是一个技术时刻，是一份双向验收契约）。没有第 6 章五道关剩下的问题，MVD 无从"最小"；没有第 7 章测出的基线，第 8 章的验收单上无字可写。

本部的第一读者是**马上要进现场的人**：乙方交付者、要带队落地的技术负责人，以及要在评审桌上提问和被提问的甲方负责人。把系统搬进生产的工程细节——数据、权限、评测、账单——是第三部的事；这门生意怎么收费，是第五部的事。本部只走一段路：从一个模糊的愿望，到一份可以签字的验收。

第 6 章 · 进入客户现场，怎么找到值得解决的问题？

> 在拿到数据之前就建立理论，是天大的错误。
> ——阿瑟·柯南·道尔，《福尔摩斯探案集·波希米亚丑闻》

在现场第一次开会，客户坐下来就给你列出七个愿望：智能报价、会议纪要、质检拍照识别、供应商邮件分类……每个都有人举手支持，每个听起来都能"用 AI 做点什么"。

听起来像是“许愿池”，但是可交付团队只有人力做其中一个。

怎么选？许多项目的失败，不是因为做得不好，而是因为一开始就选错了问题。问题选错了，团队做得越卖力、浪费得越彻底。

本章就讲这个问题：真问题去哪儿挖，挖回来怎么过五关，五关走完剩下什么。

一、真需求，往往不是在开会发现的

先说发现需求。多数项目把"发现"理解成“访谈”：约一串会，记一堆需求，回办公室整理成文档。这条路径最大的风险不是问漏了什么，而是**你听到的每句话都经过了翻译**，已经不是“第一手”的数据了。

一套流传很广的田野方法把这一步叫"影子工作法"：跟着真实用户，过完他真实的一天——不是采访他，是坐在他旁边看他工作，看他打开哪些系统、在哪些表格之间复制粘贴、在哪个环节皱眉、绕过了哪条"官方流程"。观察的重点是**变通，而不是流程**：官方流程图告诉你组织"应该怎么运转"，变通告诉你组织"实际怎么运转"。这套方法在人类学里叫参与式观察，在丰田生产方式里叫现地现物——OpenAI 的 FDE 团队在约翰迪尔的项目里就是这么干的：飞到爱荷华州，跟着农艺专家和农场主下地，看施药决策怎么做、哪些信息真正进入决策、农时这道硬期限如何支配一切。他们最终替代的那套旧流程——农艺师挨家挨户打电话、口头给设备使用建议——不会出现在任何文档里，只有下地才看得见 [1]（见图 6–1）。

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

图 6–1：真需求藏在真实动作、例外处理和未写下的规则里。

为什么必须亲眼看？因为**用户自己说不出来**——说不出来有两层。

一层是不肯直说。直接问"你觉得这个方案好不好"，得到的多半是客气话。创业圈有本方法论小书专门讲这件事：Rob Fitzpatrick 的《The Mom Test》，书名来自一个损招式的提醒——别问你妈"我这个生意点子好不好"，她因为爱你，一定会说好。客户的恭维同理。解法是把问题从观点挪到事实：不问"你觉得需要吗""将来会用吗"，只问"上一次遇到这种情况，你是怎么处理的"。问过去，不问将来；问具体的一次，不问一般情况——客气话没法核对，事实可以对质。

另一层是真的不知道要说。士兵需要路边炸弹预警工具——这个需求在任何访谈里都问不出来，因为士兵不知道"原来软件可以提供这个功能"，他们以为巡逻的忐忑本来就是巡逻的一部分［1］。国内一位企业 AI 落地顾问的说法异曲同工：痛点很少挂在嘴边，"它藏在每天重复的手工活里，藏在某个岗位没人愿意接的烂摊子里。挖掘不能靠问卷，得靠现场泡"［2］。

听不懂的时候，有个笨办法：连问三层为什么——第一层问清楚要什么，第二层问为什么现在要做，第三层问如果不做会怎样。问到第三层，真问题往往才浮上来。客户说"我要个知识库问答"，三层问下去，真问题可能是老专家快退休了，三十年的经验带不走，新人上手要半年，期间经常出错——表层需求是知识库，深层问题是经验传承。

还有一层失真更隐蔽：**你面对的根本不是使用者**。需求方是中间的数字化团队时，"你以为在给使用者做系统，其实全程对话的是一群背着数字化指标的人，他们转述一线诉求时已经过了一道滤镜"——解法也很简单，就是绕开中间层直接触达一线，"哪怕只访谈两三个真实使用者，得到的信号也比十场中

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[2]《关于 FDE 的 100 个问题》（AI磊叔）

层会议扎实"；实在见不到，就把需求假设一条条列出来，让中间层逐条确认这是一线真要的，而不是他以为要的［2］。

举一个真实例子：某公司用智能体处理退款请求。它读取每一项请求，逐条对照退款政策，拒绝所有不符合条件的申请——每笔拒绝都"正确"。几周后，合作多年的老客户开始流失。查不出原因，因为智能体拒掉的每一笔，确实都是政策规定该拒的。最后有人去和从前经手退款的人坐下来聊，才发现他们心里有一条从没写进任何文档的规矩：用企业信用卡付的订单，直接批——这些客户每月都采购，为一笔退款斤斤计较，可能丢掉整个客户账号。政策里对此只字未提，是处理者们多年前自己摸索出来的经验。解法不是换模型，是把这条检查固化成工作流里的固定规则，放在模型判断之前［3］。

这个案例把发现阶段的本质暴露得很彻底：**模型正确不等于业务正确，中间隔着一层没写下来的流程知识**。这类知识不在需求文档里，不在系统日志里，只在每天经手这件事的人的手上。同类的案例还有一个：一次文件格式迁移停滞了整整一年，所有人都以为是技术问题，直到有人坐到那位抵制迁移的工程师旁边看他干活——他一直靠双击文件看数据，而新格式没有可以双击的东西。团队当晚给他做了一个能照旧双击的工具，两天后他批准了迁移［3］。

所以发现阶段需要懂系统的人最先入场。一家投资机构的数字化负责人复盘内部实践时说，做 AI 技能封装，"最关键的是你要知道你想封装什么"——而想清楚这件事的架构能力，需要公司中层甚至更高层的人先参与进来，再去指导［4］。访谈负责收集愿望，架构师负责判断愿望背后的真实诉求——两件事得同时做。

二、第一关：这个问题重要吗

挖回来的清单，先别急着评估技术可行性。第一关问的是：**这个问题，在客户组织里够不够重要**。

Palantir 出身的高管鲍勃·麦格鲁（Bob McGrew）讲得非常直白："传统实施的范围，往往从接近产品现有能力的地方起步，但你想解决的是领导层认定的关键问题之一。如果你解决的不是 CEO 的前五大优先事项之一，那很可能行不通——他们大概没有精力陪你走完那条艰难得多的路。"［5］他给的理由完全不是技术性的：问题要是需要本地部署，你就得和客户的 IT 团队周旋；组织内部每一层都有"不点头你就动不了"的人，而这些人不像初创公司那样思考、跟最终用户的目标也不一致，你必须想办法绕过他们——所以你解决的问题必须重要到"能够请动高层的人来拍板，授予他们操作权限，允许他们使用"［5］。

重要性门槛的另一个理由是选战场：AI 要动的是哪条核心生产线——研发、合同与交付、制造质量、销售运营还是客户服务里的哪一个环节？它得跟交付、收入、毛利、风险或客户体验直接相关，而不是只"看上去省时间"；宁要高难度高价值的，也不要容易做的，否则收益非常有限。

[3] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

[4] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

[5] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[6] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

这和国内一些从业者的看法不谋而合，他们认为：现有 AI 方案能做到的只是"有限范围内的降本增效"——在旧流程上加一点 AI，效率高一点、成本低一点，"你公司原有的那些流程、人员、业务是不变的"；而"人才和组织不变革，AI 的能力就是发挥不出来"[6]。

反过来说，值得 FDE 化的问题，多半是那种顺藤摸瓜会摸到流程和组织的问题——这样才配得上 CEO 的前五大问题，也才请得动拍板的人。

Palantir 早期一个客户把第一关走成了样板。那位 FDE 工程师的第一家真正客户是空客的图卢兹工厂，他搬过去住了一年，每周四天泡在产线上。切入点不是"我们能演示什么 AI 能力"，而是 CEO 直接告诉他们的最大问题：A350 量产爬坡。团队据此只做了一件事——把分散在多个系统里的工单、缺件、质量问题（"不合格项"）整合进一个界面：可以勾选任务、追踪零件、查生产计划，还能模糊搜索历史质量问题。说穿了都是基础软件功能，但把"最佳实践"的界面真正铺进生产环境，最终帮助把制造速度提到了约四倍[7]。

方向是从 CEO 关心的最大问题倒推的，不是从技术能力正推的。

这道关为什么排在最前面？因为重要性不够的问题不解决，后面每一关都白过——验证做成了没人认领，部署卡住了没人拍板。这个问题会在第 16 章详细讨论：AI 落地为什么是一把手工程。这里先记住这个问题：**这事在 CEO 的前五大问题里吗？**

三、后四道关

过了第一关，剩下的问题再过四道关：**价值密度、可验证性、数据可达、变更容忍度**。加上重要性，一共五道关——五道关不是并列的清单，是层层收窄的通道，每一关都在砍掉一类特定的死法（见图 6–2）。

[7] Reflections on Palantir（Nabeel Qureshi）

图 6–2：被拦下的不是坏主意，而是此时还不具备落地条件的主意。

第二关：价值密度。 现场问三句话就够：这事现在靠谁在做？一天做几遍？做错了会怎样？三句答出来，问题的"体质"基本就定了。

判断价值，有三个朴素的信号：高频发生——每天都在做的事，优化一点就有大收益，一年只发生三次的事，效率提升十倍也省不了多少；现在靠人肉——全靠人工、经验、老师傅的地方，才有 AI 切入的空间；能好十倍——好一成没人愿意换，好十倍大家抢着用。

所以"只快一成的场景，碰都别碰，把力气留给高频、靠人肉、能好十倍的地方" [2] 。别忘了本章开头那份七个愿望的清单：价值密度不只算"省几个人"的账，还算"打开过去够不着的市场"的账 [8] 。

第三关：可验证性。 这个问题现在的基线是什么——基线也就是现状，当前人工做一遍耗多少时间、错误率多少、成本多少？改善拿什么测？没有基线的问题，先建基线，再谈 AI。

这一关拦下的是那种"做完大家都叫好、半年后说不清值不值"的问题：效果没法量化，预算季一来就成了第一个被砍的项目。有经验的团队会把"验收指标"放在"上线"之前——从具体场景起步，让业务的人和工程师逐条给输出打分，分数回流到迭代里。验收指标怎么建，会在第 12 章里专门讲。

第四关：数据可达。 解决这个问题需要哪些数据？数据在几个系统里？权限归谁？要多久的审批？这个问题要在发现期就摸清楚，不能拖到生产期。一个真实案例，某企业内部上马的客服 Agent 项目迭代到第二版才把真实客户数据接进来试，数据安全的顾虑立刻摆上桌面，最后被迫改成预算内的本地小模

[2] 《关于 FDE 的 100 个问题》（AI磊叔）
[8] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）
[9] AI 职觉：别再卷 Agent 了（小宇宙播客）

型部署，模型能力因此大打折扣［9］。数据安全的约束是真实的，但发现期就该把它摸清、把应对设计进方案，而不是留到生产期才发现。画数据地图时要特别记一笔：**几乎每个组织里，官方数据源和员工真正信任的数据源都不是同一个**——某个老员工私藏的表格、只在邮件里流转的报表，往往才是干活的人天天用的那份；接错了源头，系统再准也没人信。

第五关：变更容忍度。 方案落地后，谁的工作被改变？这个问题是某位主管的权力来源吗？自动化掉的流程，原来养着谁？这不是人事八卦，是项目生死：系统上线演示那天，全程配合的部门在鼓掌，角落里另一个部门的主管一言不发——他要维护的那套流程，正是被你自动化掉的那套。成熟的做法是在方案设计里就给利益受损的人安排"新生路"而不是"死路"——把把关人转型成教练，让老师傅的经验变成训练系统的素材［1］。这道关排查出的问题清单，第 15 章讲采用、第 16 章讲一把手工程时都要回头用。

关也有闯错的时候，四类错误最常见：**从最高价值的问题切入**——通常不可验证，一上来就赌大的；**从最好做的问题切入**——通常没人真痛，做成了也是自嗨；**从 CEO 的个人偏好切入**——老板拍脑袋指定的场景，往往既不高频也不可测，还没人敢反驳；**接下"先做个 Demo 给董事会看"的单**——演示的目标是好看，生产的目标是可用，两套目标做出来的东西从头就不是一回事。

四、五关走一遍

下面是一个脱敏后的真实案例：

客户是一家中型制造企业，进场时带来七个愿望——智能报价、会议纪要、质检拍照识别、供应商邮件分类、设备维修知识库、销售话术助手、合同审查。

先过第一关：这家客户的 CEO 今年只关心两件事——交付延期和原材料成本。七个愿望里，智能报价和合同审查直接挂在交付与成本上，设备维修知识库沾边（停机影响交付），其余四个都是"各部门顺手提的"。关外砍掉四个。

第二关（价值密度）：智能报价每周发生几十次，每次销售要花两小时翻价格表和历史成交，错了直接亏钱——高频、人肉、能好十倍，过。合同审查一年三十来份，单份审查两小时——真痛但不高频，这一轮先挂起。维修知识库：停机时每分钟都在烧钱，但"知识库"本身不解决停机，先退回重问一层为什么（真正的问题是老师傅退休经验带不走？还是备件查找太慢？）——挂起待进一步挖掘。

第三关（可验证性）：智能报价有干净的历史数据——过去三年的报价单和成交价就是基线，改善可测，过。

第四关（数据可达）：价格表在 ERP 里，历史成交在 CRM 里，权限分属两个部门——能接，但审批要两周，发现期就把申请提上去。过，带条件。

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

第五关（变更容忍度）：报价流程的销售总监是 CEO 的老部下，明确支持；销售团队多一步"确认 AI 建议价"，可以设计成默认采纳——过。

七个愿望，过五关、斩六将，剩下一个：智能报价，一句话问题（"销售报价从两小时缩到十分钟，错误率不升"）、一个基线（当前平均耗时与改价率）、一条数据权限清单（ERP 价格表＋CRM 历史成交，两周审批）。**砍掉六个的过程，比选中一个更有信息量**——每个被砍掉的愿望都留下了记录：为什么现在不做、缺什么条件、什么时候可以重提。这份记录本身就是下一季度的路线图。

五、发现阶段的交付物：三件套

问题发现做完了，在这个环节应该交付什么？

不是 PPT，是三样能让下一周直接开工的东西：

1. **一张问题卡**：一句话问题＋当前基线＋目标指标——就一页，写不进一页说明还没想清。
2. **一份数据可达性清单**：需要哪些数据、在哪个系统、权限归谁、审批多久——第四关（数据可达）的原始记录。
3. **一个可交互的产品原型**：不是演示视频，是让客户的人亲手点一次的东西。一位入职两个月的日本 FDE 写下他的现场心得："比起完美的规格书，（更要）能上手摸的原型。看着客户的反应，当场修正方向。"［10］规格书锁死方向，原型暴露方向——客户摸到实物时说的第一句话，常常推翻前三次会议的共识（见图 6–3）。

图 6–3：问题卡、数据可达性清单和可交互的产品原型，让发现可以进入下一周的交付。

［10］「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实（note.com · AI Shift）

三件套到位，发现阶段结束。那张问题卡上的一句话问题，接下来要变成一个真的跑在客户环境里的系统——第一个版本做多小、做多真，这正是第 7 章要重点讨论的问题。

第 7 章 · 什么才算"最小可行部署"，而不是又一个 Demo?

> 先让它能跑，再让它对，最后让它快。
> ——肯特·贝克（Kent Beck），软件工程师

问题卡到手后，业务方的下一个问题就是："那什么时候能看到结果？"

你的第一反应是不是"先做个 Demo 给你看看"?

但这里有个常见误区：许多团队把 MVD（最小可行部署）理解成"Demo 的小号版本"——功能砍掉七成，做成四不像。结果既验证不了什么，又浪费了时间。

Demo 追求"看起来能行"，MVD 追求"测出真的能不能行"。

这是两回事。怎么样的"小"才对、怎样做才能避免这个误区，这是这章要讲清楚的。

一、差距

一家保险行业的企业级 AI 应用公司从业者公开写过他们的第一天：新智能体第一次跑在客户的真实工作流上——读一堆杂乱的入站文档、做结构化和推理、把结果写回一套老旧的保单管理系统——客户自己的专家打分，约 70%；而合同里承诺的数字，是超过 98% [1] 。他的原话很有意思：我的工作里所有有意思的东西，都活在这道差距里 [1] 。

这 28 个点为什么重要？因为在演示环境里，你永远看不到它们——演示数据干净、边界友好、没有历史遗留，跑出 100% 是常态。**差距只在真实环境里显形**，而问题发现得越晚，代价越大：签了 98% 的合同才发现起点是 70%，中间的每一分都要在客户的注视下补。

MVD 的存在理由就在这里：它不是为了交付一个缩小版的系统，是为了**尽早测出这段真实差距的形状**——用最小的工程投入，在客户的真实环境里，对着真实的痛点，验证价值能不能真的发生。

顺带把 MVD 和 MVP 的差异说一下：MVP（最小可行产品）验证"要不要做这个产品"，裁判是市场；MVD 验证"这个方案在这个客户身上能不能产生价值"，裁判是这个客户的具体业务方。

这一字之差，差在裁判。这也解释了企业交付的残酷之处：一个在之前十个客户那里验证成功的方案，到第十一个客户那里仍然可能失败——数据基础不同、组织惯性不同、痛点的形状不同。价值没法从上

[1] Inside an Applied AI company（Pace 长文）

一个客户继承，只能在这一个客户身上重新验证。所以 MVD 没有模板答案，只有方法论。

二、三个"真"

MVD 与 Demo 的分水岭，展开成三个"真"：**真实数据、真实用户、真实基线**。少了任何一个，交付的还是一个换了名字的 Demo（见图 7–1）。

图 7–1：MVD 缩小的是范围，不是生产验证所需的真实条件。

真实的数据。 用客户脱敏的样例数据、或自己构造的演示数据做验证，是概念验证坟墓的第一块砖。原因不是仪式感，是真实数据里全是坑：字段含义与文档不符、三成的空值、三年前的编码规则，以及最要命的——数据本身记录着错误的流程。在假数据上成立的方案，上线那天就会撞上文档里不存在的字段，而那时候已经付了真价钱。所以硬性要求只有一条：客户必须带自己的真实业务数据来；可以脱敏、可以给沙箱副本，但 **分布要真** 。

真实的用户。 至少一个一线用户，每周真实地用它、真实地反馈——不是产品负责人代用，不是验收会上礼节性地点一遍。判断标准很朴素：这位用户休假一周，工作还会不会找上这套系统。演示是演给决策者看的，使用是长在使用者手里的，MVD 要的是后者。

真实的基线。 上线之前，把人工流程的基线测好：一件事现在耗多少时间、错多少、成本多少。没有基线，"改善"两个字就无从谈起——这正是第 6 章第三关（可验证性）在执行层的延续：过关时说好的基线，到 MVD 这儿必须变成一组写下来的数。

三个"真"齐了，MVD 才开始产生 Demo 给不了的东西：**一段可测量的真实差距，和一段被真实使用检**

验过的判断。

三、砍范围，不砍价值

三个"真"让差距测得真；下一个问题是范围——切多小、怎么切？

"小"的正确切法是三刀：**砍流程数**——只做一条端到端流程，不做平台；**砍用户群**——只服务一个团队；**砍智能边界**——Agent 只接管流程中最可测的一段，其余留人工。

三刀的前提是一条军规：**缩小的是范围，不是价值**。最常见的错误，是把 MVD 理解成"阉割版的大方案"——功能砍掉七成，做得四不像。正确的做法是不砍深度、只砍覆盖面：不追求"覆盖全公司的智能客服"，而是"只覆盖退换货这一类工单，但做到端到端、无人干预"；不追求"全集团的供应链优化"，而是"只做这条产线的排产冲突，但每周实实在在省下 20 个人时"[2]。切入点要小，才做得深；做得深，业务部门肉眼可见，才会主动传播。

至于"砍到多小算够"，也有现成的判据："用户不改变原有习惯就能用，这叫够用；为了用你的系统他得先重塑工作流，这就过度了。"[3] 曾经一个专业团队花几十万、三个月做出全链路 AI 化，结果不如一位六十多岁校长三天手搓的简化版——他只做模板生成，其余环节照旧人工填，其他校长爱用，因为不强迫他们改变习惯。

有句评语说到了点子上："衡量成败的标准从来不是技术多先进，是使用的人明天还愿不愿意打开它。"[3]

范围之外还有一道环境边界：MVD 要不要迁就客户的旧系统？

实践给出的中间路线叫"读旧写新"，即数据上深度兼容——客户的数据在哪就从哪读（注意：只读不写），哪怕在老主机和共享盘的表格里；架构上绝不耦合——代码写在自己可控的边界里，通过接口跟旧系统交互，不把代码塞进旧系统。

理由也摆在明面上：验证期的方案有一半以上概率被推翻重写，耦合越深浪费越大；写入旧系统要走变更管理流程，周期以月计，跟周级节奏根本冲突。

流程上则顺从人的习惯而不是系统的习惯——用户的核心动作发生在表格和邮件里，界面就长在表格插件和邮件里。也有国内从业者这么总结：客户核心系统用了十年，先在外围做轻量插件，"判断标准很简单：改动是否逼用户离开熟悉的系统。逼，就缓；不逼，就推。"[3]

还有一条容易忽视的阶梯原则：MVD 是第一级台阶，不是终点的缩微版。

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[3] 《关于 FDE 的 100 个问题》（AI磊叔）

[4] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

一家投资机构的周报自动化走了三级——先用对话式梳理省掉"写"，再升级到语音生成，最后连语音都省了，直接读工作留痕自动汇总出周报。每一步都能独立验收、独立叫停，前一步的真实使用就是后一步的需求依据［4］。想一步到位做出第三级的样子，通常连第一级都站不稳。

四、以周计，不以月计

第三条军规是**定死截止时间**：MVD 的验证周期以周计，不以月计——Palantir 的训练营压缩到一到五天，公开报道里最快上线四周、典型部署四到八周［2］。截止时间的意义不在快，在**逼双方如实做取舍**：凡不能在这几周里体现价值的部分，都还不是核心价值。一个六个月的"最小验证"，几乎必然重新长成一个什么都想要的大项目——那就是概念验证坟墓的又一次开工。

两周冲刺可以拆到天：

天	做什么
第 1 天	进场对齐：和客户把"单点问题"写成一句话，把验收指标写成一个数
第 2–3 天	接数据：只接这个问题需要的最小数据集；权限、脱敏、导出方式当场定死
第 4–5 天	搭骨架：能查数、能回答的最小原型跑起来
第 6–8 天	接进真实业务流：一两个真实用户开始真的用它干活
第 9 天	收集使用痕迹和问题清单
第 10 天	演示与拍板：不是给信息部门演示功能，是给业务方演示"你那个问题，现在这样了"——当场决定放大、调整还是散伙

客户方同样有三项配合要求：一个能拍板的业务方负责人在场；一个懂数据的内部接口人全程陪同；真实数据的访问权限第 1 天就位，而不是"正在走流程"——三条缺一条，两周冲刺大概率滑回传统概念验证［2］。

《FDE 交付范式革命》里把同一套节奏总结成四步：**业务理解**——进场先摸清业务链路、痛点与价值支点，不急着写 Prompt、搭 Agent；**协作搭建**——业务专家和工程师把场景一起转化成 Agent、工作流或原型，不是技术单打独斗；**工具保障质量**——从真实聊天记录、工单和业务数据里生成测试集，做自动评测和 A/B 对比；**业务结果验证**——AI 组与人工组逐环节对比，用触达率、响应率、转化率这些人效指标证明价值。四步背后是同一句话：**卖成果，不卖软件**——客户不为"用了 AI"付费，为业务真的变好付费。

五、被抛弃，就是第一版软件的宿命

MVD 做完，代码怎么处置？麦格鲁把两拨人的分工讲得很清楚：FDE 在客户现场快速交付一套定制实

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[5] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

现——只为这一个客户、允许粗糙、有坑也认；产品与工程团队的角色，是审视这套实现，从根本上思考如何把它推广到接下来的五个或十个客户，把一次性的定制变成可复用的产品 [5] 。一版代码不该同时追求这两件事：既要它十天跑通，又要它维护十年。

所以招 MVD 阶段的工程师，什么样的人不能要也很具体：讲究手艺、执着于把抽象层打磨到恰到好处、要写一套能维护十几年的软件——因为那不是这个阶段的工作。

快速原型的人写出来的代码有时很漂亮，"但通常不会，也没必要——这也不是工作的关键部分。关键是有人能按时以软件形式交付成果。可能他们写的第一版代码必须被抛弃，然后重写完整的第二版。" [5]

把这两段合起来就清晰了，MVD 真正积累的资产，**不是代码，是考题和认知**。

考题是评测集——这个客户的真实输入里，哪些该出什么答案；认知是流程理解——哪些需求是通用的、哪些是这个客户的特例、哪些坑下一个客户还会遇到。代码会扔掉重写，这两样东西全部带走。这也直接纠正一个普遍的执念——"把 MVD 代码直接改成生产代码"：省下的那点开发量，远抵不过把定制代码补成生产系统的成本，而那正是第 9 章要展开讲的。

下面是一份自检清单，大家对照着看：**用 Demo 代码直接改 MVD**——演示数据的假设全部渗进基线，测出来的差距是假的；**MVD 做成功能全家桶**——范围失控，两周变两月，验证变成开发；**"先上线再说" 跳过基线**——之后永远无法证明价值，续约谈判桌上无话可说。

最后是退出条件：MVD 阶段允许没有完整的审批链、没有监控、没有降级方案，**但不允许没有评测集**——它是 MVD 唯一必须从第一天就开始建的东西，怎么建是第 12 章的主题。带着一套跑通的评测集和一组测出来的真实差距，才答得上客户接下来必然要问的那句话——"所以，什么时候可以上线？" 这会在第 8 章里介绍。

---

[5] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

第 8 章 · "什么时候可以上线"的答案是什么?

> Good fences make good neighbours.
> 好篱笆造出好邻居。
> ——罗伯特·弗罗斯特（Robert Frost），《补墙》（Mending Wall，1914）

上线评审会开到第三个小时，甲方合上电脑，问了一个全场最朴素的问题："所以——什么时候可以上线？"会议室安静了十秒。供应商想说"效果已经很好"，业务方想问"出了错算谁的"——两句话都卡在嘴边，谁也没说出口。

这个场面之所以在行业里反复上演，是因为"上线"从来不仅仅是个技术问题，而是一份没人起草过的双向契约：供应商按事前约定的指标证明达标，客户按事前约定的清单接手运行责任。

契约没人起草，那两句话就只能一直卡在嘴边。

本章要做的，就是讨论如何起草这份契约。

一、问到上线时，到底在问什么

先站到桌子的另一边。作为要为企业引入 AI 系统的负责人，评审会上心里排着的问题清单通常长这样：数据接不进怎么办？权限归谁？答错了算谁的？出了事怎么追责？什么时候才算可以上线？

把这张清单看一遍，会发现一个共同点：**全是验收问题，没有一个是模型问题**。模型的能力在演示那天就已经定了，关键点从来不在模型上，在验收上。

"什么时候可以上线"听起来在问一个日期，实际在问的是：**由谁、按什么标准、判定这件事成了**。

答不出这三样，任何回答都是拍脑袋；答得出，日期只是水到渠成的落款。

所以标准答案不是一句话，是一份契约、三份文件：**效果验收单**（指标+基线+判定人）、**运行就绪清单**（监控/降级/回退/值守）、**责任交接表**（每个事故场景谁负责）。三份文件，一份管"值不值得上"，一份管"敢不敢上"，一份管"出了事找谁"。

二、效果验收单：五个要素，缺一项就翻车

"效果很好"是没法验收的。能验收的指标要写全五个要素：**基线值**（上线前人工做这件事的水平，第 6 章第三关的承诺、第 7 章最小可行部署（MVD）测出的那组数，到这里落进合同）、**目标值**（上线后要到的水平）、**测量窗口**（按周还是按月统计）、**分母口径**（算谁、不算谁）、**判定人**（谁说了算）

（见图 8–1）。

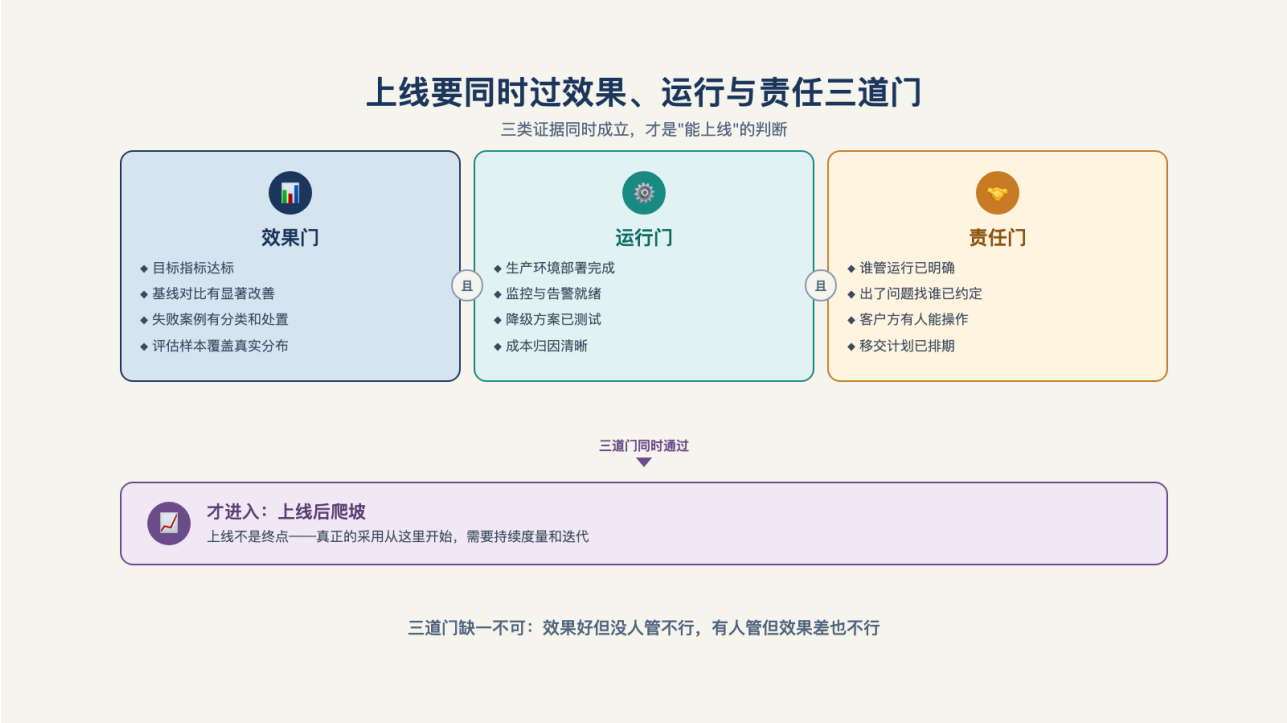


图 8–1：上线要同时过效果、运行与责任三道门，之后仍需爬坡验证。

五个要素里最容易埋雷的是分母口径——同一个"解决率 90%"，分母是全部会话、还是进入 AI 通道的会话、还是 AI 尝试回答过的会话，三个数字能差出一个档次。

这正是指标口径的纪律所在。企业部署最核心的价值指标（解决率、分流率），恰恰是各家口径最乱的指标——以 ServiceNow 的官方口径为例，一次"分流"成立要同时满足两个条件：交互后 24 小时内未提交工单，且用户随后有正向参与（如继续使用）。所以引用任何厂商的解决率或分流率之前，先问清三件事：**分母是什么、判定窗口多长、谁来判定**。一句话把这条纪律说到点子上了："口径不清的 86%，不如口径清楚的 51%。" [1]

五个要素写全的指标长什么样？前面提到的保险行业公司给过一个样板：他们和客户约定的质量标准是 99.5%——换算成可检验的说法是，每周一千次调用，做对九百九十五次 [2] 。这个数字好就好在三点：**可数**（一千次里数九百九十五次）、**有窗口**（每周）、**能复查**（客户自己数得出）。可数、有窗口、能复查，验收单上的每个指标都该经得起这三问。

验收单还有一层谈判功能：指标事前冻结，上线日就没有"这个月数据特殊"的扯皮。所有对口径的异议，都被推回到成本最低的时刻——那时候还没开跑、改一行字就行。

三、运行就绪清单：告警响了谁看

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[2] Inside an Applied AI company（Pace 长文）

效果达标只回答了"值不值得上"，第二份文件回答"敢不敢上"。四问，逐条过：

告警有人看吗？ 告警响起时有一个名字、一张值班表对得上，而不是邮件落进垃圾箱、消息沉在群聊里。监控体系怎么搭是后面的工程话题，这里只问验收级的一句：告警响了，谁看？

降级路径可用吗？演练过吗？ 包括最朴素的一条：**明确退回 Excel 和人工**。把"退回人工"写进方案，很多团队觉得脸上挂不住，仿佛写了就承认 AI 不行——恰恰相反，敢写回退的方案才敢上线：它承认系统会坏，并且提前说好了坏了怎么活。这一条在国内被反复强调，是验收清单里最容易被略过、出事时最被感谢的一项。降级路径不是失败的遮羞布，是设计出来的安全网。

运行手册存在吗？客户方有人读过吗？ 故障处置步骤写在纸上、演练过一遍，而且客户那边的对接人手里有一份、读过一遍——手册只存在乙方电脑里，等于没有。

值守谁排？ 上线后的前几周，乙方谁值守、响应时间承诺是多少，白纸黑字。这三份文件里最不起眼的一条，出事时最先被问。

反例不长，但足够疼。一位在 AI 创业公司做 FDE 的工程师在周记里记过一句：上游服务商（Cloudflare）宕机，无预案救火，自己手上所有客户的系统一起受影响 [3] 。上游故障从来不在你的控制范围内——但"无预案"在你控制范围内。运行就绪清单管的正是这一项。

四、责任交接表：每个死法都有名字

第三份文件最常缺席，也最考验双方诚意。做法是把系统可能的死法逐个列出来，每种写明**第一责任人**和**升级路径**（多长时间内、升级到谁），以下为示例：

出了什么事	第一责任人	升级路径
模型输出错误，造成业务损失	谁判定、谁拦截、谁赔付	1 小时内到双方负责人
数据源故障 / 上游服务宕机	数据方接口人	4 小时内到供应商值守
用户误操作引发故障	客户侧系统管理员	当日到联合复盘
成本超预算	平台方	周报到项目负责人

这张表为什么值得画？回到题记那句话。弗罗斯特写"好篱笆造出好邻居"，说的不是提防，是**把边界划在出事之前**——篱笆立好的时候，邻居做得成朋友；等羊踩了菜地再划界，怎么划都是伤感情。责任交接表就是上线前的篱笆：模型坏归谁、数据坏归谁、误操作归谁，白纸黑字写清楚，出事时照单找人，不伤交情。它是第 17 章完整移交的前置切片——上线时交接的是"第一响应"，长期运行责任是另一张更大的表。

三件套齐了，放到评审桌上过一遍堂，就知道成色。下面这份评审会纪要虽然不是真实案例，但每一项都来自前面的清单：

[3] 一名前线部署工程师的一周（10 个客户，50 小时）

上线评审会纪要（模拟示意）

通过项：验收单五要素齐全，判定人一栏写的是业务负责人本人，不是"业务方"；灰度第三级的回滚线已量化。

卡壳项：降级路径"退回人工"写了，但没演练过——现场约了演练日期才放行；值守表只排到上线后两周，两周之后空白。

争议项：误操作引发故障的责任归属，供应商主张客户方管理员全责，客户主张界面应有防呆设计——僵持后写入"共担+复盘定则"。

三分之一通过、三分之一卡壳、三分之一争议——一份真实的评审会纪要，大抵就是这三个三分之一。

最后看一下责任悬空的两种退化，它们都发生在没人起草契约的地方：一种是"**先免费跑三个月**"——责任无主，跑得好皆大欢喜，出一次事就翻脸，因为没有任何一页纸说过算谁的；另一种是"**效果很好，你先用着**"——供应商实际上拒绝交接，系统永远停在"试用"，永远长不成生产系统。两种退化方向相反，病根相同：契约缺席。

五、双方都要做功课

这份契约不该等到上线评审才想起。两边各有更早的功课。

甲方的自检清单。 把乙方筛选客户的防线翻个面，正好翻成甲方的自检清单：你的项目在公司内部是不是乙方眼里的"无主之地"——集团发文压下来、信息部门牵头、业务部门无人署名？你的采购流程是不是天然制造"只许看、不给碰"——要求对方先证明能力，真实数据却一份不给？你写进招标书的是不是宇宙级需求——"AI 赋能全集团"？判断很直接：最后这种需求，吓跑的是最懂行的供应商，吸引来的是最敢吹牛的。三个问题有一个"是"，先修自己，再谈上线。

供应商的功课。 一份好提案是倒着写的：第一层必须是**业务结果**，一段话、客户语言说完——"八周内，把反洗钱调查的平均处理时间从 4 小时降到 15 分钟"；第二层是**价值验证路径**——验收指标、测量方法、基线数据、每阶段的退出机制，传递的信号是"我们敢被检验"；第三层才是**交付方法**，写明需要客户配合的事项；最后是**风险与对策**——"我们想过会怎么死" [1] 。第二层就是本章三件套在售前阶段的影子：验收单不是上线时才补的作业，是从提案第一页就埋下的伏笔。这个打法有比 AI 时代早十年的先例——一家数据分析公司从 2013 年起就在免费试用期内做重型售前实施，原则只有一条：**演示就是概念验证**——永远向潜在客户要真实的数据集，绝不用精心准备的样例数据表演。

谈判桌上的现实是：甲方不懂指标时，会退回两种姿势——"要 100% 准确"（把验收单变成不可能任务）或"先免费跑三个月"（责任悬空）。两种姿势的解法是同一句话：把"效果"写成可测的数。这也是供应商的销售逻辑——**用验收单换信任**：验收写得越清楚，客户越敢签，成交越快。在被过度承诺伤害

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

过的企业市场里，敢被检验，是最稀缺的诚意。

六、上线是爬坡，而不是一个开关

契约签了，上线也不是一个开关拨过去。更像爬一段坡道：按流量或用户组分级放量，每一级都有一条**回滚线**——指标跌破约定值，退回上一级，查清再上。上线日不是终点线，是坡道的第一级。

坡道的长度里，藏着一段最容易被排期表忽略的时间。一个在中文社区流传的银行项目复盘给出过一组悬殊的数字：技术层面六到八周就做完了，试点与信任建立又花了四个月［4］。前半段是供应商能控制的进度，后半段是客户组织需要的时间——多数排期表只画了前半段。所以"什么时候可以上线"的完整答案里，必须给信任建立留出**明确**的时间预算：让一线用户从"不敢用"到"离不开"，这段时间和开发时间一样真实，怎么赢得它是采用机制那章的主题。

给全书这段路收个口。有一份开源的方法论文档把这类部署的节奏标成过参考刻度：最小可行部署的验证以周计（2–6 周），完整部署以月计（1–4 个月），并且给了一条警报——**价值实现时间持续变长，是方法论或平台本身出问题的第一信号**［1］。用三件套对照：验收单签了字、就绪清单过了堂、责任表上了墙、坡道爬完最后一级——"可以上线了"这句话才说得出口，而它真正宣布的不是系统成了，是那份契约生效了。

但契约生效只是拿到了入场券。把一个"能跑的验证"变成一个"能托付的生产系统"，中间还隔着八堵墙——数据、权限、评测、审批、成本……每一堵都真实，每一堵都有翻法。那是第三部分要讨论的事。

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[4] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

第三部 · 生产环境的每一堵墙

> 本部共六章，回答一个总问题：**把一个"能跑的验证"变成一个"能托付的生产系统"，这段工程上的路怎么走。**

第 2 部结尾留下的话在这里兑现：验收契约签了，还有很多阻碍要解决。

第 9 章先把"生产化"拆成可定义、可验收的八堵墙——并给出建议：墙要在报价和排期时数清，数不清墙的报价，就是在合同阶段埋下的雷。

随后逐堵拆：数据与权限怎么接（第 10 章，墙一、墙二）；Agent 的动作怎么约束才敢放进生产（第 11 章，墙三、墙四）；效果怎么证明没退化（第 12 章，墙五）；结果怎么取信于客户（第 13 章，墙六）；账单和稳定性怎么兜住（第 14 章，墙七）。墙八（责任可移交）在第 17 章完整兑现。

六章是一条链：数据接进来了，Agent 才能动手；动作管住了，效果才谈得上证明；效果可证了，结果才谈得上可信；结果可信了，钱和稳定还得兜住——一环松，环环松。

本部的第一读者是**要为生产负责的人**：交付的技术负责人、带队进场的工程师、要签验收单的甲方技术代表。八堵墙的翻法是工程语言，但每堵墙都从一句客户的话开始——"数据不能出域""半夜跑飞算谁的""这个数字哪来的""账单怎么这么多"。组织与采用是第四部的事，收费与生意是第五部的事；本部只回答一句：这套系统，凭什么被托付。

第 9 章 · 从 PoC 到生产，中间隔着哪几堵墙?

