

前沿部署工程 (FDE) 企业实践蓝皮书

企业数字化转型下半场：如何通过敏捷工程
与产品部署双循环驱动降本增效

FORWARD DEPLOYED ENGINEERING
(FDE) ENTERPRISE BEST PRACTICES BLUE
BOOK

研究联合编纂

FHXQTECH / 徐智昊

2026年6月

FDE 蓝皮书

FDE (Forward Deployed Engineering, 前线部署工程) 蓝皮书——从 Palantir 到 OpenAI, FDE 模式正在重塑企业软件交付的方式。本蓝皮书系统梳理了 FDE 的本质、核心模型、组织定位、落地路径、交付机制、中国语境适配以及人才画像, 为企业落地 FDE 提供完整的参考框架。

序：为什么写这本蓝皮书

我做自媒体之后，陆续有不下 20 位企业主来找我聊 AI 赋能。他们的问题大同小异：同行已经开始用 AI 了，自己团队还不知道从哪下手；听说 AI 能提效，但试了几次都是噱头大于实效；想把 100 人的团队精简到 10 个人，但不知道怎么动手。

这些问题我太熟了。我在汉得信息做过 IT 咨询的现场交付，后来又去了几家 SaaS 公司做产品。两种模式的利弊我都亲身经历过。在前司，我主导过一次 AI Native 改造，做的事本质上就是把工程师推到业务前线，和业务人员一起发现问题、解决问题、把经验沉淀下来。

后来有几个大厂的产品经理和出来单干的运营、渠道朋友跟我提到了一个概念——FDE（Forward Deployed Engineering，前线部署工程）。我去研究了 Palantir、OpenAI、Anthropic 这些公司的做法，发现 FDE 和我以前做的工作性质高度吻合，和我帮企业做 AI 改造的思路也几乎一样。只不过我一直在凭经验做事，而 FDE 把这套做法总结成了一个有名字、有框架、有方法论的东西。

于是我决定写这本蓝皮书。把我自己的经验、这段时间对 FDE 的研究、以及对中国企业实际环境的理解整合到一起，做一份系统性的科普。让更多企业主了解 AI 赋能业务的本质到底是什么——不是买一个工具、上一个系统，而是把懂技术的人推到业务前线，和业务一起解决问题，同时把解决问题的经验变成可复用的能力。

这就是这本蓝皮书的由来。

目录

- 执行摘要
- 第一章 FDE 的本质与范式转移
- 第二章 FDE 的核心模型
- 第三章 FDE 的组织定位与边界
- 第四章 企业级落地路径
- 第五章 双循环交付与即时沉淀
- 第六章 中国企业语境下的适配与风险
- 第七章 FDE 人才画像与培育
- 附录

执行摘要

FDE 是什么

FDE (Forward Deployed Engineering, 前线部署工程) 不是一个岗位名称, 是一种运营模式。它的核心是把工程师嵌入到客户的业务前线, 和业务人员一起发现问题、设计方案、做出来、跑起来——同时把前线发现的可复用模式回流给产品和研发团队。

传统的企业 IT 交付有两种模式: IT 咨询的“每个项目单独做”和 SaaS 的“一套产品所有人用”。两种模式各有致命缺陷——前者无法规模化, 后者无法个性化。FDE 是第三条路: 在前线解决个性化问题, 同时把解决方案沉淀为平台通用能力。

三个关键判断

判断一：FDE 要求组织转型，不是增加一个团队。

在现有团队旁边加一群 FDE 行不通。FDE 和传统角色的底层逻辑冲突——FDE 要求同一个人从头到尾负责、边做边沉淀；传统模式要求流水线分工、各管一段。正确的做法是转型：让现有的 IT 咨询顾问从“需求执行者”变成“问题解决者”，让 SaaS 产研团队通过双循环模型让产品本身具备吸收个性化需求的能力。

判断二：前线-产品闭环是核心机制，没有闭环就是外包。

FDE 的交付过程有两条线在同时跑。内循环解决当前项目的具体问题，外循环把前线发现的可复用模式沉淀为平台能力。两条线并行，内循环驱动外循环，外循环减轻内循环的负担。如果外循环断了——前线发现的东西没人接收、没人标准化——FDE 就退化成驻场开发，做了一堆一次性代码，没有任何积累。

判断三：前线优先是唯一的红线。

在任何决策中——需求要不要做、功能怎么设计、资源怎么分配——业务前线的真实需求优先于产品规划的完美、优先于技术架构的优雅、优先于标准化流程的规范。踩了这条红线，FDE 要么变成接需求的工具人，要么变成只看产品路线图的闭门造车。

最小可行路径

不需要一次性搭建完整的 FDE 团队。从一个试点开始，用成果证明价值，然后裂变。

1. 选一个试点。财务（报销、付款、对账）或 HR 招聘流程。不选客服、不选产研、不选采购。
2. 从两侧拉人，配对作战。从业务部门和产研部门各拉 1-2 人，两两配对，做一个完整的业务场景。
3. 120 天完成一个项目。首次集成不超过 14 天，生产上线不超过 90 天，技术侧 120 天后撤出。
4. 裂变。第一批配对完成后，业务侧出一个 FDE 种子，技术侧出一个 FDE 种子。各自回到团队培养更多 FDE。

四个决策问题

如果你是企业主或 IT 负责人，在决定是否采用 FDE 模式之前，先回答这四个问题：

1. 你当前的交付模式是不是在崩溃？ 客户的个性化需求越来越多，标准化产品覆盖不了，每个项目都在从零开始。如果你没有这个问题，你不需要 FDE。
2. 你有没有愿意投入的业务发起人？ FDE 需要业务部门的人一起参与——提供业务上下文、配合测试、推动使用。找不到这样的业务发起人，不要开始。
3. 你的产品架构能不能吸收个性化需求？ FDE 前线发现的需求能不能回流到产品里？如果产品只能做“所有人用或不用”的二元筛选，外循环跑不起来。
4. 你愿意用业务结果考核 FDE 吗？ 如果考核指标是计费工时，FDE 会变成高级顾问。只有考核 Time-to-Value、生产采用率、客户满意度，他才会是真正的 FDE。

FDE 的本质与范式转移

过去二十年，企业软件交付出现过两种主流模式。一种是以 IT 咨询公司为代表的现场交付，一种是以 SaaS 公司为代表的产品标准化。我在两种模式里都工作过，对各自的优势和缺口有切身的体会。FDE 是我看到的第三条路——它把前两者的优点合在了一起。

1.1 两套交付范式，各自留下了什么缺口

现场交付模式

我刚毕业那会在汉得信息工作过，恰好交付部门和产品部门都待过。在 2019 年左右，汉得的交付方式还是非常典型的 IT 咨询模式。

做法是这样的：产品中心开发标准化产品，交付中心带着产品到客户现场，和客户多轮沟通后，基于标准产品做二次开发。组织上，产品中心负责收集各项目的需求，形成标准化产品；交付中心在客户现场实施和二次开发。

在现场，团队分为业务顾问和技术顾问。业务顾问把客户需求梳理出来，技术顾问根据需求实现功能，业务顾问验收后和客户确认。客户认可，功能就并入当前项目的系统主线。

这套模式有一个容易被忽略的闭环。每周，各项目的需求池——当时上百个项目——会被汇总分析。如果超过 50% 的项目出现了同一种需求，这个需求就被拉到产品中心做标准化开发，完成后反哺给所有项目组，不用再重复造轮子。

我对这套模式的判断是：前线优先做得好，但标准化有结构性瓶颈。

顾问和工程师就在客户现场，离问题近，响应快。这是现场交付真正的优势。但闭环的问题比看起来更严重。

首先是 50% 的门槛太高——只有“大家都要”的需求才会被标准化。那些在少数项目里冒出来的创新、在特殊场景里发现的机会，永远过不了 50%，永远不会变成通用能力。

其次，哪怕需求过了 50% 被提交到产品中心，标准化产品的迭代周期也很慢——大概半年才会更新一次，有些情况下甚至一年。这意味着一个需求从被识别到变成标准化功能，中间隔了至少几个月。

最后，这个标准化功能只有给到之后的新客户才用得上。已经在线上跑着的旧项目，现场工程师还是得自己开发——或者等产品中心先做一个 Demo 出来，再由现场工程师并入当前项目的分支。本质上，已交付的项目并没有真正享受到标准化的红利。

SaaS 标准化模式

我后来去了三四家互联网 SaaS 公司。它们的做法和汉得完全不同，而且到目前为止，这个流程基本没有变化。

一个产品线，所有客户用同一套软件。某个客户提出需求后，运营团队收集反馈，产品经理评估，然后做一个决策：要么所有人用这个新模块，要么所有人都不用，要么直接拒绝。产品决策的逻辑是“一荣俱荣一损俱损”——没有中间态，没有“先给这一个客户做”。

整个流程是需求调研→产品→研发→测试→上线，但这个循环不围绕单个客户转，围绕整个产品转。

我对这套模式的判断是：标准化做得好，但离前线远了。

产品统一迭代，一个改进所有客户受益。但运营团队虽然在前端接触客户，不写代码、不解决技术问题——只负责“理解所有人的问题，取出可以被并入产品的部分”。产品经理坐在办公室里做决策，看不到系统跑起来之后客户那边真正发生的事情。