> 未战而庙算胜者，得算多也；未战而庙算不胜者，得算少也。
> ——《孙子兵法·计篇》

"PoC 一周能跑通，为什么生产上线还要再等大半年？"——这句话客户迟早会问出口，而多数 FDE 答得支支吾吾。

其实要回答也不难，因为真正卡住上线进度的，从来不是代码量。

一、从 PoC 到生产，隔着很多堵墙

先把"生产化"这个词拆开。它在不同人嘴里指的是完全不同的东西：有人指"加监控"，有人指"上更多机器"，有人指"过安全审计"。这些都没错，但都只是局部。

一套能在真实企业里干活的 AI 系统，分三层——**模型**、**接口**、**harness**。模型会推理，但不会行动；行动的能力全靠 harness 补上。一家保险公司AI落地系统的 harness 可能会有五样东西：**工具**（模型算不准保费，配计算器；读不了手写表单，配识别路由）、**集成**（对接各家保险公司千奇百怪的 API，没有 API 的就用程序驱动浏览器）、**文件系统**（一个持续多天的流程，状态得存下来，中断了才能接着干）、**任务编排**（简单分类给便宜的小模型，模糊条款给更强的模型，谁先谁后由任务图决定）、**监督机制**（低置信度进人工复核队列，不可逆动作等人批准，全程留审计记录） [1] 。

这个划分的价值在于诚实：一线公司给"模型和生产之间那堆东西"起了名字，承认它是产品的一半，而不是"收尾工作"。一本企业 Agent 平台工程的开源书拆得更细——把生产级部署的交付对象列成了十几项能力，从运行时、工具契约、评测到成本治理，每项都有最小代码对照 [2] 。

本书沿用这个方向，把模型和生产之间的阻隔拆成**八堵墙**，把"生产化"这个词翻译成可以逐项验收的清单——这八堵墙，需要一一拆掉，才能完整交付。

二、八堵墙

第一堵墙，数据与权限。 现场问题：哪些表能读、结果谁能看、脱敏怎么做、例外走什么流程？客户的安全审查不是走过场，是有一张表要填的。翻墙产物：一份数据契约。这是第 10 章的主题。

[1] Inside an Applied AI company（Pace 长文）
[2] enterprise_agent_platform（开源企业 Agent 平台工程）
[3] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

第二堵墙，集成。 现场问题：Agent 要写回的那个系统，是客户用了二十年的老系统——没有 API，数据格式没人说得清。Box 的 CEO 说得直白：集成就是墙，Agent 不会替你修复集成问题，它只会撞上去 [3] 。翻墙产物：一条打通新旧系统的路。

第三堵墙，工具边界。 现场问题：Agent 能调用哪些工具？参数谁校验？一个能自己发邮件的 Agent，和一个只会草拟邮件的 Agent，是两种风险等级。翻墙产物：一份工具契约。这是第 11 章的主题。

第四堵墙，运行可控。 现场问题：半夜任务跑飞了算谁的？卡在人工审批那一步，第二天还能接着跑吗？翻墙产物：运行状态机和检查点。同样在第 11 章。

第五堵墙，效果可证。 现场问题：改了一版提示词，效果是升是降？答不上来，每次改动都是赌博。翻墙产物：评测集和回归门禁。这是第 12 章的主题。

第六堵墙，结果可信。 现场问题：客户指着屏幕问"这个数字哪来的、这个结论为什么信"，除了道歉和重跑，能拿出什么？翻墙产物：证据链。这是第 13 章的主题。

第七堵墙，成本与稳定。 现场问题：演示时一次调用几分钱，上线后月账单五位数；昨天还好好的，今天系统抽风。翻墙产物：成本归因和稳定性承诺。这是第 14 章的主题。

第八堵墙，责任可移交。 现场问题：出了事谁负责？FDE 撤场之后，系统交给谁？翻墙产物：责任交接表和运行手册——第 8 章已经立过这张表的雏形，第 17 章展开完整移交。

一个具体的例子。 Cresta 的 AI Agent FDE 负责人钟钱杰分享过一个语音 AI Agent 的复杂度：背后可能同时跑二十个模型——ASR、打断判断、噪音隔离、检索、Tool Call、多模型并发 Guard Rail，再加上 PCI（支付卡行业）、HIPAA（美国医疗隐私法）合规审计动辄半年一年。花一周把端到端走通，然后要花一个月写几千几万个测试——这些不是传统 unit test，是用历史通话训练小模型去主动模拟真实场景的边界情况（手法的展开见第 12 章） [4] 。

这个例子几乎把第四堵墙（运行可控——二十个模型谁先谁后、谁能打断谁）和第五堵墙（效果可证——几千几万个测试从哪来）同时有了具体的样子：PoC 阶段"一周端到端"的兴奋，和后面"一个月测试"的现实，是同一个项目的两副面孔。

八堵墙放在一起看，会发现它们性质各不相同：第一、第二堵墙是**地基**——不通就什么都别谈；第三、第四堵墙是**风险**——不通也能跑，但随时可能出事；第五到第八堵墙是**信任**——不通则永远停在"试用"。性质不同，解法不同，验收标准也不同（见图 9–1）。

[4] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

图 9–1：模型能力只是起点，生产系统还要穿过八类工程与组织约束。

三、墙要在签合同的时候摸清，而不是等撞上了才头破血流

为什么要把墙摸清楚？因为墙的工程量必须进报价和排期。

两种不同做法的合同，结局也会完全不同。

摸清墙的合同里，每堵墙有工程量估算、有责任人、有验收标准——客户看到的报价单背后是一张地图，双方按里程碑推进，撞到墙也知道墙在计划内。

没摸清墙的合同里，"AI 能力交付"一笔打包——Demo 的掌声还在耳边，交付期就开始一堵一堵撞墙，每堵墙都变成"额外需求"，每堵墙都要重新谈钱重新排期，扯皮三个月后项目悄悄死掉。同一个项目，前者叫交付，后者叫烂尾，分岔点只在报价那天有没有摸清墙。

墙没摸清的代价还有一重，在乙方自己的账上。

行业里有一线经验的交付负责人曾警告过：一份 200 万美元的合同，如果没算清墙的工程量，配上 6 个工程师干 12 个月，算下来毛利可能是负的——账面在赚钱，实际在侵蚀企业价值 [5] 。这笔账的完整算法是第五部的事，这里先立一个警告：**墙没摸清，就是后续亏损的直接原因**。

坊间流行"AI 项目落地失败率高"的说法，听起来像在评价技术。把"失败"拆开看，大多数烂尾项目死于同一个原因：报价的时候，没把墙写进合同，也没写进价格。墙没被定价，不等于墙不存在——它只是

[5] @deployengineer（Bhaulik Patel）系列长文

会在项目最关键的时刻（交付过程中、客户注视下）逐渐显形。技术不背这个锅。

四、模型变强，墙会自己消失吗?

一个自然的疑问：模型一代比一代强，这八堵墙会不会自己消失?

会矮一半。前面四堵墙——数据接入、集成、工具生成、运行编排——属于"写代码能解决"的工程活，模型越强，这些活干得越快。有人把 FDE 的工作本身产品化成智能体，让 AI 接手前四堵墙里的重复劳动［6］，方向也是这个。

但后四堵墙很难变矮。

Databricks 联合创始人兼 CEO Ali Ghodsi 在斯坦福大学一次课程讲座上讲过的例子，把这条线划得很清楚：他们做一个能正式上线的应用连接器，原有流程约 9 个月——第一个季度做需求调研、产出几十页文档，之后才是开发、测试环境、安全验证、客户反馈。负责人自己用 AI 两天就写出了一个能跑的版本，但团队一致认为那只是 Demo：没经过完整测试，也无法向客户保证安全和稳定。更扎心的估算是：就算把 AI 直接嵌进原有流程，周期也只能从 9 个月缩到约 7 个半月——省下的时间被流程本身吃掉了。真正的转折不是换更强的模型，而是重写流程：需求阶段从一个季度压缩到一周，测试环境交给更擅长的团队并行搭建，协作方式从"一人守一个连接器"改成团队共同覆盖一组连接器——重构之后，一个季度交付了 7 个生产级连接器［7］。

两天写出 Demo 的 AI，吃掉的是前四堵墙的工程量；9 个月变 7 个半月，说明只换模型不动流程，后四堵墙一寸都不让；一个季度 7 个连接器，说明后四堵墙翻过去之后，前四堵墙的提速才能兑现成生产率。**测试、安全、协作方式、责任划分——这些墙的高度由组织和信任决定，与模型能力无关。** 这也是第 5 章那个判断的工程版：FDE 不会被 AI 取代的理由，就藏在这后四堵墙里。

五、把这张表带走

本章不给拆墙的具体方法——那是接下来五章的事。给的是一张对照表：拿任何一个"PoC 已通过、待生产化"的项目，都要问三个问题——**这堵墙拆了吗？产物是什么？谁验收?**

[6] Palantir Foundry「AI FDE」产品文档

[7] Ali Ghodsi 在斯坦福 MS&E 435 课程的客座讲座

墙	现场问题	翻墙产物	详见
一 数据与权限	哪些表能读、结果谁能看	数据契约	第 10 章
二 集成	新系统怎么写回二十年老系统	打通的集成路径	第 10 章
三 工具边界	Agent 能调什么、参数谁校验	工具契约	第 11 章
四 运行可控	跑飞了算谁的、卡住能否续跑	状态机与检查点	第 11 章
五 效果可证	改了提示词，效果升还是降	评测集与回归门禁	第 12 章
六 结果可信	"这个数字哪来的"拿什么回答	证据链	第 13 章
七 成本与稳定	账单爆炸、系统抖动怎么办	成本归因与稳定性承诺	第 14 章
八 责任可移交	出事谁负责、撤场交给谁	责任交接表与运行手册	第 17 章

三个问题里有任何一个答不上，那堵墙就是排期上的洞——而且是最贵的洞。这张表的完整版是本书的附录 A——交付检查清单按发现、最小可行部署、生产化、上线与采用四个阶段编排，本章的八堵墙集中落在「生产化」那一段，按地基、风险、信任分组展开。

从下一章开始逐堵拆墙。先拆最早撞上的那堵：客户的数据和权限。

第 10 章 · 客户的数据和权限，怎么接才既用得上又守得住?

> Trust, but verify.
> 信任，但要核实。
> ——俄罗斯谚语，经罗纳德·里根（Ronald Reagan）总统在 1987 年《中导条约》（INF Treaty）签署场合反复引用而广为人知

FDE 进场第一天，问客户的第一句通常是"数据在哪"。最常见的回答是"都在系统里"。

在实际为新客户做 AI 落地过程中，合并 7 到 8 个分散的数据源是常有的事——邮件、表格、客户管理系统、ERP，结构化的和非结构化的混在一起；被私募收购整合过的企业更极端，可能同时跑着四套 ERP，客户自己都说不清数据长什么样 [1] 。所以"都在系统里"这句话的正确翻译就是——还不知道在哪儿。

数据接入的坑，从第一个问题就开始了——而比"在哪儿"更要命的，是"谁能看"。

一、进场第一天，先问五个问题

把数据接入变成可排期的工作，起点是五个问题：**数据在哪？谁拥有？谁能看？怎么脱敏？例外走什么流程?**

五个问题一个都不少。少了"谁拥有"，权限申请会递到错误的人手里，一轮来回就是一周；少了"谁能看"，Agent 跑出来的结果可能直接泄给不该看的人；少了"例外流程"，第一次数据出域就要停工等高层拍板。

这五问答不上来的项目，排期一律先加两周。不是惩罚，是现实：数据归拢经常是进场后最先撞上的墙，撞上它的时间，早于任何权限设计 [1] 。第 6 章在问题发现时曾问过一遍"数据能不能拿到"，本节是它在工程期的延续——发现期问的是"有没有"，这里问的是"怎么拿、拿多少、给谁看"。

数据在哪摸清之后，还有一道现实要接受：Agent 要写回的那个系统，大概率是客户用了十几年的老系统。第 7 章给过解法——读旧写新：数据从哪就从哪读，通过接口交互，代码绝不塞进旧系统。

二、先契约，后数据

[1] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

五个问题问完，答案不能只记在 FDE 的笔记本里——要写成一份数据契约，再开始动数据（见图 10–1）。

图 10–1：数据可用不等于可以接入，契约、最小权限和审计缺一不可。

契约写四件事：**访问范围**（哪些表、哪些字段能读，精确到列）、**结果可见性**（Agent 产出的结果谁能看、能外发吗）、**留存与销毁**（客户数据在本方留存多久、项目结束后怎么销毁）、**审计要求**（每次访问的日志谁查、多久查一次）[2]。

这份契约有三个使用者。**客户的合规审查**用它：安全部门过审看的就是这张纸，没有它，项目在流程上就过不了关。**FDE 用它约束自己的实现**：契约写了只读三张表，代码里就只配三张表的连接——这是给工程师的边界，不是给客户的姿态。**事后的追责靠它**：真出了数据事件，双方按契约条款厘清责任，而不是互相甩锅。

顺序为什么不能反？因为没有契约的 PoC，等于在客户数仓里裸奔——出了事没有追责依据，没出事也过不了客户的安全审查。很多人觉得"先跑通再补契约"是灵活，其实是把两堵墙（数据与权限）的工程量推迟到了最贵的时刻。

三、Agent 的权限范围是继承来的

契约管数据，权限管动作。这里有一个行业里最常见的错误做法：给 Agent 申请一个大权限的服务账号——"方便，一次配好"。

[2] enterprise_agent_platform（开源企业 Agent 平台工程）

正确做法应该是：**Agent 继承发起用户的权限，并全程留痕。**

产品层面已经有现成的例子。Palantir 把自家的 AI FDE 智能体做成了产品，文档里明确写到智能体执行的所有操作都遵循用户现有的权限——用户看不到的数据，派出去的智能体同样看不到［3］。

现实约束的另一面来自 Box CEO 的提醒：Agent 只有人的原有权限，它不会绕过权限墙——卡在权限上就是卡在权限上，这不随模型变强而改变［4］。

两条合起来是一个结论：Agent 的权限＝发起用户的权限＋全程留痕，绝不申请服务账号级的超集权限。超集权限的问题不是"可能被滥用"，是一定说不清——出事的那天，审计日志里那个万能账号的操作，你既不能证明是人干的，也不能证明是 Agent 干的。

四、权限配置的活，要和客户 IT 一起干

权限落到工程上，不止挡数据读取，还要管住**动作的后果**：谁能触发外发、谁能写生产表、谁能在生产环境跑带权限的动作。一线团队的做法是把工具调用按权限、角色、场景三层控制——同一个人，在测试环境里能做的和在生产环境里能做的不同；同一个工具，读数据的和写数据的权限不同［5］。

而这件事怎么做，需要单独强调一下：一位日本 AI 公司的新人 FDE 记录过自己的配置经历——和客户的管理员一起，逐项配置 M365 和 Entra ID 的生产权限［6］。注意这个画面：**FDE 和客户 IT 并排坐在同一块屏幕前**。权限配置不是供应商单方面交付的东西，是双方一起做的工作——因为权限的每一条边界，划走的都是客户组织里某个真实的人的便利。

这也回应了一个常见的抱怨："客户 IT 不配合。"换位想：一个外部团队要在你的权限体系里开洞，你的第一反应也是审。把权限配置做成共同工作，客户 IT 从把关人变成搭档，后面每一次变更都会快很多。

五、数据敏感度，决定部署形态

数据接入还有最后一个决策：模型跑在哪——云端 API、私有化部署，还是完全本地？

这个决策不取决于技术偏好，取决于数据的敏感等级。

把数据分成四级看：**公开**（宣传材料、已公开的报表）、**内部**（流程文档、非敏感经营数据）、**敏感**（客户个人信息、交易记录）、**管制**（受行业监管的数据，如医疗、金融核心记录）。每级对应不同的

[3] Palantir Foundry「AI FDE」产品文档

[4] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

[5] AI 职觉：别再卷 Agent 了（小宇宙播客）

[6] 「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实（note.com · AI Shift）

部署形态：公开和内部数据可以走云端 API，敏感数据倾向私有化或本地，管制数据往往只能本地。

有一个一线教训，把这条路从头到尾走了一遍：落地客服 Agent 项目，前两版都顺利，到要接真实客户数据做实验时暴露了数据安全问题——预算所限，团队只能选择本地部署一个小模型，效果随之打折［5］。这个案例的价值在于诚实：**数据敏感度决定架构，而不是反过来**。当敏感数据把部署形态压到本地，模型选择、效果上限都跟着受限——这不是工程的选择，是约束的传导。

所以敏感分级要在发现期就做——第 6 章第四关（数据可达）问的"需要哪些数据、权限归谁"，就包含这一问。发现期把敏感度摸清，报价里才能留出预算；发现期没摸清，生产期的"突然要本地部署"就是一次返工。

六、每一次访问，都留下证据

最后一块拼图：审计日志。

每一次数据访问——谁发起的、读了哪张表、什么时候、用于哪个任务——都记成日志。它的价值在平时看不出来，在合规问询的那一刻是唯一答案。客户的安全部门问"你们的 Agent 到底碰了我们的哪些数据"，两种回答的分量完全不同：一种是"我们没出过事"，另一种是把日志导出来——每次访问都有据可查。

前者是承诺，后者是证据。B 端的信任不吃承诺，只认证据——这也是第 13 章（结果可信）的主题，在数据层提前出现了一次：**录下来是架构决定，不是事后补写**。审计日志从第一天就开着，是数据契约里"审计要求"那一栏的工程落点。

数据与权限这堵墙翻过去的标志，可以列成一张验收单：数据契约签了字、权限继承模型跑通、与客户 IT 的共同配置完成、敏感分级定了部署形态、审计日志开了起来。五项齐了，才谈得上一件事：让 Agent 动手干活。而 Agent 一旦能动手，下一个问题立刻出现——它的每一个动作，边界在哪？

[5] AI 职觉：别再卷 Agent 了（小宇宙播客）

第 11 章 · Agent 的动作怎么约束，才敢放进客户生产系统?

> You must first enable the government to control the governed; and in the next place oblige it to control itself.
> 既要让它管得住被治理者，也要逼它管得住自己。
> ——詹姆斯·麦迪逊（James Madison），《联邦党人文集》第 51 篇，1788 年

"我们有人工审核环节。"——这句话在每个售前会上都会出现，但值得追问三句：审批的是**动作**还是**结果**？审批人是谁，**有记录吗**？驳回之后，流程要**从头重跑吗**？这三个问题答不上来，那个"人工审核"就是一句口号。

把口号变成工程，是这一章的事——也是 Demo Agent 与生产 Agent 的分水岭。

一、先把客户担心的事情具体化

客户 IT 对 Agent 的担心不是泛泛的"不安全"，每条都很具体：

"它会不会乱传参数，打穿生产系统？"——模型生成的参数没人校验，一个拼错的表名就可能写坏数据。

"它会不会自己删数据、发邮件？"——一个有删除权限和外发权限的 Agent，幻觉一次就是一次真实事故。

"它半夜跑飞了，谁发现？"——定时任务在凌晨两点陷入循环，早上九点上班时已经跑了几千次。

"出了错，我们查得到是哪一步错了吗？"——黑盒跑完只给一个结果，错了无法定位，只能整个重来。

注意：他们关心的不是"模型够不够聪明"，而是在问**动作有没有边界、过程有没有记录、出错有没有退路**。四条担忧，对应四个机制——本章的四节。

敢放进生产系统的 Agent，每个动作要过四个"可"：**可校验、可暂停、可恢复、可追责**。这些机制在开源工程社区已经有成熟的字段级设计 [1]，这里讲清楚每一项"为什么需要、长什么样"。

二、可校验：参数过不了关，动作就出不了门

[1] enterprise_agent_platform（开源企业 Agent 平台工程）

第一道闸门在工具的入口。

Agent 每调用一个工具，参数都要过一遍**结构校验**：这个工具叫什么名字、哪个版本、接受什么类型的参数、每个参数的合法范围是什么——写成一份明确的契约，调用先对照契约，对得上才执行，对不上直接拒绝并报错。

举个最小的例子：一个"查询订单"的工具，契约写明只接受两个参数——订单编号（字符串，固定格式）和查询原因（枚举值，三选一）。模型生成调用时多塞了一个"客户手机号"，校验不过，动作不执行。就这一道闸门，挡住的是"模型顺手多拿一点数据"这类最隐蔽的越界。

每个工具还要**明确写下调用边界**：只读、写入、外发，三选一起写明。只读工具出问题最多是读错，写入工具出问题会改状态，外发工具出问题会把数据送出边界——三类的风险等级完全不同，声明在先，后面的分级管控（第五节）才有依据。

现场自检时需要问一句："你们每个工具的参数校验，在哪一步发生？"答案如果模糊，这一关就是虚的。

三、可暂停、可追责：审批不是弹窗，是工单

高风险动作要等人批准——"加人工审核"这半句人人会说，工程在哪？在于把审批建模成一张**带字段的工单**，而不是一个界面上弹出的确认框。

一个合格的审批请求，至少带五个字段：**属于哪次运行**（不孤立地批一个动作，要知道它前后是什么）、**卡在第几步**（整个流程走到哪儿了）、**想做什么**（具体动作加参数，一目了然）、**谁审批**（指定到人，不是指定到"业务方"）、**批注写了什么**（批准和驳回的理由，留档）。

弹窗和工单的差别就是口号和工程的差别。弹窗关掉就没了；工单是**可归档、可审计的记录**——半年后客户问"当时这笔外发是谁同意的"，翻出工单：谁、何时、基于什么理由、批了什么。四个问题一张纸答完。这也是移交时的伏笔：客户接手运行的那天，审批工单上"谁审批"一栏，就是责任转移的落点（第 17 章再展开）。

四、可恢复：审批隔了一夜，不用从头再来

审批有一个天然属性：慢。审批人在开会、在休假、在另一个时区——一等一夜很正常。

如果流程不支持续跑，会发生什么？审批通过的第二天早上，系统从头重跑——前面已经跑过的十几个步骤、已经花掉的调用成本、已经消耗的几个小时，全部重来一遍。更糟的是行为扭曲：等不起的业务方会开始想办法绕过审批——要么逼着把动作改成免审，要么干脆不用这个流程。**等不起的审批，等于没有审批。**

[1] enterprise_agent_platform（开源企业 Agent 平台工程）

解法是检查点：每跑完关键一步，把当前状态存一份快照；人工审批通过后，从卡住的那一步**续跑**，前面的进度原样保留。这套机制在开源的工程实现里叫状态机加检查点——每一次运行有编号、有生命周期（等待、规划、执行、等待人工、成功、失败），卡在"等待人工"这一步时，状态冻结、快照落地、随时可续［1］。

它顺手解决了第一节担心清单的第三条：半夜跑飞的任务，状态机里有超时和重试上限，循环到阈值自动挂起转人工——"凌晨两点跑几千次"变成"凌晨两点挂起，早上九点有人处理"。

五、分级放权：不是每个动作都要人批

四个"可"立起来之后，最容易走过场的是另一端：所有动作都等人批。审批是稀缺资源——每个动作都批，等于每个动作都不批（人会疲劳，会闭着眼睛点同意）。

正解是按风险分级：

动作级别	例子	处理
只读	查询、检索、汇总	自动执行
可逆写	写入可回滚的中间结果	自动执行+事后抽检
不可逆写	写生产表、删除、覆盖	必须审批
外发	发邮件、调用第三方接口	必须审批

分级的依据就是第二节里每个工具**明确写下的调用边界**——契约在前，分级在后，一环扣一环。

分级之前其实还有更早要问的一层：**这一步压根该不该交给 Agent**。一个落地团队给过具体的切分参照：把一条工作流拆成约十个步骤，大致五步要做成确定性的硬编码——数学计算、合规校验这类不能出错的，根本不给模型发挥空间，更不允许犯错；三四步交给 Agent，允许弹性；两步留给人工审核。反面例子也直白：财务对账这类流程不该让 Agent 判断，它要的是确定性结果，不是聪明［2］。风险分级的第一问不是"要不要审批"，是"这一步要不要智能"。

下面我们用一次被拦下的灾难案例（已经过脱敏处理）把本章的机制串起来走一遍。

一个报销审核 Agent 执行到"反馈结果"一步时，生成的外发动作是：把本季度全部报销明细（含员工姓名和银行卡号）作为附件，发送到一个第三方邮箱——那个邮箱来自一封伪装成财务系统通知的邮件，被模型当成了合法配置。接下来发生的事，按秒排：**结构校验**先亮红灯——外发工具的契约写明"附件仅限汇总报表，禁止明细数据"，参数不合法，动作没有执行，流程停在原地；**风险分级**判定外发属必审动作，一张审批工单生成：属于哪次运行、卡在第几步、想发什么、发给谁；客户财务负责人看到工单，**驳回**，批注"该地址不在白名单"；系统按检查点**续跑**，改用站内通知，前面已完成的十一个步骤原样保留。整个过程没有数据离开边界，客户的 IT 部门在第二天的日志里看到了完整记录。灾难不是没

[2] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

发生——是被工程挡在了门外。

六、四个"可"，对客户是一句话

把四节收拢：**可校验**（参数过契约才执行）、**可暂停**（高风险动作等工单）、**可恢复**（审批后从断点续跑）、**可追责**（谁批的、批了什么有单据）（见图 11–1）。

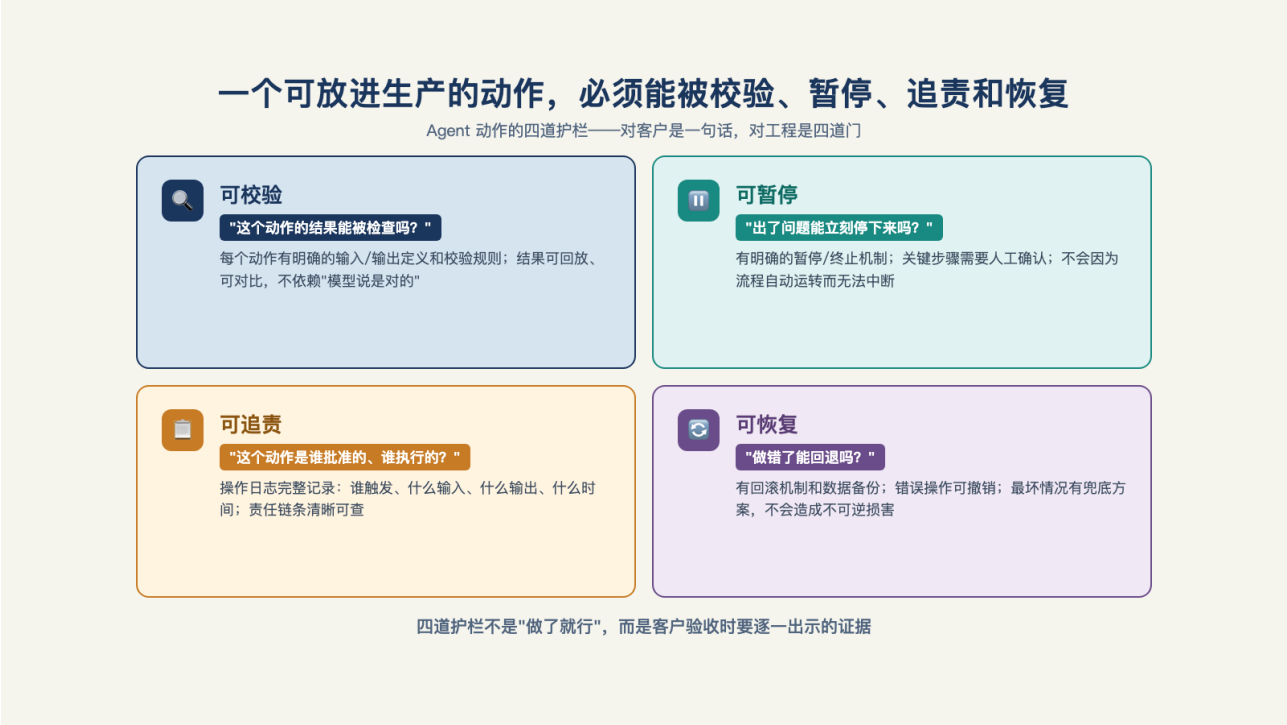


图 11–1：一个可放进生产的动作，必须能被校验、暂停、追责和恢复。

这四个"可"加起来，就是一线公司说的 Harness 五要素里的最后一项——**监督机制**：低置信度进复核队列、不可逆动作等人批准、审计记录全程跟随 [3] 。叫法不一，实质相同：把"人盯着 Agent"变成"架构盯着 Agent"。一线团队把这套东西按权限、角色、场景做到工具层——同一个工具，不同人在不同环境里能触发的后果不同 [4] 。

而对客户，不用讲这些术语。就一句话：**它做的每个动作，有边界、有记录、有阻断。** 客户担心的事情，就全部解决了。

动作有了边界，只回答了"它能不能做"；改了提示词、换了模型、加了工具之后，另一个问题马上出现：它还做得对吗？如果每次变更都答不上，光有前面立好的边界，也守不住交付质量。下一章，我们把这个问题变成一套可复算的评测。

[3] Inside an Applied AI company（Pace 长文）

[4] AI 职觉：别再卷 Agent 了（小宇宙播客）

第 12 章 · 评测体系怎么从零建起来?

> What gets measured gets managed.
> 被衡量的，才被管理。
> ——管理学界流传箴言

系统在周一把一笔退款判为"可自动处理"。周二，工程师换了一句提示词，同样的订单却被转给人工。业务负责人问："到底是变好了，还是变差了？"如果昨天的样本没留下，好坏也说不清楚，团队只能靠感觉争论。

"改了提示词之后，效果有没有退化？"——这个问题答不上来的系统，不配进生产。

这话听起来很严重，拆开看全是常识：没有评测集，每次改动都是赌博——赌注押在工程师的直觉上；每次客户质疑都是罗生门——"变差了""没变，是你们变挑剔了"，谁也说服不了谁。评测不是上线前补的一个测试环节，是生产系统的底线。

这一章讲从零到生产怎么建——四步：**先建基准样本，再建回归集，然后自动判定，最后接进变更流程**。

一、从 70% 到 98%，怎么跨过去

第 7 章的那个案例：第一天真实环境跑分约 70%，合同承诺超过 98%。中间那 28 个百分点，靠什么补?

不是靠调提示词——调提示词是最后一步的微操。靠的是一个循环：**评测发现失败的模式，针对性修，回归验证，再发现**。那位保险行业从业者把它叫评估循环，质量标准就是第 8 章那条每周一千次、做对九百九十五次的可检验口径 [1] 。

同一条爬坡曲线在公开案例里也有完整的版本。一家客服软件公司的智能客服产品，第一代平均只能解决 23% 的会话；更换底层模型后升到 51%；再针对具体客户深度定制，最高做到 86%。它的提供商自己拿它做客服，从上线起持续调优两年，解决率升到约 79%，每月解决约 56 万次会话。

三个数字之间隔着的不是三次模型升级——51% 到 86% 那一段，全是对着评估集一轮一轮改出来的。同样的路，另一家软件公司把自己家的客服智能体调了整整一年，"答不上来"的比例才从三成压到一成以下。

[1] Inside an Applied AI company（Pace 长文）
[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

评估为什么能驱动这一切？因为 AI 系统的质量是连续的、带随机性的、随场景而变的——同一个回答在演示里惊艳，在具体业务语境里可能是灾难；而"好"的定义权在业务方手里，不在工程方手里[2]。没有评测指标的迭代就是蒙眼狂奔：调了半个月，质量是升是降，没人说得清。评测体系就是把业务专家脑子里那句"什么叫好"，变成系统每天都能执行的打分标准。

二、第一步：基准样本——标准从真实案例里长出来

基准样本是一批**带着明确判定标准的真实案例**：输入是什么、正确输出是什么、为什么算对。它是整个评测体系的地基，来源只有两个：**最小可行部署（MVD）期间攒下的真实案例**（系统答过的、用户用过的），和**一线用户标注的争议案例**（答得可疑的、被投诉过的）。不能用工程师坐在办公室编出来的案例——因为编不出现场的混乱，也编不出业务专家在意的那根线。

冷启动需要多少条样本案例？一般而言，五十到一百条可用的，就能起步。追求一次建几百条是常见的弯路——条数多不如口径准。

比条数更关键的是**判定人**。同一条输出，一线用户判及格、FDE 判良好、客户业务负责人判不及格——三个口径都对，因为三个人看的东西不同：一线用户看顺不顺手，FDE 看技术上对不对，业务负责人看赔不赔得起。判定人要在建评测数据集的时候就定下来，并写进每一条样本：基准样本不是一堆题，是一堆**由具体角色判过标准的题**。这也是第 8 章验收单里"判定人"一栏的根——验收口径，在评测集建集那天就开始成形了。

讲冷启动的最佳例子来自一个金融场景：一家机构里几百条隐性业务规则——一笔贷款怎么批、哪个字段异常要上报——散在几位老员工的记忆里，文档上没有。做法是拉着老员工做几轮真实案例推演：拿一笔真实业务，让他边做边讲为什么这么判，同步记成规则条目，喂给模型当评测集；第一轮一定漏，补上再跑，循环几轮就好起来了；固化之后，老员工抽审系统判定，他点头才算数[3]。这段操作把评测集的本质说透了：**基准样本是从人脑子里长出来的标准**——把老师傅的隐性判断，显性化成一条条可执行的题。

三、第二步：回归集——只进不出的错题本

基准样本立起来之后，评测体系开始自我生长：**每一次真实失败，都变成一条新样本进集**。

系统答错了一个案例——进集；用户投诉了一个输出——进集；线上审计抽查发现问题——进集。集子随着运行越长越大，它其实就是错题记录本：每个都是一次真金白银的错误，记录的目的是让它永远不再发生。回归集只增不减——唯一的例外是口径变更（比如判定标准调整了，老样本按新口径重判，而不是直接扔掉）。

[3]《关于 FDE 的 100 个问题》（AI磊叔）

两条纪律防污染。**演示数据不入集**：演示环境的干净数据会让评测集对生产环境的混乱失去分辨力。**标注分歧的样本单独归档**：两个人判不到一起去的案例，不是烂样本，恰恰是最有信息量的样本——分歧本身就是口径没对齐的信号，单独放着，定期复盘口径。

到这里可以看清基准样本和回归集的分工：基准样本是**地基**（标准从这里定，量小而精），回归集是**防护网**（持续生长，量大而杂）。前者保证"好"有定义，后者保证"坏过的不重犯"。

要让回归集更"狠"，还有一个更有效的做法：拿历史数据训练一个专用模型，比如在语音Agent里，用历史通话数据去训练一个小模型，专门模拟真实场景会出现的说话方式——急躁的客户、说话吐字不清的患者 [4] 。这类"对抗性测试集"比标准问答集（QA）更贴近生产会撞见的边界情况，也是回归集从"记录已发生的错误"进化到"主动制造边界情况"的一种手法。

四、第三步：自动判定——三级判定，加一道人工校准

样本有了，每次改动后总不能全靠人重判一遍。自动判定分三级，按成本递进：

第一级，精确匹配和规则。答案能精确比对的（分类、提取、结构化输出），写规则判定——成本几乎为零，结果完全确定。能用规则判的，永远不用模型判。

第二级，模型裁判。开放性任务（摘要写得好不好、回复得体不得体）没有标准答案，用一个模型当裁判，按写在提示词里的评分标准打分。这是当前工程实践的主流做法，但有一条容易被忽略的纪律：**裁判的评分标准本身也是一次"提示词改动"**——它改了，判定口径就变了。所以裁判的评分标准也要进回归管理：改裁判和改系统，走同一道门禁。

第三级，人工抽检。定期抽一小批样本人工重判，对照模型裁判的结论——不是不信自动化，是自动化会**漂移**：裁判模型对某种表达风格系统性偏爱、对某类错误系统性失明，这类漂移只有人工抽检能发现。从不校准的裁判，等于自动地一路错下去。

成本意识贯穿三级：不是每条样本都要跑最贵的判定。规则能判的走规则，模型裁判只判开放性任务，人工只抽检——评测体系本身的运行成本，也是成本（第 14 章的账上要记这笔）。这些机制在开源工程实践里已经沉淀成了标准件——离线评测防退化、在线审计测真实、带成本感知的变更门禁 [5] 。

五、第四步：门禁——没过回归集的变更，不许部署

前三步攒下的家底，在这一步变现：**任何提示词变更、模型切换、工具改动，必须先过回归集，才能部署**。

[4] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

[5] enterprise_agent_platform（开源企业 Agent 平台工程）

这一条把"我觉得没退化"变成"回归通过率 99.2%"。别小看这个措辞变化——前者是感觉，进入不了任何决策；后者是数字，可以进验收会、进变更单、进客户的信任账户。第 7 章埋的那句话在这里兑现：MVD 唯一必须从第一天就建的东西就是评测集——因为它是变更的门禁，而变更是生产的常态。

门禁也是评测与第 8 章验收单的接口：上线验收的指标从哪来？从评测集的口径来。基线值是评测集跑出来的历史水位，目标值是双方在评测口径上约定的承诺——验收单上的每个数字，背后都站着一个可以复算的评测过程。两份文件对不上的项目，一定有一边是拍脑袋的。

六、在线与离线：闭环即评估循环

门禁管的是变更那一刻，运行中的系统还需要另一只眼。评测分两线：

离线评测防退化——变更前对着回归集跑，守住"不比原来差"。**在线评测测真实**——生产环境持续抽样审计，用户反馈（点赞、纠错、投诉）回流进回归集。两线合起来，就是本章开头那个循环的完整版：离线挡住退化的改动上线，在线发现新的失败模式，新的失败变成新的样本，样本让下一轮离线评测更严。转起来之后，它就是那位从业者说的评估循环 [1] （见图 12–1）。

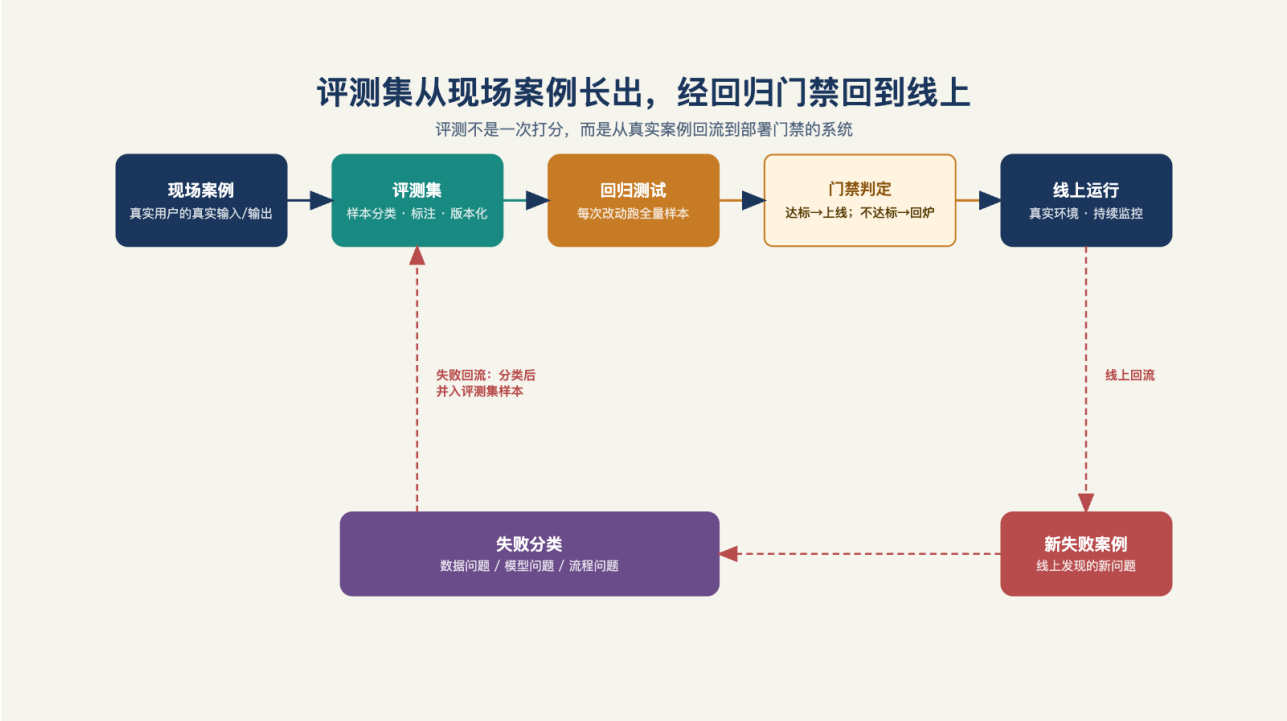


图 12–1：评测集从现场案例长出，经回归门禁回到线上，再把新失败带回样本。

这条方法论有一个公开实践里的完整对照，可以当本章的镜子。领先团队的做法固定成三步：**评估集从真实案例里长出来**（不能靠工程师编）、**让业务方当评委**（把评估做成业务专家能参与的形态——并排的输出对比、简单的优劣标注、定期评审会；评估集越来越准的同时，业务方看着系统在自家案例上一

[1] Inside an Applied AI company（Pace 长文）

次比一次好，信任是亲眼看着攒起来的）、**把评估接到生产回路上**（上线不是终点，持续抽样、分数下滑立即告警）。本书的四步与这三步不冲突：那三步说的是方向（真实、业务方、闭环），本章的四步是冷启动的操作序（先基准样本、再回归、再判定、再门禁）。

评估先行的效果，最完整的公开样本在农业。一家农机制造商要解决除草剂浪费：传统喷药机整田覆盖，新技术用 36 个摄像头加机器视觉，在行进中只喷杂草。但农民要的不是技术，是可信的建议——AI 团队的工程师飞到农场，跟着农艺师下地，先和专家一起评审数百个真实作业案例、建起定制评估体系，再迭代模型，且必须赶上农时。最终化学品使用量最高减少七成，农户互动频率提升六倍［2］。先有评估体系定下的"好"，才有这两个数字。

七、评测集是 FDE 带得走的资产

最后算一笔账，把视角从单个项目拉到整个团队。

项目结束，代码可能会被抛弃，数据必须还给客户，集成方案下个客户不通用——那什么沉淀下来了？**评测集，和建评测集过程中长出来的流程认知**。有位从业者有句话说得透彻：数据是客户的护城河，所以它不可能同时成为供应商的护城河；真正可累积的，是每次部署沉淀的隐性知识与评估循环［1］。

下一次部署，模型可能又换了一代，集成方案重新做——但上一个客户那里攒下的失败模式、判定口径、裁判标准，直接搬进下一个项目的回归集起步。**评测体系是少数不会因模型升级而贬值的工程资产**——模型越换代，越需要一把不跟着换代的尺子。这笔复利怎么在组织层面变现，是第 22 章（现场经验沉淀为产品能力）的主题，这里先把资产本身建起来。

三个负面清单，对照自检：**先建评测平台、后建评测集**——顺序反了，平台是为评测集服务的，没有评测集的平台是空转的；**评测集只在上线前跑一次**——那不是评测体系，是上线仪式；**裁判从不校准**——自动化的漂移无人看守，自动化就成了自动化的错误放大器。

评测把"效果可证"立住了。但生产环境里还有一类时刻评测管不到：客户指着屏幕上昨天的一个结果问"这个数字哪来的、当时为什么这么判"——这要的不是分数，是证据。

[1] Inside an Applied AI company（Pace 长文）

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

第 13 章 · 客户质疑结果时，拿什么自证?

Res ipsa loquitur.
事实本身，即是证据。
——英美普通法侵权法古老法律格言（直译"事情本身会说话"），常见于过失责任认定

季度经营会上，客户方的财务总监指着屏幕上的一个数字问："这笔应付账款的分析结论，数字哪来的？"会议室安静下来。系统如果能调出来源文件和段落定位——发票、对账单、合同各是哪份、各在第几页——质疑当场就能核对；如果只能回答"我们回去重新查一下"，即使数字最终正确，信任也已经在等待里受了损。

质疑时刻是信任的清算时刻，应对清算需要的不是态度，是证据。

一、可信度是一座三层建筑

客户对"这个结果可信吗"的追问，拆开其实是三层：**这个答案的依据是什么？这个判断的过程是什么？这次出错的原因是什么？**——引用可回溯、决策可回放、异常可诊断（见图 13–1）。

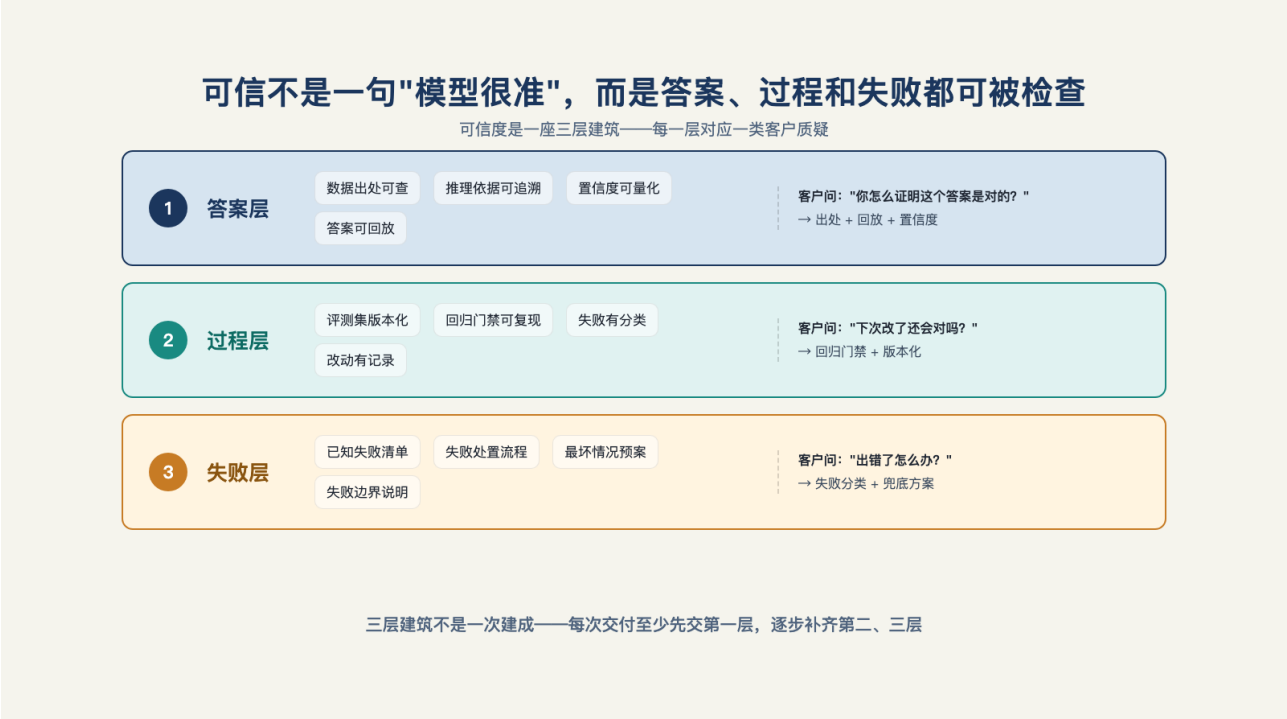


图 13–1：可信不是一句"模型很准"，而是答案、过程和失败都可被检查。

[1] Inside an Applied AI company（Pace 长文）

这三层有一个共同的性质，也是本章最重要的一句话：**它们不是给客户看的汇报材料，是系统运行时自然留下的副产物**。证据链不是被质疑之后补写的材料包，是架构里预埋的记录能力——录下来是架构决定，不是事后补救。开源的企业级 Agent 工程实践已经把这三层做成了标准件，一线公司 Harness 五要素里的监督机制，同样要求审计记录跟着每个动作走——记下 Agent 做了什么、为什么做［1］。

二、第一层：每个答案，带着出处

最基础的一层：Agent 给出的每个结论性答案，都挂着**证据引用**——依据了哪份文档、定位到哪里、置信度多高［2］。

上一章结尾那个问题的完整版就在这里：客户问"这个数字哪来的"，答案分三段呈现——来源是什么（文件名、系统、时间）、定位在哪（段落、页码、字段）、把握多大（置信度）。三段齐了，客户可以顺着引用自己核——核一次，信一分。

这一层还藏着幻觉治理的另一半。大家都指望模型变强了就不胡说，但工程上的纪律更硬：**检索失败的答案，宁可说"没找到"，也不许编**。说"没找到"只是失败，不是欺骗；编一个看起来圆满的答案，是一次对信任的透支。所以架构上要约束：没有挂上证据引用的结论性输出，不允许直接发出去。检索质量不过关，引用链会先断，后面的推理与解释也无从谈起。

三、第二层：每步决策，都要可以回放

第二层往深一步：不只答案带出处，**做答案的过程全程可回放**。

工程上的形态叫运行轨迹：Agent 的每一次运行，逐事件记录——读了什么数据、调了什么工具、传了什么参数、拿到了什么结果、中间怎么判断的。一条时间线，从头到尾［2］。

回放的价值在三个场景里兑现。**客户质疑时**：不只能给"数字哪来的"，还能给"当时怎么一步步推到这个数字的"——第二节的证据引用是照片，运行轨迹是全程录像。**事故复盘时**：出了错，逐事件回放定位到具体是哪一步开始偏的——第 11 章的审批工单记录了"谁批的"，运行轨迹补上"批之前系统看到了什么"。**评测失败时**：回归集里挂掉的一条样本，打开轨迹看它挂在哪一步——这也接上了第 12 章的诊断，并引出本章最后一层。

四、第三层：失败要分类，混在一起就没法改

有了轨迹，异常就能诊断了。但诊断有个前提：**失败必须分类，因为不同类的失败修法完全不同**。

[2] enterprise_agent_platform（开源企业 Agent 平台工程）

生产环境里的失败至少分四类：

- **检索失败**——没查到该查的。修法在数据上：补索引、调召回、改切分。系统再聪明，喂不到料也白搭。
- **推理失败**——查到了，用错了。修法在模型和评测上：改进提示、加判定规则、把这条案例进回归集。
- **工具失败**——调了工具，工具没成。修法在工程实现上：超时、重试、降级（第 14 章的韧性四件套）。
- **数据失败**——源头就是错的。修法在客户那边：上游数据质量治理，这可能要回到第 10 章的数据契约去谈。

四类失败，四个方向、四种责任归属。混在一起（最常见的混法是全部笼统叫"模型不行"），就不知道从哪儿改起——往检索失败上砸模型升级，往数据失败上砸提示词，钱花了不少，错照出。运行轨迹是分类的依据：回放到失败那一步，看它卡在取料、判断、执行还是源头，类就分出来了。

五、自证的章法：四步标准动作

架构给了三层能力，怎么用它们面对质疑？有一套四步的标准动作：

第一步，质疑具体化。不说"我们的系统总体是可靠的"，说"您指的是哪一个案例，我把它调出来"——把泛泛的质疑落到具体的单据上。

第二步，证据上桌。证据引用加运行轨迹，当场放出来——来源、定位、过程，三样摆在桌面上，而不是"我们回去查了给您答复"。

第三步，归因分类。如果确有问题，当场分类——是四类失败里的哪一类，责任边界在哪（数据源头的问题归客户那边治理，推理的问题归我方修复）。

第四步，修复承诺。这条失败案例进回归集（第 12 章的错题本），修复后过门禁再上线——给一个有验收标准的期限，不给一句"我们优化一下"。

这四步把"自证"从一场辩解，变成一次质量改进的现场演示。同样是"你们错了"的质问，走过四步的客户看到的不是一家公司在认错，而是一套系统在运转——很多长期合作关系里，一次处理漂亮的质疑，抵得上十次顺利的汇报。这套动作与第 8 章的责任交接表、第 17 章的移交清单前后呼应：责任表写了"出了事找谁"，这里回答"找到人之后，对方拿什么给你看"。

六、留痕的成本，与一次审计的回报

证据链不是免费的。逐事件的全量轨迹意味着可观的存储和记录开销——每个运行都存完整录像，账单会先于价值到来。所以记录要分级：**关键运行全录**（涉资金、涉审批、涉外发的），**常规运行抽样录**

（抽检比例与评测抽检对齐）。

分级的依据与第 11 章的风险分级同一套——高风险的动作配高规格的留痕［2］。这笔存储账最终记进第 14 章的成本治理，这里先立住原则：留痕的规格跟着风险走，不跟着洁癖走。

最后用一个场景，看这堵墙翻过去的商业回报：一家客户的审计部门例行抽查，随机抽了 20 个历史输出，要求逐条提供依据来源。供应商当天返回：20 条里 18 条证据链完整可溯——来源文件、定位、运行轨迹齐备；2 条因当月记录策略调整缺失，附了说明与补录方案。审计部门没有再问第二句。续约谈判桌上，这个 18/20 的成绩后来变成了供应商方案里的一页——**证据链完整率，成了续约的筹码**。

至此，八堵墙里最难量化的两堵——效果可证、结果可信——都立起来了。第三部分的最后两组墙是同一枚硬币的两面：成本账单和稳定。这是下一章要讨论的。

---

[2] enterprise_agent_platform（开源企业 Agent 平台工程）

第 14 章 · 演示很便宜、上线账单爆炸、每天抖动，怎么办?

> 善战者之胜也，无智名，无勇功。
> ——《孙子兵法·形篇》

演示时一次调用几分钱，上线后月账单五位数；演示跑一百遍都顺，上线第三天系统抽风。这两件事经常被当成两个问题——一个归财务，一个归运维。

其实是一个病：**演示架构里没有成本和故障的位置**。演示环境天然短链路、低并发、干净数据、人工盯场——成本与故障还没有充分暴露。生产环境把这些保护撤走，病根才显形。解法是把成本和稳定性都变成**明确的工程对象**——成本按运行归因、可设预算熔断，稳定性按承诺管理、可降级回退。

一、账单解剖

月底账单到手，五位数。项目经理逐笔看过去，没有哪一笔是错的——每一笔都是系统在"正常干活"。成本失控从来不是偶发故障，是结构性的。

病因一，上下文膨胀。演示走的是单轮问答，生产走的是几十步的长链路——每一步都把前文带上，上下文越滚越大，而计费跟着上下文走。链路比演示长一个量级，账单自然也长一个量级。解法：任务链按步骤切分，每步只带必需的上下文，中间结论落盘而不是塞进对话。

病因二，无缓存。同一个问题，白天问了二十遍，每遍都全价重算。解法：高频问题的答案缓存复用，变更触发重算——省的是重复，不是质量。

病因三，模型错配。简单任务用最贵的旗舰模型，等于开着劳斯莱斯幻影送外卖。解法是第三节的主题：路由。分类、抽取、格式转换这类活，小模型又快又便宜，质量不差。

病因四，重试风暴。失败自动重试，没有上限——一个下游服务抖十分钟，重试把调用量放大几倍，最贵的那几分钟恰恰是故障的那几分钟。解法在第五节：重试要有限、要退避，还要有熔断兜底。

这四类病因构成账单爆炸最常见的结构性来源，但逐项看，没有一项是"浪费"：每一笔都合规、都必要、都在干活。

成本问题的第一课：先归因，再谈优化——不知道钱花在哪，优化就是无头苍蝇。

二、成本归因

[1] enterprise_agent_platform（开源企业 Agent 平台工程）

归因的工程形态：每一分钱，落到**哪次运行、哪个租户、哪个工具**头上［1］。

按运行归因，账单从一串总额变成一张地图：这次账单涨上去，是哪类任务带来的？哪个客户的用量在爬坡？哪个工具的调用在膨胀？成本看板不是财务的报表，是运行手册的一部分——值班的工程师每天扫一眼，异常趋势当周处理，不等月底。

看板之上是刹车：**预算熔断**。单次运行的花费超过阈值，自动暂停、转人工确认——不是省钱的小气，是止损的保险丝：一个跑飞的任务链（第 11 章担忧清单的第三条），有熔断就是"暂停待确认"，没熔断就是"跑了一夜，账单见"。熔断复用的正是第 11 章的机制：超阈值触发审批工单，人工放行后从检查点续跑——用的是同一套机制，只是多了一个场景。

拿一组数字做账单会诊：月账单四万八，拆开看——上下文膨胀两万一（一条三十步的流程链每步全量带上下文）、无缓存九千（高频重复问题全价重算）、模型错配八千（抽取任务全走旗舰模型）、重试风暴一万（一次下游抖动被无上限重试放大）。四项各治各的：链路切步、高频缓存、任务路由、重试设限。若分别压住这些结构性来源，次月账单可降至一万三，功能不受影响。

成本优化的空间在结构里，不在功能里。

三、模型路由

错配的正面解法是模型路由：**按任务难度分级，把任务派给够用的模型**——分类和提取走小模型，模糊条款的复杂推理走大模型；涉及关键写回时，除了看能力和价格，还要切到经过验证的受控执行路径，先校验、后写入，必要时等待审批［2］。

在实际工作里，成本约束下的模型选择从来是常态。路由设计的关键是**进架构，而不是事后补救**——事后补救的路由是补丁摞补丁，架构期的路由是任务分解的自然产物。

这里与第 9 章的判断对上了：模型编排（哪个任务给哪个模型）本身就是第 9 章所说 Harness 的五要素之一［2］——它不是成本优化的技巧，是生产架构里优先级很高的一件事。

四、抖动的来源和解法

接着说稳定性。先把"抽风"的来源说清楚。

生产系统的抖动有四个来源：**依赖服务抖**（下游接口、模型服务超时）、**模型输出抖**（同一输入，两次输出不一致——概率系统的本性）、**数据源抖**（客户数据更新、字段漂移）、**负载抖**（并发峰值，月底结账时全公司一起用）。

[2] Inside an Applied AI company（Pace 长文）

四个来源，没有一个是"修好就不再发生"的——抖动是常态，承诺的方式才是工程。

这个领域的通行做法是给服务水平承诺（SLO，Service Level Objective），而 SLO 的正确写法是**三个维度分开承诺**[1]：

- **可用性**——多大比例的请求正常返回（99.9% 与 99% 是两种完全不同的系统）；
- **延迟**——多大比例的请求多快返回（平均快没有意义，要看长尾）；
- **质量**——多大比例的输出达到合格线。

第三条最容易被漏，也最要紧：**质量承诺的口径直接来自第 12 章的评测集**——"合格"不是形容词，是回归集定义的水位。三个维度分开写，是因为它们的成本曲线不同：可用性每加一个 9，冗余成本翻着涨；延迟压长尾，靠的是架构不是算力；质量的水位，靠的是评测循环。混成一句"系统稳定运行"的承诺，等于三个都没承诺。

五、韧性四件套：坏了怎么办

SLO 是承诺，韧性是兑现承诺的机制。四件套，件件针对一类死法：

超时——别等死。每个外部调用设时限，到了时限就算失败，交给下一步处理。无超时的系统里，一个卡住的服务能拖死整条链。

重试——有限、退避。重试要设上限（防风暴，第一节的病因四），两次之间要指数退避（给下游喘息），盲目立刻重试等于给着火的房子泼汽油。

幂等——重试不重复扣款。同一个操作重发一次，效果和发一次相同。没有幂等的重试是危险的：超时之后你不知道对面到底成没成功，再发一次可能就是两笔。

降级——预案分级。小毛病小模型顶替（路由降档），中等故障规则引擎顶替（退回确定性逻辑），最坏情况**回退人工和表格**。这也在第 8 章里提到过：敢写"退回人工"的方案才敢上线——降级路径不是失败的遮羞布，是设计出来的安全网。客户对这套分级的感受，恰恰是安全感的来源：系统坏了有去处，天塌不下来（见图 14–1）。

[1] enterprise_agent_platform（开源企业 Agent 平台工程）

图 14–1：上线后的工程问题要同时看成本归因、模型路由和故障时的可恢复性。

六、一页纸运行手册

把本章所有内容收拢成给客户的一页纸——运行手册：告警谁看（第 8 章四问）、出了什么事故翻哪页（四类失败，第 13 章）、什么时候按回退按钮（降级三级）、成本看板在哪、熔断阈值是多少。**写给客户方值班的人看，不是写给工程师看**——手册只存在乙方电脑里，等于没有（第 8 章的纪律，这里再强调一次）。

最后是成本的一个追问：压完结构、写完承诺，客户还是嫌贵，怎么办？先看一个维修调度的算账场景：智能体每天的成本是两千美元，它替代的是"派哪位工程师去维修"的决策；如果派错人的损失远高于两千美元，账就不能只拿调用费和零比。这个场景要问的不是话术，而是成本的第一性问题：**和什么比**？与被替代的人工决策成本比，与出错的成本比——而不是与零比。

第三部的八堵墙到此拆完：数据与权限、集成、工具边界、运行可控、效果可证、结果可信、成本与稳定、责任可移交（最后一堵在第 17 章分析）。

回头望去，这八堵墙立着一条共同的原则——**生产系统的第一设计约束不是"更聪明"，是"坏了怎么办"**。可预期的小毛病，好过不可预期的聪明；善战者无赫赫之功，运维良好的系统没有救火事迹。

但工程底座垫好，只回答了"系统能不能站稳"。系统站稳之后，真正的考验才刚开始——没有人用，前面的一切都是成本。那是第四部要讨论的事。

第四部 · 系统上线之后才真正开始

> 本部共四章，回答一个总问题：**系统在技术意义上"交付"了，怎么让它在组织意义上"活"下来——有人用、有人撑、撤了你自己也能转。**

第三部收官了：数据接上了，动作管住了，效果可证，账单可算。但生产系统的敌人不止在工程里，更在组织里——一套验收满分的系统可以在三个月里安静地死掉，死因不在代码，在激励。

第 15 章先解剖这个静默死亡：上线不等于激活，三个死法全发生在"上线成功"之后，解药是关于激励的三个问题、渐进嵌入与两类盟友的经营。

第 16 章把钥匙交到它真正的主人手里：人工智能落地天然跨部门，预算、流程、数据、受益四张牌只有一把手同时握得住——一把手工程不是口号，是预算、激励、失败三种背书，两道门，一个越滚越厚的飞轮。

第 17 章定义交付的终点：验收、采用之后还有第三段——移交；撤场三件事、移交阶梯、三个反模式，核心一句：离不开 FDE 的系统不是交付，是寄生。

第 18 章把目光转回中国：采购结构、信任结构、定义权都不一样，四条结构差异长出两个本土形态——中国的约束更紧，正好把本书的方法当极限测试。

四章是一条递进的链：先让系统被用起来（采用），用起来要有组织撑腰（一把手），撑稳了才算交接（移交），交接的难易最终由土壤决定（中国）。本部的第一读者是**要为"系统活了"负责的人**：带交付的项目负责人、客户的信息化负责人、以及被"上线了没人用"困住的经理。工程到此退半步，组织上前半步——因为这一部的每一步，都是从一句会议室里的话开始的："大家怎么还不肯用？"

第 15 章 · 系统上线了，为什么没人用?

> 给我看激励，我就能给你看结果。
>
> —— 查理·芒格

系统终于上线了。

验收会开得很体面：技术指标逐项打钩，双方项目负责人握手合影，甲方信息部门的一句"系统正式投入运行"念完，掌声整齐。

三个月后回访，却看到了另一种景象：上一次的登录还停留在上线后的第二周，业务骨干的桌面上还是那张用了六年的电子表格，问起来，回答是"那个系统啊，偶尔开一下"。

没有事故，没有投诉，没有反对——系统就那么安静地、体面地死了。

这一章讨论的问题是：系统上线之后，技术验收全过、指标全绿，但业务方为什么就是不用?

一、上线走的是流程，激活发生在行为里

先拆一个被混用的词。"上线"走完的，是采购和验收的行政流程：合同履约、部署完成、验收签字。而系统能不能算真的活了，标准只有一个——目标用户群体在日常工作中，形成了对它的稳定使用习惯：不是演示会上鼓掌，是三个月后没人催，它依然被高频使用。

企业软件行业内从来不缺这样的项目：功能清单百分之百打钩，但用的人只有百分之五；系统可用性做到四个九，业务部门却宁愿继续用电子表格。

为什么企业部署普遍死在激活这一环？因为它要过的三道关，全都不在代码里：用户嫌它不顺手——比旧习惯多一步，就没人用；用户觉得它不可信——错一次，信任就清零；用户觉得它与自己无关——那就没人碰。 [1]

三道关没有一道能在总部解决，只能到客户的会议室、车间和工位上，一道一道去过。

这正是 FDE 与"交付即走"的传统实施的分水岭：传统实施把验收单当终点，FDE 把客户行为的改变当终点。

二、"上线成功"之后平静地死去

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

一位一线 FDE 从业者在播客里把失败项目的原因归成以下三条：

第一，一把手对 AI 能做什么有超出现实的预期。以为上了 AI，企业就起飞，把它当超人而不是工具——预期过高注定崩盘，崩盘的结果就是弃用。

第二，让技术部门立项，而不是让业务部门立项。业务部门把自己当甲方、把技术团队当乙方，这样立项的系统，几乎一定会失败——懂业务的是业务团队，技术团队提供不了选品、预估、谈判这些知识和经验。系统于是成了技术团队的业绩，不是业务的工具。

第三，也是最底层的：组织的激励机制没有跟着变。嘉宾原话是——"大家不要把它看成一个工具的上线，它是一堆员工上线，一堆新的生产力来了，生产关系要改变了"。[2] 翻成大白话：系统不是多了一个软件，是多了一批"新员工"——新员工上岗，考核、分工、汇报关系都要跟着重排；如果还按管理一件工具的方式去管它（不动考核、不动分工），这批"新员工"自然干不起来。

注意这三条死法的共同点：它们没有一条发生在上线之前，全都发生在"上线成功"之后。上线那一刻项目还是满分，死是在后面的三个月里，一天一天安静地发生的。而三条死因里，前两条指向一把手（会在下一章里详细展开），第三条——激励——是这一章重点要讲的，先看下其他来自一线的真实反馈。

一位在客服场景里做了四版迭代的工程师，四版方案从编排框架一路演进到带工程外壳的本地部署，技术上是成功的；被问到项目对业务的实际帮助时，他的回答是：[3] 不裁员，就体现不出投资回报。他们的客服团队一个人没裁，省下的时间大家去"创造更好的价值"——但投资回报的账因此算不清了，因为在多数老板的算法里，省时间不算数，省人头才算数。投资回报的账和员工的利益，在这里直接冲突了。

还有一位从业者抱怨道："我们用了 AI 反而是 997，不用 AI 是 996。"——工具增加而不是减少了工作量：系统崩了要抢救，记忆丢了要重喂，流程卡了要手动疏通。[4]

这些真实反馈都不是"项目失败"的抱怨——技术能力没问题。

真正卡住的是结构：省下来的时间折不进投资回报的账，多出来的工作量没人重新分配。

项目做对了，用的人处境却没有变好，这才是真正问题所在。

三、培训解决不了的问题，激励能解决

把第三条死法展开成一个可操作的诊断工具。对一线用户，用不用取决于三个问题——不是问出来的，是用户在心里自己算的：

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

[3] AI 职觉：别再卷 Agent 了（小宇宙播客）

[4] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

其一，用了这个系统，我的考核变了吗？如果考核还是老指标，那么学习新工具的每一分钟都是无偿加班。

其二，我省下来的时间归谁？如果效率提升的收益全部归公司，成本却由我承担（学新工具、兜新错误），这笔账没人愿意签。

其三，系统出错算谁的？如果新系统的错误算我的，旧流程的错误算"正常的"，那么理性人的选择就是守住旧流程。

三个问题里任何一个答不好，培训做得再漂亮也没用——培训解决"会不会"，激励决定"愿不愿"。前文那位从业者举过一个正面对照：销售团队采用新工具时，配套把考核从百分之百看结果，改成八成看结果、两成看过程——用新系统沉淀的过程数据可以直接换算成考核积分，工具的采用才真正落了地。

激励设计还有一个更彻底的正面样本。一家住房租赁平台里有个"管家"角色，管租客的大事小情：邻居家狗叫、空调漏水，全是带情绪的琐事。人工智能团队进场后没有做"自动回复"，而是先重新划了分工：机器接走全部琐事应答，人去做有温度的关怀——再配一个探查销售机会的智能体，提示管家"这家租客养猫、这几天要出差，可以推上门喂养"。

一年下来，人效从一个人对五百个租户提到一个人对一千二百个，第二年目标两千。关键的是：这个部门没有裁员。[2] 提效不与减员挂钩，员工才不会把系统当敌人——"不裁员"在这里不是道德姿态，是激励设计本身。

四、员工的敌意：怕，会变成不配合

这两年，"教会 AI，自己就下岗了"变成了打工人之间心照不宣的黑色幽默——"蒸馏"本来是模型训练的术语，"蒸馏同事"也开始流行起来，专指"把老员工的经验喂给 AI，然后老员工被优化掉"这件事。

玩梗背后是真实的时代背景：密集的 AI 相关裁员新闻，把"我的经验值不值钱"变成了每个一线员工心里随时会响的一道警报。

自保是本能，而本能触发之后，很少有人会当面说不——更常见的是把警惕悄悄嵌进日常配合里：教是要教的，但留一手；配合是配合的，但能拖就拖，能刁难就刁难。

激励设计解决的是"划算不划算"，但账算得再清楚，也拦不住员工从心底里不信任这套系统——这种不信任比"不划算"更难对付，因为它经常不说出口。

一家做企业智能体交付的公司，在给一线老师傅配"AI 学徒"时，"老师傅会说，你把我的经验蒸馏了，把我的东西都学走了，那我之后在企业的价值也没了。"[2] 这句话精确地说出了前几节没讲透的另一半：前面讲的是"这笔账划不划算"，这里讲的是"就算是划算的，可我就是不想用你"。

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

那家公司的应对措施拆开是两层。第一层，互惠先行："你要学习他的东西，首先你要给他付出一点东西。"他们的 AI 学徒进场不是先偷师，是先干活——老师傅要排班时给建议，要做营业预估时提建议，老师傅先觉得"这徒弟有用"，才会真正教它。第二层，重新定义价值："一个能被蒸馏出经验的优秀基层员工，其实是企业最大的资产，你不应该放弃他"——因为商业场景每天在变，AI 能帮你想很多事，但替代不了一线的判断和决策。

更进一步，这种员工的价值不是变小了，是变大了：以前一个金牌销售厉害，是自己能卖两三倍，再厉害一点，是能带出一个团队卖两三倍；现在，他积累的判断和方法论能够复制给全国一千个、一万个销售——对企业的价值只会更大，对应的激励也必须跟着涨，不能维持原状。老师傅在系统里的新身份，是训练师和权威，不是被替代的对象。

同样的恐惧，来自于一位企业 AI 落地顾问遇到的客户："它都全自动了，还需要我点一下干嘛？我这个人有什么必要再存在呢？"[5] 这不是矫情，是真实的存在焦虑：如果一个人在流程里的全部职责被压缩成"点一下确认"，这个岗位本来就是悬空的。采用设计必须正面回答"人在这条流程里的新角色是什么"，哪怕新角色只是判断与担保，也要明确告诉他、写进新的岗位职责里——不说清楚，员工只能自己在焦虑里瞎猜，猜的结果通常是往最坏处想。

怕不会总是说出口，更多时候会悄悄变成行为，而不是摆在台面上的反对。最常见的四种：

拖——该反馈的问题拖着不说，等系统自己撞上去；护——明里配合、暗里继续用老流程"保底"，出了事好证明"我早就说了不行"；刁——鸡蛋里挑骨头，用一堆边角需求和"不好用"的抱怨拖慢验收，让系统看起来比实际更不成熟；散——用最负面的措辞把体验讲给同事听，把观望者提前拉进对立面。

这四种共同的杀伤力在于，它们都披着"配合"的外衣，比明着反对更难发现、更难拆解。

化解这种不配合，实战中有三招屡试不爽：别让系统以替代人的姿态出现（人工智能处理重复活，人处理要判断的活）；先挑用户最烦的重复劳动开刀，让他先尝到甜头；让受益者替你说话——受益用户的现身说法，比你自己解释一百句管用。[6]

前面管家案例里"这个部门没有裁员"的承诺、周报自动化里"从写周报变成把工作留好"的渐进过渡，本质都是先接住这层恐惧，再谈激励——恐惧不解除，账算得再清楚也没人敢用，甚至会有人暗中盼着系统出错。而这层恐惧真正的解药，往往不在项目团队手里：员工要看的不是某个负责人怎么说，是组织最高层怎么表态，这道门谁来开、怎么开，下一章详细展开。

五、渐进式变革：别一步到位

激励之外，采用的第二战场是系统进入工作流的方式。

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[6]《关于 FDE 的 100 个问题》（AI磊叔）

这里有一条被反复验证的规律：嵌入旧习惯，胜过另起炉灶。

一家投资机构的周报自动化，走过了教科书式的三个阶段：第一阶段，在协作平台上搭了个日报机器人，填写还是要靠人；第二阶段，改成语音对话——下班路上跟系统聊几句"这周忙了什么、卡在哪"，系统梳理成周报；第三阶段，连语音也不需要了，直接授权系统读取工作留痕——聊天记录、文档、日历——自动汇总生成。人从"写周报"变成"把工作留好"。［4］注意这条路的方向：不是一次性把人甩出流程，而是每一阶段都让人少做一点、多保留一点确认权。

反面的教训更深刻。有一个流传很广的案例：一群执行了多年、共十一道工序的老员工，拿到了一个"一步到位"的智能体版本——结果他们停用了。改成保留原有步骤、让智能体逐步完成每一步、且人能看到中间产物之后，采用才回来。［7］道理不玄：中间产物是人的确认点，确认点是信任的载体。把所有中间步骤藏进黑盒，等于要求用户用职权为黑盒担保——没人愿意。

所以，做采用设计时先问一个朴素的问题：我做的东西，是想改造他的工作方式，还是他愿意顺手用起来?

这个问题在"全自动"场景下会被放大到极致。一位企业人工智能落地顾问提到客户对全自动质量的判断："AI 全自动出来的东西，往往就只有六七十分。"［5］原因很朴素：全自动的水平就是模型的能力，而模型是所有人共享的——模型进化，大家的能力一起进化，等于谁都没进化。全自动的平庸因此是采用的天花板：用户很快发现系统产出与自己的职业声誉不匹配，弃用只是时间问题。反过来，那被保留下来的人的二十分判断增量，恰恰是系统输出的及格线与优秀线之间的差距——这也是渐进嵌入要保留"中间产物"和"确认点"更深一层的原因：不是流程上舍不得，是那二十分离不开人。

人应该是问题的答案，而不应该是问题的障碍。

六、把组织经营成采用的基础设施

激励与嵌入解决的是"要不要接住"和"怎么接住"，员工的敌意解决的是"信不信得过"；把时间拉长到整个项目周期，采用还有一个组织层——不是防御恐惧，是经营盟友。两类人物值得单独点名（见图 15–1）。［1］

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[4] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[7] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

图 15–1：采用发生在用户行为里，需要工作流、激励、支持与度量一起运转。

支持者：客户内部真心相信这件事、愿意押上自己信誉为你开路的人。对他的经营有三件套——给他能讲出去的材料，把他的远见变成他的战功，失败时替他挡在前面。一个被善待的支持者，胜过十场产品宣讲会。

影响者：每个组织里的无冕之王——车间老师傅、部门里"问他就行"的人。他们不掌权力，但掌握信任；一句"这玩意儿还真行"，抵过管理层三封全员邮件。让他们第一批试用、认真对待每一条吐槽、把采纳建议显性化（"这个字段是按王师傅的意见加的"），是破解"不是我做的东西我不用"的最短路径。

最后是节奏。部署期的迭代要以"天"计，而不是以"版本"计：上午业务用户说输出少了字段，下午字段就加上；今天车间主任说戴着手套点不准按钮，明天按钮放大一倍。这种响应速度的意义远超功能本身——用户在第一个月就会形成判断：这个团队是来真的，还是又一个交完就走的。配套的排序规则也和产品期相反：不按功能重要度排，按"使用阻塞度"排——什么在阻碍明天的使用，什么就是今天的最高优先级。

培训亦然：培训不是宣讲，是陪跑——陪跑真实工单一周，比一堂课有用。

七、采用要有度量，口径先立规矩

采用的改进要有仪表盘，两个指标就够用。

激活率：多少目标用户每周真实使用系统——注意三个口径要素，分母（目标用户是谁，按第 8 章验收契约里定的人头算）、窗口（每周还是每月）、判定方式（登录算不算，还是要产生真实任务）。

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

分流率：同类任务里，有多少从旧路径流进了新系统——它比激活率更诚实，因为它量的不是"有没有打开"，是"活有没有被接走"。 [1]

口径纪律与第 8 章同一条：模糊口径下的漂亮数字，不如清晰口径下的难看数字。"80% 的用户曾使用过系统"和"60% 的目标用户每周在系统里完成至少一项真实任务"，后者小，但只有后者能指导改进。

这一章从失败原因讲到激励、再讲到渐进嵌入，讲的其实是同一件事：系统进入组织，是一场利益与惯性的博弈。

而这场博弈的规则，只有一个人才能改变，那就是一把手——因此，下一章，我们讲一把手工程。

第 16 章 · 为什么 AI 落地是一把手工程?