同一个缺口

两种模式放在一起看，有一个有意思的地方。

现场交付解决了“离客户近”，但标准化慢，项目间割裂。SaaS 解决了“标准化”，但离客户远了，个体需求被“所有人用或不用”的逻辑过滤掉。

两条路各留下了一个缺口，而且恰好是对方解决过的那个。

有没有一种模式，能把现场交付的前线优势和 SaaS 的标准化优势合并起来？

这就是 FDE 要回答的问题。

1.2 FDE：两条路的合题

FDE 不是对前面两种模式的否定，是合题。它从现场交付模式里取了前线优先的基因，从 SaaS 模式里取了平台化标准化的基因，用一种新的组织方式把它们接到一起。

FDE 有四个核心原则。

前线优先

四个里面最重要的一个。如果只留一个，就留这个。

前线优先的意思是：组织的所有设计——人才标准、考核方式、资源分配——都围着前线转。不是产品中心定义好东西让前线去卖，是前线发现问题、解决问题，然后把发现的东西带回来。

在汉得的现场交付模式里，前线已经有了，业务顾问和技术顾问就在客户那里。但前线被拆成了两拨人：一拨理解问题，一拨解决问题。FDE 的做法是把两拨人合成一个人，或者合成一个极小的单元。一个人（或两三人的 pod）嵌入前线，从理解问题到写出方案到上线到运营，全程负责。不用翻译，不用交接，不用等需求文档。

工程师前移

如果前线优先是原则，工程师前移就是操作层面的第一步。

传统模式下，工程师坐在产品中心或研发中心，等需求文档传过来。FDE 把工程师直接推到前线。不是顾问去前线、工程师留在后方，是工程师自己就在前线。这意味着工程师需要具备传统交付模式下不具备的能力：不只写代码，还要理解业务、和客户直接对话、独立判断什么值得做什么不值得做。

产品-前线闭环

这是从 SaaS 模式里取来的基因。

现场交付模式有闭环，但太慢、门槛太高。SaaS 模式的标准化快，但不来自前线。FDE 的闭环是：前线在解决一个客户的具体问题时，就在想“这个模式能不能抽象出来给下一个人用”。不需要等 50% 的项目有同样需求，不需要等一周汇总。

根据过往的经验，前线大约 70% 的工作成果可以沉淀为可复用的平台能力或模板。这个数字不是拍脑袋的——如果沉淀率太低，FDE 就退化成了“更贵的驻场开发”；如果要求 100%，前线就没有空间处理真正的特殊情况了。70% 是一个合理的区间。

迭代共创

前三个原则在时间维度上的展开。

FDE 不是”先想清楚再动手”，是边做边学。工程师嵌入前线后，和客户一起发现问题、定义问题、构建方案、上线验证、再调整。每个周期都在缩短”理解问题”和”解决问题”之间的距离。

四个原则的关系：前线优先是锚点，工程师前移是实现它的手段，产品-前线闭环保证前线经验不浪费，迭代共创是具体的工作方式。

1.3 从 Palantir 到 OpenAI：一个模式的扩散

被迫发明

FDE 最早的公开来源是 Palantir。

2000 年代中期，Palantir 做美国情报机构的生意。CIA、NSA 这些机构的数据高度机密、格式混乱、没有标准。传统软件销售模式——开发一个产品，卖给客户，客户自己配置——在这种场景下完全行不通。没人能把需求文档写清楚，因为需求本身就是混乱的。工程师不到现场，不知道数据长什么样、系统怎么运转、用户怎么工作。

Palantir 把工程师直接派到现场，和情报分析师坐在一起，边看边做。这个做法后来被命名为 Forward Deployed Engineer。

Palantir 随后把模式正式化：Delta 面向单一客户，在前线做技术驱动的价值创造；Dev 面向所有客户，负责平台能力建设。Delta 在前线发现的问题和模式，回流给 Dev 变成平台能力；Dev 的新能力，交给 Delta 在前线使用。

Palantir 不是先有了 FDE 理论再去实践的。他们碰到的是极端场景——需求没法文档化、数据没法标准化、环境没法远程访问——被逼出了这套做法。这个极端场景把两种旧范式的结构性缺陷放大到了无法忽视的程度：既需要前线优先（需求在现场），又需要平台化（每个客户的底层问题其实是相似的）。

从一家公司的做法到行业共识

Palantir 之后，FDE 没有停在一家公司。到 2025-2026 年，OpenAI、Anthropic、Vercel、Scale AI、Databricks 都在公开招聘和组织公告中使用了 FDE 这一角色标签。

OpenAI 成立了专门的部署公司，募资 40 亿美元，直接把前向部署工程师嵌入战略级客户，现场编写代码，定制化实施 AI 系统与决策工作流。Anthropic 与顶级投资机构成立部署合资企业，募资 15 亿美元，将 AI 研发力量深植于大型金融、保险及零售企业。

风投机构 a16z 的数据显示，FDE 岗位需求在 2025 年增长了超过 800%。不是一家公司在招一个特殊岗位，是一个行业在用脚投票：AI 时代，旧范式交付不了复杂系统，FDE 可以。

FDE 的核心模型

2.1 Delta + Echo：从“分开干”到“一起扛”

汉得信息的前身

在汉得的时候，现场团队分为业务顾问和技术顾问。这两个人之间的配合，其实就是 Delta 和 Echo 的雏形。

业务顾问（Echo 的前身）负责理解客户的业务需求，把客户的真实痛点翻译成技术可以执行的东西。技术顾问（Delta 的前身）负责把需求变成可运行的系统。

但这套组合有一个关键缺陷：两个人属于不同的专业线，考核标准不同，甚至利益冲突。我在现场见过太多次这种冲突——业务拿回来一个需求，看起来对技术层要求很高，短期内实现不了，或者实现成本高到客户大概率无法接受。又或者需求本身有明显的业务漏洞，上线后容易出问题。这时候冲突就来了：一边说“这东西实现不了”，另一边说“这就是客户的要求”。

这种冲突在业务顾问能力不足的时候尤其频繁。好的业务顾问会在和客户沟通时就把技术可行性和成本边界考虑进去，避免把不可能需求带回团队。但不是每个业务顾问都有这个能力。

Delta + Echo 的升级

FDE 的 Delta + Echo 双子星单元，本质上就是业务顾问 + 技术顾问的组合，但做了一个关键升级：不是两个人各管各的，是同在一个 pod 里，对同一个业务结果负责。

建设性对抗依然存在——Delta 会质疑 Echo 带回来的需求是否技术可行、成本合理；Echo 会质疑 Delta 做出来的东西是否真正解决了客户的问题。但这种对抗是被设计的，不是意外的。它的目的是防止两种极端：Delta 脱离 Echo 容易陷入技术自嗨，做出架构优雅但对业务没有价值的东西；Echo 脱离 Delta 容易变成只能做 PPT 的传统顾问。

我在汉得见过的那种冲突——“这东西实现不了” vs “客户就是要这个”——在 FDE 模型里不会被回避，但会被更早地解决。因为 Delta 和 Echo 从一开始就一起在现场，一起听客户说话，一起判断什么值得做什么不值得做。冲突发生得更早，解决得也更快。

就应该是一个人

理论上，最理想的情况是一个人同时具备 Delta 和 Echo 的能力——既懂技术又懂业务，既写代码又和客户对话。我在第一章里说的“把两拨人合成一个人”，就是这个意思。

有人可能会说，同时具备顶级工程能力和顶级业务理解力的人太稀缺了，双人组合更现实。但我的判断是：就应该往一个人去培养。原因很简单——哪怕技术顾问在现场，他也一直在忙于开发，根本没有时间去听业务顾问和客户的沟通内容。业务顾问在和客户聊需求的时候，技术顾问在写代码；技术顾问在写代码的时候，业务顾问在和客户确认方案。两个人虽然在同一个项目上，但信息是割裂的。

FDE 要解决的就是这个割裂。一个人既听了客户说什么，又自己动手把东西做出来，中间不需要任何翻译和交接。这才是真正的“前线优先”。

当然，现实中 Delta + Echo 的双人组合可以作为过渡形态存在。但方向必须是合一的——不是“两个人各管一段然后拼起来”，是“两个人都在往同一个全栈方向成长”。

2.2 前线-产品闭环飞轮

闭环是怎么闭环的

第一章讲过，汉得的闭环是“每周汇总需求池，50% 门槛标准化”，周期慢、门槛高、旧项目享受不到。FDE 的闭环机制完全不同。

具体分四步：

第一步：前线构建解决方案。 Delta 在客户现场解决一个具体的业务问题时，不是只做一个一次性的定制开发。他在做的过程中就在想：“这个问题其他客户会不会也遇到？这个解法能不能抽象成一个通用的模式？”

第二步：定期 review。 产品和平台团队定期 review 前线的解决方案，识别其中可复用的模式。这个“定期”不是汉得的“每周汇总”，而是一个持续的过程——每个解决方案完成后都会被审视。

第三步：沉淀。 约 70% 的前线解决方案会被重构为平台功能或标准模块。这个重构不是前线工程师自己做的——有专门的 Platform Engineer 负责把前线“粗糙但管用”的方案变成“通用且可靠”的平台能力。