> 上下同欲者胜。
>
> —— 《孙子兵法·谋攻篇》

先看一个对照实验。同一个 AI 落地项目，在两家几乎一样的公司，立项方式不同。

A 公司走技术部门立项：立项书规整、选型合理。项目死在了第 4 个月——回过头复盘，死因是要调用销售部门的数据；销售总监说，这是你们信息部门的项目，凭什么给你开权限。

B 公司走一把手立项：项目启动会上老板讲了十五分钟为什么这件事关乎明年的生意，六个月后系统跑进了三个部门的日常。

两家公司的技术方案是同一套。差别只有一个：A 公司的项目有一个好架构师，B 公司的项目有一个能调动全公司的签字人。

"一把手工程"这四个字，在国内企业软件圈说得快要磨出茧子了，以至于很多从业者听到就本能地反感——它听起来像一句推卸责任的口号，项目黄了，一句"没有一把手支持"就交代了。这一章想把这四个字从口号还原成结构：为什么人工智能落地几乎必然是一把手工程？它的正确含义是什么？错位的形态长什么样？以及它什么时候可以不是。

一、跨部门的难题

人工智能落地有一个别的技术项目没有的特点：它天然跨部门。

拆开任何一个真实项目，你会看到一张四散的地图——数据在 A 部门（客户信息在销售系统里，历史记录在运维手里），流程在 B 部门（要改造的审批流归财务管），预算在 C 部门（立项要过信息技术预算，受益却算在业务头上），受益在 D 部门（效率提升的账记在一线团队的报表里）。

一个中层项目负责人，手里能同时调动这四样吗？一项都难。数据权限要 A 部门点头，流程改造要 B 部门让渡权力，预算归属要 C 部门认账，受益确认要 D 部门配合度量——每一项都是跨部门的谈判，而中层手里只有"协商"，没有"裁定"。组织里唯一同时握着四项调度权的位置，就是一把手。

这不是推演，一线已经有实例。

先看一个反例：一家大厂交付团队——客户某个业务部门想做业务智能体，但没有预算；预算在 IT 团

[1] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

队手里，而 IT 与业务分属不同部门——乙方只能把方案设计成"既能满足 IT 的先进性，又要满足业务的效果"，最后发现"IT 要的先进性不一定是业务要的效果，业务要的效果可能不一定是最先进的"，生产关系怎么都摆不平。 [1]

对照的正面例子来自一家制造业：董事长本人明确要拥抱人工智能，选定销售端、选定区域，自上而下推——财务、IT、销售线全部动了起来；试点从最落后的一个省做起，半年后成了全公司最先进的省，再复制到全国。同一道数学题，一把手在不在等式里，答案完全不同。

还不止于此。推动改变的是惯性和利益，而改变工作方式（惯性）的权力，同样只在一把手手里——考核指标、编制增减、部门职责边界，这些是公司治理层面的东西，不是任何业务或技术部门能单方面动的。所以这不是"请领导重视"的姿态问题，是一道权限的数学题：需要的权限总和，只有在公司管理层的金字塔尖才凑得齐。

二、优先级达不到，落地路走不下去

那么，是不是随便什么项目，只要找到一把手就万事大吉？也不是。帕兰提尔早期高管鲍勃·麦格鲁（Bob McGrew）给过一个更锋利的筛选标准：切入点必须是领导层认定的关键问题，最好在首席执行官最关注的五个问题之列。他的理由很现实——只有这个量级的问题，才能帮你碾过企业内部的官僚阻力；不到这个优先级，客户组织没有动力陪你走完落地路上那些艰难的部分。 [2]

这个标准的反面同样重要。你作为乙方实际能接触到的，常常是信息负责人或者某个赞助人，离首席执行官还隔着几级——而这几级的距离会改变问题的性质：隔层的人有他自己的考核，他宁愿你解决对他容易的问题，也不愿你碰对他危险的问题。于是项目在源头就被"降维"了：本来该解决业务的核心痛点，最后做成了一个安全的、边缘的、谁也不得罪的演示性项目。这正是第 2 章里提到的一种组织层面的失败方式——不是问题不值得解决，是你够不着那个问题。

一条更完整的失败链，来自一位企业软件公司首席执行官的观察：董事会要"更多人工智能"，首席执行官于是找咨询公司做一个集中式大项目——脱离具体的业务部门，做完没人知道它怎么工作、运营没有对齐，"这些项目会失败"。 [3] 注意这条链的讽刺之处：它不是一把手缺位的失败，恰恰是一把手只在开头出现、中间全程缺席的失败。签了字，然后呢？

三、"三不懂"：交集是空的

为什么人工智能项目特别依赖一把手把人拉到一张桌子上？因为组织里存在一个结构性的空洞，业内流

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[3] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

[4] 有在做 FDE 这个工作的佬吗（linux.do 帖）

传的概括是三句话：懂业务的不懂人工智能，懂人工智能的不懂业务，都懂的没有权限。 [4]

这不是能力问题，是组织内分工的问题。业务专家的知识长在多年现场经验里，人工智能团队的能力长在技术栈里，两套语言、两套考核、两个圈子；而同时跨过两边的人，在多数组织里职级不高——他们太新，还没爬到能调资源的位子上。三者交集为空的结果，就是每一个项目都在做"翻译接力"：业务把需求讲给产品，产品翻译给技术，技术做出来再翻回去——每转一手都会丢上下文（第 6 章的转述层损耗，在组织内部原样重演）。

能把这三类人同时按在一张桌子前的，只有一把手。把这几方拉到一起，是让"业务的不懂"和"技术的不懂"在项目内部互相抵消——业务专家现场纠偏，技术专家现场评估，权限的问题当场定。FDE 之所以被称为"前线部署工程师"，之所以要坐进客户现场，本质上就是把这个翻译接力的中间环节拉到同一张桌子上。

四、一把手工程的真实含义

把口号拆开，一把手工程在项目里的实际形态是三种背书，缺一不可。 [5]

预算背书：跨部门项目的成本归属，只有一把手能裁定。人工智能项目的受益部门和出资部门几乎总是分离的——钱从信息技术预算出，效率账记在业务部门头上，这种项目在任何一个部门的账本里都是"亏的"，只有站在公司整体的账本上才算得平。谁看公司整体的账本？一把手。

激励背书：第 15 章里讨论的激励问题，在部门层面无解，只有在一把手层面才有答案。考核要不要改、编制要不要动、出错算谁的——这些是公司治理的动作，不是项目经理能承诺的。一把手不公开站台，第 15 章提到的三个死法就会一个一个应验。

失败背书：允许第一个项目死得有意义。人工智能项目的第一个交付，很大程度上是一次组织学习——学怎么跟智能体协作、学怎么度量、学怎么定验收口径。第一个项目的学费，需要有人以组织的名义承担：否则每个部门都会学会一件事——别碰，碰了出错算自己的。

三种背书形态，有几个实际例子。

一个是需求方的态度变化：企业一把手对人工智能的态度变了，"天壤之别——以前一直在讲怎么用、能不能用，现在直接跳过进入下一阶段：什么时候开始帮我用上"，拐点出现在能执行任务的智能体出现之后——问题的性质从"要不要"变成了"多快"。 [1]

另一个背书的姿态则最扎实、最贴近一线：第一节那家制造业的董事长，一年之内亲自给全员和管理干部讲课超过十五场——"我先学会，我再给员工讲，那员工才能去理解我的思想、能用好这些系统"。 [1] 思维转型不是仅仅开一场动员大会就能解决的，是一把手把自己变成第一讲师——这同时覆盖了

[1] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

[5] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

上面提到的三种背书：真金白银的时间投入（预算）、最高规格的公开站台（激励）、以身试错先交学费（失败）。

陆骁鹏（Ventus AI 联合创始人，服务北美医疗行业）曾分享自己的观察：能真正推动 AI 的老板只有一个共同点——自己得 AI native（自己就是 AI 的重度用户）。他的策略是"先搞定标杆，再向下推"［6］。

这不是抽象的态度问题，落到执行上是实打实的摩擦成本。

陆骁鹏对比过两个规模相近的客户：一个老板的推动理由是"别人都推我也想推"，另一个是"我很确信 AI 现在一定要开始推，要投入多少就投入多少"——两边做变更管理的难度完全不一样；有 CEO 层面的坚定支持，推起来会快很多［6］。

这给"三种背书"添了一层机制解释：预算背书和失败背书之所以要紧，不是仪式性的表态，是因为老板对"要不要做"这件事的确信程度，会直接传导成组织执行时的摩擦系数——跟风的老板给得出预算，给不出"以身试错先交学费"的那份笃定。

用这三条去检验，"出席动员大会"算不算一把手工程？不算。出席是十五分钟的事，而三种背书要贯穿项目落地的那几个月。

真假背书的判断标准也很朴素：项目在第四个月遇到部门墙时，一把手是出面裁定，还是说"你们自己内部协商解决一下"？

五、错位的形态清单

一把手参与不足或参与错位的项目，会长出几种典型的形态，每一种都有识别信号。［5］

- **技术部门主导型**：系统是信息部门的业绩，不是业务的工具。识别信号：项目周报里全是技术里程碑，没有一个业务指标。这是三大死法之二，上一章已展开。
- **挂名型**：一把手出席了启动会，此后再未出现。识别信号：项目升级到需要跨部门裁决时，会议纪要里找不到比部门更高的签字。乙方的对应信号：售前阶段能见到的人，交付阶段再也约不到。
- **孤岛型**：项目在单一部门内闭环，数据、权限、流程跨不出去。识别信号：系统很好用，但只在三十个人的团队里好用，扩张计划永远在"下一季度"。孤岛型是四形态里最体面的——它至少活着——但它验证不了规模化的部分。
- **咨询依赖型**：就是那条"董事会→首席执行官→咨询公司→集中式大项目"的失败链。［3］识别信号：项目文档越来越厚，一线使用记录越来越薄。

[3] a16z：Box CEO 谈 AI Agent 与企业为何跟不上（YouTube）

[5] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

[6] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

清单不是给别人贴标签，是拿来检查自己手里的项目：如果你是甲方项目负责人，你手里的项目此刻更像哪一种？如果你是乙方，你的合同签在了哪一种上？

六、两道门与一个飞轮

一把手工程不是"搞定领导"的一次性动作，它有两道门，过完第一道还有第二道，过完两道之后还能滚出一个飞轮（见图 16–1）。

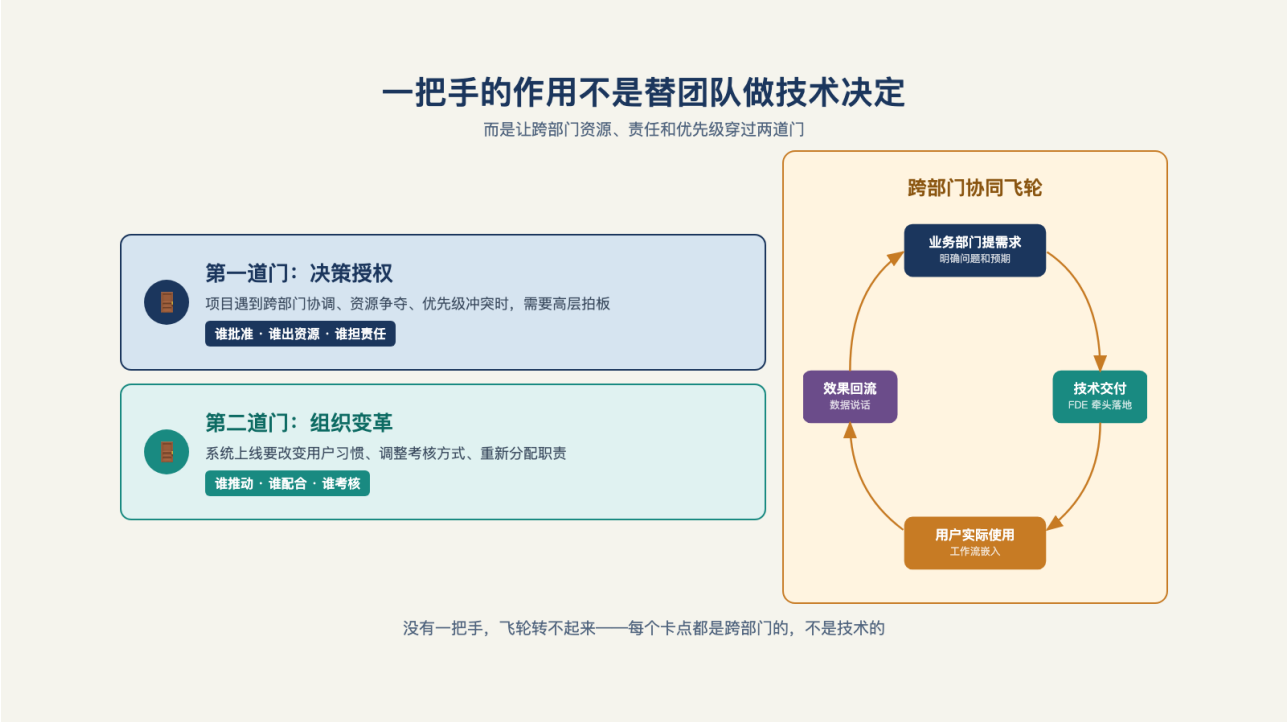


图 16–1：一把手的作用不是替团队做技术决定，而是让跨部门资源、责任和优先级穿过两道门。

第一道门，**校准一把手的预期**。上门要讲清四件事：人工智能能做什么、不能做什么、多久看到效果、要投入什么。预期不对，上线后任何一个环节不达预期，老板的感受都是"被骗了"——三大死法之一在门口就埋下了。一个常见的校准场景：老板开口就是"帮我减员三成"或"替代三千人"，这时候要当场把对话拉回价值——人工智能现在替代的是任务，不是岗位；一个岗位八成以上的核心任务都被接管，才谈得上替代，现在差得远；把"承诺裁三千"换成"交付七个可验证的提效结果"，双方才都活着。[7] 这道门为什么难跨？因为多数老板对人工智能的判断，来自昨晚刷到的短视频和网上流传的爆款案例，不是系统性认知——有从业者直言，落地过程里最大的隐性成本，就是老板的认知成本。[8] 校准预期，本质是把这层认知成本花在门槛上，而不是留到验收时才爆。

第二道门，**化解员工的恐惧**。第 15 章专门用一节拆过这层恐惧——它在部门层面无解，员工要看的不是某个负责人怎么说，是组织最高层怎么表态。一把手不出面，这扇门就从里面闩着。三招：别让系统

[7]《关于 FDE 的 100 个问题》（AI磊叔）

[8] AI 职觉：别再卷 Agent 了（小宇宙播客）

以替代人的姿态出现；让第一批用户成为受益者，再让受益者去影响观望者；以及最关键的一招——一把手必须公开站台，只私下批预算没用，要在公开场合表态"这是为了让大家做更有价值的事"。[7] 私下支持和公开背书的区别，员工用脚投票时分得很清。

两道门都过了，还有一项实操：把关键人表贴进周报——谁是支持者、谁是对接人、谁是潜在阻力、各自的关切是什么，每周更新，让组织图从静态的名片夹变成活的地图。

过门的功夫会不会每个项目重来一遍？一位从业者的做法把这件事变成了复利：第一次搞定一个不懂人工智能的老板最累，但每搞定一个，手里的材料就厚一层——培训稿、对齐话术、常见误区清单，下次直接复用；一线踩出的真问题还反哺总部产品，让下一个客户更容易接受。所以"老板换了一茬又一茬，手里的框架越来越厚"不是安慰，是滚雪球——个人手感一点点攒成组织资产；这门功夫本身就是前线部署工程师带得走的资产之一。

七、两面工具

最后把本章收成两张清单，一张给甲方，一张给乙方。

甲方项目负责人的向上管理清单：向一把手要三样——预算归属的明确裁定（谁出钱、怎么分账）、具体授权（书面指定项目负责人及其跨部门调度权）、激励背书（公开站台，最好配一条考核调整）；汇报时用第 8 章的验收口径说话——业务指标、时间窗口、判定方式，而不是技术进展的甘特图；每季度向一把手同步一次进展：这个项目在他的五个优先事项里排第几；排位跌出前五，要么把项目升回去，要么趁早体面收场。

乙方的售前尽调清单：合同里的一把手参与信号有哪些——立项会上谁出席？验收标准谁签字？跨部门数据调用的授权路径走不走得通？一个只和技术部门谈的合同，要按"项目大概率死在第四个月"来定价风险（第 20 章收费与第 27 章接单判断会接住这条）。还有一条善意的提醒：如果尽调发现客户的一把手只想在年报里写"人工智能"四个字，最好的生意是不做这门生意。

八、什么时候可以不是一把手

回到开头那个本能的反感——"一把手工程"被讲成万灵药时，确实会遮住一个降维的真相：规律的核心不是职级崇拜，是**权限与受益对齐**。[7]

当项目能在一个部门内闭环、数据不出部门、流程不碰别人的奶酪时，"一把手"可以降维成"流程负责人"——那个对这条流程的权限和受益同时负责的人。车间主任对车间的排产系统、财务总监对财务的审单智能体，都可以是有效的"部门级一把手"。

[7]《关于 FDE 的 100 个问题》（AI磊叔）

所以这一章的结论收在一句话上：人工智能落地的最小组织条件，不是"有一把手"，是"权限的覆盖面不小于问题的跨度"。问题跨了四个部门，权限就得站在四个部门之上；问题只跨一条流程，流程负责人就够。前者是例外还是常态？到目前为止，前者仍是常态——因为值得 FDE 出手的问题，几乎总是跨部门的。

一把手让你进门，也让项目活过了采用。但活的系统总要问最后一个问题：交付的终点在哪里——是验收签字，还是你能放心地走？下一章，我们讨论移交。

第 17 章 · 怎么把系统交给客户自己运行?

> 不幸的国家才需要英雄。
>
> —— 贝托尔特·布莱希特《伽利略传》

FDE 什么时候可以撤？两种撤场，结局天差地别。

第一种，体面地走：验收通过，庆功宴散，工程师撤出。三个月后客户没人提这个系统，监控面板一片绿灯——不是系统在跑，是没人再用。

第二种，也是体面地走：同样的流程，但六个月后客户内部开季度会，讨论的是"下个季度让智能体再接两个环节"，没有人再提"这是人工智能项目"——它已经长成了工作本身的一部分。

两种撤场在撤离当天看起来一模一样。区别要到后面才显形，而决定区别的动作，全部发生在撤之前。这一章回答交付的最后一段：怎么把系统交给客户自己运行，才算交付完成?

一、交付是三段，验收单只覆盖一段

传统项目制对"交付完成"的定义是一张验收单。但交付其实是三段：**验收**——系统在约定口径下达标（第 8 章）；**采用**——用户真的在用（第 15 章）；**移交**——客户自己能运行、维护、改进它（本章）。三段全过，能力才算留在了客户组织里；只有第一段，留下的是一套没人负责的软件。

有一位长期观察这个行业的从业者写过一句听起来自相矛盾的话：最好的前线部署组织，致力于消除自身的工作——建模板、造工具、把一次性的客户代码变成可配置的平台能力；目标不是消灭这个角色，而是把它移向更有价值的问题。 [1] 这句话的原意是说产品化（第 21 章展开），但把它挪到移交上同样成立，甚至更贴切：一个不能让自己变得不必要的交付团队，把每个项目都做成了自己的长期饭票——这不是敬业，是没让客户断奶。

判断断没断奶的标准很朴素：撤场六个月后，系统的小变更、小故障、新需求，还要不要回到你手里？要，说明判断力和运行知识没交出去；不要，才算交完。

二、撤场三件事

[1] @deployengineer（Bhaulik Patel）系列长文

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

一位从业者把 FDE 撤场的完成条件概括成三件事：业务的融合、知识的治理、系统的对接——"这三件事情都做好了，他才能够离开"。 [2]

第一件：业务融合。 系统要长进业务流程里，不再是"那个人工智能项目"。可核验的动作有三个：流程文档更新——智能体参与的环节写进部门的标准作业流程，替换它就得改流程文件，组织惯性反过来保护系统；嵌进日常——系统的使用发生在用户本来的工作界面里（第 15 章的嵌入原则），而不是一个要专门登录的孤岛；指标归属落地——系统的业务指标记进业务部门的报表，不是记在信息部门的项目总结里。这一件的判据，就是本章开头那句话的反面：六个月后没人说"这是人工智能项目"，它只是工作的一部分。

第二件：知识治理。 运行系统的知识，要从 FDE 的脑子里搬进客户的组织。搬什么？三样资产，正好都是前面各章的产出：运行手册（第 14 章的一页纸：告警谁看、出事翻哪页、什么时候按回退按钮）；评测集（第 12 章那套"带得走的复利资产"——以及它的维护责任：谁往里加错例、多久跑一次回归）；故障处置手册（第 13 章的四类失败分诊：每类故障的第一响应动作）。三样资产在移交清单上是实体条目，不是"已沟通"三个字。反面教材第 15 章见过："记忆丢了要重喂"——运行知识不文档化，人一走系统就退化。 [3]

第三件：系统对接。 运维责任接进客户体系：监控告警接进客户自己的运维平台（而不是只有你们公司看得见）；升级路径明确——谁批准变更、谁执行、停机窗口在哪；供应商责任边界写进合同——出什么级别的事故找谁、响应时间多少、什么情况下收费。第三件最容易在"关系好"的时候被跳过，然后在第一次半夜故障时被发现从来没人约定过。

三件事的共同点：它们都不是技术动作，是责任和知识的转移。技术早就交出去了，责任没有。

三、交付是下台阶，不是跳悬崖

三件事不可能在撤场日一次完成，交付要走一个阶梯，每一级有时间、有退出条件（见图 17–1）。 [2]

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

[3] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

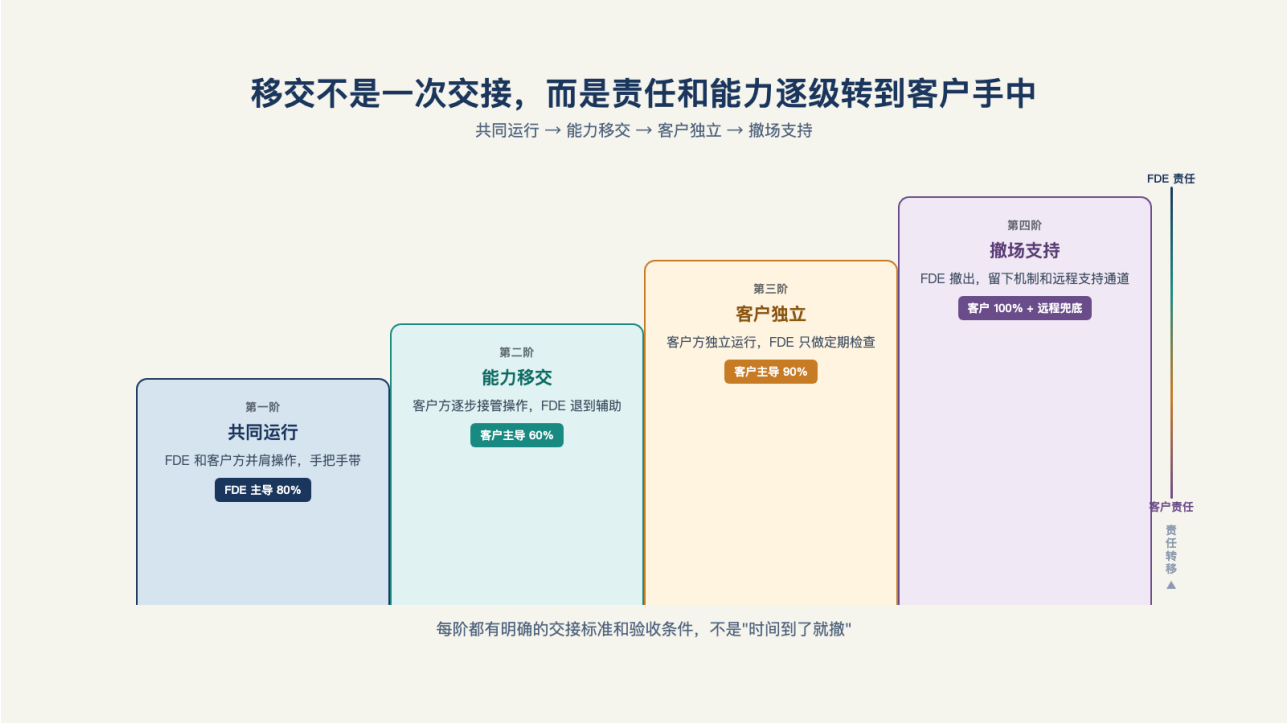


图 17–1：移交不是一次交接，而是责任和能力逐级转到客户手中。

第一级：陪伴运行。 客户团队开始操作，FDE 在旁边盯着——出错就修，但修的动作让客户看见。这一级结束时，客户能独立完成日常运行。退出条件：连续若干周无人工干预的常规运行。

第二级：影子运行。 客户操作，FDE 旁观——只记录不动手，每周碰一次，把客户自己没发现的问题攒成清单。这一级真正在交付的是判断力：什么情况算正常、什么情况要警觉。退出条件：客户对异常的处置决策，连续若干次与 FDE 的事后判断一致。

第三级：撤出。 只留升级支持——约定的响应级别、约定的渠道，其余全归客户。

阶梯的两侧各有一个坑：右侧是"永远撤不了"——每一级都无限延长，影子运行变成常驻保姆，交付失败但没人承认；左侧是"突然消失"——验收完直接跳到第三级，客户第一次独立面对故障时孤立无援，弃用从那天开始倒计时。每一级停多久、什么时候进下一级，这个节奏感，就是新手和老兵的区别。

四、三条弯路

移交的失败有几个高频形态，每一个都对应一个正确做法。

弯路一：移交就是发文档。 文档打包发过去，签字，撤场——三个月后没人读过第二页。企业文档写完没人读是常态，除非它做到两条：以任务组织而不是以功能组织（"如何处理一笔异常退款"，而不是"退款模块功能说明"），且嵌在使用现场而不是躺在共享盘里（在用户卡住的地方就地出现）。[4] 配合移交期的小考核更有效：让客户方的接手人当着你的面走一遍故障处置——他讲，你听，讲不下去的地

[4] 范冰《前线部署工程师（FDE）》（XDash 开源书）

方就是文档的漏洞。

弯路二：评测集不给客户。 运行手册可以照抄给客户，评测集却常被留在自己手里——它是资产，也是下一单的议价筹码。这个选择要在合同期就谈定，而不是撤场时博弈：评测集连同维护责任一起移交，客户得到的是持续改进的能力，供应商得到的是"能干净移交"的声誉——这笔交换在第 20 章收费模式里会再算一次账。这里先把立场说清：抱着评测集不撤手的团队，本质上是把"持续收费的权利"看得比"交付完成的义务"重，客户迟早会感觉到。

弯路三：只移交操作，不移交判断。 客户会按按钮，但不会改参数：阈值要不要调、新错例怎么归因、要不要触发回退——每个小问题都得回头再买一次服务。这一条最隐蔽，因为它在合同期内看不出来，甚至对营收还很好看。破解它的机制有一个现成样本：某家人工智能公司与一家金融科技公司的合作，把"培训培训师"写成了核心设计——识别客户组织里的积极分子，把他们培养成内部讲师和内部答疑人，给官方认证、给专属支持通道、给在高管面前曝光的机会；合作的总目标一句话："转移知识，让客户能独立构建和扩展自己的智能体。"交付做到最后，是教会客户教自己。 [4]

五、移交之后：五面镜子

撤场不是终点。移交质量的高低，要在撤场后的一两年里照镜子才看得清。企业客户流失的原因，可以归纳为五类，每一类背后都对应着移交时欠的账。

- **价值蒸发**：系统还在跑，但没人记得它解决了什么问题——立项时的价值叙事断了档，新上任的管理者只看见一笔运维费。对冲它的动作在移交清单里：把价值口径和度量责任一起交给客户，让"这系统值多少钱"始终有人回答。
- **支持者离场**：你的内部盟友升职调岗，继任者没亲历过当初的选择，系统成了"前任的政绩"。对冲动作：关系网格化（一个支持者之外至少再发展两条关系线）、价值组织化（把系统写进流程文件和集体记忆）。移交期就是网格化的窗口。
- **质量漂移**：业务在变、数据在变、模型在更新，输出质量缓慢下滑，等管理层注意到时已经晚了。对冲动作：评测集移交的真正意义在这里——回归测试由客户自己跑，质量一漂移，第一时间发现的就是客户自己。
- **成本反噬**：用量越大账单越高，财务的审视越严。对冲动作：第 14 章的成本看板和预算熔断随系统一起移交，让客户自己看得懂、刹得住。
- **供应商戒断反应**：客户对"被绑定"的警惕——咨询行业甚至有预测说，多数企业终将因为成本过高与内部技能不足而放弃这类方案。［4］对冲动作恰恰是彻底的移交：客户怕的是"离不开你"，不是"你在这里"。

五面镜子照出的结论是一致的：移交不是交付的尾巴，是交付质量的验收单——你在场时欠的每一笔账，撤场后都要连本带息地还。

[4] 范冰《前线部署工程师（FDE）》（XDash 开源书）

六、撤不了的场怎么办

诚实地说，总有一些场撤不了：客户没有承接的意愿，或者没有承接的能力。这时候要做的不是硬撤（系统会死），也不是赖着（人天依赖），而是把这种说不清的状态写进合同——托管运行，作为一种明码标价的服务：客户明确知道自己买的是持续托管而不是已完成交付，供应商明确知道自己收的是运维的钱而不是交付的钱，双方对"这个系统归谁"没有错觉。 [2]

托管与假装移交的区别，商业上是一张合同，伦理上是一句实话。前者客户可以随时决策"自建团队接回去"，后者客户以为已经拥有、实际租用——等他发现真相那天，失去的不止一个系统，还有对你所有承诺的信任。

最后补一个商业视角，为第五部搭桥：移交能力本身就是产品化的前置。能干净移交的团队，才有资格谈规模化复制——因为移交清单上的每一条（模板、手册、评测集、连接器），同时都是下一个客户的启动资产；不能移交的团队，每个项目都在用人天堆人天，堆到团队规模的上限就停。 [1] 这个逻辑第 21 章会展开成"定制量递减"的完整论证。

撤场那天可以开一个很短的会，议程只有三行：三件事逐项过清单，每项有客户方签字人；影子运行期的周报模板定稿；下一次"只谈升级"的会，约在半年后。会开完，人才撤。

本章讲的是理想情况：客户想要这套系统，组织配合移交。但把目光转向中国市场，采购结构、信任结构、定义权都不一样——采用和移交长在另一套土壤里。

下一章，我们聊聊中国的另一套打法。

---

[1] @deployengineer（Bhaulik Patel）系列长文

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

第 18 章 · 中国企业的 FDE 落地，为什么是另一套打法?

> 地图不是疆域。
>
> —— 阿尔弗雷德·柯日布斯基

先来看两个同样做 FDE 的人，一天的工作时间都花在哪。

第一份来自硅谷。一家模型平台公司的 FDE 把约七成半的时间花在软件工程与模型优化上，咨询与客户关系分摊其余（时间账细目见第 1 章）。 [1]

第二份来自中国。一位在国内做企业人工智能落地的顾问同样算过：大约 60% 的时间花在帮客户梳理业务问题，或者干脆在"做人情"——维护关系、聊天、吃饭、挖需求；技术只占剩下的一小截。 [2]

两份放在一起，讲的不是谁好谁坏，而是想说明：同一个 FDE 岗位名称，在两个市场里干的是两份工作。

硅谷的 FDE 假设了一个已经就位的世界——需求是清晰的、数据是在线的、付费机制是成熟的，他的时间自然可以七成半投在工程上。中国的从业者面对的是另一个世界：需求要在饭桌上挖，数据要从笔记本里誊，付费要按人头算。把硅谷的打法原样搬过来，搬得越像，死得越快。

这一章我们讨论下：差异到底差在哪些结构上，打法要怎么变，以及中国市场长出了哪些自己的形态。

一、结构差异，打法变形

差异不在技术——模型是同一个模型。差异在结构，一共有四条（见图 18–1）。 [3]

[1] Baseten：我做前线部署工程师学到了什么（Het Trivedi）

[2] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[3] 《关于 FDE 的 100 个问题》（AI磊叔）

图 18–1：当决策、数据环境和组织采用的约束不同，FDE 的推进顺序也必须变形。

第一条，采购与预算。 美国市场能接受订阅制、按量计费甚至成果分成；中国企业的肌肉记忆是"买断加项目制验收"：年度预算、立项招标、分期验收付款。打法变形：把最小可行部署切进既有验收节点——第 8 章那套验收契约（业务指标、时间窗口、判定方式）恰好长得像项目制的验收单，这反而是顺水推舟；照搬订阅制，先死在采购流程里。

第二条，数据约束。 "数据不出门"在中国客户那里不是偏好选项，常是硬约束——尤其国资与涉敏行业。打法变形：私有化与本地部署的架构决策要提前到设计期，而不是补救期。一线的例证：有团队因预算所限做本地开源模型加检索的冷启动——成本约束下的本地化不是中国特色插曲，是常态。 [4]

第三条，付费习惯。 为人天付费，市场成熟了一百年；为结果付费，采购科目里没有这一栏。打法变形：按结果计费的教育成本要计入定价（第 20 章展开怎么算），或者用"阶段验收 + 价值指标写进条款"的混合制过渡。

第四条，人才供给。 既懂业务又懂人工智能的复合型人才，市场共识是"不是贵就是能力不行，大部分公司只能自己培养"。 [5] 打法变形：转岗加现场练——大厂普遍把产品、前端工程师转岗做前线部署，送去甲方现场收集需求、出原型。

四条再往下挖一层，还有一个共同的地基问题。阿里瓴羊首席执行官朋新宇，其团队服务大几百到上千家中大型客户，他给出的判断是：美国企业在信息化阶段有 Salesforce、SAP 这样的公司，"奠定了很多的工作流，以及奠定了很多的数据标准"；中国缺这一层，"每一家都没有一个比较平稳的地基"——所

[4] AI 职觉：别再卷 Agent 了（小宇宙播客）

[5] 有在做 FDE 这个工作的佬吗（linux.do 帖）

[6] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

以中国的交付经常要"从第一天打桩盖楼再到上面装修，一系列事情都要干完"。[6] 这个判断同时解释了本章的多条观察：数据约束严，是因为数据地基本来就要现打；人情占比高，是因为没有标准工作流可依，需求只能从关系里挖；付费认人天，是因为"中国每个大的企业都有一个 IT 团队"，纯功能集成的交付客户内部也能做，乙方必须做出超出内部团队的价值才收得到钱。

二、定义权：这一公里由谁说了算

四条差异之上，还悬着一个更根本的问题：**最后一公里由谁定义？**

同一个项目，有人觉得从演示到上线是一公里，有人觉得到一线员工愿意用起来才算到头——差出好几倍。谁来定义终点，直接决定 FDE 干的是技术活还是扯皮活。硅谷叙事里，这一公里由 FDE 自己定义：他坐在客户现场，看着人干活，亲手把断点接上，再把经验带回总部。而在国内，定义权常常根本不在 FDE 手里。

一个被从业者反馈的真实案例：一家国企，合同是大老板签的，落地的执行人提不出需求，就让 FDE 自己找、自己提、自己做——最后做了七个智能体，结果却没人用。这最后一公里被拖成了泥潭，因为从头到尾没人说清终点在哪里。链路上还常隔着代理商：国内不少项目由云厂商的代理商承接，需求隔了好几层传到现场——每一层都在放大和扭曲承诺。[3]

定义权之争的打法不是对抗。是把"定义"这件事从口头变成文件：用第 8 章的验收契约，把"做成什么样才算完成"写成双方共同签署的一页纸——终点由谁说了算的问题，变成终点白纸黑字的问题。签字之前，你是在给别人对不上号的期望打工。

三、两种土壤，两套推进逻辑

结构差异落到具体客户身上，中国市场的另一个特点是双轨：私企和国企，是两块完全不同的土壤。

私企的逻辑是"搞定事"。老板只关心一件事：花几百万，增收还是降本？人工智能不人工智能，他不在乎。打法：直通拍板的人，用结果说话，别绕。

国企的逻辑是"搞定人"。组织里的权责博弈比技术方案复杂十倍，怕犯错是底色。打法：先理清谁负责、谁背锅；把项目进展翻译成对接人能拿去向上汇报的语言——给他材料交差，而不是跳上去替老板做决定；同时把这一周能推进的小事列出来，能落一厘是一厘。其中有一条：**推动进程，绝不替人拍板**——权责不清是客户自己的病，你开不了刀；越界担责，项目黄了锅全是你的。

两套逻辑混用就会出问题。而有一个动作对两边都管用：最好能见到签合同的那个人——离源头越近，需求越真。[3]

[3] 《关于 FDE 的 100 个问题》（AI磊叔）

进场前花十分钟判断土壤，比后面花三个月补救划算。

四、中国现场的一线观察

前面那位自述"60% 时间做人情"的 FDE 从业者有三条一手观察。 [2]

观察一：先建数据，再谈契约。 "很多水下生意，数据是没有的，客户信息记在笔记本上。"第 10 章的数据契约假设"客户有数据可接"——中国现场常常更早一步：先把笔记本里的信息誊进系统，把老师傅脑子里的流程画成图，契约才有的谈。

观察二：最后一公里卡在最 Low 最土的地方。 有门店客户还在用装不上新软件的 XP 老电脑；配好的机器人某天"罢工"，查了半天，是被人随手关了机。部署方案的第一个技术决策，常常不是选模型，是适配哪一年的操作系统。

观察三：人情是基础设施。 花在人情上的那 60% 时间，要换个角度看：关系不是润滑剂，是承重墙——需求从关系里挖，信任从饭桌上长（这笔时间账要提前算进成本和报价，第 27 章接单定价会用到）。信任从哪来——大厂背书、过往案例，个人 IP，还是熟人转介绍——直接决定了你是哪种从业者。

陆骁鹏是极少数两边一线都做过 FDE 的人——在美国创业做北美医疗 AI 落地（Ventus AI），此前回国工作过两年，接触过国内版本的 FDE。他对国内客户心态的概括很扎心："两三个人捯饬捯饬也能做出来"——客户甚至直接反问："你给我两个人做项目，我为什么不能自己招两个人考核？" [7]

这句反问，和本章第三条结构差异（付费习惯）说的是同一件事：中国客户默认外部团队卖的是"人天"，人天买得起，就没道理外包；而"为结果、为持续判断力付费"这件事，本身就要先教育市场——这不是中国客户不理性，是两地对"外部团队到底在卖什么"的认知差还没被填平。

横跨中美两国看同一个问题，反而更容易看清：这不是某个市场的毛病，是"可教性低的复杂度"（很难教给客户自己上手做的复杂问题）这门生意在任何一个信任结构尚未成熟的市场里，都要先过的第一关。