第四步：反哺。 新的平台能力反向赋能未来客户的落地，进一步减轻单个 FDE 项目的负担。下一个客户遇到类似问题时，不需要从零开始，直接调用平台能力就行。

一个真实的闭环案例

我在汉得做的是报销系统。每个客户项目都需要做移动端，但每个客户的报销政策和用户填写内容都不一样，所以每个项目的移动端都是单独开发、单独设计的。

做了两三个项目之后，我发现了一个关键点：虽然每个客户的报销政策不同，但移动端的信息采集方式是可以被标准化的——字段可以配置，流程可以配置，展示方式可以配置。

跑了三四个项目之后，我把移动端的模块做成了一个标准功能。这个标准功能可以让业务顾问直接在前端和后端完成所有配置，不需要技术人员做二次开发。

这个变化的直接效果是：业务顾问获得了类似 FDE 的能力。他们不再需要找技术顾问写代码，自己就能把客户的移动端需求交付掉。

这个案例也解释了一条演进路径：从每个项目单独开发 → 发现可标准化的部分 → 做成配置化模块 → 业务人员直接交付。2020 年之后，这条路变成了低代码构建平台；AI 时代来了之后，又进一步变成了 Agentic Coding 平台——业务顾问直接通过 AI 把要做的东西做完。逻辑是一致的：让最接近客户的人，直接产生可交付的成果。

哪些东西可以拿回产品？

不是所有前线的东西都值得沉淀。大致有三类：

类型	例子	是否值得沉淀
通用模式	多个客户都需要的数据集成方式、认证流程、权限模型	是，优先级最高
行业模板	某个行业的典型工作流程、合规要求、数据模型	是，但需要抽象
纯定制需求	只有一个客户在用的特殊业务逻辑	否，留在定制层

判断标准是一个问题：这个东西会不会让下一个客户的部署更快？如果会，就值得沉淀。

怎么加快闭环的速度？

闭环的最大敌人是时间差。前线发现一个模式，如果等半年才沉淀到平台，黄花菜都凉了。这里有几个我认为可行的加速手段：

把沉淀当成从第一天就启动的事。不是“做完项目再想沉淀”，是在项目过程中就持续识别可复用的模式。Platform Engineer 和 FDE 的配合不是事后交接，是并行工作。

用分层架构分离关注点。客户定制层（FDE 直接操作）→ 平台适配层（可复用组件）→ 核心平台层（通用能力）。FDE 在定制层工作，沉淀下来的东西自然往上推。这比“做完一整个项目再把一部分抽出来”快得多。

强制 handoff。每个 FDE 项目结束时，必须有一个结构化的知识转移：这个项目解决了什么问题、用了什么模式、哪些可以复用、哪些不能、为什么。不是把代码扔给平台团队就完了。

70% 是经验值，不是起点。刚开始做 FDE 的时候，沉淀率可能只有 20-30%。随着平台能力越来越完善，前线的可复用素材越来越多，沉淀率会自然上升到 70% 左右。重要的是闭环机制从第一天就存在，而不是等到沉淀率达标了再建。



2.3 FDE 与传统角色的本质区别

素材里有大量的对比表，比较 FDE 和 SE、SA、CSM、专业服务的区别。这些表格在维度上很全面，但我觉得它们没有抓住最核心的东西。

最本质的区别是一句话：FDE 是手艺人，传统角色是流水线工人。

手艺人从头到尾负责一件作品。他理解需求、设计方案、动手制作、交付成品、处理售后。他对结果负责，不是对某个工序负责。

流水线工人只负责流水线上的一环。他接收上一个环节传过来的东西，做完自己这部分，传给下一个环节。他不关心上游为什么这么做，也不关心下游拿去干什么。

映射到具体的角色上：

角色	工作方式	产出	对结果负责吗
FDE	纵向负责一个场景的全流程	可运行的生产系统 + 可复用的模式	是，端到端
解决方案架构师 (SA)	设计方案，不写代码	架构文档、评估报告	只对设计方案负责
销售工程师 (SE)	售前演示、搭建 POC	演示环境、POC	只对签单负责
客户成功经理 (CSM)	售后培训、提升采用率	培训计划、运营方案	只对续约率负责
专业服务/咨询	按 SOW 交付	项目交付件	只对合同范围负责

不是说流水线不重要——流水线的效率比手艺人高，标准化程度也好。问题是：当交付的东西足够复杂、足够定制化、足够不确定的时候，流水线的协调成本会超过它带来的效率。你需要一个人（或一个极小的团队）从头到尾看着这个东西，随时判断方向对不对、做得对不对。

这就是 FDE 存在的理由——在复杂场景下，手艺人比流水线更高效。

FDE 的组织定位与边界

3.1 汇报线决定行为模式

FDE 必须挂在产研部门

FDE 的汇报线应该挂在哪儿？我的判断是：必须挂在产研部门（工程或产品），不能挂在销售或 GTM（Go-to-Market）下面。

原因只有一个：闭环。

FDE 在前线拿到真实需求，做出客户可用的 Demo 和解决方案。这些解决方案需要回流到产品和研发团队，被内化进标准化产品。如果 FDE 挂在产研部门下面，这个闭环是自然的——前线和后方是同一个团队，同一个目标。如果 FDE 挂在一个独立的交付部门，闭环就断了。交付部门的核心考核是“按时上线、不亏钱”，没有人有动力、也没有精力去思考“这个东西能不能做成标准功能”。

我在汉得的时候，交付部门和产品中心是分开的。上百个项目在现场同时跑，每个人都在为了按时交付冲刺。每周的需求汇总和 50% 门槛的标准化机制，理论上是闭环的，但实际运行中，没有人会在现场会主动想“我能不能把这个功能做成标准化产品”。能力上谁都能做，但激励不往那个方向推。交付部门考核的是项目交付，跟产品贡献没关系。

三种汇报模型

行业里总结过三种典型的组织模型：

工程主导。FDE 向工程副总裁汇报，所有 FDE 都是正式的工程师，参与核心产品的开发流程。Palantir 和 Anthropic 用的是这个模式。好处是从根本上避免了咨询陷阱——FDE 做的所有工作最终都会回流到产品里。

产品主导。FDE 向产品副总裁汇报，和 PM 紧密协作，前线需求直接驱动产品路线图。OpenAI 和 Cohere 用的是这个模式。适合产品已经相对成熟、但定制化需求仍然很多的阶段。

GTM 主导。FDE 向首席收入官（CRO）汇报，KPI 变成计费工时和签单额。这个模式有一个可预测的结局：18 个月内，你会拥有一个咨询公司，产品会越来越腐烂。

三种模型的区别在于激励方向。挂在产研下面，FDE 的行为模式是“解决问题 + 把解法变成通用能力”。挂在 GTM 下面，行为模式变成“完成 SOW + 按时开票”。同一个工程师，放在不同的汇报线下，会做出完全不同的决策。

闭环为什么在独立交付部门里长不出来

汉得的例子不是孤例。独立交付部门的通病是：闭环依赖于“流程”而不是“结构”。

在汉得的模式下，闭环是这样运行的：每周汇总各项目需求池 → 分析共同需求 → 超过 50% 的需求提交产品中心 → 产品中心开发标准化功能 → 分发给项目组。这个流程在纸面上是完整的，但它的运转依赖几个条件同时成立：前线有人愿意主动上报、产品中心有资源快速响应、新功能能被旧项目用上。

这些条件在现实中经常不成立。前线忙于交付，没有时间和动力做结构化的知识总结。产品中心排期有限，半年才更新一次。旧项目已经在线上跑了，标准化的新功能到了也用不上。

如果把 FDE 放在产研部门里面，闭环不再依赖流程，而是依赖结构。FDE 和平台工程师是同一个团队，他们之间不需要“每周汇总”，每天都可以同步。FDE 做出的解决方案，不是“上报需求等半年”，是直接把可复用的部分交给同团队的 Platform Engineer 去重构。闭环的速度从“月”缩短到“天”。

3.2 消灭所有业务人员的边界

素材里怎么说

关于 FDE 和现有角色的关系，行业里主流的处理方式是划清边界：售前工程师（SE）负责签单前的技术演示和 POC；解决方案架构师（SA）负责售前架构设计；FDE 负责从设计到生产系统的落地；实施团队负责标准化配置和部署；客户成功（CS）负责上线后的采用和续约。每个角色管一段，各司其职。

我的判断：方向不是划界，是合一

我认为这个思路在方向上是错的。

这些角色确实在做不同的事情，这没有问题。问题在于，如果你接受“每个角色各管一段”的前提，你就在接受交接的存在。而交接正是传统交付模式效率流失的根源。

第二章里我说过，FDE 是手艺人，传统角色是流水线工人。手艺人从头到尾负责一件作品。如果把 FDE 放在流水线的环节里——“你负责从设计到落地”——你还是流水线，只不过换了一个更贵的环节。

我的判断是：方向不是给 FDE 划一块地盘，而是让所有人都往 FDE 的方向走。

现有的 SE、SA、实施、CS，这些角色的核心能力其实都指向同一件事：理解客户需求，解决客户问题。区别只在分工——你管这段，我管那段。而 FDE 要做的，恰恰是消除这种分工。

一个 SE 做完 POC 之后，为什么不继续把这个 POC 变成生产系统？SA 设计完架构，自己动手搭起来不就行了？CS 经理跟进使用情况的时候，直接改客户的部署——有什么不可以的？

做得到，但组织把他们框在了一个环节里。他们被定义在流水线的环节里，做好自己这一段就完成了。FDE 的组织设计，应该是打破这些环节，让同一个人（或者同一个极小的团队）从前到后负责。

过渡期的现实

当然，现实不可能一步到位。你不可能明天就把所有 SE、SA、CS 都变成 FDE。过渡期的做法是：

从最高价值的场景开始。不是所有场景都需要 FDE。标准化程度高的产品配置、简单的技术支持，现有的角色模型就够了。FDE 应该先放在“产品边界未定、需求高度定制、业务价值最高”的场景里。

让 FDE 成为其他角色的标杆。当 SE 看到 FDE 一个人从前做到后、交付速度比自己协调三个团队还快的时候，SE 会想“我能不能也这样”。当 CS 看到 FDE 直接在现场解决客户问题、而不是提工单等后台处理的时候，CS 会想“这个能力我也需要”。FDE 不是来抢其他角色的活，是来展示一种更好的工作方式。

逐步扩展。随着平台能力越来越完善（更多可复用的模板、更标准化的模块），对 FDE 个人能力的要求会降低。今天需要一个顶级全栈工程师才能做的事，明天一个有基本技术素养的 CS 就能通过平台完成。所有人往 FDE 方向成长的路，会越走越宽。

3.3 考核：前端交付为主，反哺为辅

FDE 考什么

FDE 的考核应该分两个维度：

第一个维度：前端业务交付结果。这是主要考核。具体指标包括：

- **Time-to-Value**：从项目启动到客户真正用起来的时间有多长
- **响应速度**：客户提出问题后，FDE 解决它的速度有多快
- **生产采用率**：做出来的东西客户到底在不在用，还是上线了又退回手动操作
- **客户满意度**：客户对 FDE 的交付质量和服务响应的直接评价

为什么前端交付是主要考核？因为这四个指标反映的是 FDE 的核心使命——在前线解决客户的实际问题。FDE 存在的意义不是写代码，是让客户用起来。

第二个维度：对后端的反哺。这是辅助考核，不是主要考核。

FDE 在前线解决问题的过程中，应该持续把发现的需求、做出的 Demo、识别的可复用模式反馈给产品和研发团队。但这个“反哺”不应该被量化成硬性 KPI，比如“产品团队采纳了多少 FDE 的需求”或“FDE 贡献了多少标准化模板”。

为什么不能把产品化率作为 FDE 的考核

这里有一个悖论。

如果用“产品团队接收了 FDE 多少需求和 Demo”来考核 FDE，产品团队就变成了 FDE 的评分人。一旦产品团队不接——可能是因为排期、优先级、或者纯粹的观点分歧——FDE 的考核就难看。这个权力关系会把 FDE 和产品推向对立：FDE 为了完成考核会拼命向产品推东西，产品为了保持自主性会本能地排斥。

更糟的是，这会扭曲 FDE 的行为。如果“产品采纳率”是考核指标，FDE 在前线解决客户问题的时候，脑子里想的不是“怎么最快最好地解决眼前的问题”，而是“这个东西能不能让产品团队接受”。前线优先变成了产品优先——方向反了。

这里要澄清一个容易误解的地方。第二章提到的 70% 沉淀率，不是我们拍脑袋定的目标，是行业里的一个观察——当 FDE 和产研部门的协作运转良好的时候，沉淀率自然会在 70% 左右。我们要做的，是让整个团队的核心努力点都放在把这个沉淀率往上推——让所有人都倾向于把产品做得更高效、更好。这个数字反映的是组织协作的健康程度，不是一个可以被压给某个人完成的 KPI。把它拆成个人指标，就把两拨人放到了对立面上了。

考核的反面：什么信号意味着 FDE 在退化

比“怎么考核”更重要的是“怎么判断 FDE 在退化”。几个红灯信号：

你开始用计费工时（Billable Hours）考核 FDE。这意味着 FDE 被当成了按天卖的人力资源，而不是解决问题的工程师。一旦考核逻辑变成“在现场待的时间越长越好”，FDE 就没有动力快速解决问题、快速沉淀、快速撤离。

FDE 向销售负责人（CRO）汇报。汇报线决定行为模式。FDE 向 CRO 汇报，很快就会变成高级售前——帮助签单才是核心任务，交付质量和产品化变成了可有可无的事。

客户侧的一次性定制代码在 90 天内没人问。如果 FDE 做完一个项目，写了一堆定制代码，但 90 天后这些代码还躺在客户仓库里没人去审视和抽象，说明闭环断了。FDE 在做一次性开发，不是在做产品化交付。

这些信号不需要复杂的考核体系来捕捉。一个简单的定期 review 就够了：FDE 最近做完的项目里，有哪些东西被平台团队拿走了？如果答案是“没有”，不需要立刻拉警报，但需要问为什么。

企业级落地路径

4.1 第一个试点怎么选

选什么

FDE 的第一个试点，应该选企业内部信息流传递的节点，不要选直接面向客户或供应商的环节。

我推荐两个方向：财务和 HR 招聘。

这两个方向有一个共同点：它们都是企业内部信息流传递的节点，不直接面向外部客户，试错成本可控，同时摩擦很大、效率肉眼可见地低。

财务是最强候选。 每个公司都有报销、付款、对账这些流程，它们跨越多个部门，涉及多套系统，里面有大量手工操作和信息搬运。FDE 嵌入进去，第一天就能看到问题在哪里，第一周就能做出一个可用的原型。这种场景特别适合 FDE——需求明确、价值可量化、技术复杂度适中。而且财务流程的优化可以直接量化成节省的时间、减少的错误，ROI 一目了然。

HR 的招聘流程是第二个优先方向。 招聘是一个信息密集的协作流程——简历筛选、面试安排、评价收集、Offer 审批，每一个环节都在不同的人、不同的系统之间传递信息。不同公司的招聘流程差异很大，有的 5 步搞定，有的 15 步起步，标准化的 HR 软件很难覆盖这种差异。但这种差异背后又有共同的模式——都是在收集信息、做判断、走审批。FDE 嵌入招聘流程，可以在保持灵活性的同时把重复性高的部分（面试排期、评价汇总、Offer 生成）自动化。

不要选什么

同样重要的是知道什么不该选。

不要选客服。 客服是高度个性化的岗位，要求即时响应，每个客户的问题都不一样，很难在短期内抽象出可复用的模式。把客服作为第一个试点，你会花大量时间处理 edge case，而不是验证 FDE 的核心价值。

不要选产研。 产研团队自己就是技术团队，不存在“业务和技术之间的信息断层”——填平这个断层恰恰是 FDE 存在的最大理由。放一个工程师到一群工程师里面，解决的问题他自己就能解决，FDE 的价值体现不出来。

不要选采购。 采购的摩擦点要么在外部（供应商），要么在合规（改不动）。有一定规模的企业，采购已经被 ERP 覆盖了，流程虽然慢，但合规要求卡死了，改不动。FDE 最擅长的是“发现痛点、快速试、快速改”——采购这两点都卡不住。

选试点的核心逻辑是：不要选外部面向的（客服）、不要选没有业务-技术断层的（产研）、不要选被合规卡死的（采购）。第一个试点的目的不是证明 FDE 无所不能，是证明它能在一个具体场景里产生可衡量的价值。做到了，后面的扩展才有说服力。

4.2 团队搭建：从两侧拉人，配对作战

第一个 FDE 从哪来

素材里有一个 90 天培养计划，前提是“配对一个资深 FDE 做带教”。但问题是：如果你连一个 FDE 都没有，谁来带？

所以第一个 FDE 团队的搭建逻辑，不是“招人→培养→上前线”，而是从现有的业务团队和产研团队各拉人出来，配对作战。这和第二章说的 Delta + Echo 双子星单元是一回事——业务侧的人负责理解需求、做 Demo、验证方案；技术侧的人负责把 Demo 变成可运行的系统。只不过在这个阶段，他们还不是“成熟的 FDE”，他们是在做 FDE 的过程中变成 FDE 的。

具体怎么做：从业务部门拉一两个人，从产研部门拉一两个人。两两配对，选一个具体的业务场景，从头做到尾。业务的人先根据对业务的理解做一个 Demo——用现有工具、现有数据，把“理想的工作流”搭出来。技术的人拿到这个 Demo，理解业务到底在做什么，然后基于产品的现有能力，判断哪些可以直接用、哪些需要二次开发、哪些做不到。两个人一起把这个工具在真实业务环境里跑通。

跑通之后，把这个成果反哺给产研团队——哪些能力是产品应该内置的、哪些配置化就能解决、哪些确实需要定制。这就是第二章讲的闭环，只不过现在是由这一对“FDE 种子”来驱动。

为什么是现在：AI 编程的前提

在继续往下说之前，有一点必须先讲清楚：业务人员能自己做工具，这件事在五年前是不可能的。是 AI 编程工具把写代码的门槛降到了业务人员够得着的地方。

第二章讲过一条 IT 咨询的演进路径：每个项目单独开发 → 标准化配置 → 低代码 → Agentic Coding。现在到了 Agentic Coding 这一步，业务侧的人才能真正“自己做”。没有这个前提，“业务侧学会自己迭代小工具”就是空谈。