五、先行者的答卷

中国市场不是没有答卷。把公开可查的先行者排开，能看到两条清晰的路子：服务商怎么把自己产品化，大厂怎么把这件事建制化。

服务商的产品化，做的是同一件事的两个层面：先立边界，再定阶段。一家不动产与设施管理行业的本

[2] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[7] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

土服务商，在官网上用三句话划清了它和驻场外包的界限：按阶段交付、按结果验收，而不是按工时结算；带着成型的产品来做工程，而不是从零现写现改；做完会走、能力沉淀在系统和客户团队里，而不是越驻越久。它还建了劝退机制——场景未验证的不接、数据一张表能导出的不接、只想了解人工智能的不接。这三句话加一套劝退，几乎是把本书第二部和第四部的要点做成了广告语。

边界立住之后，下一步是把方法论做成可复制的阶段：一家金融服务商沉淀出三阶段交付——场景诊断一到两周（识别价值最高的三到五个场景、量化投资回报）→ 嵌入式交付八到十六周（从模型选型到生产集成）→ 生产部署与持续演进（随业务迭代），同时明确"数据不出域"。[8] 注意它的结构：诊断期就是第 6 章的发现与筛子，交付期就是第 7 章的最小可行部署，演进期就是本章的交付之后——方法论是通的，只是长在了中国的采购节奏上。

大厂走的是另一条路：不靠边界和阶段说服客户，靠建制化本身。火山引擎立专职 FDE 团队；其总裁在公开采访中的表述与本书的定义几乎逐字呼应——"不是销售，也不是售前，必须具备很强的技术落地能力"，且团队成员刻意配置多元行业背景。[8] 电商公司得物的效率工程负责人则把万人规模的人工智能转型拆成三步：先全员共识、降低门槛让人用起来；再把场景按容错空间分象限，容错大的先交给智能体；最后设知识运营小组，把专家的隐性经验搬进智能体可见的空间——共识、场景、知识，顺序不能反。

建制化走得最深的样本，可能是阿里瓴羊首席执行官朋新宇所在的团队。作为阿里数据中台方法论的创立者，他的团队在"FDE"这个词流行之前就叫 CSM（客户成功），做了五年，先靠数据平台业务服务了大几百到上千家中大型客户——用他自己的话说："没有 FDE 这个词之前，我们干的就是这个事情……我们原来 5 年前就一直有这么一个团队就叫 CSM。"[6] 他们把这套做法收敛成三条原则：以业务结果为导向（问题必须有业务目标、业务数字和历史记录，能构建评测集与指标体系）、以企业数据为基座、以岗位标杆为预设——AI 客服的起点人设就是"一个工作五到十年的客服坐席"。[6]

落到具体案例：一家 3C 数码客户的"催发货"售后环节，被拆出来是 260 多步——少发整机、漏发配件、赠品未达三类催法，内部外部系统混杂、横跨京东天猫多个平台，覆盖 95% 以上的场景；而现状的人工客服回应"以小时或者天为单位"，客户等不起就退单。[6] 选场景的标准也给了可直接抄的三问：哪里耗人最多（打电话、接电话、处理投诉）、耗钱最多（赔款止损、营销投放）、耗时最多（催货查询）——这三问与第 6 章的价值密度筛子几乎说的是同一件事。组织上，他们的答案是"我们的做法不是一个人，而是一个组织"：以业务结果为导向的 BA（美妆、家电、汽车这些行业专家，负责编排流程、设定评测标准）、AI 架构师（把业务问题翻译成用什么模型、哪个环节人干预哪个环节机器接管）、再从客户方选出最好的客服和销售组成调教团队——与第 24 章能力模型、第 26 章团队建设的讨论直接对话。

这些答卷背后压着一个共同的财务事实：2025 年年报，用友人均营收约 48 万元、金蝶约 62 万元；同一年，帕兰提尔的人均营收约 100 万美元——十倍级的人效差。[8] 这就是"项目制诅咒"在财务报

[6] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

[8] 范冰《前线部署工程师（FDE）》（XDash 开源书）

表上的长相：大公司要定制，厂商做一单赔一单，交付完代码离场作废，最顶尖的工程师消耗在高度定制的非标交付里。也是同一组数字，给出了中国市场对这门生意最大的想象空间：谁能把交付工程师的产出从"人天"定价改写成"结果"定价，谁就同时改写了数十万人的市场价格和这门生意的财务结构。

需求方的声音也值得一听。一份面向企业首席信息官的定向调研给出了三个并排的数字：73.3% 的受访首席信息官认可前线部署模式、愿意试点；同时 60.0% 担心供应商对自己行业理解不足，53.3% 认为市场缺少可量化的投入产出证明。 [8] 翻译成一句话：认可是真的，顾虑也是真的——客户不是在犹豫要不要买一个新岗位，是在评估一整套还没被验证过的价值实现机制。这份顾虑清单（行业理解、可量化的投入产出），恰好就是本书第 6 章和第 8 章提到的。

六、两个真实形态

结构约束之下，中国的前线部署长出了两个自己的形态——它们与硅谷原版的责任结构都不同，但都在各自的结构里真实运转。

形态一：大厂驻场原型工程师。 社区里的一线从业者描述：公司把产品和前端工程师"二合一"派去甲方现场收集需求，用人工智能现场出原型——纯演示数据的前端页面，根据甲方要求现场改；需求确定后，交回公司内的后端开发做实现和联调。 [5] 按第 2 章的三判据检验，它覆盖的是光谱上"发现 + 原型"那一段，生产和回流在别处。在中国的采购结构（代理商隔层、需求模糊、验收导向）下，这是理性的分工——原型是撬动甲方确认需求的最快杠杆，而且它恰好绕开了定义权的扯皮：原型摆出来，改三轮，需求就自然定义完了。

形态二：个人 FDE 与独立服务商。 社区里另一位从业者的自述：做智能体开发定制三年，多数时间线上接单，自建了案例库和可复用模块——但"自己干没背书，打工又苦"。 [9] 这句话戳中了独立形态的两个真实约束：个人背书的信用成本（没有大厂光环替你开第一扇门），和打工形态的激励损耗。它的成熟形态，是那些以顾问与销售能力为主轴、聚焦中型企业、按行业纵深复制案例库的"企业人工智能落地服务"生意——工程能力排在第三位，前面是顾问与销售：不是工程不重要，是在中国市场，挖得出问题、谈得下单子比搭得出系统更稀缺。 [2] 这个形态的完整讨论在第 27 章展开（个人能不能独立做 FDE）。

两个形态放回第 1 章的责任上，结论是：中国样本普遍窄于硅谷原版——但"窄"不等于"错"。形态是约束的函数：采购结构决定你只能收到原型需求，你就把原型做到极致；信任结构要求你先做人情，你就把人情做成方法论。评判形态优劣之前，先看清它回应的是什么约束。

[2] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[5] 有在做 FDE 这个工作的佬吗（linux.do 帖）

[8] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[9] 在武汉做 FDE，寻伙伴（linux.do 帖）

七、约束更紧的极限测试

在此前提下，本章的结论是三句话。

第一句：中国市场不是 FDE 的低配版，是另一套约束逼出的另一套活法。采购结构决定了你多半只能先做原型；信任结构决定了没有大厂背书的人只能靠人情和案例一点点攒。驻场原型工程师、个人FDE，都不是"退而求其次"的将就——是在各自面对的约束下，能站得住的打法。

第二句：中国的约束不是"更多一条"，是"更紧"——本章一路讲下来的采购规则、数据现状、付费习惯、定义权归属、人情投入，加在一起就是一句话：同样一件事，在中国要多花几道工序才能立住。第二部和第四部的那些方法，在约束松的市场是加分项；在这种紧约束下，是不做可能就会死的必答题。

第三句，也是给所有读者的：把中国样本当作极限测试。一套交付方法如果在中国现场的约束下依然成立——依然找得出值得解决的问题、依然收得拢验收口径、依然撤得了场——那它在约束更松的市场里，大概率更能成立。

组织、采用、移交都谈完了。但这一切之上还悬着更根本的一问：这么重的交付背景下，这门生意在商业模式上到底成不成立？第五部，我们来算这笔账。

第五部 · 这门生意怎么才能成立

前三部讲怎么把系统做成、做稳、用起来，这一部回到最冷酷的一问：这一切凭什么是一门生意，而不是一支高级外包队。

四章算的是同一笔账的四个侧面：第 19 章算总账——招 FDE 是公司层面的损益决策，不是招聘决策，算不清大账就别动；第 20 章算单笔账——人天、固定范围、按结果三种收费模式的背后是风险分配，中文现场的三条定价公式都锚在客户自己能核算的账上；第 21 章算曲线账——第 N 个客户的定制量必须下降，否则毛利、速度、质量随客户数线性恶化，靠英雄主义硬撑是最贵的；第 22 章算机制账——现场经验回流成产品能力是被设计出来的接口，FDE 负责把现场经验送回产品，也是产品的第二客户。

写给正在决策"要不要建前线部署团队"的管理者，和已经在账上挣扎的创始人：这一部的每一章都可能算出"不该做"——那是这些账最值钱的输出。

第 19 章 · 为什么"招几个 FDE"，要算大账，不是小账？

> 你该投资那种好到傻瓜也能经营的企业，因为迟早会有傻瓜来经营它。
>
> —— 沃伦·巴菲特

"这个客户很重要，我们先招两个 FDE。"

这句话在项目会上听起来几乎没有问题。客户需要深度部署，销售不愿失单，现有工程团队也抽不开身；两个能驻场、能救火的人，看上去是最快的解法。

但招聘决定一旦做下，问题就不再只是多付两个人的薪水。下一个客户来了，还要不要再招两人？这两个人为客户写下的代码、跑通的流程和积累的经验，能不能带到下一个项目？

"招几个 FDE"表面是人事问题，实际上是在决定公司将来靠什么增长。这一章要算的，正是这笔账。

一、小账与大账，先分清再动笔

小账是招聘视角：这个人什么背景、多能干、要多少薪水、能不能吃苦驻场。大账是投入产出视角：这个团队的投入产出结构是什么？它随客户数怎么变化？它会把这个组织变成什么？

两组问题摆在一起，差别一目了然。小账问"这个人一年成本多少"，大账问"每增加一个客户，成本增加多少、毛利变化多少"；小账问"他能不能搞定客户"，大账问"他搞定的东西，下一个客户能不能用上"；小账问"团队要几个人"，大账问"团队规模是不是必须随客户数线性增长"。

本书的立场很明确：先答大账，再答小账。大账不成立，小账答得再漂亮也是给一个错误的组织招人。这也是第 26 章（什么时候建团队、怎么建）的前置——组织设计的一切细节，都从这一章的账本里长出来。

二、六个指标，把大账落到数字上

大账不能只是姿态，要能算。六个指标，每个说清两件事：怎么算，什么样算健康。 [1]

其一，实施杠杆率：一名 FDE 能把多少客户收入做上线？ 分子是新进入生产环境的客户收入，分母是

[1] @deployengineer（Bhaulik Patel）系列长文

投入的工程师人数。它回答的是这门生意最基本的问题：一个人扛得动多少？

其二，价值实现周期：客户从签约到第一次可衡量的业务见效，多久？ 参考口径：最小可行部署的验证以周计（2–6 周），完整部署以月计（1–4 个月）；这个周期持续变长，是方法论或平台出问题的第一信号。 [2]

其三，服务成本：部署期和上线后各要多少工程投入？ 这项要算全——差旅、支持、返工这些隐性项最容易漏。

其四，扩展效应：有 FDE 参与的客户，是不是比没有的客户长得更快？ 老客户的新增收入里，有多少不需要新的交付？它检验的是"深度服务换来信任扩张"这条逻辑。

其五，产品杠杆率：为一个客户做的工作，有多少变成了后续客户可复用的能力？ 一项平台组件如果能替每个未来部署省下两个月，它创造的是公司级杠杆。

其六，也是看得最紧的一个：定制递减率。 第 N 个客户的定制量，应该显著小于第一个客户——连续三个客户不降，立即检视产品回流机制。这一条是第 21 章重点讨论的，大家先有个印象。

六个指标不必凑齐——盯住与当前阶段最相关的三到五个，胜过铺满一面墙的仪表盘。 [2] 但一个都不看的组织，就是闭着眼在算小账（见图 19–1）。

图 19–1：FDE 的大账要看六个彼此关联的指标，而不是只看新增收入。

三、英雄主义的账

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

六指标里，最先让"英雄主义"现出原形的是实施杠杆率。

什么是英雄式 FDE：个人能力极强，什么都自己扛——最难的单子给他，最难的客户给他，救火也给他。十个客户的时候，英雄是公司的资产：他的名字就是口碑，他的存在让三个单子起死回生。但一百个客户的时候，英雄是公司的灾难——因为组织能力沉淀在他个人身上，而不是在流程和产品里：他带不动一百个客户，他教不会一百个工程师（知识在他脑子里），他走了公司就塌一角。

比英雄个体更深的坑是英雄文化。一个没有边界的前线部署团队，会成为组织里"所有不好归类的东西"的默认去处：销售把承诺过的功能塞过来，产品把填不上的缺口塞过来，工程把不想做的集成塞过来，客户成功把升级投诉塞过来——很快，什么活都往这里塞，最后全靠几个英雄硬撑：少数几个杰出的工程师靠蛮力让客户活着，业务依赖的是人而不是系统。十个客户的时候运转良好，一百个客户的时候就转不动了。

解法在第 22 章，把个人能力回流成产品能力，但在这里先记下：**英雄是线性增长的资产，而这门生意要么复利，要么退化，没有线性档位。** 一位从业者把这层意思说到最透：最好的前线部署组织，致力于消除自身的工作——建模板、造工具、把客户特定的代码变成可配置的平台能力，把角色不断移向更有价值的问题（见图 19–2）。 [1]

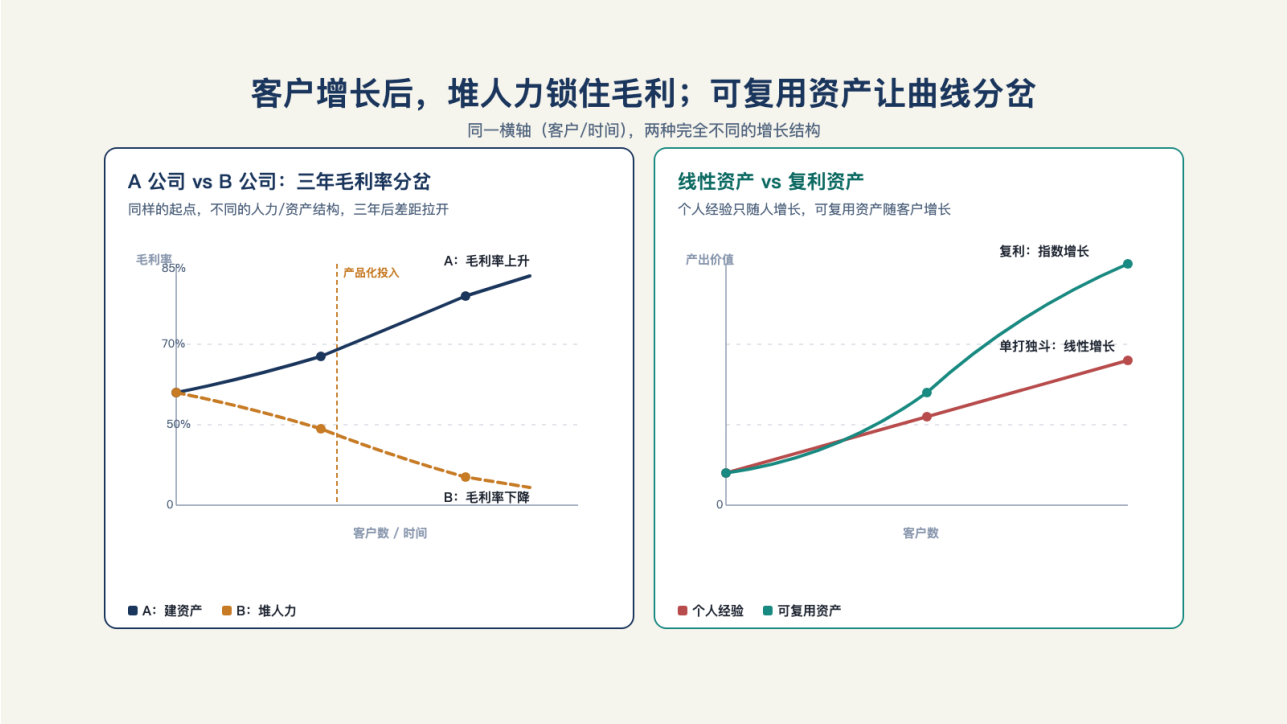


图 19–2：客户增长后，英雄式人力会锁住毛利；可复用资产才让曲线分岔。

四、麦格鲁的两笔口径

[1] @deployengineer（Bhaulik Patel）系列长文

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

聊大账绕不开麦格鲁，他是这套模式最成功的亲历者，也是账算得最细的讲述者。他在播客里给过两笔口径。 [3]

第一笔，关于 KPI。他区分了两种策略：产品市场契合策略里，你希望为每个客户做更少的工作、降低成本、保持合同金额不变；而 FDE 策略正相反——你要驱动合同金额增长，在同一位客户上做越来越有价值的事，因为做的事更有价值，所以为单个客户保持一定的定制量是可以接受的。因此内部的衡量标准是合同金额，而不一定是每个客户的工作量。

乍听之下，这与"定制递减"矛盾——不是说好定制要降吗？别急，这个张力是真实存在的，第 21 章会正面拆解：递减的正确度量是定制占交付的比例，而不是绝对工时。合同变大、单位定制量不变时，占比仍在降、毛利仍在改善——两种读法量的是同一个健康度。

第二笔，关于利润率的路径。他说：为客户做一次新部署，早期实际上可能是在亏钱的；随着你在客户那里时间越长，产品因发现过程而更贴合需求——你不再需要一大群人驻场摸索——并且你赢得了处理更重要问题的权利；于是你交付成果的单位价值成本下降，利润率从负转正——可能要一年，也可能是好几年。 [3]

这笔账解释了一件事：为什么成熟的 FDE 组织敢在前几个客户上"亏本"——那不是不会算账，是把早期亏损当作产品发现的投资，赌的是合同随时间变大。它同时给六指标补了一条时间轴：前两个客户的所有指标都难看是正常的，要看的是方向，不是起点。

五、一张三年损益的对照表

给一张可以换上自己数字重算的演示表。

两家同规模的供应商，起点几乎一样：各三个客户、每客户合同额相当、团队各六人。A 公司从第一天起把每个客户的定制请求打上标签，第三个客户起复用连接器，第五个客户起一半流程走配置；B 公司每单都从零干起，因为"客户着急、复用太慢"。

第一年，B 公司的收入更高——不复用就意味更多的人天，更多的人天可以直接卖出去。第二年，分岔出现：A 公司的第六到第十个客户交付周期缩短四成，边际成本压向产品成本，交付毛利率从三成爬向六成；B 公司的毛利率被人力成本锁死在三成以下，且团队规模必须随客户数线性扩张。第三年，A 公司的人均产能是 B 公司的两倍以上，老客户续约时新增的服务几乎零交付成本；B 公司最资深的三个工程师职业倦怠离职——他们手上的四个客户，各需要两名新工程师重新摸索半年。

上面的具体数字不是真的，但分岔的机制是：毛利率从纯人力期的两三成向平台复用后的六成以上爬升，"不上升的 FDE，是咨询公司"。 [2]

[2] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

六、自建、买，还是混合

账算完，落到决策。管理者面对"要不要有自己的 FDE"，实际是三选一。 [1]

自建：适用条件是差异化就在交付里（你的市场里深度部署就是产品的一部分）、且管理层接受前一两年指标难看。失败形态：没有边界的团队沦为英雄文化的温床（本章第三节），或在平台能力还没成形时硬上。

买：向外部服务商按项目采购前线部署能力。适用条件是需求间歇、场景非核心。失败形态：知识从不沉淀在自己组织里，第二个项目从零买起（第 17 章移交三件事全部缺席）。

混合：核心场景自建，峰值外包。适用条件最宽，但失败形态也最隐蔽：自建团队慢慢被外包"惯坏"，能力空心化。判据是自建团队是否始终保有最难的那几个客户。

三种选择没有优劣，只有与大账的匹配。唯一错误的选项是第四种：没算过账就招人（见图 19–3）。

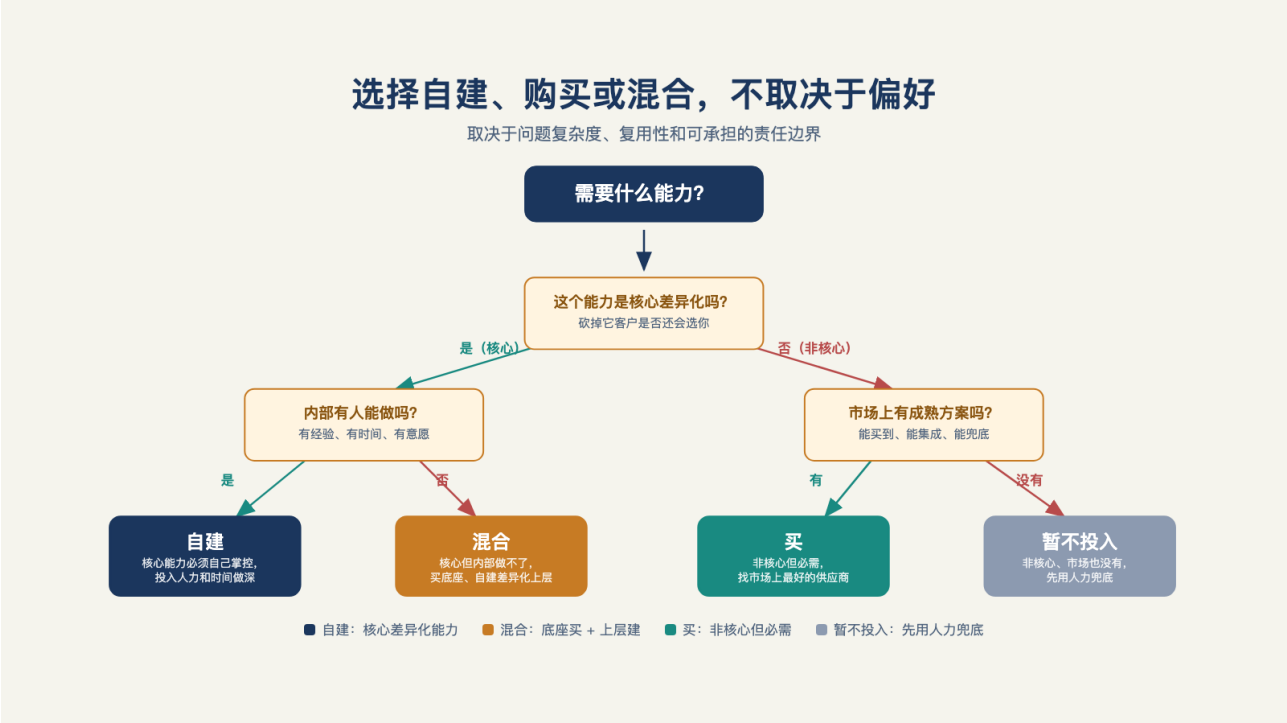


图 19–3：选择自建、购买或混合，不取决于偏好，而取决于问题、复用性和可承担的责任。

七、不要轻易尝试

最后是麦格鲁的劝退：他的第一、第二、第三条建议是同一句话——不要轻易尝试。 [3] 原话的分量

[1] @deployengineer（Bhaulik Patel）系列长文

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

在于说话的人：把这套模式从 Palantir 带到 OpenAI 的最成功实践者，给出的却是全场最响亮的劝退。

把这句话放回本章的框架，它其实是"大账思维"的最后一条检验：算大账，最终可能算出的答案是"不该做"。这不是失败，是这笔账最值钱的输出——每一家想做 FDE 的公司都该先问自己能不能承受这个答案，问不出口的，多半已经在坑里了。

大账算清了结构，还有一笔更细的账等着要算：每一个合同，钱怎么收？下一章，我们谈一下如何收费。

第 20 章 · 怎么收费：按人天、按范围，还是按结果?

> 战略的本质，是选择不做什么。
>
> —— 迈克尔·波特

"不达标不收费"，是最容易让客户心动的一句话。

它把一件原本说不清的事说得很简单：项目做出效果再付钱，做不出效果就不付。对刚经历过烂尾项目、又不想再为一堆人天账单买单的客户来说，这几乎很难拒绝。

可是，当双方开始把这句话写进合同，简单的承诺会立刻长出细节。补货准确率提高多少才算有效？滞销库存下降，到底是系统的功劳，还是季节、促销和采购策略的共同结果？三个月后才看得见的收益，能不能覆盖供应商前三个月已经投入的人和成本?

一张写着"不达标不收费"的报价单，真正开始谈的并不是价格，而是这些问题。不同的收费方式，差别只在于把这些不确定性放在哪里：留给客户、留给供应商，还是由双方分担。

这一章拆解收费这个黑箱：三种计费模式各在什么条件下成立、按结果收费的前提、一个从业者的定价方法，以及合同里必须写清的几件事。

一、不同的模式，不同的风险

按人天。 最古老、最成熟、最好卖——客户懂，采购流程认。风险全压在客户一边：买到的是时间，不是结果。它的失败形态是第 2 章判据三的失守——卖时间的人不对结果负责，做着做着就滑向高级外包。按天计费，增长的上限永远封在人天上，分不到结果增值的那部分。 [1] 一家企业客服 AI 平台商（Cresta）走得更彻底：他们"从来不是以 FDE 的小时来计费的"——FDE 的人力投入折进平台订阅与成功费的总账，不单独逐项列账；对意向强但还没签单的客户，甚至会主动倒贴人力先免费做出效果，"就像送货上门，你已经试穿了，你说你再退掉也不愿意"，把 FDE 成本当作获客投资而非计价单位。 [2] 这条路径不是本章后面要展开的按结果分成（客户仍然按订阅付费，只是内部不把 FDE 工时暴露给客户），而是提醒一句：把 FDE 人力从计价单位里拿掉，本身也是一种风险分配方式——供应商把交付成本前置的风险自己扛了，换来的是更短的成交周期。

[1] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

[2] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

固定范围交付。 验收制环境里最常见的形态——尤其与中国项目制的采购节奏合拍。风险双方各担一半：乙方担范围内的交付，客户担范围定义不清的后果。它的失败形态有两种：范围蔓延（"顺手再改一下"累积成第二个项目）和墙没摸清——第 9 章说过，报价期数不清八堵墙的合同，就是烂尾的合同级起因。

按结果分成。 结果导向对齐得最彻底：客户为零效果付的钱最少，乙方分享增值。但它的失败形态也最锋利：基线争议（省下的钱到底有多少算系统的功劳？）和回款滞后（交付成本前置、分成收入后至，现金流先被压垮）。所以它从来不是"更先进"，只是**更贵的前提换更深的对齐**（见图 20–1）。

图 20–1：收费模式的差别，本质是范围和结果风险如何在双方之间分配。

二、按结果收费的四个前提

"不达标不收费"听起来，像供应商把自己和客户绑在了一起：项目没有产生效果，客户就不必为一堆工时、会议和中间产物买单。比起按人天计费，这当然更像一次对结果的承诺。

但真正开始谈合同，问题很快会变得具体：什么叫"达标"？销售额上涨，是系统带来的，还是旺季、促销和销售团队共同带来的？效果三个月后才看得出来，前面几个月的交付成本谁来垫？

有人把这种安排称为"结果即服务"（Result as a Service）：供应商不只交付一套系统，也把一部分收费条件绑在业务结果上。客户买到的，不再只是开发工作，而是一项被写进结算规则的结果承诺。

这也解释了它为什么诱人，又为什么难做。一句"不达标不收费"背后，至少要先把四件事谈清：效果从哪里开始算、效果该归给谁、等多久才能结算、最差情况下谁来兜底。少了其中任何一项，结算时都可

能变成另一场项目。

下面这四个前提，决定了按结果收费究竟是一种可执行的合作方式，还是一场看似漂亮的对赌。

基线可测。 效果要有事前双方签字的基线——这正是第 8 章验收契约的商业意义：没有基线的分成，结算是扯皮。

归因可信。 效果提升要能归到系统头上。销售额涨了三成，是智能体的功劳还是季节？归因逻辑要在签约前写死（对照期、对照门店，或至少一个双方认可的折算规则）。

周期可承受。 交付成本在前、分成收入在后，中间的时间差要算清现金流能否活过——按结果收费饿死的公司，比做砸的多。

下行有底。 保底费用要覆盖交付成本。纯对赌不是定价，是赌命；一家现金流健康的公司没有理由用下一季的工资单去对赌客户的业务指标。

四前提背后有个共同的名字：可测量性。第 12 章的评测体系在这里完成它的商业闭环——**评测不只是质量工具，是定价的地基。** 评测口径说不清的生意，只能按人天卖。

三、三个报价锚点

一位服务过多家企业的顾问，将自己见过的报价方式归成三类。它们不是通用范式，只是提供一个思路：

公式一，降本。 他举例：一家有五十名研发人员的公司，人均年成本二十万元；若人数从五十人降到三十人，节省四百万元，收费可以锚定在这笔差额的一部分。客户不必先相信供应商的报价，只要先算清自己的成本账。

公式二，固定成本置换。 一家电商的商品详情页，每年要花两三百万元在摄影、模特、修图和设计上；为其搭建人工智能生图系统，收费一百万元。被替代的存量开支，就是价格的锚；为实现这种替代所需的额外工程成本，也要计入报价。

公式三，增收。 销售人员月业绩从十万元做到三十万元，增量二十万元里按约定比例分成。价格只落在新增部分，客户也更容易把它和原有业绩区分开来。 [3]

三条公式外形不同，内核是同一句话：**价格锚在客户自己能核算的账上。** 降本锚在人力差额、置换锚在存量开支、增收锚在增量流水——没有一个锚需要客户"相信"乙方的报价，每个锚客户自己都能按计算器。这与第 2 章判据三（对什么收钱）和第 8 章基线测量完全互证：报价的说服力，来自客户能在自己的账本上复算它。

[3] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[4] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

大厂供应商的价目表，与这三条独立顾问的公式对得上。阿里瓴羊的销售口径是双轨：一类按坐席——客户原有五百人的客服团队，今天用相当于原成本八到九折的投入做到同等效果，锚的是被替代的人力开支（公式一的变体）；一类按效果分成——营销、投放类场景，转化率超出原有基线的部分才分成，锚的是增量（公式三的变体）。[4] 更值得注意的是他们对"不达标不收费"的叫法："不叫对赌，还是叫做测评"——先在客户的真实业务单元上做最小价值验证，客户验收通过才继续服务，并且明言，算不过账的客户不会连续续费。[4] "对赌"能不能改口叫"测评"，分界线恰好就是本章第二节的四个前提：基线可测、归因可信，才有资格叫测评；测不了，就只能叫赌。

四、混合结构：一种更稳妥的安排

对交付成本高、效果滞后的项目，一种更稳妥的结构是：**固定费 + 效果奖金 + 运行费。** 固定费覆盖可预见的交付成本，效果奖金分享增值，运行费对应移交后的托管运行。运行期仍在持续消耗工程与支持资源；不单列收费，这部分成本就会侵蚀交付毛利。

一个条款参考骨架是：固定费按里程碑分三期，各挂验收口径（第 8 章）；效果奖金设两条档线，基线以下为零、达标以上按约定比例封顶；运行费按年计，写明响应级别、升级路径与责任边界（第 17 章）；任何一方提前终止的结算规则。

五、合同里必须写清的五件事

模式之外，是条款。五件事写不清，什么模式都会翻车。

其一，评测口径与判定人。 用哪个评测集、什么算达标、谁说了算——第 12 章的评测体系在这里变成合同附件。

其二，甲方配合义务。 数据与权限的配合是交付的前置（第 10 章）——甲方拖延导致延期，责任怎么算？没有这条，所有延期都是乙方的。

其三，验收节点与灰度规则。 分几期验、灰度比例怎么爬（第 8 章），写进合同而不是留在邮件里。

其四，移交后的责任边界。 验收之后、托管期内的故障分级与响应，接第 17 章的责任交接表。

其五，知识产权归属。 定制代码、评测集、可复用模块各归谁——这条与第 22 章的产品回流直接冲突（回流的原料就是现场资产），乙方必争，但争的方式可以设计：定制代码留客户、可复用组件留乙方、评测集随移交（第 17 章的立场）。

为了防止被"白嫖"，还有一条来自一线从业者的做法：**方案要分拆。** 售前方案只交代问题判断、技术路径和报价；真正可执行的分阶段步骤、验收指标和落地细节，签约后再共同细化。原因也很直白：客户

[3] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

拿着一份已经能执行的方案去找别人做，或者交给内部团队照着做，乙方前面的售前工作就打了水漂。[3]

六、买方视角：怎么读一张报价单

最后站到桌子的另一边——甲方。拿到报价单后，先问三件事。

第一，这笔钱买的是什么？ 是工时、明确范围，还是已经定义的结果？先看收费结构，再看总价，才能知道项目失控时风险会落在谁身上。

第二，低价漏掉了什么？ 集成、数据治理和运行支持是否已经算进报价？这些成本没有消失，只是可能在交付中以变更单的形式回来。

第三，指标由谁定义？ 报价里写"预计提升 40%"，分母是什么、观察窗口多长、谁来判定？定义不清的承诺，到了结算时很难不引发争议。

单笔合同算得再好，也只是小账的胜利。第 19 章的大账里最难的一条曲线——第 N 个客户的定制量——不下降，任何收费模式都救不了毛利率。下一章讨论如何实现定制递减。

第 21 章 · 第 N 个客户的定制量，为什么必须下降？

> 今天的问题，来自昨天的解决方案。
>
> —— 彼得·圣吉《第五项修炼》

"这个客户和前九个客户一样，还要再派一组工程师吗？"

第十个客户刚签下，项目经理开始排期：接系统、配权限、改流程、做评测、上线后支持。每一项在前九个项目里都做过，排期却仍然像面对第一个客户。

真正危险的不是交付团队不够努力，而是公司没有把前九次交付变成第十次可以直接使用的能力。第 N 个客户的定制量为什么必须下降？这就是本章要回答的问题。

一、批判的不是定制，是不可递减

前线部署模式靠定制起家——这一点全书没有含糊过。第一个客户百分之百定制是必然：你连这个行业的流程长什么样都是第一次见，除了定制没有别的动作可做。把"定制"当贬义词的批评打偏了靶子。

真正致命的是另一个东西：**定制量不随客户数下降。** 第一个客户全定制，是模式的开端；第十个客户定制还占八成，是模式的失败——因为毛利、交付速度、质量稳定性会全部随客户数线性恶化，下面两节会去拆解。

所以本章的问题不是"怎么少做定制"，而是"怎么让第 N 个客户的定制量显著小于第一个客户"。指标已经有了：定制递减率，第 19 章提过、参考书写明判读法的那个——连续三个客户不降，立即检视产品回流机制。 [1]

需求方也在同一方向上移动。一位大厂交付负责人说，客户买人天的投入这种订单会越来越少——人天定制对双方都是"从养鸡场到养鸡到下蛋，所有都干完"的低效模式；连以服务收入著称的 Databricks，他也辨析为"帮客户找出有价值的问题，而不是帮客户实现他列出来的 to-do list 的 100 个功能需求"。 [2] "按人天买功能"退潮，逼着交付方回答本章的问题：不靠人头堆定制，靠什么。

二、两条成本曲线

[1] 范冰《前线部署工程师（FDE）》（XDash 开源书）

[2] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

先用数学把"慢性死亡"说清楚。

不递减的世界里，成本结构是一道乘法：**总定制成本 = 客户数 × 单位定制成本。** 假设每个客户五十人天，十个客户就是五百人天——团队规模必须随客户数线性扩张——这正是第 19 章那张三年对照表里 B 公司的路径：毛利率被人力成本锁死，B 公司最资深的工程师在第三年职业倦怠离职。毛利率陷阱（第 19 章）和这条成本曲线，是同一张账的两个侧面。

递减的世界里，第 N 个客户的交付成本逐步靠近产品的复制成本。

到第十个客户，两种世界的单客户成本差十倍——而递减世界里省下的不只是钱：交付周期会缩短，质量稳定性也会同步上升。递减不是省钱项，是这门生意从"人力生意"变成"产品生意"的门槛。

三、当定制能力只长在人身上

问题不在于某位 FDE 能力太强，而在于他的判断、取舍和踩坑记录只留在自己脑中。第一个客户做过的工单集成，第四个客户来了，他仍凭经验再做一遍；这叫更熟练的定制，不叫可复制的交付。

个人熟练度可以提高，客户数也会增长。若每次交付都没有留下组织可用的资产，客户最终会多到个人能力接不过来。

解法与第 17 章对外移交同构：**对外交付系统，对内沉淀知识。** 每次定制结束，都应留下可复用资产清单：哪些集成能力可以组件化，哪些流程可以配置，哪些评测结构可以迁移，哪些部分只属于当前客户。

四、定制不降，合同也能增长吗?