这也是 FDE 模型在现阶段才成立的核心原因——不是组织理论忽然变了，是技术条件终于到了。

业务侧和技术侧各自的成长

FDE 对两侧的人有一个双向要求：业务人员要学会编程，程序员要搞明白业务。

这个过程结束后，会发生一件事：

业务侧的人学会了用 AI 工具做东西。他发现自己不需要等产研排期，通过 AI 编程工具和现有产品的组合，就能解决业务问题。他开始理解产品的能力边界，在和业务部门沟通时知道什么可以做什么做不到。他从一个“提需求的人”变成了一个“能自己动手解决问题的人”。

技术侧的人不再埋头只写代码。他亲自看到了业务是怎么运转的、痛点在哪里、为什么有些需求看起来简单但做起来复杂。他从一个“接收需求文档写代码的人”变成了一个“能判断什么值得做什么不值得做的人”。

这对参与 FDE 的产研人员来说，成长速度会明显快于留在后方的同事。他们直接面对业务，做完一个项目效果立刻看得见——这个流程从 5 天变成 1 天，那个环节从手动变成自动。而没参与 FDE 的产研人员还在排期做功能，效果要等半年才知道。这种“结果可见性”和“成长速度”的差异，会让更多产研人员主动想参与 FDE。不需要设计正式的竞争机制，让成果透明就够了。

这就是 FDE 的种子。一对配对完成一个项目后，业务侧出一个 FDE，技术侧出一个 FDE。他们各自回到自己的团队，可以进一步推广和实施——业务 FDE 在业务团队里培养更多的业务 FDE，技术 FDE 在产研团队里培养更多的技术 FDE。

然后裂变

第一批种子跑通之后，裂变就可以开始了。

第二批：从业务和产研团队再各拉两三个人，分别和第一批的 FDE 配对。这次有了带教的人，培养速度会更快。

第三批：不再需要从外部拉人了。业务 FDE 在业务团队内部识别合适的人，技术 FDE 在产研团队内部识别合适的人，自然扩展。

这个裂变模型的好处是：它不需要你一开始就有完整的 FDE 编制和预算。两三个人就能起步，用实际成果证明价值，然后逐步扩展。比“一次性招一整个 FDE 团队”风险低得多。

配套的硬性规则

团队搭建过程中，有几条硬性规则：

首次集成时间不超过 14 天。技术侧的人进入项目后，14 天内必须完成第一次和业务系统的集成。做不到，要么是项目选错了，要么是技术能力不够。

生产上线时间不超过 90 天。如果一个项目 90 天还上不了线，直接终止。拖得越久，沉没成本越高。

业务侧必须有对接人。如果业务部门没有人愿意配合，项目不启动。FDE 不是单打独斗，需要业务侧有人提供业务上下文、配合测试、推动使用。

每个项目至少产生一个核心产品改进。做不到的项目，需要复盘原因——是项目本身太定制化，还是没有去识别可复用的模式。

上线 120 天后，技术侧 FDE 撤出。注意，这里撤出的是技术侧——也就是产研团队的人。他回到产品团队，把前线的经验、发现的可复用的模式带回去用于孵化新的 FDE。而业务侧的人不会撤出，因为他就

站在原地。经过了这一个完整的项目，业务侧已经多了一个能自己迭代小工具的人。这个人不需要再等产研排期，他能用现有的产品和工具自己解决问题，同时还能把做出来的东西继续反馈给产研团队，作为可用的工具开发方案。闭环没有断——只不过从“两个人一起做”变成了“业务侧自己做、持续反哺产研”。

启动之前的一道门槛

团队搭建的倒计时不是从拉人那天开始的。在那之前，有一件事必须搞定：业务发起人。

没有业务发起人的试点，大概率会变成一场技术自嗨。业务发起人是那个愿意把团队的时间腾出来配合 FDE 的人，是那个在 FDE 做出来的东西和现有流程冲突时拍板“用新的”的人，是那个在项目遇到困难时不会撤退的人。

如果找不到这样的人，不要开始。等。等到有一个业务部门的主管主动说“我这里有问题，你们能不能来帮我看一下”，再启动。

4.3 项目执行：三步走

试点选定、团队搭好之后，FDE 进入项目执行。整个过程分三步。

第一步：跟着走一遍。 FDE 到了现场，先到业务人员的工位旁边坐下来，看他怎么工作。不是画流程图，不是写需求文档，是亲自动手跟着走一遍完整的业务流程。

第二步：第一天就交付一个能跑的原型。 挑一个最痛的点，直接动手做一个粗糙但能跑的改进。做出来就给业务人员用，用完反馈，立刻改。第一天的原型不需要完美，但必须能跑、必须能解决一个真实的痛点。

第三步：逐步深入和扩展。 原型验证方向之后，把粗糙的方案打磨到可以日常使用，把覆盖范围从一个环节扩展到整个流程。每解决一个问题就立刻让业务人员用起来，增量交付。

4.4 与需求方的协作

FDE 和业务人员的配合方式，不是“我给你做东西，你验收”。两个人（或一个极小的团队）一起发现问题、一起设计方案、一起验证效果。业务人员提供业务上下文和组织感知，FDE 提供技术落地和判断可行性。项目做完之后，业务人员应该学会用 AI 工具自己解决日常问题，技术侧 FDE 在 120 天后撤出但保持持续的通道——业务侧随时能找到技术侧，闪光点持续回流给产研团队。

4.5 沉淀机制

发现的那个瞬间就报上来

沉淀机制最关键的一步，不是做完项目之后的复盘，是在发现可复用模式的那个瞬间就报上来。

我在汉得做报销系统的时候，不是做了四五个项目之后才突然意识到“哦，移动端可以标准化”。是在做第二个项目的时候，就发现移动端的信息采集方式和第一个项目几乎一模一样——字段不同，但采集的逻辑是相同的。那个发现是在项目过程中冒出来的，不是事后复盘想出来的。

发现之后，我做了产品设计和方案输出，然后把开发工作交给标准化部门（当时的产品中心）完成。这个分工很重要：前线的人负责发现和设计，后方的人负责重构和标准化。不是前线工程师自己把标准化也做了——他还在跑项目，没那个时间。

必须跑够足够多的项目

沉淀机制有一个前提条件：FDE 必须在外部跑了足够多的交付项目，才能看到可以标准化的东西。

一个项目看不到模式。两个项目开始有感觉。三到四个项目之后，哪些是通用的、哪些是定制的，就比较清楚了。所以沉淀不是靠聪明，是靠经验积累。你必须在不同的客户、不同的场景里看到同一个问题反复出现，才能确认这是一个值得标准化的模式，而不是某一个客户的特殊需求。

这个前提也解释了为什么 FDE 不能频繁换项目。一个 FDE 在一个领域里待的时间越长，他看到的重复模式就越多，沉淀出来的东西就越有价值。频繁轮换的 FDE，每个项目都是从零开始，和传统的驻场开发没有区别。

沉淀的路径：从发现到平台化

沉淀的路径大致是：在项目中发现模式 → 立刻记录和上报 → 由平台团队负责重构 → 新能力反哺后续项目。这条路径里有几个关键点：发现是即时的不是事后的；记录不需要完美但必须抓住瞬间；重构不由前线负责而是后方 Platform Engineer 的工作；反哺速度取决于同步频率。

沉淀的具体机制、即时捕捉的方法、以及什么值得沉淀什么不值得，第五章会详细展开。

双循环交付与即时沉淀

5.1 双循环模型：交付一条线，沉淀一条线

在第四章，我们已经讲了 FDE 在项目里怎么执行、怎么和业务配合。这一章聚焦在另一个维度：两条线怎么同时跑、跑多快、怎么度量。

FDE 的交付过程有两条线在同时跑。一条线是当前项目的交付——发现问题、设计方案、做出来、跑起来。另一条线是平台能力的积累——在做项目的过程中发现可复用的模式，实时上报，产研团队即时标准化，新能力立刻供后续项目使用。

素材里把这两条线叫做“内循环”和“外循环”：

内循环：单个项目的交付闭环。FDE 进入一个具体的业务场景，做六件事：发现真实的业务问题 → 画出当前的工作流 → 设计改进方案 → 构建可运行的系统 → 部署到真实环境 → 追踪业务效果。这个循环转一圈就是一个项目的完整交付。

外循环：平台能力的持续演进。内循环过程中产生的每一个定制化方案、临时接口、应急补丁，都是潜在的“闪光点”。这些闪光点实时回流给产研团队，被抽象为平台通用能力、行业模板、或者评估框架，然后供所有后续项目直接使用。

两个循环不是串行的——不是“先做完项目，再考虑沉淀”。它们是并行的。FDE 在做项目的同时就在识别可复用的模式，产研团队在接收到前线反馈的同时就在做标准化。内循环驱动外循环，外循环减轻内循环的负担，形成飞轮。

OpenAI 的案例最能说明这种“两条线同时跑”的状态。OpenAI 的 FDE 在为某呼叫中心开发 AI 语音助理集成时，构建了一套实时评估框架来捕捉语音延迟和吞吐波动。这套框架是前线为了解决具体问题临时做的——这是内循环。但做完之后，FDE 意识到这个评估逻辑对所有语音场景都有用，于是即时上报。最终，这套前线评估框架重塑了 OpenAI 官方 Realtime API 的底层机制，同时促成了 Agent SDK 的定型——这是外循环。一个为单个客户做的临时工具，变成了平台的核心能力。如果 FDE 只做了内循环，那套评估框架就永远留在那个呼叫中心的系统里。因为有了外循环，它变成了所有人都能用的东西。