麦格鲁在播客里说：FDE 策略要驱动合同金额增长，在同一位客户上做越来越有价值的事，因此**为单个客户保持一定的定制量是可以接受的**，内部衡量标准是合同金额而非每客户工作量。 [3] ——"定制量大致不变"？这与本章标题直接冲突了。

关键在度量口径：**递减衡量的是定制占交付或合同的比例，而不是绝对工时。** 单个客户的定制量可以暂时不变；只要产品杠杆推动合同金额和交付价值增长，定制占比仍在下降，毛利仍有改善空间。

降分子（少定制）与增分母（做更大、更有价值的合同）是同一个健康度的两种读法。多数团队先要做到前者；后者成立的前提，是定制内容确实在持续回流成产品能力。

五、递减的三类可复用能力

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

口径立好了，问题是拿什么降。三个引擎，每个都有现场证据支撑。

先立一个总纲：客户数据属于客户，不能被带到下一个项目。真正能随部署累积的，是对场景的判断、评测与验证的方法，以及把一次跑通变成稳定交付的工程流程。 [4]

能力一，集成能力组件化。 可复用的不是客户数据，也不只是接口代码，而是一套可配置的集成能力：认证方式、字段映射、同步规则、异常处理和权限边界。客户差异留在配置层，稳定部分进入平台组件。

能力二，流程模板化。 同一行业的业务流程，抽象成可配置模板。这里有一个容易搞混的界限：**配置不是定制——配置改参数，定制改代码。** 模板化的目标，是把"改代码"的需求降级成"改参数"的请求：审批层级、字段命名和阈值区间进配置表；流程拓扑本身变化，才动代码。

能力三，评测结构迁移。 跨客户的评测模式可以沉淀：任务类型、难度分层和判定规则可复用，案例数据不可复用。带走的是"怎么考"，不是客户的"考卷"。

三类能力共用一个特征：**每次复用都在让下一次交付更轻。** 平台组件补齐异常处理，模板积累参数经验，评测结构扩展失效样本。复用不是把旧方案原样搬走，而是让可复用的部分越攒越多，下一次交付就轻一分（见图 21–1）。

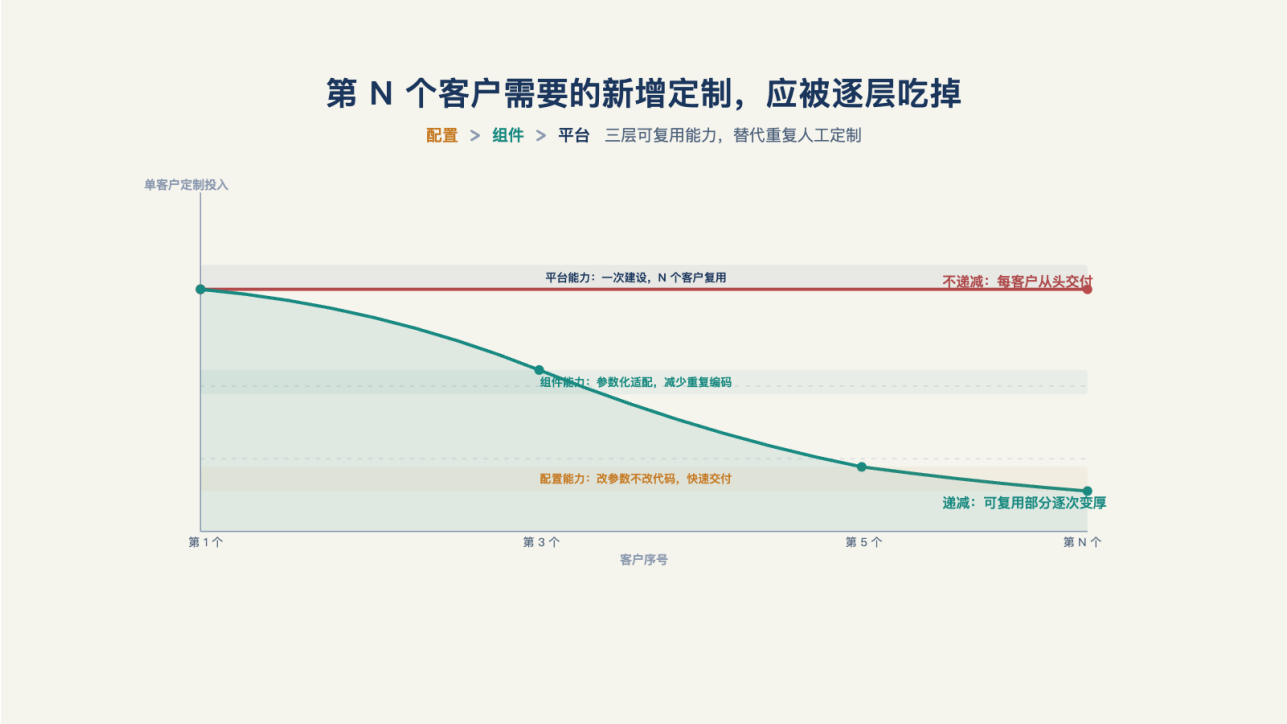


图 21–1：第 N 个客户需要的新增定制，应被组件、配置和平台能力逐层吃掉。

六、配置与定制的分界线

[4] Inside an Applied AI company（Pace 长文）

引擎转起来之前，每个需求进门要先过一道闸。三问依次问下来：

第一问：能否配置解决？ 参数、模板、规则表能搞定的，禁止进代码。

第二问：能否作为下一个客户的默认能力？ 这个需求是这家客户独有，还是别的客户也会要同样的能力——后者即使今天要写代码，也按"未来默认能力"的标准写：进主干、带抽象、留参数。

第三问：是否只能定制？ 本客户特殊、且无复用预期——可以定制，但定制代码必须**打标签**：一次性代码明确标记，与产品代码物理隔离。标签的用处不是仪式感，是防止定制代码伪装成产品能力——下一段要讲的死代码，一半就是这么混进去的。

这三问背后还有一个更根本的方向判断，来自一位大厂交付负责人的三段论：定制化、标准化、个性化。过去的软件交付是"用定制化的方式去解决标准化的问题"——把企业已经跑顺的流程刻画进系统，刻画得到位能解决三成，刻画不到位"天天有七成的新增量"，项目变成"又长又久的黑天鹅工程"。 [2] 智能体时代的机会是反过来的："用标准化的方式去解决个性化的问题"——把工作流、数据流、企业上下文这些环节标准化之后，同一个产品就能在不同客户那里自主长出各自的特性；而"既然定制化，就很难做到个性化"——定制锁死的恰恰是变化的能力。 [2] 落到操作上，就是三问闸门要推的方向：让"改代码"持续降级成"改参数"，再让"改参数"沉淀为产品出厂就带的预设能力。

七、产品化的时机：何时抽象，何时保留定制

引擎和闸门都就位后，剩下的是时间问题：什么时候把定制升级成产品。第一个客户通常要走完全程，产品团队再从中抽象出能服务后续客户的通用版本。 [3]

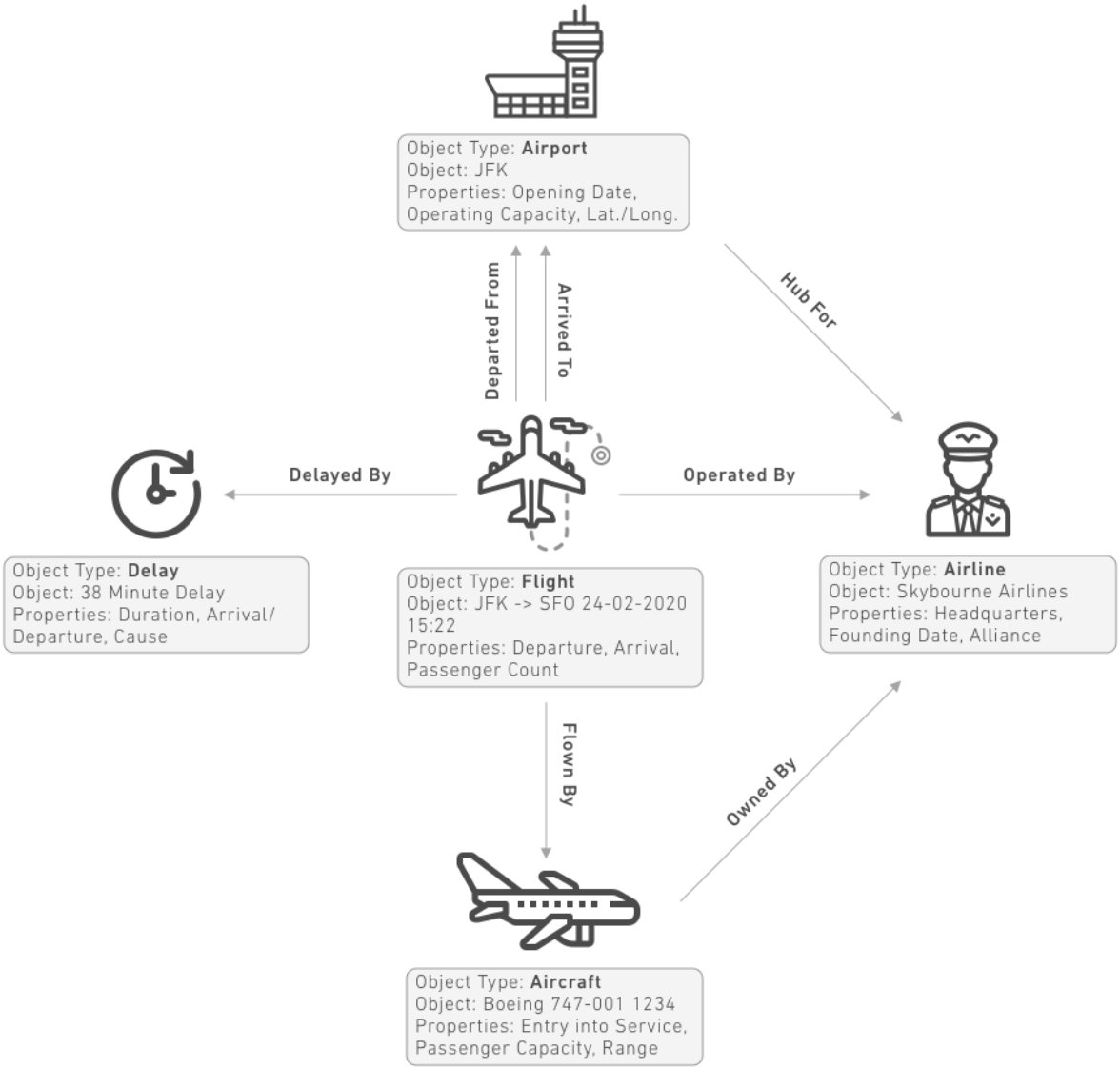
这里有两个常见误判。

把一个客户的方案直接塞进产品。 它只解决了一个客户的问题，未来客户却都要为它付维护成本。正确的抽象必须比客户带来的问题更通用一级。Palantir 早期从为"人员""资金"等具体对象分别建表，转向用对象、属性、关联和操作来描述业务——这套建模方式后来定型为**本体论（Ontology）**：不为每个具体对象单独建表，而是用统一的对象类型、属性、关联类型和操作类型来描述任何一类实体（见图 21–2）。 [3] Palantir 官方把本体论定位为"组织的数字孪生"，并特别强调它不是一层薄薄的语义层——数据、逻辑、操作、安全四者的整合，靠贴标签式的字段字典做不到。 [5] 客户差异留在建模与配置层，通用能力留在产品层；本体论怎样跨客户持续生长、反哺下一次交付，第 22 章展开。

[2] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[5] Palantir 官方文档：Ontology 架构与权限体系说明



A simple ontology of 5 object types displays some of the properties and relationships within airline industry datasets.

图 21–2：Palantir 官方本体论教学案例（航空业），图中英文标注为原图，未译；图片来源：Palantir 官方文档。[5]

共性信号还没出现就急着产品化。 只有第一个客户时就为未来客户投入，容易把猜测做成平台功能。更稳妥的纪律是：没有第二个客户之前，不急着铺开；有了第三个同类需求仍不处理，就该检查回流机制。

判断信号可以来自三个地方：同一段流程被两个客户以不同变体提出；售前反复演示同一套集成；新客户主动要求沿用上一套方案。还有一个前提：后面的客户必须走相近的业务路径。一位从业者的个案

[5] Palantir 官方文档：Ontology 架构与权限体系说明

[6] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

是，服务三到五家同行业客户后，流程才逐步成型为面向该行业的智能体产品。 [6]

八、不会递减的部分，按服务生意核算

有些定制天然不递减：超大型客户的独有流程（那是它竞争力的载体，不会为你的复用让路）；强监管行业里各家不同的合规要求（监管文本不同，改造义务就不同）。这类需求的正确处理不是硬套递减叙事，而是承认它：**这就是服务生意，按服务生意定价**（第 20 章的人天与混合结构），把它单独核算（第 19 章的服务成本），别让它混进"我们也在做产品"的故事里抬高估值预期。

把天然不递减的部分诚实标注为服务，反而保住了递减部分的含金量——递减曲线上的每一分下降都是真的产品化；混在一起报"产品收入占比"，才是毛利率陷阱里最隐蔽的那一种。

曲线下降的背后，是一只手在把现场的经验搬回产品。这个机制怎么建、搬什么、谁负责、怎么激励——下一章，产品回流。

第 22 章 · 客户现场的经验，怎么沉淀为产品能力？

> 好的意图不管用，机制才管用。
>
> —— 杰夫·贝索斯

"这个坑怎么又来了？"

第二个客户上线前夜，FDE看着报错记录愣了一下。问题和上一个客户几乎一样：两个系统对同一件事用了不同叫法，流程一接起来，权限和审批就全乱了。

他知道怎么修。上一次他已经修过一次。

只是上一次的规则、映射和测试，都留在另一个项目里。于是他又从头做了一遍。客户的问题解决了，公司却第二次忘掉了答案。

一、现场的答案，为什么没有回到产品里

一家公司做前线部署时，实际上同时运行着两套东西。

第一套在客户现场。FDE 用这家客户的数据、权限和流程，把一个具体问题跑通；目标是让项目上线。第二套是供应商自己的产品平台：可供多个客户复用的数据模型、集成组件、流程能力和应用工具；目标是让下一次交付不必从头开始。

现场与平台之间并不是天然连通的。FDE 在现场发现了问题，也写出了能用的解法；如果它只作为项目交付物留在客户目录里，平台就没有学到任何东西。久而久之，平台功能看上去不少，现场却仍要反复定制，最后实际做的还是外包。

问题的本质是：需求信号产生在交付团队，产品决策发生在平台团队，中间没有固定的传递机制。交付团队优先赶里程碑，产品团队优先守路线图；谁都没有错，经验却会停在项目里。

回流因此不是一句"多写文档"，而是一套被设计出来的接口：把现场形成的资产识别出来，带入产品决策，再让新的产品能力回到下一次交付。

二、先把现场问题翻译成共同业务语言

本章开头那个问题，表面上是字段映射，实质上是两套系统没有说同一种业务语言。一个系统叫"服务请求"，另一个叫"工单"；FDE 临时把两者连起来，客户就能继续使用。可下一个客户又会换一套名称、状态和审批方式。

要让这类经验跨客户留下来，不能只保存某次映射规则，还要沉淀一套描述业务的共同语言。这就是**本体论（Ontology）**：它规定业务里有什么对象、对象之间怎样关联，以及在什么规则和权限下可以执行什么动作。[1]

以工单场景为例：工单、客户、设备和服务人员是对象；"某张工单属于哪台设备""由谁处理"是关联；派单、升级、关闭是动作；金额阈值、审批层级和可见范围是规则。客户可以保留自己的字段名和原始数据，平台沉淀的是这套建模方式。

这套建模方式里，还有一层容易被"对象、关联、动作"三个词盖住的维度：**权限**。谁能看到这张工单、谁能派单、谁能改字段——这些边界本身也要沉淀，不是事后补一层访问控制。Palantir 官方用一家医疗制造企业的例子说明这一点：生产团队需要设备的全局遥测权限，仓库员工的权限按所在区域收窄，供应链分析师的权限精确到行和列；当这三类角色都开始用 AI 智能体去操作同一套业务对象时，每个智能体的权限必须能继承自它所代表的人类用户或所属项目的权限结构，不能另开一套规则。[2] 对象、关联、动作、权限，合起来才是这套共同语言的完整定义——但定义写清楚只是起点，它怎么从一次性的映射规则，变成下一个客户能直接拿去用的资产？

"蒸馏"这个词，是"回流"更通俗的说法。陆骁鹏（Ventus AI 联合创始人）的表述最锋利：散落在每个人脑子里的经验不叫沉淀，汇聚成具象化的知识库或 Skill，才叫沉淀。钟钱杰（Cresta）给出了更进一步的产品化形态：他的团队把蒸馏后的成果——可能是一份 Markdown、一个 CLI、一段脚本——封装成智能体（agentic）工具，客户说想做什么，后台自动调用最佳实践去尝试[3]。业务本体是"蒸馏"出来的共同语言，下面这六类资产，是这套语言之外还能被蒸馏出来、并能封装成工具反哺现场的其余部分。

三、回流的不是"经验"，而是六类资产

业务本体是共同语言，但它不能独自完成交付。现场还会留下五类可搬运的资产。

连接器。 连接器不是"接了一个接口"的口头经验，而是可复用的集成组件：认证方式、数据读取、字段映射、同步规则、异常处理和权限边界。它解决的是数据怎么稳定地接进来，而不是把客户数据带走。

工具与审批模板。 现场为约束智能体动作而搭出的工单系统、审批链、风险分级和操作界面，若能被不同客户配置使用，就不该永远留在一次项目里。

[1] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[2] Palantir 官方文档：Ontology 架构与权限体系说明

[3] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

流程模板。 同行业反复出现的流程，可以抽象成参数化模板。配置改参数，定制改代码；模板的价值，是把越来越多的"改代码"变成"改参数"。

评测结构与失败模式库。 可带走的是任务类型、难度分层、判定规则和失效模式，不是客户案例数据。它让下一个项目从"怎么验证"开始，而不是从"先试试看"开始。 [4]

需求信号。 哪些请求在不同客户处反复出现、哪些只是一家客户的例外，这些记录决定什么值得进入产品路线图。它往往最容易被扔掉，也最接近产品方向。

一位 Palantir 前员工回顾，现场反复出现的导数据、做可视化和搭页面需求，后来逐步沉淀为数据摄取、可视化和应用构建工具。产品不是先画好再被客户接受的；很多时候，它从重复出现的现场请求里长出来。 [5]

四、让资产真正回到平台：三步与两道闸

第一步，FDE 在现场构建并记录问题的上下文：客户为什么要这个能力、替代方案为什么失效、哪些部分属于当前客户。

第二步，FDE 带着这些上下文回到产品团队，共同讨论"正确的通用版本"。只交一份需求文档不够，因为文档通常丢掉了形成需求的约束和取舍。

第三步，邀请服务其他客户的 FDE 一起校验。三个略有不同但同源的工作流摆在眼前，团队才更容易分清什么是行业共性，什么只是单个客户的习惯。 [1]

三步前后还要加两道闸。第一道闸管进：需求先分为一次性、行业共性和通用三类，一次性需求可以交付，但不直接进入产品队列。第二道闸管出：回流评审必须给每项资产明确去处——进入产品路线图、进入组件库、进入模板库，或明确不做（见图 22–1）。

[1] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[4] Inside an Applied AI company（Pace 长文）

[5] Reflections on Palantir（Nabeel Qureshi）

图 22–1：不是每个客户需求都应产品化；通过两道闸，才把可复用的现场经验送回平台。

钟钱杰团队的做法是这条出口路径的一个具体范例：蒸馏后的资产不只是"进组件库"这么模糊，而是直接封装成一个智能体（agentic）工具——客户说想做什么，工具自动调用沉淀好的最佳实践去尝试，不行就换下一个方案再试 [3] 。

产品化也不能只听客户开口要什么。一个需求至少要同时过三问：它是否在相近客户处重复出现？是否服务客户真正重要的业务目标？能否比当前客户方案再抽象一层？三问里任何一问答不上来，就不应急着把它做成平台能力。

五、让回流成为双方都愿意做的事

清单和流程改变不了激励，就只能停在纸面上。做交付的人要能看见回流的收益：自己沉淀的组件、模板和评测结构被后续项目复用时，应计入交付成果。

产品团队也需要为现场信号预留空间。没有明确容量和决策权，现场需求永远排不过既定路线图；只收件、不处置，回流评审只会变成收集箱。

回流也不该是单向的。平台每沉淀一个可配置组件、流程模板或评测结构，都应优先回到 FDE 手里，让他在下一个客户处少拉人、少写一次性代码、把时间用在更难的问题上。麦格鲁将 FDE 称为产品的另一类关键客户，指的正是这种关系。 [1]

[1] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[3] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

回流跑没跑起来，不看会议纪要，看 FDE 的选择：面对新需求时，他会不会主动用平台组件拼装，而不是绕开平台再写一套临时方案。

六、三种让回流失效的做法

把评审开成甩锅会。 交付团队吐槽产品，产品团队辩解路线图，却没有产出可处置的资产、做出明确的决定。评审只讨论可处置的资产，每项都要有明确去处。

把评审开成只进不出的收集箱。 需求被记进待办，却没有人能调整排序；几个月后，现场自然不再提交。回流评审必须拥有影响产品优先级的权限。

把第一个客户的答案直接产品化。 这不是回流，而是把定制代码搬进平台。没有第二个相近客户前，不必急着投入；同类需求持续出现后仍没有动作，才应回头检查机制。

第 21 章问的是：第 N 个客户为什么不能继续重复第一个客户的劳动。本章给出的答案是：要把现场反复得到的答案，翻译成共同语言、组件、模板和评测，再通过一套有决策权的机制送回平台。

第六部 · 人与组织

账算完了，墙翻过了，剩下的变量只有一个：人。

六章按三个视角排开：先是**看清楚这份工作**——第 23 章剥开滤镜看一天与一周的真实情况，第 24 章立起三大支柱能力模型与十二问自查；再是**走通这条路**——第 25 章讲工程师怎么转型，第 26 章换到雇主一侧讲什么时候建团队、怎么建，第 27 章讨论个人能不能独立单干；最后第 28 章收束全书——经验会保值吗、角色会留下吗、这个结构会消失吗，以及那句话：AI 项目卡在没有人为最后一公里负责，FDE 是这个责任此刻的名字。

写给正在犹豫要不要入场的工程师、正在盘算建制的管理者、正在考虑要不要单干的从业者——也写给读完全书、想检验自己判断的行业观察者。

第 23 章 · FDE 到底在干什么?

> 管理者的活动，特征是简短、多变、不连续。
>
> —— 亨利·明茨伯格

杭州，晚上十一点半，一位 FDE 的微信响了：客户打来语音，说周报机器人不发消息了。他远程连上系统查了一圈，一切正常——最后派人第二天跑一趟现场，发现是那台电脑不知道被谁关机了。 [1]

同一周，硅谷，另一位 FDE 的十个客户系统同时报警：上游的 Cloudflare 宕机，没有预案，救火一晚上。

这是同一份工作的两种夜晚。这一章拿掉职位滤镜，从四个真实样本里拆出这个角色的真实节奏、任务构成与情绪体验：硅谷的两种打法（多客户并行、驻场深耕）、一位刚入职的日本新人，和杭州这头的中国日常。

一、两种节奏：多客户并行 vs 驻场深耕

第一份样本来自硅谷，一位有十二年经验的工程师，近两年转入前线部署式的角色。他自己发了一篇"一周日记"：**十个活跃客户，一周约五十小时，会议与真正写东西的时间约各一半**，外加计划外的救火——开头那次 Cloudflare 宕机，就是他这十个客户的系统一起报警。 [2]

值得逐条看的是任务构成：并行推进多个客户的构建、售前电话、救火、周报。没有一条是"研究算法"。有人在这篇日记下问他与传统软件工程师的差别，他的回答很直接——某种意义上更有成就感，因为解决方案从头到尾归他所有。 [2]

第二份样本节奏完全不同，同样在美国：Baseten 的那位 FDE（就是第 1 章时间账的作者，七成半在工程与模型优化），交付完成后转入支持角色，继续守着同一个客户。 [3]

对照着看：一个是**多客户并行型**（十个客户、高频切换、售前也管），一个是**驻场深耕型**（少数客户、工程占比更高、交付后接着运维）。同一个头衔，物质基础、节奏、能力侧重都不一样——第 18 章讲过这是市场结构决定的，这里先记下一个事实：**谈"FDE 的日常"之前，先问哪一种 FDE。**

二、新人的第一天，不是写算法

[1] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[2] 一名前线部署工程师的一周（10 个客户，50 小时）

[3] Baseten：我做前线部署工程师学到了什么（Het Trivedi）

[4] 「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实（note.com · AI Shift）

第三个样本来自一位入职两个月的日本新人，她的记叙纠正了最常见的想象。她原以为这是"在客户现场一个人战斗"的工作，两个月后看到的真实是：项目刚上线，正处在等客户真正上手使用的阶段；她的日常是**整理需求、有时写代码、跟进到客户的愿望真正实现为止**。 [4]

两个细节最有分量。其一，她做的事包括**和客户的管理员一起，把生产环境的权限与认证配置好**——新人 FDE 的起点不是算法，是接系统（第 10 章的全部）。其二，她的工作信条是：**与其写完美的规格书，不如做一个能让客户实际上手试用的原型，看客户的反应，当场修正方向**——这不就是第 7 章最小可行部署的新人版吗。 [4]

三、拼一张一周结构图

三个样本各有侧重，拼一张综合的周结构图：**周一**，现场或计划会——对齐本周目标与风险；**周中**，构建与联调——写代码、接权限、跑评测（第三部的日常）；**周五**，客户周报与评测回顾——用数据说话（第 12、13 章）。

但理想循环只是一半。另一半，是随时冒出来的生产事故（半夜的微信、宕机的上游）、临时加塞的需求（老板要的演示、客户临时拉会），和维系关系的人情——中国样本里这一项不是润滑剂，是承重墙。两个一半的比例，基本决定了这份工作的体验是哪一种（见图 23–1）。

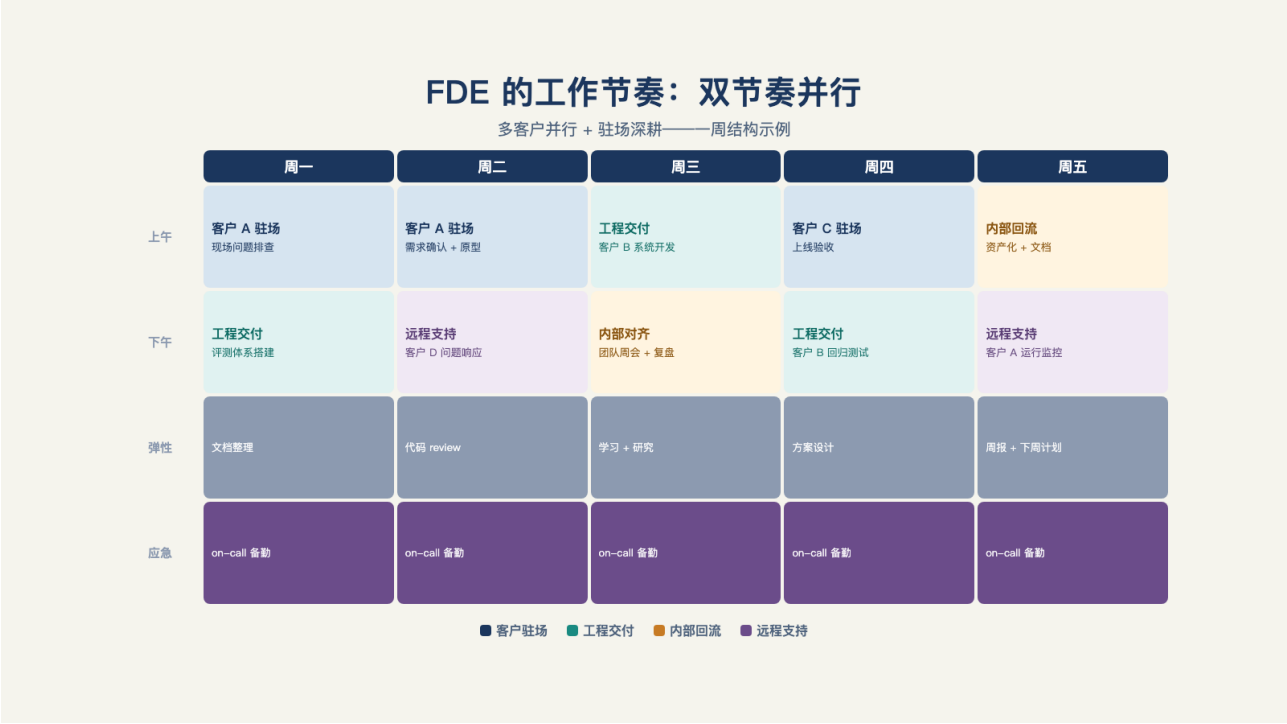


图 23–1：FDE 的工作节奏在客户现场、工程交付、内部回流与持续支持之间切换。

[4] 「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实（note.com · AI Shift）

四、中国的一天

把目光转回中国——这里的物质基础完全是另一套。开头那通深夜电话——周报机器人不发消息，一查是电脑被顺手关了机——就是这套物质基础的产物；第 18 章那笔**六成时间在业务梳理与人情**的时间账，落到一天里就是这种节奏。 [1] 同一份记叙里还有一个更荒诞的细节：客户拿着 XP 系统的电脑来装部署包，这可是2026 年，还在用XP系统。 [1]

真正的分岔在这里：社区观察到的国内"驻场原型型" FDE，一天的主轴是守着一个客户做原型、改原型——责任半径窄，但深度和依附性强；硅谷的多客户型，责任半径宽，切换成本高。 [5] 同样是"一天"，装的东西完全不同——这正是第 18 章结构差异落到个人日程上的样子。

五、正在变宽的上限

现在回到第一节的两个数字：十个客户、五十小时。他能同时扛十个，除了个人能力，还有一个正在起变化的因素——**工具杠杆**。一位硅谷从业者给出的方向性判断：录音转写、企业内部搜索一类工具，短期内可能把单个 FDE 能同时扛的项目数从两三个提到五六个；长期则会让角色分化——一端是啃最难问题的高端 FDE，另一端是服务中小客户、甚至不上现场的远程 FDE。 [6]

这与第 5 章"执行层贬值"的判断同构：工具把"一天"的产出上限往上推的同时，也在把角色拉开档次。**"一天"不是常量，是被工具改写中的变量。**

六、情绪的真话

先说正面情绪。回到第一节那位硅谷工程师的话：端到端的所有权感——自己看见问题、自己解决、自己收到客户的反馈，这是十二年工程师转岗后最直接的收获。 [2]

反面的痛感则有两层。第一层在甲方：认知误区、技术栈不统一、代码做不到端到端交付、安全没保障，返工不断，全压在一线团队身上。 [7] 第二层，这股痛感会沿着责任链传导到乙方：甲方的 997，就是 FDE 的半夜微信、崩了要抢救的系统、多客户并行反复切换的消耗。

同一份工作，在不同人身上是天堂或地狱。分界变量只有两个：**对模糊性的耐受，和对所有权的需求**——耐得住打断、受得了模糊、并且强烈需要"这事从头到尾归我"的人，在这份工作里如鱼得水；反过

[1] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[2] 一名前线部署工程师的一周（10 个客户，50 小时）

[5] 有在做 FDE 这个工作的佬吗（linux.do 帖）

[6] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

[7] AI 职觉：别再卷 Agent 了（小宇宙播客）

来则是持续的消耗。这两个变量，就是下一章能力模型的最后一道门槛。

七、想去这个方向，先试三件事

不必先辞职再验证。想检验自己适不适合，可以在当前岗位里先试试这三个场景：

给内部用户做一次真实交付。 找一个跨部门的内部工具需求，从需求访谈做到上线运维——体验"用户就坐在旁边"的交付。

跟一次完整的客户会议。 从售前听到售后，数一数一场会里有多少个"这不是技术问题的问题"。

接手一个上线后的系统，管一个月。 哪怕只是值班——上线之后的每个报警、每句"怎么又不好用了"，就是这份工作的日常质地。

三件事做完，对模糊性耐受与所有权需求这两个变量，你会得到比任何测评都准的答案。

这一天、这一周的真实质地，算是摸清楚了。谁能扛下来、扛得好——下一章，从能力模型说起。

第 24 章 · 什么样的人适合做 FDE？

> 狐狸知道很多小事，刺猬知道一件大事。
>
> —— 阿尔基罗库斯，经以赛亚·伯林《刺猬与狐狸》转引

关于 FDE 岗位要什么能力，网上有两种说法，吵得不可开交。

一种说法：业务知识可以在工作中学，技术不行——技术是硬门槛，业务是软门槛。

另一种反过来：构建早已交给智能体，业务理解才是真正稀缺的东西。 [1] 两种说法都说得斩钉截铁——该信哪一种？

先别急着下结论，它们是同一个市场在前后两个阶段、两种立场下给出的答案。

真实的判据要等两个框架搭完再回来：三大支柱说清"要什么"，两层结构说清"什么会归零、什么会复利"——最后你会发现，那两种打架的说法各抓住了其中一层。

一、三大支柱：交集，不是单科

FDE 的能力是三根支柱的交集。 [2]

支柱一，工程交付能力。 不是算法研究，是"把东西弄到能运行"的工程：集成、权限、评测、运维——第三部的全部内容。它首先反驳"FDE 靠嘴"的想象：驻场样本里工程师把七成半的时间花在工程与模型优化上，靠嘴的人活不过第一堵墙。 [3]

支柱二，业务理解能力。 把客户的模糊目标拆成可测的流程。它分三层递进：第一层，听得懂行业术语；第二层，看得懂流程逻辑——这家的单子为什么这样走；第三层也是最稀缺的一层，读得懂激励结构——谁的绩效被这个流程撑着、谁的利益会被新系统动到。第三层恰好是第 16 章"三不懂"的反面：不懂组织的一把手做不了这个角色。

支柱三，项目推动能力。 知道谁拍板、谁受影响、谁的抵制要处理。甚至有人把"一把手认知"放在能力地图最前面、叫"第一道门"——这道门进不去，后面的能力无处施展；用它的比喻：门、流程、肌肉，只练肌肉的人会被挡在门外。 [2] 这是第 15 章三组人物（支持者/影响者/受损者）的操盘版。

三大支柱抽象，一家公司的真实招聘要求能把它落到具体门槛上。

---

[1] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

[2] 《关于 FDE 的 100 个问题》（AI磊叔）

[3] Baseten：我做前线部署工程师学到了什么（Het Trivedi）

[4] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

Cresta 的 FDE 负责人有一条明确"不招谁"的标准：**不招资历浅的 FDE**——一个项目现场往往只有一两个人和客户共创，资历不够建立不起信任。三大支柱在他的招聘语言里对应成三层：技术上必须是"合格的开发者，开发测试过 AI agent"；沟通上要有"过硬、让人信服的面向客户的经验"，能跟对方 CTO、IT 总监、安全负责人对话，也能把复杂问题讲给非技术人员听懂；剩下是靠谱、韧性、能扛不确定性的特质——他偏好招创始工程师、有过大风大浪经验的人（其称之为迷你 CTO，实干型，能写代码、能跟客户交流、非常靠谱）。 [4]

三大支柱是交集不是全满：**及格线以上即可入场，长板决定风格**——工程长板偏交付型，业务长板偏发现型（本章第三节的两种角色），组织长板偏推动型。"全栈全能超人"是误解，三类长板各成一派。

三大支柱也不是固定配方——权重随市场结构变。硅谷样本七成半时间在工程； [3] 中国一位服务过三十多家企业的从业者却把这活拆成顾问、销售、工程、教学四块，工程排第三，六成时间花在业务与人情上。 [5] 差异不在谁对谁错，在有没有产品底座（第 18 章）：底座替你扛住复杂性，工程时间自然变成交付；没有底座，顾问与销售才是瓶颈。**读者要学的不是背下哪个排序，而是判断自己面对的市场该用哪个排序。**

还有一层关系要点破：三大支柱是"建制所需合成能力"的岗位化表达。支柱模型描述的从来不是某个超人，而是一支小队能力的分解图。团队形态下，三根支柱可以分属不同角色——第 26 章的中国三人组（BA、AI 架构师、业务教练/陪跑团队）就是一支小队各长一角；个人形态下，三根支柱压在一个人身上。

二、两层结构：什么归零，什么复利

比排序更要紧的是时间维度。

行业层：某行业的流程、术语、数据形态。换行业即归零重学——零售的补货逻辑帮不上金融的风控。

元能力层：问题拆解、基线测量、验收设计、责任管理——这是元能力层原来就有的四项。还要再加一项更底层的：学习本身的能力。模型、框架、行业黑话年年在变，新术语、新流程层出不穷；元能力层里最抗折旧的，不是记住了什么，而是"多快能吃透一套新东西"这个本事，以及愿不愿意持续接收新东西的心态——开放程度本身，也是一种元能力。跨行业复利——第 8 章的验收契约、第 12 章的评测方法，换个行业照用；换了新工具、新术语，靠的也是同一套"重新学习"的肌肉（见图 24–1）。

[3] Baseten：我做前线部署工程师学到了什么（Het Trivedi）

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

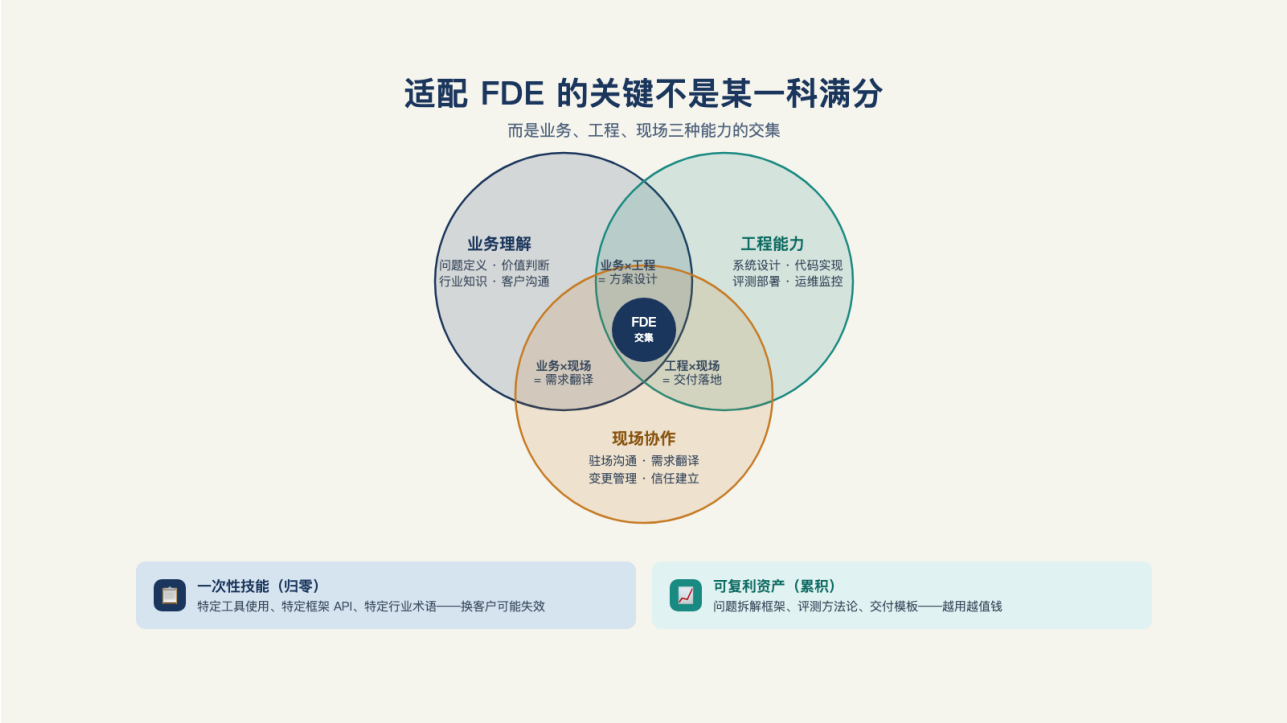


图 24–1：适配 FDE 的关键不是某一科拿满分，而是三种能力的交集和可复利资产。

样本佐证：那位十二年经验的工程师转入新角色，可迁移的显然不是任何行业知识，而是工程元能力——生产意识、排障直觉、对系统的手感。 [6] 同理，一位 FDE 从零售换到制造，归零的是行业层，带走的是元能力层。

投资含义直接：**培养与招聘的预算应该向元能力倾斜**——行业知识可以买到（顾问、专家访谈、驻场几个月自然长出来），元能力买不到。这也是对开头两种说法的回答。第一种，"业务可学"，在元能力层上成立。第二种，"业务理解才稀缺"，在行业层和激励结构层上同时成立。两层都对，合起来才完整。

三、发现者与落地者：同一种角色的两种人

三大支柱之上还有一层角色内分层——同一个 FDE 团队里，发现者与落地者招的是两种人。麦格鲁给过两类人的画像——发现者与落地者。第 1 章定义过这套分工，这里看它在招聘上的含义： [7]

发现者（Echo 型）：从所服务的领域里来——他举的例子是前陆军军官、医疗行业老手这样的领域老兵；并且必须是"叛逆者"：既懂现状、又认定现状不够好。为什么必须是叛逆者？因为新软件要带来三倍或十倍的阶跃式改变才值得做——安于现状的领域专家看不出为什么非得是这个倍数，也说服不了任何人。

落地者（Delta 型）的反画像更扎心：工匠型工程师是错误人选。 沉迷抽象的正确、追求"可维护十几

[6] 一名前线部署工程师的一周（10 个客户，50 小时）

[7] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

年"的代码——而落地者的工作是快速写出真正能运行的原型、承受大量的痛苦，并且**第一版代码通常是要扔掉重写的**（麦格鲁的原话在第 7 章 MVD 的代码处置处引过）。性格与背景与发现者完全不同，两者不能互换。

这段反画像会给工程师读者带来冲击，但要读对地方：它不是说工匠型工程师不好，是说**工匠价值观与这个岗位的交付节奏错配**——原型是拿来验证问题的，不是拿来传世的。

四、资深工程师的优势与短板

三大支柱+两角色合起来，资深工程师转 FDE 的体检单就出来了。

天然优势是**系统级判断力**。一个来自国内交付一线的教训：给客户做能力封装时，最关键的不是封装本身，而是**知道想封装什么**——这需要架构判断，需要公司的中层甚至更高层以架构师的角色第一顺位参与。［8］资深工程师十年攒下的功底，恰好补在这一项上。

常见短板是项目推动能力：习惯了对代码负责、不习惯对人负责；习惯了自己写、不习惯看着别人写；最不习惯的是在没有明确规格的地方开工——而现场根本没有规格。所以**转岗培训的重心应放在短板而非长板**：让工程骨干去跑客户会议、做需求访谈、挨一次投诉，比教他们新技术有用得多。

两家公司的招聘哲学能互相印证，也能互相较劲。钟钱杰（Cresta）这边不招任何 junior FDE，只招做过 AI Agent 开发测试的人；陆骁鹏（Ventus AI）这边招过两个还在读书的年轻人——一个刚高中毕业准备去哈佛、一个 NYU 肄业——但他自己也承认"差点意思"，原因是早期公司的 FDE 要独立承担客户沟通和项目管理，"没自己踩过几次坑，很难有项目管理的意识"［9］。两条招聘线合起来印证的是本章"三大支柱不是全满、但门槛不能降"的判断：Cresta 靠资历筛门槛，Ventus 靠承认短板控制风险，路径不同，但都不敢用没经历过项目推动能力考验的人补位——恰恰是本节说的短板，两边都躲不过去。

招聘漏斗的数字也可以参考：钟钱杰的团队一个岗位收到过 7000 份简历，发出的 offer 不到 20 个［9］。这组数字印证的正是本章开篇的判断——三大支柱交集不是写书的人事后总结出来的框架，是招聘现场真实在用的筛子，而且筛得比想象中更狠。

五、适配度自查

十二个问题，帮你自查——不是心理测评，别当判决书用。每问按"经常如此/看情况/几乎不"给自己记 2/1/0 分。

[8] 一路瞳行 EP134：从 AI 辅助到数字员工（小宇宙播客）

[9] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

1. 别人给一个模糊目标（"把这件事搞定"），我能自己拆出第一步。
2. 被打断后回来，能较快捡起之前的思路。
3. 和陌生业务的人聊天，我能问出对方真正着急的事。
4. 我能对着不懂技术的人把技术方案讲到对方能决策。
5. 没有标准答案时，我敢基于有限信息拍板并承担后果。
6. 我愿意对"结果"负责，而不只对"代码质量"负责。
7. 客户半夜找我，我的第一反应是解决问题而不是烦躁。
8. 我能同时挂着三件以上的事而不漏。
9. 看到别人的业务流程乱，我手痒想理顺它。
10. 我能接受第一版方案将来被扔掉重写。
11. 我愿意花时间经营关系（吃饭、闲聊、人情）并视其为工作的一部分。
12. 换一个行业从头学起，我觉得兴奋多于恐惧。

18 分以上：三大支柱的底子和心理门槛基本齐备，值得一试（第 25 章给路径）。**12 到 17 分**：具备部分条件，先看失分集中在哪一支柱，用第 23 章第七节的三件事定向补。**11 分以下**：不是能力不够，是适配存疑——先看第六节的三思清单再决定。

六、不适配的三思清单

最后保留反方：三种人转 FDE 前应三思。注意，这不是能力问题，是**对齐问题**——错配的代价是双输。

偏好深度专注单问题的人。 这份工作的默认状态是高频切换（第 23 章）——频繁切换对有些人只是家常便饭，对偏好深度专注的人却是持续的消耗。

需要明确规格才能开工的人。 现场的规格要自己写，而且写了会改。等待清晰指令的工作习惯在这里是奢侈品。

只愿对代码负责、不愿对结果负责的人。 这份工作的考核对象是客户业务的变化，代码只是手段——把手段当全部责任边界的人，会在每一次"系统对了但业务没变"的复盘中失语。

七、公司视角：买不到，就自己养

个人视角之外，我们从公司视角看 FDE。

复合型人才不是贵就是能力不行，大部分公司只能自己培养。[10] 内部的培养路径因此比招聘更重要：**工程骨干＋现场轮岗＋影子交付**——从工程团队挑元能力底子好的人，放进真实客户现场轮岗一到

[10] 有在做 FDE 这个工作的佬吗（linux.do 帖）

两个季度，先做影子（跟着资深 FDE 看全流程），再独立扛一个小客户的完整闭环。

能力模型立起来了。下一个问题非常实际：一个已经在岗的工程师，怎么走完从写代码到扛结果的这段转型——下一章，转型路径。

第 25 章 · 工程师怎么转型 FDE?

> 疤痕，标志着真正下场玩过。
>
> —— 纳西姆·塔勒布《非对称风险》

两个"三十天"。

一个在硅谷的社交平台上：一条"如何在三十天内成为百万美元级前线部署工程师"的帖子，三十六万次浏览，附三十天路线图与薪酬档位。 [1] 中文的"FDE 培训"课表，只能用四个字来形容——"一言难尽"：第一节课教怎么在电脑上打印 Hello World，通篇没有一点落地训练，请一堆名师，收两万块。 [2]

转型的焦虑是真的，围着这份焦虑长出来的培训市场，最先吹起了泡沫。这一章不再重复"怎么准备"——这个话题已经被写烂了，只谈两个常被忽略的问题：**被点名转岗时该怎么判断，雇主招人时到底在看什么。**

一、转型路径

五类背景各有各的近路和坑：后端出身，工程交付力现成，要补的是业务还原与组织推动；产品经理出身，业务与组织的底子在，缺的是端到端的工程手感；咨询顾问出身，客户沟通不是问题，最难迈的坎是从"交一份方案"变成"交一整套能跑的系统"；数据分析师出身，评测和数据的底子在，要补系统集成与责任承担；算法工程师出身，最容易犯的错是高估模型、低估集成——第 9 章的八堵墙里，模型甚至不占一堵，集成和数据权限才是先撞上的墙。

有一句所有背景都通用的话：**转型的入场券不是学完一门课，是拿出一个能给人看的交付证据。**

二、被点名转岗：是机会还是坑

谈转型的内容，默认的都是"你想转"。但现实里，大量转型是被推着走的——一位外企后端工程师的帖子写得很实在：他 AI 用得熟练，在研发团队里沟通表达也算好的，领导因此想安排他转 FDE；可他喜欢技术，"一想到要走这条路，就感觉自己可能会离技术越来越远"，心里不甘。 [3] 碰到这种情况，

[1] 「30 天百万美元」课程营销帖（@gregisenberg）

[2] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[3] 研发转 FDE，是升维还是转行做实施？（linux.do 帖）

不该急着"接受"或"拒绝"，该先做三步检验。

第一步，做一次三重检验， 标准来自第 2 章。 问三个问题：我写的代码上不上生产？现场的经验回流到哪？这个岗位考核什么？三问都能得到明确的答案，是真 FDE；支支吾吾、只会说"先干着看"的，多半是实施换了张皮。

第二步，验组织支不支持：公司是真有一个业务问题要解决，还是"别人都有我也想有"？有没有业务方愿意投入？还有一个更直接的信号：这个岗位挂在哪个部门、向谁汇报。挂产品或工程线，收入记在平台产品上，FDE 成功等于产品被采用；挂销售或交付线，按人天收费，容易沦为售前的人力池、退化成高级实施。 [4] [5] 面试时可以直接反问"FDE 向谁汇报"，比自己猜更准。孤岗 FDE——全公司就你一个人，没有回流通道，没有业务接口——是个高风险配置：干好了是个人的功劳，干砸了没有组织兜底。

第三步，验自己对不对得上（第 24 章的十二问）。分数高低不是重点，真正要看的是"离技术越来越远"这个担心站不站得住脚：真 FDE 的工程含量并不低（第 23 章那七成半的工程时间就是证据），离得远的其实是算法研究式的深度专注——先弄清楚自己在意的到底是哪一种"技术"。

三步走完，结果无非三种：三关都过，全力转；岗位是真的但组织支持弱，就谈边界——要业务接口、要回流机制再接；岗位本身是假的，那就在内部换个轨道，别把职业生涯押在一个换了皮的头衔上。

三、雇主看什么

还有一个常被忽略的角度：招转岗者的公司，凭什么给你机会。这决定了转型者该把力气花在哪。

算得上信号的证据：给内部用户做过的真实交付——有真实用户，有上线后的运维痕迹；跨部门项目里的推进记录——会议纪要里你是那个跟到底的人；维护过一个上线后的系统——处理过报警、接过抱怨、做过迭代。

算不上信号的证据：课程证书；没有真实用户的演示合集；"学过某某 Agent 框架"——框架很快就会过时，元能力才留得住（第 24 章的两层结构在招聘上的体现：雇主买的不是你会用哪个框架，是你在没人给需求的时候，自己做过架构选择）（见图 25–1）。

[4] 《关于 FDE 的 100 个问题》（AI磊叔）

[5] 范冰《前线部署工程师（FDE）》（XDash 开源书）

图 25–1：转型靠一组可验证的交付信号，而不是把岗位名称写进简历。

所以"先学完再动"是个策略错误：学是必要的，但学的产出——证书、课程作业——恰恰是最没说服力的信号。**在现在的岗位上做出一个能给人看的交付证据，比上任何课程都值钱**——这也是为什么第 23 章那份一天体验清单（内部交付、客户会议、管一个月上线系统）同时也是转型者的简历工程。

四、培训市场：怎么判断靠不靠谱

回到开头那两个"三十天"。中美两个样本说的是同一件事：转型培训是这条链路上最先吹泡沫的一环——焦虑出现在需求成熟之前，而焦虑向来是最好卖的东西。 [1] [2]

选课或选训练营，只有一条硬判据：**有没有真实项目代练。** 有真实客户、真实交付、真实验收的训练，贵也值；没有的话，课表越漂亮越要留心——"三十天百万美元"和"第一节课打印 Hello World"，本质是同一种生意。多说一句提醒：上面那位批评培训泡沫的从业者自己也在办训练营，卖点正是"真实项目代练"——观点和他的生意方向一致，听道理归听道理，留一分清醒。 [2]

五、最省钱的验证方式

不辞职也能验证这条路——第 23 章那三个场景（内部交付、跨部门项目、客户会议）本来就是最省成本的试错，不用另外再想一套。做完它们，你会同时拿到三样东西：一个能给人看的证据（第三节）、

[1] 「30 天百万美元」课程营销帖（@gregisenberg）

[2] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

一份合不合适的答案（第 24 章）、以及"会不会离技术越来越远"这个问题的答案（第二节）。

六、降薪与预期

最后一个诚实的问题：转型要不要降薪？市场给不出可信的数字——转岗定价这件事正处在混乱期，同一座城市、同一个头衔，价差能大到离谱。不给数字，给一个结构性的判断：**混乱期里，个体的差异比岗位的差异更大**——那份能给人看的交付证据有多硬，对你身价的影响，比"FDE"这三个字母本身更大。所以谈判该锚定的不是头衔对标，是证据对标：拿着那份内部交付的记录去谈，和拿着一句"我会做 FDE"去谈，站的位置完全不一样。

个人这条路走完了。再看桌子另一边：一家公司什么时候需要第一个这样的人、把他放在哪、怎么让团队长起来——下一章，雇主那边的问题。

第 26 章 · FDE团队怎么建?

> 没有什么，比把根本不该做的事，做得很有效率，更没有意义了。
>
> —— 彼得·德鲁克

公司的 AI 项目落不了地，第一反应通常是三连问：模型不够强？加两个算法工程师。人不行？换一茬团队。工具跟不上？买一整套平台。

钱花了，人换了，工具也升级了——项目还卡在原地。

这一章告诉你，真正缺少的，可能正是一个从头到尾对结果负责的角色。症结找到了，什么时候招、招几个、怎么长成一支队伍——答案自己会浮出来。

一、先诊断：你缺的到底是什么

开头那三连——模型不够强、人不行、工具跟不上——背后至少藏着三种不同的病，药完全不同。

病因一，问题选择错。 选了一个不值得解决的问题（第 6 章五道关没过）——这是方向病，换人没用，重选问题才有用。

病因二，工程墙没翻。 问题和方向都对，人倒在第三部那八堵墙上——数据与权限没理清、集成打不通、效果证不出来、结果换不来信任——这是工程病，需要的是交付工程能力。

病因三，没人对结果负责。 系统上线了，没人用——这是组织病，需要的是一个对结果端到端负责的角色。

三种病因只有第一种可能涉及换人，多数时候正确答案是**加一个角色，而不是换一个团队**。

诊断工具推荐项目失败原因分析：把失败项目逐个按八堵墙（第 9 章）+四个责任段（问题发现/生产工程/业务采用/产品回流）归因——归因集中在"做不出"是模型与工程能力问题，集中在"没人用"是交付闭环所有者缺位。同样的诊断，埃森哲与微软 2026 年设立联合前线部署工程实践的公告里也写进了官方文案：企业 AI 卡住，缺的不是技术，是能把工程能力放对位置的人。 [1]

二、时机的三个信号

[1] 埃森哲推出微软前线部署工程实践

诊断出缺口，第二个问题是现在建不建。三个信号，齐了再招。

信号一，有明确的业务问题。 不是"探索一下 AI"，是一个说得出来的问题——哪个环节、什么代价、谁在着急（第 6 章）。"想搞 AI"是预算的起点，不是建制的起点。

信号二，有业务方愿意投入。 一把手背书、业务方出人出数据出时间（第 16 章两道门）——没有内部同盟的交付是空降，空降的存活率极低。

信号三，已经跑过一次尝试。 哪怕是外部服务商做的一个最小可行部署（第 7 章）——公司至少见过一次"从问题到上线"长什么样，知道自己要什么。

三信号不齐时的正确动作不是憋编制，是**先买小服务**：外部评估、单点交付，用别人的交付帮你把信号凑齐（那个市场下一章展开）。建制是重资产，确定性不够就不要押上。

三、第一个 FDE：画像、放置、考核

信号齐了，招第一个人。三件事定成败。

画像：资深，不是新人。 第一个 FDE 要的是资深工程交付力+业务还原力（第 24 章三大支柱的前两条强、第三条可练）。不能是新人——新人需要有团队罩着的成长环境：第 23 章那位入职两个月仍在客户确认期的日本新人，她的成长环境是有团队罩着的；[2] 把同样的人直接扔进没有支持的单岗，就是让她自己一个人在深水区扑腾。第一个 FDE 得是能自己搭脚手架的人。

放置：业务交付线，不是纯技术线。 汇报给业务或交付负责人，不是研究线——汇报线决定他优先响应什么。同时，与产品团队的接口从第一天就定义：哪怕只是每月一次的回流评审（第 22 章机制的最小版）。没有这个接口，第一个人就会一步步变成外包团队。[3]

考核：对结果考核，不对工时。 验收指标加采用指标（第 8、15 章），不是代码量或人天利用率——考核错位会把 FDE 用成普通开发，这是第 2 章判据三的组织版：卖时间的人不对结果负责，考核什么，人就是什么。

四、外部参照系（一）：双角色与三职能

建制没有标准答案，但有几组高信度的参照。第一组是角色分工的两种现代形态。

发现/落地双角色（第 1、24 章引过画像）：Echo 型的驻场分析师管发现与关系，Delta 型的部署工程师管落地——两角色按画像分开招，不能互换。[4]

[2] 「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实（note.com · AI Shift）

[3] @deployengineer（Bhaulik Patel）系列长文

[4] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[5] Inside an Applied AI company（Pace 长文）

[6] 硅谷 101 E240：聊硅谷最火新职位 FDE（YouTube）

三职能与账户制：一家应用人工智能公司把自己组织成三块——工程（平台与基础设施）、市场拓展（获客漏斗）、应用人工智能（桥梁：接住温热商机、定义价值证明、把 Agent 部署到生产环境）；应用人工智能内部再分两角——**Agent 产品经理**，从价值证明一路带到生产之后，技术好到自己能搭 Agent 但不必写生产代码；**前线部署工程师写代码、反哺产品**，且要有足够的商业感，能直接参与销售对话——因为"真正会去造这个东西的人，是房间里最有说服力的声音"。［5］另一家公司（Cresta）的变体给出了配比：一位前线部署产品经理带两到三位 FDE，产品经理可同时管理多个项目；团队规模化后还会有意识地按行业（如医疗保险）和技术领域（如支付、搜索）培养专长，而不是让每个人都当通才。［6］

两个样本合起来的启示：**"懂客户的"与"能落地的"从第一天就该是两个明确的角色**——要么两个岗位，要么一个岗位明说的两顶帽子；把这些要求全压在一个人身上，还要他样样全能，是很难招到人的。

五、外部参照系（二）：中国版——三人组与极限形态

第二组参照来自中文市场与巨头端。中国版给出的建制答案更完整：不是把工作切两半，是三兵种合成的一支小队。

走得最深的是瓴羊的"组织三人组"（建制背景与案例细节见第 18 章）。他们的判断为："我们的做法不是一个人，而是一个组织"——要一个人有这样的全才"非常难"。［7］组织因此分成三类角色：**BA 管方向**，以业务结果为导向——编排两百多步的客服流程、找出 AI 的最佳工作模式、设定评测标准，他们本是美妆、家电、汽车这些行业专家；**AI 架构师管技术**，把业务问题翻译成模型选型与系统设计——用什么尺寸的模型、哪个环节人干预、哪个环节机器接管；**客户方调教团队管业务（陪跑教练）**，从客户里选出最好的客服和销冠，像教练一样在过程中持续反馈、调教 AI——供应商在这里主动退后。［7］

把它与 Palantir 的 Echo/Delta 摆在一起，结构同源、切法不同：

	Palantir Echo/Delta（美国版）[^c06]	瓴羊三人组（中国版）[^b19]
方向：该做什么	Echo 驻场分析师与客户经理	BA：业务结果、流程编排、评测标准
技术：怎么做出来	Delta 部署工程师	AI 架构师：模型、数据、系统设计
业务：怎么用好	隐含在 Echo 的关系维护里	明确建制：客户方调教团，教练式持续反馈

差异集中在第三行：美国版两分（发现/落地），"业务教练"是 Echo 的隐含职责；中国版把它升格为正式建制的第三角——标杆业务知识散落在客户方（第 18 章的"岗位标杆为预设"），不把客户的人编进建制，模型就调不准。

弹性的口径同样来自中文市场。一位从业者曾分享过，国内企业做 AI 改造，真正需要的是三个角色：

[7] 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE（小宇宙播客）

[8] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

项目经理、工程师、产品经理；这三个角色可以是一个人，可以是两个人，也可以是一个团队，不强求编制。 [8] 说到底是同一句话：需要一个懂业务、能把需求翻译给 AI 工程师的人。三个角色怎么拼，看公司大小和信号强弱（第二节的三信号）——信号刚齐，一个人兼三个角色也能起步；信号扎实，再拆成三个岗位。三人组是标准形态，弹性拼法是它的裁剪。

极限形态：埃森哲与微软的联合实践——数千名人工智能技能工程师的规模，官方口径"从想法到生产以天计而非以月计"，充当"企业人工智能转型的门户"。 [1] 供参照，不供模仿——数千人是交付工业化之后的形态，不是起步形态。

两组参照横向对比，能提炼的共同点有两条。其一，**FDE 的完整形态不是一个人，是一支建制小队**——特种部队式的：小规模、多兵种合成、嵌入前线、对任务结果负责；单人 FDE 是这支小队被裁剪后的形态，本章第三节"第一个 FDE"说的正是裁剪到一人时怎么保住关键件。其二，**FDE 必须与产品团队有制度化的回流通道**——没有回流通道，交付就会在不知不觉里变成外包，这是跨市场的共同警告。 [3]

六、团队怎么从一个人长起来

第一个人站住后，团队怎么长?

第一步，跑通最小闭环。 一名 FDE＋一个产品接口人＋半个平台支持（后端/基础设施工程师的分摊），跑通两到三个完整交付。这一步验证的不是人，是**闭环本身**：问题定义→交付→采用→回流，全链路在公司里走通一遍。

第二步，按瓶颈扩，不按规划扩。 交付排队——加 FDE；同类定制反复出现（第 21 章的触发信号亮了）——加产品化人力，把定制收进平台；客户多到服务质量下滑——先问是人的瓶颈还是回流的瓶颈（是后者的话加人只会加乱）。每一步扩张用第 19 章的六指标复盘——扩张不看感觉，看实施杠杆率与定制递减率的方向（见图 26–1）。

[1] 埃森哲推出微软前线部署工程实践

[3] @deployengineer（Bhaulik Patel）系列长文

图 26–1：团队增长不是人数堆叠，而是让现场责任逐步被协作机制和平台能力接住。

这个路径的反面是"一步到位招一个二十人交付部门"——人员先于闭环到位，组织会在没有回流习惯的时候就固化成人海形态，第 19 章的英雄文化与咨询化陷阱都是从这个形态里长出来的。

七、三种反模式

收尾照例是镜子。

跟风设岗。 竞争对手有 FDE，所以我们也要有——第 3 章"外包换皮、一阵风"的组织版。信号不齐就设的岗位，最后都会变成挂新头衔的旧实施。

招了不管。 招了人、给了客户，但没有业务接口、没有回流通道、没有结果考核——三个缺失任占其二，这个人就在为竞争对手培养人才。这类人的离职信通常写得很客气，原因都不在信里。

用管咨询的方式管 FDE。 按人天考核、按利用率排产、把回流时间视为"没在干活"——管理动作全是咨询公司的，却期待产品化的结果。考核什么，得到什么；按工时管理，得到工时。

公司这边至此闭环。还剩最后一组人：不进公司、自己单干的人——个人能不能独立、兼职或创业做 FDE？下一章。

第 27 章 · 个人能独立做 FDE 吗?

> 精通一门手艺的技术活，不等于懂得经营一家靠这门手艺盈利的公司。
>
> —— 迈克尔·格伯《E-神话：再谈创业》

"FDE 这活，自己干没背书，打工又苦。"

一位独立从业者在论坛上留下这句自我总结。他管自己叫 OPC——One Person Company，一人公司；没有背书，他能想到的办法是放低身段，先攒够经验、方案和基础设施；帖子结尾，他在找同伴：一起探索客源、方法论、基础建设。 [1]

这句话问到了本章讨论的问题：一个人到底能不能以 FDE 的身份接企业客户的活？哪些单能接、哪些接了就是坑？

一、先把一个问题拆成两个

"个人能不能做 FDE"这个问题——其实混合了两个完全不同的问题。

问题一：单人能不能完成一个窄范围的原型或评估？ 能，而且 AI 显著放大了这个能力——一个熟练者用现成的模型、工具与框架，几周内给单个客户做出可评估的原型，今天完全可行（第 24 章"组装思路"）。

问题二：单人能不能可持续地承担多客户的生产级交付？ 极难。因为生产交付不是一项技能，是**并行的责任**：业务发现、系统接入、安全合规、工程交付、客户沟通、培训采用、线上支持、产品回流——八项责任挂在同一个人身上，其中任何一项在客户的生产系统里出事，都得有人顶着。线上支持意味着客户系统报警时你在，多客户并行意味着三个客户同时报警时你也在——7×24 的责任结构，一个人在数学上就扛不住——交付远不止写代码，参见撤场三件事：业务融合、知识治理、系统对接， [2] 每一件都是持续责任。

换个角度看更清楚：这八项责任本来就不是为一个人设计的。FDE 的完整形态是一支建制小队（第 26 章的三人组是它的一个编法）——OPC 走不通的根源，正是它要把一支小队压缩进一个人。

所以答案不是"能"或"不能"，是：**OPC 在逻辑上并非不可能，但作为商业命题，不能当主流组织形态来做。** 能不能走通，取决于你选的模式与条件——这正是下面两张表要给的。

[1] 在武汉做 FDE，寻伙伴（linux.do 帖）

[2] 十字路口 × Rolling AI：FDE 对谈（小宇宙播客）

二、FDE 模式之父的劝诫："不要轻易尝试"

被问"想尝试这套打法的人该做什么"，麦格鲁的建议是一句话——不要轻易尝试（don't try this at home）："它可能对你不利，你很可能最后做成服务业务。只有当真的努力不做却失败了，那么，也许它对你来说真的是一条护城河——如果它是你市场上唯一行得通的路径。" [3]

这句话对个人命题的预警意义比对公司更高：**个人做 FDE 的默认结局，是变成一人服务商**——按人天卖自己、承担无限责任、没有产品杠杆、没有复用资产。

但注意这句话的结构：它没有说"不可能"，它说"默认结局不好，除非满足特定条件"。

条件是什么？就是接下我们要讲的。

三、四种模式：条件表

模式	成立前提	不应混同为
分数型 FDE / 顾问（按固定节奏投入，如每周两天）	客户有明确决策人、系统边界有限、客户侧有维护负责人、目标可量化	全天候结果兜底
固定范围评估 / 上线就绪审计	交付物、数据访问、责任边界在合同写明	完整交付项目
垂直 FDE 工作室	少数客户交付中沉淀出可复用资产（连接器/评测/模板）	按人天驻场外包
FDE 引领的产品创业	先以少量设计伙伴验证工作流，再把重复部分抽成产品	无限定制项目堆积

四种模式按责任从轻到重排列：前两种**不承担生产责任**——交付物是报告或原型，交完即止；后两种承担部分生产责任，但用资产（复用库）或组织（合伙人）对冲了单人极限。

每一行最值钱的是右列——**不该混同的东西，正是个人接单翻车的标准姿势**：谈的是模式一的价格，干成了兜底的活；签的是评估合同，做着做着客户说"顺手也把上线管了吧"。模式的边界要在第一封邮件里说清，不能在第三个月才发现被混同了。

用条件表回头看开篇那位实践者：三年接单+自建案例库+可复用模块——**资产项已具备模式三（垂直工作室）的雏形**，这正是他自己说的"攒基础建设"；卡点也清楚：背书（客户凭什么信一个人）与渠道（活从哪来）——所以他放低身段、找伙伴、去社区发帖。 [1] 他的路径直觉与条件表完全吻合：一个人扛不住全部责任，就用资产复用+合作网络把责任拆出去。一个社区样本不构成结论，但它真实存在：模式表不是理论，是有人正走在这条路上的地图。

[1] 在武汉做 FDE，寻伙伴（linux.do 帖）

[3] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[4] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

Ventus AI 是一家 10 人规模、A 轮阶段的公司，联合创始人陆骁鹏本人就是 FDE。他描述的早期形态，几乎是模式三（垂直工作室）与模式四（产品创业）的混合体：FDE 从售前跟到售后，直接背商业指标——"帮客户省两百万还是多赚两百万"，这是本章"八项责任挂在一人身上"论点的具名版本。更值得记的是他的驻场策略：Ventus 的 FDE 基本不驻场，原因是"很清楚给客户提供什么、能收多少钱"；他区分了驻场的两种动机——好的动机是给大客户安全感、有明确交付目标；坏的动机是"我也不知道给你做什么，但你是大客户，先占坑"——后者他直接判定为不健康，产生不了价值 [4] 。本章开头那位社区里的从业者证明了条件表描述的路径确实有人在走，Ventus AI 证明了这条路径走通之后长什么样——一个能报出具体数字、具体客户关系、具体商业模式的对照组。

四、哪些客户是你的：市场分层

选客户先看客户体量。 [5] 大型企业（数百人以上、数亿营收）是平台大厂千万级单子的地盘，中小团队顶多做配套培训；微型企业预算只够付一次培训或咨询；**最适合个人切入的是中型企业**——几十到几百人、营收千万级起：决策链短（老板能拍板）、单个问题价值高（他自报的口径：单个问题值一千万，收一百万成立）。选择框架四象限：价值高低×决策快慢——**取"高价值×快决策"那一格**，其余三格要么养不起你，要么拖死你（见图 27–1）。 [5]

图 27–1：个人的优先切入点是高价值、快决策的客户，而不是名气最大的客户。

五、审计：个人的第一个可售卖物

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[6] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

如果四象限里选中了客户，第一件卖什么？一份从业者给了具体到动作的答案：**流程审计**。做法：争取坐到客户公司里干活最重复的那个人旁边，观察一个小时；按第 6 章的路数（记录追问→过滤→量化）产出一份审计报告——这家公司某个环节的真实流程、哪些步骤适合交给 AI、经济价值多少。 [6]

审计是**企业真正愿意付费**的服务；它是**任何项目开发前的第一阶段，也是最大的瓶颈**；报告本身就是你的展示物——申请 FDE 职位或向企业推销业务时，它证明你见过真实流程。 [6] 对照条件表：审计就是模式二（固定范围评估）的标准产品形态——**不需要生产责任，天然规避八项责任里单人扛不住的那几项**。个人入门的完整路径由此成形：审计（见真实流程）→窄原型（模式二加深）→加入合作网络（第八节）或沉淀为工作室（模式三）。

六、接单前的止步清单

反过来，什么单不能接。六个预警信号，中两条以上，要么不接、要么改结构（改固定范围、拉合作方分责）：

其一，客户没有明确决策人。 谈了三个月还在"我们内部再对齐一下"——付费能力和拍板能力都不存在。

其二，系统在关键生产链路上且要求不间断。 7×24 的责任与单人的现实直接冲突（第一节）。

其三，要你独自承担安全合规的签字。 签字是组织行为，个人签字等于个人兜底——无限责任。

其四，客户没有内部维护负责人。 上线之后的维护没有对接人，意味着这个人默认是你，免费且永久。

其五，"先做着看"的无边界需求。 范围都不存在的项目，价格没有意义（第 20 章）。

其六，驻场邀约说不清目标。 好的驻场是客户已经想清楚要交付什么、给大客户一份安全感；坏的驻场是"我也不知道给你做什么，但你是大客户，先占个坑" [4] ——分不清是哪一种，先问清楚再接。

七、兼职：算不算、卖的是哪一种

顺带说清兼职：**兼职做原型与评估，可行**（模式一二本就限边界限周期）；**兼职做生产交付，对客户不负责任**——这不是保守主义，是责任结构的推论：线上支持与兼职在时间上不兼容，客户买生产交付时买的就包括"出事时你在"。想清楚卖的是哪一种，要在谈价格之前。

八、不单干的三条出路

[4] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

[6] FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

最后，不想单干的人的结构化出路。

其一，加入合作网络。 第 26 章那位从业者的体系就是活样本：核心团队掌握获客与验收，外部顾问按项目合作。 [5] 对个人，这解决了单干最致命的两缺——背书（网络的品牌替你背）与渠道（单子从网络来）；代价是让渡一部分定价权。

其二，加入建制。 巨头正在建数千人规模的交付组织（第 26 章埃森哲×微软形态）——个体加入建制，杠杆大于单干：同样的能力，踩在别人的平台、品牌与回流机制上。 [7]

其三，把工作室做垂直。 锚定一个行业，用三到五个客户把流程做穿（第 21 章），让产品产生复利。

单干也好，建制也好，合作也好——这条路走上五年、十年，通向哪里？最后一章，终局。

---

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[7] 埃森哲推出微软前线部署工程实践

第 28 章 · FDE 的终局是什么?

> 君子不器。
>
> —— 《论语·为政》

五年后回头看，一位FDE的工作清单里，总有一部分已经不值钱了：某个框架的熟练度、某套提示词的手感、某款产品的操作路径。工具更新得太快，昨天的新技术，明天可能只是默认功能。

但另一些东西不会随版本一起过期：知道该评什么、怎样验收、谁该为结果负责；知道一个方案在演示里能跑，和在客户现场能活，是两回事。

也许到那时，"FDE" 这个头衔本身已经不常见了。它可能被拆进产品、工程、行业解决方案或交付团队，换成另一个更时髦的名字。

但客户系统出问题时，仍然会有人要回答：这个方案到底能不能跑到结果，跑不起来谁负责?

所以，这个职业会不会消失，不是终局的第一个问题。先问两件事：你今天积累的经验，五年后还值不值钱？而那个把新技术扛到生产结果的责任，届时又会由谁承担？本章从个人、角色和产业三层回答这两个问题。

一、个人层：经验的资产负债表

先答最切身的：干这行攒下的经验，五年后值钱吗？把工作所得列成三栏。

贬值项：具体框架的使用技巧、提示词的手感、某个产品的操作细节——工具层的经验折旧最快。模型、工具和工作流每迭代一轮，这一栏都会被重新估值。

保值项：评测方法、验收设计、责任管理、组织推动力——元能力层，跨行业复利，且随案例积累增值。

增值项：行业纵深＋可复用资产——连接器、模板、评测集——这一栏的特殊之处在于它能**交易**：能跟着人走，成为你的市场价；也能留在组织里，变成平台的一部分和你的议价权。

职业建议一句话：**定期把贬值项换仓成保值项**——每察觉自己在熟练使用某个即将过时的东西，就问一句：它背后有什么我该抽出来的方法。换仓勤快的人，保质期无限延长。

二、三条路径与各自的门槛

这些资产不会只留在简历上。随着经验加深，它们会把人带向三个不同的位置：把交付经验变成团队能力，变成产品能力，或变成一家公司的持续交付能力。

管理线：把经验变成团队能力。 从交付负责人走向更大的管理岗位，真正的转折不是带更多项目，而是从"自己扛结果"变成"让一套机制持续扛结果"。能否培养人、安排责任、处理冲突，决定了这条路的上限。

产品线：把经验变成产品能力。 现场反复出现的问题、评测结构和交付模板，都可以沉淀进平台。Retool 的 FDE 团队曾孵化出十多家初创公司，正说明一线经历可能成为产品和创业的起点。[1] 这条路的门槛，是辨认什么值得标准化，以及愿意从解决单个客户的问题，转向解决一类客户的问题。

创业线：把经验变成持续交付能力。 垂直工作室和产品创业的难处，不是接到第一单，而是把获客、方案、交付、支持和产品化组织成一个能反复运转的系统。行业的人工智能需求刚从尝鲜走向真实预算时，这套能力尤其稀缺；太早，需要教育市场；太晚，优势又会被同质化吞掉。

这也是端到端现场经验会成为创业者训练场的原因：它迫使人同时理解客户为什么付钱、方案怎样落地，以及交付之后谁来承担结果。

前 Palantir 员工纳比尔·库雷希观察到，尽管 Google 的员工数量约是 Palantir 的 50 倍，YC 每一批项目中前 Palantir 员工创办公司的数量通常反而更多。[2] 这个比较无法分开训练效应和人群自选择，却提示了一点：习惯在现场从头扛到尾的人，更容易看见可以独立成立的问题和生意。

三、头衔会变，责任留下

这个问题没有共识——两位一线从业者给出的是相反预测。钟钱杰（Cresta）认为 FDE 会是很长一段时间的主干："一个公司有好的销售，销售也很难被替代掉"；FDE 属于"非常接地气、有工程能力、又能用好 AI 的人"，这部分也很难被 AI 化。陆骁鹏（Ventus AI）则认为 FDE 会越来越往客户成功方向偏："对技术的要求越来越低了，如何 out-compete 别的公司？在于提供情绪价值、更好的服务、更好的理解需求"[3]。两种预测都来自身处一线的人，却指向两个方向——这恰好印证了本章开篇的立场：对终局的回答不该只是预言，该盯的是能被现实检验的信号（见图 28-1）。

[1] @jamiecuffe 关于 Retool 孵化产品线的自述

[2] Reflections on Palantir（Nabeel Qureshi）

[3] 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

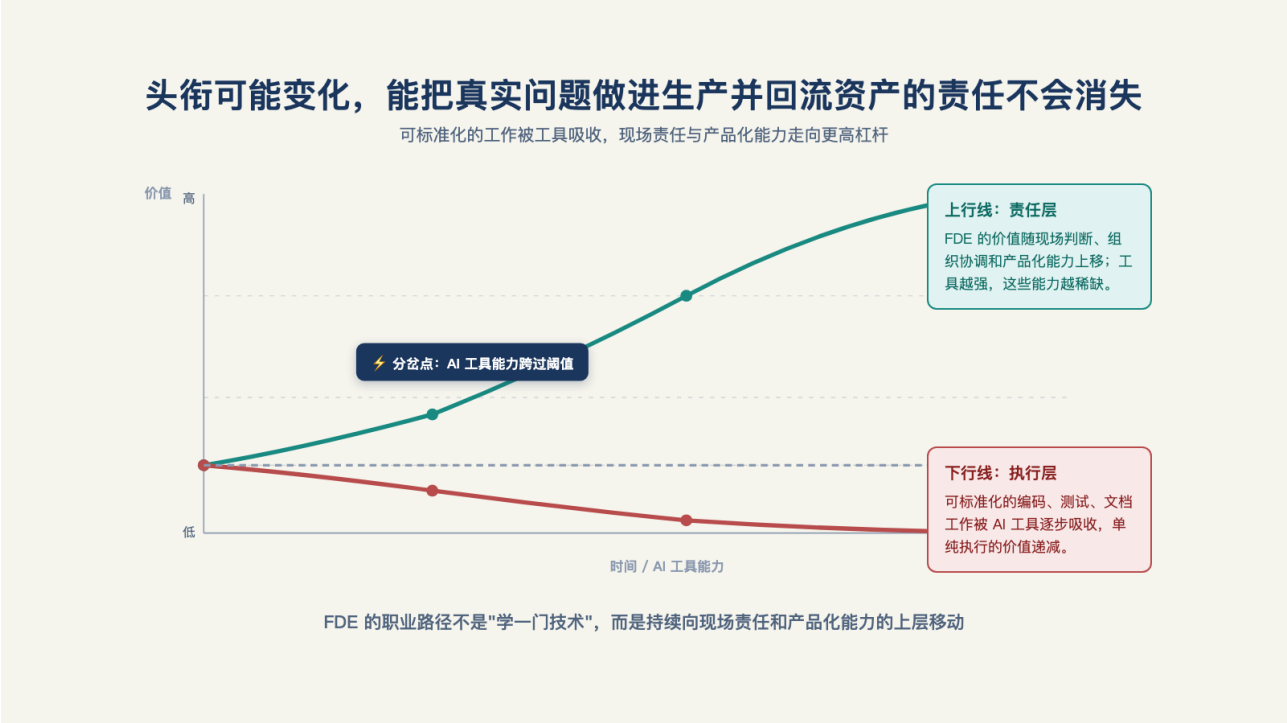


图 28–1：头衔可能变化，能把真实问题做进生产并回流资产的责任不会消失。

FDE 这个头衔五年后还在吗？未必。但这不等于这类工作消失，而是工作正在被重新分配。

一部分执行会被人工智能和平台接走。Palantir 已将部署、配置和问题定位写入 AI FDE 的产品能力；这只是产品口径，却清楚指向一个方向：重复、标准的执行会越来越少地依赖人工完成。 [4]