5.2 外循环：即时沉淀的机制

不是周期性的复盘，是即时的捕捉

这是我要强调的一个关键区别。

FDE 的沉淀不应该是周期性的。不应该等每周的例会才把发现的东西上报。应该是即时的：FDE 在项目中发现了一个可复用的模式——一个“闪光点”——立刻记录，立刻和团队负责人沟通，立刻交给产研团队评估。

这个过程没有固定的时间点，就像敏捷开发里不会等两周的迭代结束才修一个紧急 bug。需要沟通的时候直接沟通，该做的事直接做掉。

闪光点长什么样

闪光点不一定是宏大的架构发现。大多数时候，它是很小、很具体的东西：

- 两个客户都用同样的方式处理某种数据格式——这个数据适配逻辑可以抽成一个标准组件
- 某个审批流程在三四个项目里反复出现——这个流程可以做成配置化的模板
- 前线为了解决一个问题写了一段临时脚本，但这段脚本的逻辑是通用的——可以重构为平台的内置能力

这些闪光点在做项目的过程中会不断冒出来。关键是在冒出来的那个瞬间就抓住它，而不是指望事后回忆。

闪光点的流转路径

闪光点被捕捉到之后，流转路径简化为四步：记录 → 讨论 → 交给产研 → 供后续使用。

FDE 即时记录——几句话描述清楚发现了什么、出现在哪个项目里、目前的临时解法是什么、为什么觉得可以复用。不需要正式文档，一个即时消息就够了。然后和团队负责人快速讨论——这个发现值不值得做标准化？如果值得，优先级多高？这个讨论可能是五分钟的对话，不是正式的评审会议。讨论完交给产研团队，由他们负责把前线的临时方案重构为通用且可靠的平台能力。标准化后的模板、组件、API 放进平台，下一个 FDE 项目直接调用。每一个新项目都在前一个项目的积累上起步。

这条路径的运转速度取决于一件事：沟通是不是即时的。如果 FDE 发现了一个模式，但等到每周例会才说，这个信息就要在脑子里存一周，很可能忘掉或者记不清。如果发现了就说，五分钟讨论完，产研团队当天就能开始评估。

从闪光点到平台能力：什么值得沉淀

不是所有闪光点都值得变成平台能力。第二章讲过三类：

通用模式——多个客户都需要的数据集成方式、认证流程、权限模型。优先级最高，一旦沉淀，所有后续项目直接受益。

行业模板——某个行业的典型工作流、合规要求、数据模型。值得沉淀，但需要抽象：不能把某个客户的特殊做法直接模板化，要提炼出行业共性。

纯定制需求——只有一个客户在用的特殊业务逻辑。不值得沉淀，留在定制层。

判断标准始终是一个问题：这个东西会不会让下一个客户的部署更快？ 如果会，就值得沉淀。

5.3 作战节奏

双循环模型要持续运转，需要不同层级的节奏来支撑。这些节奏服务于不同的人、不同的决策层级，彼此不冲突——即时沉淀是前线节奏，周/月/季是治理节奏。

每日 standup

前线团队每天同步当天进展和障碍。这不是汇报会，是协同会。每个 FDE 用两三分钟说三件事：昨天做了什么、今天打算做什么、有没有卡住的地方。

standup 的核心价值不是让管理者知道进度，而是让问题在出现的当天就被看到。FDE 说“某个 API 的返回格式变了，我的集成跑不通了”——这个问题不需要等到周会，当天就有可能有人能帮忙解决。

每周 pod 评审

每周一次，pod 内部做一次更深入的评审，关注三件事：风险、依赖与采用。

风险是那些可能在接下来一周内爆掉的事——第三方系统下周要升级了、某个审批流程的负责人要离职了。依赖是 pod 外部的人或团队——产研团队那个 API 还没交付、业务部门那边培训还没排上。采用是做出来的东西有没有人真的在用——上线一周了，使用率怎么样，有没有人绕过新系统继续用老办法。

pod 评审不讨论技术细节——那是每日 standup 和即时沟通的事。它关注的是项目的整体健康度。

每月架构/安全评审

每月一次，平台团队和安全/治理团队参与。关注两件事：模式抽象与控制继承。

模式抽象是看前线的多个项目中，有没有出现值得沉淀的共性——这个月报上来的闪光点里，哪些可以变成平台通用能力？哪些可以做成行业模板？控制继承是确保前线的项目在安全、合规、数据治理上没有各自为战——新项目有没有继承平台的权限模型？有没有用统一的审计日志？

这个节奏的核心是：让前线跑得快的同时，不让平台和安全变成后续的债。

每季度 portfolio review

每季度一次，由 Head of FDE 和业务发起人共同参与。做一次全局的盘点：哪些试点 kill、哪些 scale、哪些 productize。

Kill 是终止——试了三个月，价值不明确、采用率上不去、业务方不配合，及时止损。Scale 是扩展——试点在一个部门跑通了，模式清晰，可以复制到相邻的部门或流程。Productize 是产品化——前线做出的某个能力，已经经过了多个项目的验证，可以正式进入产品的路线图。

portfolio review 是最高层级的治理节奏。它不关心单个项目的技术细节，关心的是：我们的 FDE 投入有没有产生可复用的资产？哪些方向值得继续投入？哪些方向应该退出？

分层的重要性

这四个节奏服务于不同层级，不冲突：

- 每日 **standup** 是前线节奏，FDE 之间的即时协同
- 每周 **pod 评审** 是项目节奏，关注单个 pod 的健康度
- 每月 **架构/安全评审** 是平台节奏，关注复用和治理
- 每季度 **portfolio review** 是组织节奏，关注投资回报和方向调整

即时沉淀——发现闪光点的那个瞬间就上报——发生在每日的节奏里，可能是几分钟的对话。治理——闪光点被标准化、被产品化——发生在周/月/季的节奏里，需要不同层级的人参与。两者并行，互不干扰。

5.4 关键指标

双循环模型运转得好不好，不能只凭感觉。需要几个核心指标来量化。

Time-to-Value

从启动到第一次可衡量的业务价值。

这个指标衡量的是 FDE 的交付速度。不是从启动到代码写完，而是从启动到业务人员真的感受到“这东西帮我省了时间/减少了错误/提高了效率”。

怎么算：从项目 kickoff 到第一个可量化的业务效果出现的时间。比如报销流程优化项目，从启动到“某个环节的平均处理时间从 3 天降到 1 天”的那一刻。

什么算好：第四章讲过，首次集成不超过 14 天，生产上线不超过 90 天。Time-to-Value 应该在生产上线后的两周内就能看到——如果上线两周了还没有可量化的效果，说明方案要么没切中痛点，要么用户还没真正用起来。

生产采用率

用户真的在用，不只是部署了。

部署不等于采用。很多项目上线了，但用户还是用老办法——新系统成了摆设。生产采用率衡量的是：目标用户中，有多少人在日常工作中真正使用了 FDE 做出来的东西。

怎么算：活跃用户数 / 目标用户总数。活跃的定义要严格——不是登录了算活跃，是真正用核心功能完成了至少一次业务操作。

什么算好：Morgan Stanley 与 OpenAI 的合作案例中，财富管理顾问团队的使用率超过了 98%。这是天花板级别。对一般的 FDE 项目，上线后一个月内达到 50% 是及格线，三个月内达到 80% 是目标。如果持续低于 50%，需要审视的不是推广力度，是方案本身——用户不用，多半是因为没解决真问题。

模式复用率

沉淀下来的东西被后续项目复用的比例。

这是衡量外循环是否有效的核心指标。如果前线的闪光点被沉淀了，但后续项目没有复用，说明沉淀的方向错了，或者沉淀的质量不够。

怎么算：被后续项目调用的沉淀资产数 / 总沉淀资产数。沉淀资产包括模板、组件、API、评估框架等。

什么算好：第二章提到，FDE 的目标是约 70% 的前线解决方案能被重构为平台能力。70% 沉淀率是长期目标，但模式复用率是更直接的指标——沉淀了 10 个组件，后续项目实际调用了几个？如果复用率低于 30%，说明沉淀的颗粒度不对，或者平台团队的抽象质量有问题。

反哺平台贡献

前线方案被平台化的比率。

这个指标衡量的是 FDE 是不是在“越做越轻”。如果每个项目都是从头开始定制，FDE 就退化成了传统的外包团队。反哺平台贡献越高，说明前线的工作成果越能被复用，后续项目的边际成本越低。

怎么算：进入核心产品或平台的前线方案数 / 总前线方案数。另一个角度是看“平台抽象提取率”（Abstraction Extraction Rate, AER）——被核心研发合流为主干通用原语的现场代码占总现场代码的比例。

什么算好：AER 达到 20% 以上是健康的。如果持续为零，说明 FDE 团队正在陷入纯定制开发——每个项目都从零开始，没有任何资产积累。这就是素材里说的“咨询陷阱”的预警信号。

四个指标之间的关系

这四个指标不是孤立的，它们构成一个整体：

Time-to-Value 衡量速度——FDE 交付得快不快。生产采用率衡量效果——做出来的东西有没有人用。模式复用率衡量沉淀质量——沉淀下来的东西有没有被后续项目用上。反哺平台贡献衡量飞轮效应——整个组织是不是在“越做越轻”。

如果 Time-to-Value 很快但采用率很低，说明做得快但没切中痛点。如果采用率很高但复用率很低，说明单个项目成功了但没沉淀出东西。如果四个指标都在健康区间，双循环飞轮就在真正运转。

5.5 运营 playbook

变更管理

Azure CAF 有一条很适合 FDE 的理念：变更是最常见的问题来源。

FDE 不是“探索完再交给正式 IT 上线”，而是端到端地负责交付。这意味着变更管理不是后置的审批流程，而是内生的设计要素。如果 FDE 把变更管理当作“上线前要走的手续”，就会在安全、运维和业务审批环节全部卡死。

具体怎么做：对变更设定风险分级。低风险变更——比如修改一个前端文案、调整一个阈值——不需要正式审批，FDE 直接做，做了记录就行。中风险变更——比如增加一个新的数据源、修改一个工作流的步骤——需要 pod lead 确认，可能需要和相关团队同步。高风险变更——比如涉及敏感数据的流向、影响多个部门的流程调整——需要多名高级工程师评审，渐进式推出，主动监控。

对所有生产变更，都要有验证步骤和明确的回滚策略。变更前能验证，变更后能回滚，这两点是非谈判条件。

培训与上岗

FDE 的培训不是“上三天课然后扔到项目里”，而是一个 3-4 个月的分阶段培养过程：

第一个月：平台能力。新 FDE 不能直接去见客户。先熟悉核心产品的能力边界，读完关键的架构决策记录（ADR），在沙盒环境里完成演练，能讲清楚现状架构。这一步的目的是让新 FDE 知道平台能做什么、不能做什么。

第二个月：shadowing。配对一个资深 FDE 或 Pod Lead，跟着参与一个正在进行的项目。看他怎么做 discovery，怎么和业务人员沟通，怎么做技术判断，怎么处理突发问题。完成一次流程映射、一个小型交付件。

第三个月：独立负责。独立承担一个子工作流，从设计到上线。参与一次 go-live，独立完成一个子模块的上线方案和 runbook。

第四个月及以后：沉淀与贡献。进入 pod 的主交付节奏，同时开始模板化沉淀——提交可复用的模板、文档或 playbook 更新。到这一步，FDE 不只是自己能做项目，还能把做出来的东西变成别人也能用的资产。

最小文档集

FDE 不需要写大量文档，但有几个关键工件是必须的。这不是“文档负担”，而是规模化 FDE 的最低控制面：

Discovery Brief——业务目标、现状流程、痛点量化、系统清单、数据敏感级别、成功指标草案。在项目启动前写完。

Pilot Charter——试点范围、排除范围、赞助人、pod 组成、阶段门限、停机/冻结窗口。和业务发起人一起签。

ADR (Architecture Decision Record) ——关键架构取舍、替代方案、决策者、回退条件。每个重要的技术决策记一条。

Go-live Checklist——权限验证、监控就绪、告警阈值、回滚路径、培训完成、值班安排。上线前逐项检查。

Incident Runbook——触发条件、分级、负责人、诊断命令/查询、回滚动作、沟通模板、事后复盘入口。上线后立即可用。

Handoff Pack——服务目录、SLO/SLA、依赖图、常见故障、升级路径、模板与代码仓地址。技术侧 FDE 撤出时交付。

这些工件不需要完美，但必须存在。没有 Discovery Brief 就不启动项目，没有 Go-live Checklist 就不上线，没有 Handoff Pack 就不撤出。

治理红线

有几条红线是不能碰的：

不要把 FDE 当外包。如果业务部门强行要求 FDE 开发完全无法复用的定制表单，FDE 团队必须有一票否决权。FDE 的考核是“抽象提取率”和“Time-to-Value”，不是“计费工时”。如果 KPI 挂在人天上，FDE 天然会倾向于把问题复杂化、延长驻场时间——这正是传统咨询的死穴。

不要让 KPI 错位。FDE 越快在现场用最少的定制化代码跑通业务、越快提炼出通用模式并撤离现场，其获得的激励应该越高。反过来，如果一个 FDE 在一个项目上待了六个月还在做定制开发，这不是“深耕”，这是组织设计的失败。

设立“平台收割委员会”。现场团队为了交付速度，难免写一些粗糙的临时代码。如果没有后台的审查，这些临时分支会无限制地野蛮生长。专职的平台团队每周检索一线的代码变更，发现高频出现的相似逻辑，必须在 30 天内启动抽象合流——要么重构为平台通用能力，要么下线清理。

5.6 这个模型能不能运转，取决于几件事

前线和产研之间的信任

即时沉淀的前提是信任。如果 FDE 上报一个闪光点，产研团队的回应是“这个我们已经知道了”或者“排不上”，报两次之后 FDE 就不会再报了。

避免这种情况的方法是：产研团队对每一个前线上报的闪光点都给出明确的反馈——接还是不接、为什么不接、如果接的话大概什么时候能做。哪怕是不接，也要说清楚原因。FDE 需要的不是每一个发现都被采纳，是每一个发现都被认真对待。

业务人员的 AI 编程能力

第四章讲过业务人员要“学会编程”。这个要求在 AI 时代已经不是天方夜谭，但也不是自动成立的。企业需要提供基础的工具和培训——让业务人员能用上 AI 编程工具、知道基本的调试方法、理解产品的能力边界。

不需要每个人都变成开发者。但每个人都需要能达到“遇到小问题自己能解决”的水平。

沉淀的节奏由业务驱动，不由日历驱动

最后再强调一次：整个双循环模型的运转节奏不是固定的。不是“每周必须报三个闪光点”，不是“每月必须做一个标准化”。节奏完全由业务状态决定——项目多、发现多的时候，沉淀就密集；项目少、发现少的时候，沉淀就稀疏。

强行规定节奏，只会让 FDE 为了凑数而报一些不值得标准化的东西，或者让产研团队为了完成 KPI 而把不该标准化的东西标准化。保持即时性，保持业务驱动。

中国企业语境下的适配与风险

6.1 “包工包料”的甲方预期

一个非常常见的场景

在中国企业里，甲方和乙方的关系有一个根深蒂固的模式：甲方动动嘴皮子，乙方把东西全做出来。

我在汉得的时候，见过无数次这种场景。客户提一个需求，团队花两周做出来。客户看了说这里不行那里不行，改。改完再看，又说之前那个版本其实更好。改了很多版之后，客户选择使用第一版。

这不是个案，这是中国企业甲乙双方关系里的常态。甲方的心态是“我付了钱，你就应该把我想要的东西做出来”。他们不认为需要自己参与设计、验证、迭代。他们的角色是“提需求”和“验收”，中间的过程是乙方的事。

这种预期和 FDE 的工作方式有一个根本冲突。FDE 要求甲方的人一起参与——一起发现问题、一起设计方案、一起验证效果。但甲方的预期是“你做给我看”。如何弥合这个预期差距，是 FDE 在中国企业落地的第一个挑战。

怎么处理

在合同里就把角色定义清楚。不要在口头上说“我们是一起做”，要在商务条款和项目章程里写明白：甲方需要指定业务对接人，每周投入多少时间参与方案设计和验证；FDE 的角色是“联合创新”而不是“代工交付”。如果甲方不同意这个条件，这个项目就不适合用 FDE 模式。

用“联合实验室”或“创新中心”的形式降低阻力。这是素材里一个有用的建议。与其在传统的甲乙双方合同框架里硬塞 FDE 的概念，不如换一个形式——双方成立联合团队，共同投入资源，共同对结果负责。这种形式在中国企业里更容易被接受，因为它打破了“我出钱你出力”的二元对立。

让甲方的人从第一天就动手。不解释为什么甲方需要参与，直接让甲方的人参与。FDE 做出一个原型后，把工具交给甲方的业务人员操作，让他们自己跑、自己发现问题。一旦他们开始自己用，就回不去了——他们会意识到参与比旁观更高效。

6.2 现有团队怎么转型

为什么不可能共存

第三章讨论过 FDE 和现有角色的关系，我的判断是：方向不是划清边界，是消灭边界。第六章要把这个判断在中国企业的语境下再讲透。

在中国的 IT 服务市场里，最常见的团队配置是：售前团队负责拿单，实施团队负责交付，售后团队负责维护。三个团队各管一段，交接是常态。如果在这种结构里”嵌入”FDE，结果是可以预见的——FDE 变成实施团队的一个高端版本，做的东西和传统实施没有本质区别，只是人更贵。

共存是不可能的。因为 FDE 和传统角色的底层逻辑是冲突的——FDE 要求同一个人从头到尾负责、边做边沉淀；传统模式要求流水线分工、各管一段。把这两种人放在同一个组织里，只会产生内耗：传统实施团队觉得 FDE 抢了他们的活，FDE 觉得传统实施团队拖了后腿。

转型路径

正确的做法不是共存，是转型。把现有的人变成 FDE，而不是在现有团队旁边加一群 FDE。

从 IT 咨询团队转型。IT 咨询团队最大的问题是”接需求、做需求”——用户说什么就做什么，从来不思考这个东西能不能标准化、该不该标准化。转型的方向是让他们从”需求执行者”变成”问题解决者”。不是客户说”我要一个审批流”就去做审批流，而是先问：这个审批流解决的是什么业务问题？有没有更好的解决方案？这个模式在别的客户那里出现过吗？问完这些问题之后再动手。