另一部分会被产品化接走。某类问题一再出现，解决方法逐渐稳定，它就会从一次次现场交付，变成平台上的默认能力。

两端吸收完，中间剩下什么？**判断与担保**——对"这个方案在该客户的生产环境里成不成"的判断，和"成了我负责、不成我兜着"的担保。B 端采购的永恒需求不是代码，是**风险转移**：客户买的从来是"有人替我把这个新技术扛到结果"。这个内核不被吸收——AI 能替执行，不能替客户承担对结果的责任。

四、技术越强，前线为什么还在

把视野放到整个产业：模型公司、应用公司和企业客户之间存在同一种落差——上游拿到的能力超前，下游真正能用上的部分滞后。填平这段落差，需要把新能力放进具体流程、权限和组织里；这不是待消除的缺陷，而是持续出现的商业空间。

一位中国从业者看到的是旧流程的阻力：老板容易高估 AI 无所不能的一面，却低估它被既有制度和协作方式困住的一面。他预期市场最后会收敛为两种形态——标准化产品，或标准化服务。 [5]

[4] Palantir Foundry「AI FDE」产品文档

[5] 破局点 TURNING POINT 第 31 期：中国式 FDE（小宇宙播客）

[6] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

另一位美国实践者看到的是能力与采用之间的时差。模型一代代升级，人们实际用起来的速度却未必跟上；人工智能需要有人探索、推动，也需要有人承受落地时的摩擦。 [6]

两种观察并不矛盾。前者解释为什么新技术会被旧组织拖住，后者解释为什么能力再强也不会自己发生。模型公司像总部的产品团队，应用公司和客户现场则不断把新能力变成可用的东西；每一层都在为下一层承担前线工作。 [6]

这个节奏并非人工智能独有。经济史学者卡萝塔·佩蕾丝描述过技术革命从少数先行者的建设、验证，走向大范围扩散的过程。 [7] 能力与采用之间的最后一公里会长期存在，只是不断上移：今天在企业现场，明天可能在行业流程和组织协作之中。

五、三类读者现在怎么做

对终局的回答不该只是预言。下面给出可跟踪的动作和信号：如果判断错了，读者能及早发现。

个人：往资产负债表的哪一栏投资——少背贬值项，多攒保值项与增值项；三年一次换仓检查。

公司：先确认三件事：有没有明确的业务问题，业务方是否愿意投入人、数据和时间，是否已经跑过一次小范围尝试。三件事不齐，先买小服务，不急着设岗位。

行业观察者：盯三个信号：AI FDE 类产品的自动化边界扩张多快，按结果计费在企业合同中的比例是否上升，以及"FDE"头衔是否消失而职能仍以别的名字存续。它们会告诉你，这一章的判断正在怎样兑现或失效。

六、最后

答案可以一句话说尽：**AI 项目卡住，往往不是因为做不出，而是没有人为最后一公里负责。** 模型在云上、能力在增长，真正的难处在"用得上、用得久、算得清"。

FDE 是这个责任此刻的名字。名字会变——会变成产品能力、变成平台职能、变成别的头衔；责任不会变——只要企业采购的仍是"把新技术扛到生产结果"这件事，就总要有一个角色对它端到端负责。这本书教的是这个责任的全部技术细节；而读完这本书的你，无论站在交付、产品还是组织的位置上，都已经在用同一套视角看这个问题了。

路铺到这里。剩下的，在现场自己探索发现吧。

[6] Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

[7] 卡萝塔·佩蕾丝《技术革命与金融资本：泡沫与黄金时代的动力学》

附录 A · FDE 交付检查清单

使用说明

- 每项标注： 已满足 / 部分 / 未满足 / N/A 不适用
- 原则：任一阶段的"关键项"（标 ★）未满足，不应进入下一阶段
- **后果说明**：不满足时的常见结局（红线风险）
- **正文对应**：已标注章节，便于查阅细节

阶段一：发现（对应第 6 章）

阶段目标：从客户工作流中定位高频、高价值且可验证的问题；先定义基线和成功指标。

关键项（★）

- **★ 问题定义清晰**
- 要素：一句话问题 + 当前基线数据 + 目标指标
- 正文参考：第 6 章
- 不满足的后果：
- 问题定义模糊 → 后续验收永远对不上账
- 没有基线数据 → 上线后无法证明"比原来好"
- 没有目标 → 项目永远"做得还不错，再试试"
- 检查清单：[ ] 一句话清晰定义 [ ] 基线数字有出处 [ ] 目标值与时间窗口已冻结

- **★ 基线已测量**
- 要素：人工流程的耗时/错误率/成本有具体数据（不是估算）
- 正文参考：第 6 章
- 不满足的后果：
- 只有定性评价（"很慢""很多错"）→ 上线对比无法量化
- 测试样本太小或非代表性 → 数据不可信，客户不认可
- 测量方法一致性不确保 → 上线前后用不同尺度衡量，数字爆炸
- 检查清单：[ ] 样本量≥30 [ ] 测量方法有文档 [ ] 至少重复测过一轮

- **★ 用户已尝试过可交互的原型并给出真实反馈**

- 要素：用真实数据、真实用户完成一次端到端交互，收集真实反馈（不只是"点了点头"）
- 正文参考：第 6 章、第 7 章
- 不满足的后果：
- 原型只在演示数据上跑过 → 上线后发现数据分布、格式完全不同
- 用户点了头但没真实使用 → 需求理解停留在客户的一句话
- 没有拿到真实反馈（客户怎么用、怎么吐槽） → 第二版设计又是猜测
- 检查清单：[ ] 用了真实数据（分布、量级都真实） [ ] ≥1位真实一线用户尝试过 [ ] 记录了用户反馈/疑问/困惑

非关键项（可选但推荐）

- **数据可达性清单完成**
- 内容：数据在哪/谁拥有/谁能看/脱敏规则/例外流程
- 正文参考：第 10 章
- 早做的好处：发现数据缺失/权限缺失/法律问题趁早补救

- **变更容忍度确认**
- 内容：流程所有者已知、受影响人群已识别、变更会引起的冲击范围已算清
- 正文参考：第 15 章
- 早做的好处：避免"上线完美、用户崩溃"

阶段二：最小可行部署 MVD（对应第 7、12 章）

阶段目标：用真实数据、真实用户快速验证方案是否可行；用最小范围确保验收成本可控。

关键项（★）

- **★ 真实数据（分布真实，非演示数据）**
- 要素：数据量级、分布、格式都与生产环境一致；不能是"我们选最好的 100 条"
- 正文参考：第 7 章
- 不满足的后果：
- 演示数据分布简单 → PoC 效果 95%，上线效果 65%，被客户投诉"缩水"
- 演示数据量太小 → PoC 秒级响应，上线百级延迟
- 演示没有边界情况 → 某类数据一进去系统就崩
- 检查清单：[ ] 数据量级代表生产环境 [ ] 包含≥5%的边界情况 [ ] 来源可追溯

- **★ 真实用户（至少一个一线用户每周真实使用）**

- 要素：不是"老板看了一眼"，是实际干活的人、每周都用、能给反馈
- 正文参考：第 7 章
- 不满足的后果：
- 只有管理层同意 → 最后执行人不配合，功能设计完全脱离使用场景
- 用户不是真实干活的人 → 没发现工作流的真实堵点
- 用户不经常用 → 间断使用发现不了系统稳定性问题
- 检查清单：[ ] 用户列表有名字/部门 [ ] 用户是实际操作者（不是采购方） [ ] 有周使用记录

- **★ 基准样本准备（代表性例子≥50条，有指定判定人）**
- 要素：收集50条代表一线真实场景的例子（"这个问题应该这样解决"）；明确指定一个人负责判定这些例子是否正确
- 正文参考：第 12 章
- 不满足的后果：
- 样本太少（<20条） → 无法代表真实多样性，评测结果无效
- 没有判定人 → 样本标注纠纷无法解决，做完了还得返工
- 样本有偏向 → 只有客户最关心的场景，遗漏了80%的实际问题
- 检查清单：[ ] 样本数≥50 [ ] 样本覆盖≥3个主要场景 [ ] 指定一个判定人 [ ] 样本与最终评测集的版本管理有记录

非关键项（可选）

- **范围界限明确**
- 内容：清晰划定边界——只做1条流程、1个用户群、Agent只做可测段
- 正文参考：第 7 章
- 推荐早做：避免最小可行部署变成"试试所有功能"

阶段三：生产化（对应第 8–14 章）

> **阶段目标**：把 PoC 补齐成能在真实企业环境中独立运行的系统。对应第9章的"八堵墙"逐堵拆除。

关键项（★）— 必须拆的墙

地基性（不通则什么都别谈）

- **★ 第一堵墙：数据与权限 → 数据契约签署**
- 内容：范围/可见性/留存/审计 四项一张表

- 要素：哪些表能读、结果谁能看、脱敏规则怎么做、异常权限走什么流程
- 正文参考：第 9 章、第 10 章
- 不满足的后果：
- 上线后才发现某表无权读 → 流程卡住
- 没有脱敏约定 → 客户投诉泄露敏感信息
- 结果可见范围没定 → 有人看不到，项目失败
- 检查清单：[ ] 一张表列出所有要读的数据源 [ ] 每个源标注权限方 [ ] 脱敏规则附表 [ ] 审计日志方案确认

- **★ 第二堵墙：系统集成 → 打通新旧系统的集成路径**
- 内容：Agent 要写回的那个系统怎么接（API/数据库/RPA/其他）
- 要素：接口协议确认、数据格式映射、失败重试策略、幂等性保证
- 正文参考：第 9 章、第 10 章
- 不满足的后果：
- 没事先验证 API 可用性 → 上线后发现接口从未真正能用
- 没做失败重试 → 偶发网络抖动导致数据丢失
- 没做幂等性检查 → 重试后客户数据被重复处理
- 检查清单：[ ] 集成方案已做 PoC（真实连接过一次） [ ] 网络异常、timeout、返回格式错误都测过 [ ] 幂等性设计文档 [ ] 失败回滚方案明确

风险性（能跑但随时出事）

- **★ 第三堵墙：工具边界 → 工具契约**
- 内容：Agent 能调什么工具、参数谁校验、什么情况下拒绝调用
- 要素：工具清单/参数校验规则/高风险工具的防护（如发邮件、修改数据）
- 正文参考：第 9 章、第 11 章
- 不满足的后果：
- 没有参数校验 → Agent 生成错误的邮件、删除错误的记录
- 没有工具白名单 → Agent 调用不该有权限的工具
- 没有速率限制 → Agent 给外部 API 发起 DDoS 级的请求
- 检查清单：[ ] 工具清单白名单化 [ ] 每个工具的参数校验规则写进代码 [ ] 高风险工具（写操作、外发）有审批工单 [ ] 调用速率/并发限制确认

- **★ 第四堵墙：运行可控 → 运行状态机 + 检查点与恢复**
- 内容：任务失败怎么办、卡在人工审批能否续跑、中途宕机能否断点续传
- 要素：状态流转图/检查点/恢复策略
- 正文参考：第 9 章、第 11 章
- 不满足的后果：
- 没有检查点 → 任务失败只能全部重跑，客户数据被重复处理
- 没有人工审批挂起点 → 需要人工确认的事卡住了，整条链路堵死

- 没有降级路径 → 系统故障就是业务故障，没有人工兜底
- 检查清单：[ ] 状态流转图已画 [ ] 至少 3 个关键检查点已实现 [ ] 人工审批节点能挂起并续跑 [ ] 降级到人工/Excel 的路径演练过

信任性（不通则永远停在试用）

- **★ 第五堵墙：效果可证 → 评测集 + 变更门禁**
- 内容：改了提示词/模型/工具后，效果是升是降有数据说话
- 要素：回归集/对标版本/变更前后对比
- 正文参考：第 9 章、第 12 章
- 不满足的后果：
- 没有回归集 → 每次改动都是赌博，一周改崩系统
- 没有对标版本 → 改进还是倒退看不出来
- 没有变更门禁 → 谁都能改，出了事无法追责
- 检查清单：[ ] 回归集≥200条 [ ] 基线版本冻结并记录 [ ] 变更前后有自动对比 [ ] 变更日志可追溯

- **★ 第六堵墙：结果可信 → 证据链**
- 内容：客户问"这个数字哪来的""这个结论为什么信"能拿出证据
- 要素：调用执行记录/原始输入/中间推理步骤/最终决策依据
- 正文参考：第 9 章、第 13 章
- 不满足的后果：
- 没有执行记录 → 客户质疑结果只能重跑，无法溯源
- 执行记录不完整 → 客户看到结果但看不到推理过程
- 没有原始数据存档 → 数据变了，无法做事后复盘
- 检查清单：[ ] 每次调用记录了输入 [ ] 中间步骤（检索结果、模型输出）可查 [ ] 最终决策依据可展示给客户 [ ] 日志存储周期明确

- **★ 第七堵墙：成本与稳定 → 成本归因 + 预算熔断**
- 内容：账单怎么算、超了怎么办、系统怎么保证服务水平目标 SLO
- 要素：每次调用的成本分摊/成本预警/超预算自动降级
- 正文参考：第 9 章、第 14 章
- 不满足的后果：
- 没有成本分摊 → 月底账单爆炸，客户怨声载道
- 没有预警 → 等发现的时候已经超预算数倍
- 没有自动降级 → 成本高时还在跑高耗能的调用，恶性循环
- 检查清单：[ ] 成本模型文档 [ ] 每 1000 调用的平均成本已测 [ ] 预算预警阈值设定 [ ] 超预算时自动降级策略已实现

- **★ 第八堵墙：责任可移交 → 责任交接表 + 运行手册**
- 内容：出了事谁负责、撤场后系统交给谁

- 要素：事故分类/第一责任人/升级路径/运行文档
- 正文参考：第 8 章、第 17 章
- 不满足的后果：
- 没有责任表 → 出事时指责无门，关系破裂
- 运行手册只在乙方 → 客户接手不了
- 没有升级路径 → 半夜告警，找不到人
- 检查清单：[ ] 责任矩阵（事故类型×责任人）已列 [ ] 升级时间与升级对象明确 [ ] 运行手册客户已评审过 [ ] 故障演练至少做过一次

非关键项（检查但可分阶段）

- **高风险动作审批工单**
- 内容：写操作前要有人批、记录 run_id/approver_id/操作详情
- 正文参考：第 11 章

- **服务水平目标 SLO 三维承诺（可用性/延迟/质量）+ 降级路径**
- 内容：服务水平协议 SLA 合同里写的指标，要用三维来定
- 正文参考：第 14 章

- **风险分级表（只读/可逆/不可逆/外发）**
- 内容：把所有操作按风险等级分类
- 正文参考：第 11 章

阶段四：上线与采用（对应第 8、15、16 章）

阶段目标：效果已证、系统已稳，从试用到生产的最后一关：客户要敢接、员工要想用。

关键项（★）

- **★ 效果验收单（指标+基线+窗口+分母口径+判定人，事前冻结）**
- 内容：五要素齐全，打印一张贴在评审室
- 要素：
- 基线值（第6章测出的人工流程水平）
- 目标值（要达到的水平）
- 测量窗口（按周/按月）
- 分母口径（算谁、不算谁）
- 判定人（一个人的名字，不是"业务方"）

- 正文参考：第 8 章
- 不满足的后果：
- 指标事前没冻结 → 上线后因"这月数据特殊"改指标，扯皮无解
- 分母口径模糊 → 客户说"没达标"，乙方说"你算错了"
- 判定人不明确 → 最后谁也说不通，项目悬而未决
- 检查清单：[ ] 五要素全部写进合同 [ ] 判定人签字 [ ] 灰度阶段的各级回滚线已量化

- **★ 运行就绪清单（告警有人看/降级可用/手册有人读过）**
- 内容：四项核查，一项都不能缺
- 要素：
- 告警有人看：告警响起时有值班表对得上
- 降级路径可用：包括退回 Excel 和人工，已演练过
- 运行手册：客户有一份、读过一遍
- 值守排期：上线后第一周谁值守、响应时间承诺白纸黑字
- 正文参考：第 8 章
- 不满足的后果：
- 告警响了没人看 → 系统故障客户先知道
- 没有降级路径 → 系统一坏就是业务停机
- 手册只在乙方 → 客户半夜故障无人接
- 值守表空白 → 谁来处理第一个告警?
- 检查清单：[ ] 值班表有覆盖 [ ] 降级流程演练过一次 [ ] 客户确认收到并读过手册 [ ] 值守承诺服务水平协议 SLA（响应时间）已签

- **★ 责任交接表（每个事故场景的第一责任人）**
- 内容：把系统可能的死法逐一列出，每种写明第一责任人和升级路径
- 示例：

出了什么事	第一责任人	升级路径
模型输出错误，造成业务损失	谁判定、谁拦截、谁赔付	1小时内到双方负责人
数据源故障/上游宕机	数据方接口人	4小时内到供应商值守
用户误操作引发故障	客户侧系统管理员	当日到联合复盘
成本超预算	平台方	周报到项目负责人

- 正文参考：第 8 章
- 不满足的后果：
- 没有责任表 → 出事时"这不是我们的问题"←→"这怎么会是我们的问题"，关系破裂
- 没有升级路径 → 半夜故障无人接，问题升级成客户投诉
- 只写了法律责任不写处置流程 → 纸上有名字，现实中没人接
- 检查清单：[ ] 列出至少 5 种可能的故障场景 [ ] 每种标注第一责任人（用人名，不用部门名） [ ] 升级时间与升级对象明确 [ ] 双方签署

非关键项（推荐但可分阶段）

- **灰度计划（分级放量 + 每级回滚线）**
- 内容：第一批 10% 用户→20%→50%→100%，每级有明确的回滚条件
- 正文参考：第 8 章

- **★ 激励三问已回答**（字典：考核变了吗/省的时间归谁/出错算谁的）
- 内容：用户为什么要用这个系统
- 正文参考：第 15 章

- **支持者网络（每团队一个 champion）**
- 内容：每个受影响的部门里，有一个人懂系统、能回答同事的问题
- 正文参考：第 15 章

- **激活度量口径（激活率/分流率，分母与窗口明确）**
- 内容：怎么算用户真正在用系统
- 正文参考：第 15 章

阶段五：移交与回流（对应第 16、17、21、22 章）

阶段目标：从 FDE 拉手交给客户；把这一次交付的经验和工具沉淀成可复用的产品能力。

关键项（★）

- **★ 三资产移交**
- 内容：三个东西客户必须接住（缺一个就还不算"移交成功"）
- 第一项：**运行手册**
- 包括：故障排查树、常见问题 Q&A、值守流程、人工回退步骤
- 客户验收标准：至少一个人能按手册处理一次实际故障
- 正文参考：第 17 章
- 第二项：**评测集 + 维护责任**
- 包括：评测集（用于后续验证）+ 定期补充的计划（谁负责）
- 客户验收标准：能自己跑评测，明白怎么加新样本
- 正文参考：第 17 章
- 第三项：**故障应对指南**
- 包括：常见故障的诊断与处置步骤，每条配例子
- 客户验收标准：见过指南里的场景，知道怎么处理
- 正文参考：第 17 章
- 不满足的后果：

- 手册没演练过 → 真出事时还是要回头找 FDE
- 客户不懂评测集怎么维护 → 系统漂移无人感知
- 没有故障应对指南 → 每次故障都是紧急求救
- 检查清单：[ ] 三份文档客户都收到并评审过 [ ] 各项都演练过一次（不是"看过"，是"做过"） [ ] 客户能独立完成最常见的 3 种故障处理

- **★ 定制请求打标（一次性/行业共性/通用）**
- 内容：这次项目的需求，哪些只是这个客户的、哪些是行业通性、哪些能沉淀成产品
- 正文参考：第 21 章、第 22 章
- 不满足的后果：
- 全当"一次性需求"处理 → 下一个客户还是从零开始
- 全当"行业共性"承诺 → 承诺太重，后续项目成本爆炸
- 没有打标 → 回流评审时不知道该沉淀什么
- 检查清单：[ ] 给全部定制需求标注了分类 [ ] 一次性需求有明确的退出日期 [ ] 通用能力的初版代码已提交

非关键项（推荐）

- **移交阶梯执行（陪伴→影子→撤出，各级有时间与退出条件）**
- 内容：第一阶段 FDE 陪伴，第二阶段 FDE 影子跟着，第三阶段客户自己干
- 正文参考：第 17 章

- **客户方操作者通过考核（不只发文档）**
- 内容：客户的对接人要实际操作过、出过问题、自己解决过
- 正文参考：第 17 章

- **回流评审至少一次（交付+产品双席）**
- 内容：项目结束后，交付侧和产品侧坐下来，梳理这次有什么能力值得沉淀
- 正文参考：第 22 章

- **定制量趋势记录（第 N 客户 vs 第 1 客户）**
- 内容：定制量是上升还是下降，是产品–市场匹配的指标
- 正文参考：第 21 章

使用建议

对 FDE 团队

- **阶段1–3**：项目启动时就对照清单评估，逐项标注 //

- **阶段4–5**：上线前两周拉上客户，逐项过堂，落实 ★ 标记的必须项
- **持续维护**：每个项目后做一次回顾，补充这份项目特有的坑

对客户采购方

- 把这份清单的"关键项"部分写进采购需求书
- 把第8章的"甲方自检清单"和这份清单的"关键项"对应，评估你们的采购流程是否天然制造"只许看、不给碰"的困境

对项目经理

- **前期评估**：用这份清单快速评估一个"PoC 已完成"的项目离生产还有多远
- **风险管理**：把这份清单转成甘特图，八堵墙的各项工程量和依赖关系清晰可见

附录 B · 术语表

本书使用的概念，按全书六部分组。每条含中文名、英文、一句话定义、首次出现的章。

一、角色与边界

词条	英文	一句话定义	首现
前线部署工程师	Forward Deployed Engineer (FDE)	被派驻到客户现场、对系统在真实业务里跑出结果负责的工程角色；为单个客户交付许多能力，而非为许多客户交付单一能力。正文统一称「前线部署工程师」，也常称「前沿部署工程师」，简称 FDE	第 1 章
责任四分法	the Four Responsibilities: Discovery / Production / Adoption / Feedback	定位交付类角色的四格坐标：问题发现、生产工程、业务采用、产品回流；说明位置，不评价高低——只承担其中一两格是常态，不是评分表。	第 1 章
Echo / Delta	Echo / Delta	一种两人组分工：Echo 判断该做什么（发现与关系），Delta 负责把它做出来（落地与工程），源自 Palantir 实践；区分的是责任，不是职级或技术栈。	第 1 章
AI FDE	AI FDE	把 FDE 的部分工作产品化的方向：自然语言指令转为平台操作、按用户既有权限执行、全程留痕、异常暂停待澄清；是 Palantir 的产品名，为其产品文档描述的能力声明，不等于独立验证的效果。	第 4 章
三个评判标准	the three tests: where the code goes, where the experience goes, how it is priced	判断一个角色是不是 FDE 的三条判据：代码的去向、经验的流向、报价方式与责任；用于替代「按岗位职责表对号入座」这种不可靠的判断方式。	第 2 章

二、发现与交付

词条	英文	一句话定义	首现
五道关	the five filters	问题发现阶段的五道筛子，逐关淘汰候选问题，最后只留一个切入点；第一关是「这个问题重要吗」，其后四关依次收窄。	第 6 章
发现三件套	the discovery deliverables	发现阶段该交出的三样东西：一张问题卡、一份数据可达性清单、一次可摸的原型交互；不是 PPT，判据是「下一周能不能直接开工」。	第 6 章
最小可行部署	Minimum Viable Deployment (MVD)	能在真实业务里跑通并被度量的最小交付，而不是又一个演示；与 MVP 的区别在「部署」二字，必须有真实用户、真实数据、真实流程。	第 7 章
三个「真」	the three "real" conditions: real users, real data, real workflow	MVD 的三个必要条件：真实用户、真实数据、真实流程；缺任何一个，做出来的仍是 Demo。	第 7 章
效果验收单	outcome acceptance sheet	上线前约定「算不算成功」的单据，五个要素缺一项就翻车；与「运行就绪清单」分工不同——这张管效果，那张管运行。	第 8 章
运行就绪清单	operational readiness checklist	上线前约定「谁来运行」的清单：告警响了谁看、值班怎么排、手册在哪；手册只存在乙方电脑里，等于没有。	第 8 章
责任交接表	responsibility handoff table	把每一种失败方式写上名字，并指定它归谁负责的表；「每个死法都有名字」是它的设计原则。	第 8 章

三、工程底座

词条	英文	一句话定义	首现
八堵墙	the eight walls	从 PoC 到生产之间的八类能力缺口：数据与权限、集成、工具边界、运行可控、效果可证、结果可信、成本与稳定、责任可移交；模型能力增强只矮化前几堵，后几堵由组织与流程决定。	第 8 章
harness	harness	模型之外把模型变成可用系统的那一层：工具、集成、文件系统、编排、监督；本书不译、沿用原文，与「模型能力」「接口」构成三层划分。	第 9 章
数据契约	Data Contract	动数据之前先写清四件事的书面约定：访问范围、结果可见性、留存与销毁、审计要求；使用者有三方——客户合规审查、FDE 自我约束、事后追责。	第 5 章

词条	英文	一句话定义	首现
工具契约	Tool Contract / ToolSpec	约定 Agent 可调用哪些动作、参数如何校验、越界如何拒绝的规格；与「数据契约」分管两侧——数据契约管读什么，工具契约管做什么。	第 9 章
运行状态机	Run State Machine	把一次 Agent 运行拆成可观察、可暂停、可恢复的状态流转；是「检查点与恢复」「审批工单」得以成立的底层结构。	第 9 章
审批工单	approval ticket (HITL)	不可逆动作在执行前生成的待办单据，由人审批后才继续；不是弹窗——弹窗即时且易被点掉，工单可追责、可隔夜。	第 11 章
检查点与恢复	Checkpoint & Resume	运行中断后从最近状态继续，而不是从头重跑；审批隔了一夜也不用从头再来，是它存在的直接理由。	第 9 章
分级放权	tiered authorization	按动作风险高低决定是否需要人工介入，而不是所有动作一律要人批；与「审批工单」配套——工单是机制，分级是策略。	第 11 章
基准样本	Golden Set	从真实案例里长出来的标准集，用于定义「什么算对」，是评测体系的第一步；区别于「回归集」——基准样本立标准，回归集防倒退。	第 12 章
回归集	Regression Set	只进不出的错题本：每次出错都补进去，此后每次变更都要重跑；「没过回归集的变更不许部署」是它的执行方式（门禁）。	第 12 章
模型裁判	LLM-as-Judge	开放性任务没有标准答案时，用一个模型按写定的评分标准打分；裁判的评分标准本身也是一次「提示词改动」，改它要和改系统走同一道门禁。	第 12 章
门禁	release gate	变更必须先通过回归集才允许部署的强制关卡，与「回归集」是机制与执行的关系。	第 9 章
评测集	eval set	基准样本、回归集与判定规则的统称，是 FDE 带得走的资产之一；可带走的是任务类型、难度分层、判定规则和失效模式，不是客户案例数据。	第 7 章
可信度三层	the three layers of trust	客户质疑时自证的三层结构：每个答案带出处、每步决策可回放、失败要分类；三层缺一层，自证就断在那一层。	第 13 章
证据引用	EvidenceRef	输出结论时一并给出所依据的原始出处，是可信度第一层的实现形式。	第 13 章
回放	Trace / Replay	把一次运行的每步决策原样重现，用于事后核查，是可信度第二层；与「留痕」的区别在于——留痕是记录，回放是可重演。	第 13 章

词条	英文	一句话定义	首现
SLO	Service Level Objective	写进合同的服务水位承诺：可用性、延迟、质量三个维度分开写；三者成本曲线不同，混成一句「系统稳定运行」等于三个都没承诺，质量水位直接来自评测集。	第 14 章
成本归因	cost attribution	把账单拆到每一次运行、每一个环节，而不是只看总额；没有归因就没有优化对象，是成本治理的前提。	第 9 章
模型路由	model routing	按任务难度把请求分派给不同档位的模型，针对「模型错配」这一病因——简单任务用旗舰模型等于开豪车送外卖。	第 14 章
预算熔断	budget circuit breaker	消耗触及阈值时自动停止或降档，防止账单失控；与「降级」的区别在于——熔断因成本触发，降级因故障触发。	第 14 章
韧性四件套	the four resilience primitives: timeout, retry, idempotency, graceful degradation	针对四类死法的四种机制：超时、重试（有限＋退避）、幂等、降级；降级的最后一级是「回退人工和表格」，敢写回退的方案才敢上线。	第 13 章

四、采用与移交

词条	英文	一句话定义	首现
激活	activation	系统真正被一线用起来、进入日常行为的那一刻；与「上线」严格区分——上线走的是流程，激活发生在行为里，「上线成功」之后平静地死去是需要警惕的情形。	第 15 章
一把手工程	executive sponsorship	跨部门的优先级、预算归属与权限裁定必须由决策层直接背书，否则推不动；不是万能口号，也存在「什么时候可以不是一把手」的边界。	第 6 章
撤场三件事	the three exit conditions	移交前必须完成的三件事：业务融合、知识治理、系统对接；「交付是下台阶，不是跳悬崖」是它的设计意图。	第 17 章
定义权	who defines the last mile	「最后一公里」的终点由谁说了算，直接决定 FDE 干的是技术活还是扯皮活；可以解释中美打法差异——硅谷叙事里由 FDE 自己定义，中国现场常常不是。	第 12 章

五、商业与产品化

词条	英文	一句话定义	首现
大账与小账	the big ledger vs the small ledger	小账算工时与调用费，大账算被替代的成本与出错的代价；成本的第一性问题是「和什么比」——与被替代的人工决策成本比，而不是与零比。	第 19 章
六个指标	the six metrics	把大账落到数字上的六项，含实施杠杆率、定制递减率等；是个人归纳的框架，本书借其口径但不作行业基准。	第 19 章
结果即服务	Result as a Service (RaaS)	供应商把一部分收费条件绑定在业务结果上，而不只交付一套系统；「不达标不收费」背后至少要先谈清效果从哪算、归给谁、等多久、谁垫成本。	第 20 章
定制递减	declining customization	同类需求在后续客户处所需的定制量必须逐次下降，否则生意不成立；批判的不是定制本身，是不可递减的定制——配置改参数、定制改代码，是它的分界线。	第 19 章
六类资产	the six reusable assets	可从现场回流的六类东西：业务本体、连接器、工具与审批模板、流程模板、评测结构与失败模式库、需求信号；回流的不是「经验」，散落在个人脑子里的经验不算沉淀。	第 22 章
三步与两道闸	three steps and two gates	让资产真正回到平台的流程：记录上下文、回到产品团队共议、请其他 FDE 校验；进口一道闸分类需求，出口一道闸给每项资产明确去处。只收件不处置，回流评审就会变成收集箱。	第 22 章

六、人与组织

词条	英文	一句话定义	首现
三大支柱	the three pillars	FDE 能力的三个方面，要的是交集而不是单科突出；权重随市场结构变——硅谷样本工程占比高，中国样本顾问与销售在前。	第 24 章
发现者与落地者	discoverer and deliverer	同一种角色的两种人：一种擅长找到该做什么，一种擅长把它做出来；与 Echo/Delta 同构，可用于讨论招聘与自查。	第 24 章
三人组	the trio	中国市场常见的最小建制：策略师＋行业顾问＋AI 全栈工程师，可以是一个人、两个人或一个团队；可视为 Echo/Delta 的中国变体，不强求编制。	第 1 章
前线部署产品经理	Forward Deployed Product Manager (FDPM)	与 FDE 配对的产品角色，一位 FDPM 通常带两到三位 FDE（Cresta 的配比）；是 Echo/Delta 的另一种现代变体。	第 26 章
土 FDE	(community coinage) independent FDE	中文社区对独立服务商形态的自称，多时间线上接单、自建案例库与可复用模块；非正式术语，英文版建议保留拼音或改写，不直译。	第 27 章
分数型	fractional	以非全职、按比例投入的方式为多家客户提供 FDE 服务；与「兼职」的区别在于责任结构，不只是工时比例。	第 27 章
市场分层四象限	the market segmentation quadrant	按价值高低×决策快慢划分客户，取「高价值×快决策」那一格（框架来自访谈嘉宾，其数字均为自报）。	第 27 章

共 50 条。

附录 C · 引用

本书的每一处引用，都指向这份表里的一个条目。正文脚注只写来源编号（如 `[c06]`），完整的出处统一记在这里——**一个来源，全书只定义一次**。

需要追溯某句话的依据时，记下正文里的编号，到本附录对应条目即可。共 38 条，即本书全部证据基础；正文没有引用到的材料不列入此表。

来源清单

1. enterprise_agent_platform（开源企业 Agent 平台工程）

- **出处**：GitHub · datagallery-lab/enterprise_agent_platform，Apache-2.0
- **出现章节**：第 9、10、11、12、13、14 章

2. Anthropic 与黑石、H&F、高盛组建企业 AI 服务合资公司

- **出处**：anthropic.com 官方公告，2026-05-04
- **出现章节**：第 4 章

3. 埃森哲推出微软前线部署工程实践

- **出处**：Accenture 官方新闻稿，2026-03-18
- **出现章节**：第 4、26、27 章

4. Palantir Foundry「AI FDE」产品文档

- **出处**：palantir.com/docs/foundry/ai-fde
- **出现章节**：第 4、5、9、10、28 章

5. 十字路口 × Rolling AI：FDE 对谈

- **出处**：小宇宙播客，2026-06-03
- **出现章节**：第 2、15、16、17、20、27 章

6. AI 职觉：别再卷 Agent 了

- **出处**：小宇宙播客，2026-06-21
- **出现章节**：第 6、10、11、15、16、18、23 章

7. 一路瞳行 EP134：从 AI 辅助到数字员工

- **出处**：小宇宙播客，2026-04-10
- **出现章节**：第 6、7、15、17、24 章

8. Baseten：我做前线部署工程师学到了什么（Het Trivedi）

- **出处**：baseten.co 博客
- **出现章节**：第 1、18、23、24 章

9. 一名前线部署工程师的一周（10 个客户，50 小时）

- **出处**：Hacker News 讨论帖
- **出现章节**：第 8、23、24 章

10. 「以为 FDE 是在客户现场独自作战」——入职两个月看到的真实

- **出处**：note.com · AI Shift，日文
- **出现章节**：第 6、10、23、26 章

11. Inside an Applied AI company（Pace 长文）

- **出处**：X（原 Twitter）长文
- **出现章节**：第 7、8、9、11、12、13、14、21、22、26 章

12. 破局点 TURNING POINT 第 31 期：中国式 FDE

- **出处**：小宇宙播客，约 2026-08
- **出现章节**：第 3、5、6、15、18、20、21、23、24、25、26、27、28 章

13. Reflections on Palantir（Nabeel Qureshi）

- **出处**：nabeelqu.co，2024-10
- **出现章节**：第 3、6、22、28 章

14. 硅谷 101 E248：和阿里瓴羊朋新宇聊中国式 FDE

- **出处**：小宇宙播客，2026-08
- **出现章节**：第 16、18、20、21、26 章

15. 腾讯研究院 AI 透镜圆桌 06：来自硅谷一线创业者的 FDE 非共识与落地指南

- **出处**：微信公众号，2026-07-08
- **出现章节**：第 2、9、12、16、18、22、24、27、28 章

16. a16z：Box CEO 谈 AI Agent 与企业为何跟不上

- **出处**：YouTube，2026-04-28
- **出现章节**：第 3、5、9、10、16 章

17. Y Combinator：The FDE Playbook for AI Startups（Bob McGrew）

- **出处**：YouTube，2025-09-08，约 50 分钟
- **出现章节**：第 1、5、6、7、19、21、22、24、26、27、28 章

18. Every engineer should do a stint in consulting（Hacker News 讨论帖）

- **出处**：Hacker News，2021
- **出现章节**：前言

19. @deployengineer（Bhaulik Patel）系列长文

- **出处**：X（原 Twitter）
- **出现章节**：第 2、3、9、17、19、26 章

20. 欧洲远程 FDE 职位说明（@afahmy_dev 转帖）

- **出处**：X（原 Twitter）
- **出现章节**：第 1 章

21. 「30 天百万美元」课程营销帖（@gregisenberg）

- **出处**：X（原 Twitter）
- **出现章节**：第 25 章

22. 在武汉做 FDE，寻伙伴（linux.do 帖）

- **出处**：linux.do 社区
- **出现章节**：第 18、27 章

23. @jamiecuffe 关于 Retool 孵化产品线的自述

- **出处**：X（原 Twitter）
- **出现章节**：第 28 章

24. 有在做 FDE 这个工作的佬吗（linux.do 帖）

- **出处**：linux.do 社区
- **出现章节**：第 1、2、3、16、18、23、24 章

25. 研发转 FDE，是升维还是转行做实施？（linux.do 帖）

- **出处**：linux.do 社区
- **出现章节**：第 25 章

26. FDE 工程师 101：弥合 AI 与业务的鸿沟（中文汇编视频）

- **出处**：小红书，未署名，本地导出
- **出现章节**：第 6、8、15、24、27 章

27. linux.do 帖 #2580046

- **出处**：linux.do 社区，2026-07-14
- **出现章节**：第 1、3 章

28. 硅谷 101 E240：聊硅谷最火新职位 FDE

- **出处**：YouTube，嘉宾为 Cresta FDE 负责人与 Invisible 企业业务 VP
- **出现章节**：第 5、6、10、11、20、23、24、26 章

29. Palantir 官方文档：Ontology 架构与权限体系说明

- **出处**：palantir.com/docs/foundry/architecture-center/ontology-system
- **出现章节**：第 21、22 章

30. The Dynamo and the Computer: An Historical Perspective on the Modern Productivity Paradox

- **出处**：Paul A. David，《American Economic Review》80(2), 1990–05, pp. 355—361
- **出现章节**：第 4 章

31. Ali Ghodsi 在斯坦福 MS&E 435 课程的客座讲座

- **出处**：课程官网材料页，Week 4（2026–04–24）
- **出现章节**：第 9 章

32. 腾讯研究院《FDE 模式行业观察与实践》

- **出处**：公开分享的中文参考材料
- **出现章节**：第 1、3 章

33. 《关于 FDE 的 100 个问题》（AI磊叔）

- **出处**：公开开源分享电子书
- **作者主页**：https://my.feishu.cn/wiki/QWQHwA9uYibmtzkZLJBccaEhnNd
- **出现章节**：第 6、7、12、15、16、18、24、25 章

34. 范冰《前线部署工程师（FDE）》（XDash 开源书）

- **出处**：开源电子书，v1.0.23，2026–08
- **出现章节**：第 6、7、8、12、15、16、17、18、19、21、25 章

35. 卡萝塔·佩蕾丝《技术革命与金融资本：泡沫与黄金时代的动力学》

- **出处**：公开出版物（经本地读书笔记转引）
- **出现章节**：第 28 章

36. MIT Project NANDA《The GenAI Divide: State of AI in Business 2025》

- **出处**：MIT Media Lab Project NANDA 研究报告，2025–07
- **出现章节**：前言

37. Forward deployed engineer is AI's hottest job as OpenAI and Google race to hire

- **出处**：The New Stack 报道
- **出现章节**：第 4 章

38. We'd Better Watch Out（书评）

- **出处**：Robert M. Solow，《New York Times Book Review》，1987-07-12，第 36 页
- **出现章节**：第 4 章