转型的第一步是让咨询团队里的人学会用工具自己做东西。就像第四章说的，从业务侧拉出来的人要学会用 AI 编程工具。咨询顾问如果只能写 PPT 和需求文档，他永远不可能变成 FDE。他需要能自己搭原型、自己验证想法、自己判断技术上可不可行。

从 SaaS 产研团队转型。SaaS 产研团队最大的问题是”客户要什么都不给”——因为每个需求都要经过”所有人用还是不用”的筛选，大量合理但不够通用的需求被直接拒绝。

表面上看这是人的态度问题——产品经理不愿意接定制需求。但根本原因是产品架构层面的问题：产品没有能力承载不同企业的个性化需求。一个需求如果能被”所有人用”，它才能进产品；如果不能，就被拒绝。这个二元筛选逻辑注定了大量合理的个性化需求会被丢掉。

转型的方向不是让产研的人去前线”找平衡”。是通过第五章讲的双循环模型，让产品本身具备吸收个性化需求的能力。内循环在前线发现不同企业的个性化场景，外循环把其中可复用的部分抽象为平台能力。随着外循环持续运转，产品能覆盖的场景越来越多，能配置化的部分越来越多，下一个企业的个性化需求就不需要从零开始做了。

第二章讲过报销系统的案例：移动端模块从“每个项目单独开发”变成“配置化”——字段可配、流程可配、展示可配。做完这个转变之后，业务顾问自己就能把客户的移动端需求交付掉，不需要技术人员二次开发。这就是产品架构层面的转型——不是产品功能变多了，是产品变得更灵活了，能承载更多不同的企业场景。

转型的第一步：产研团队需要建立的是平台可扩展性——配置化能力、模板化能力、API 化能力。每一条个性化需求被满足后，不是留在一个客户那里，是通过外循环变成平台能力，让下一个企业直接受益。

6.3 遗留系统与数据孤岛

现实

中国企业 IT 环境有一个普遍特征：系统多、老、杂。

一个中大型企业可能同时跑着 ERP（SAP 或用友）、OA（泛微或致远）、CRM、HR 系统、财务系统、自研业务系统，以及若干部门级工具。这些系统之间的数据通常是不通的——同一个客户的信息在三个系统里有三个版本，同一个审批流程在 OA 里走一遍、在 ERP 里又走一遍。

FDE 进入这种环境，面对的不是“给你一个干净的沙盒你可以随便搭东西”，而是“你的解决方案必须跑在一堆乱七八糟的旧系统上面”。

怎么处理

不要试图替换旧系统。 替换一个运行中的遗留系统的成本和风险远大于在它上面加一层适配。正确的做法是用“适配层”的方式——FDE 做的新东西通过 API 和旧系统对接，旧系统继续运行，新能力逐步叠加。等新能力覆盖了旧系统的核心功能，旧系统自然会被淘汰。这比“一次性推倒重来”安全得多。

数据问题先治标，再治本。 数据不通不是一天造成的，也不可能一天解决。FDE 在做项目的时候，先把眼前需要的数据打通——用脚本搬运、用接口对接、用中间表做映射。这些临时方案不优雅，但能让业务先跑起来。跑起来之后，再把数据治理作为平台团队的长期工作来做。

安全合规提前设计，不要临上线才想起来。 中国企业（尤其是国企和金融机构）对数据安全和合规的要求很严格。FDE 在做 discovery 的时候，就要把数据分级、权限模型、审计要求纳入考量。等到东西做完了才去过安全评审，大概率会被一票否决，前面的工作全部白费。

6.4 最典型的失败与红线

两种失败，一个根源

我在两种模式里都工作过，见过两种截然不同但同样致命的失败：

IT 咨询的失败：FDE 变成了现场外包。这是汉得模式下最常见的退化。工程师到了现场，客户说什么就做什么。不问为什么，不想能不能标准化，不考虑做出来的东西和产品的关系。完全变成了接需求、做需求的执行者。做了一堆定制功能，没有一个能沉淀到产品里。项目结束了，代码留在客户仓库里，再也没有人碰过。

SaaS 的失败：动不动就拒绝合理需求。这是我后来在 SaaS 公司见到的。产品经理坐在办公室里，收到客户需求后先做判断：“这个需求够不够通用？能不能放进产品路线图？”答案经常是不能，于是需求被拒绝。客户提了十次合理的需求，被拒绝了十次。客户学乖了，不再提需求了。产品和客户之间的信息通道就这么断了。

两种失败看起来完全相反——一个是什么都做，一个是什么都不做。但根源是同一个：把产品当核心，没有把业务需求当核心。

IT 咨询的失败表面上是“什么都做”，实际上是因为不关心业务问题本身——只关心把需求文档上的功能做完、把项目按时交付。做着做着就变成了外包。

SaaS 的失败表面上是“什么都不做”，实际上也是因为不关心业务问题本身——只关心产品路线图上的功能排期。客户真正需要的东西，如果不在路线图上，就不是“我们的问题”。

红线

一条红线：前线优先。

这是第一章讲的四条原则里最重要的那一条，也是整本蓝皮书的底线。如果只保留一条规矩，就留这条。

前线优先的意思是：在任何决策中——需求要不要做、功能怎么设计、资源怎么分配——业务前线的真实需求优先于产品规划的完美、优先于技术架构的优雅、优先于标准化流程的规范。

踩了这条红线的表现：

- FDE 在现场变成接需求的工具人，不思考、不判断、不沉淀
- 产研团队以“产品路线图排不上”为由拒绝前线的合理需求
- 考核体系把 FDE 推向“计费工时最大化”而不是“业务价值最大化”
- 产品团队从来不看前线做出来的东西

一旦出现这些信号，不需要等到项目失败再反应。这些信号本身就是失败的早期症状。

FDE 人才画像与培育

7.1 FDE 需要什么样的人

素材里有一个说法叫“工程师外交官”——既是能打硬仗的技术高手，又是能打通关节的组织协调者。这个说法不是什么正式的职位名称，但它确实抓住了 FDE 最核心的人才要求。具体拆开来看，一个合格的 FDE 需要在五个维度上达标。

第一，业务逻辑理解 + 系统设计能力。

FDE 首先得知道业务是怎么运转的。不是泛泛地知道“财务管报销、HR 管招聘”，而是理解具体的业务逻辑——一个报销流程为什么设计了五个审批节点，哪个节点真正在控制风险、哪个节点只是在走形式；一个招聘流程里，简历筛选到面试安排之间到底卡在什么地方。

理解了业务逻辑之后，还要知道怎么把它变成一个系统。拿到需求文档不能直接开始写代码，得先判断：这个需求背后的业务问题是什么？系统能怎么解决？哪些环节可以自动化、哪些必须保留人工干预？

这一条对应第二章说的 Delta + Echo 合一。业务理解力和系统设计能力不能分在两个人身上。一个人只懂业务不知道系统能做什么，会提一堆技术上不现实的需求；一个人只懂技术不理解业务，会做出技术上优雅但没人用的东西。

第二，组织洞察与协调能力。

FDE 在客户现场工作，一定会遇到组织政治。部门之间的墙、利益冲突、被隐瞒的低效流程、担心被替代的业务人员——这些东西不会写在任何需求文档里，但它们决定了 FDE 做的东西能不能真正被用起来。

FDE 需要能识别这些问题。到现场不能带着天真——“我做好了东西他们自然会用的”——要学会观察：为什么这个流程一直在用但没人觉得有问题？为什么那个部门不愿意配合？为什么这个需求提了很多次但从来没有被满足？

识别出来之后，还要能处理。找到推进的方式——谁能拍板、谁的利益需要照顾、怎么让各方都能接受一个新的工作方式。

第三，结果驱动的执行能力。

FDE 到了现场，目标只有一个：让系统跑起来。遇到阻力想办法绕过去或者推过去，碰到技术问题自己解决，碰到人配合不了就想办法让人配合。

Palantir 总裁 Shyam Sankar 说过，FDE 的完美画像往往是其所在领域中的“叛逆者”或“异端”，有打破常规的充沛精力和主动性。这个说法有点夸张，但方向是对的。FDE 面对的是高度不确定的环境、不配合的利益相关方、随时会变的业务需求。按部就班在这种环境里不够用，得有那种“我一定要把这个事情做成”的驱动力。

这和第六章讲的红线是一致的——前线优先。遇到阻力就退缩的 FDE，前线优先就成了一句空话。

第四，平台反哺意识。

FDE 在前线解决一个问题的时候，需要能判断：这个问题是只有这一个客户会遇到，还是其他客户也可能遇到？这个解法能不能被抽象成一个通用的模式？

这个意识和个人的技术能力、业务能力没有直接关系。一个技术很强的工程师可能完全不具备这个意识——他只关心眼前的问题怎么解决，不会去想解决方案的通用性。但 FDE 必须有这个意识。

第三章讲过，70% 的沉淀率是行业里的观察——当 FDE 和产研部门的协作运转良好的时候，沉淀率自然会在 70% 左右。对个人的要求是他要能持续提出有价值的发现，和产研部门一起推动沉淀率往上走。

第五，行业前沿意识。

最后一条，也是最容易被忽略的一条。FDE 需要主动让自己站在行业前沿，无论是业务侧还是技术侧。

业务侧的前沿意识意味着他持续关注行业的趋势——新的监管政策、新的业务模式、新的竞争对手。技术侧意味着他持续跟进新的工具、新的方法、新的可能性。

这一条为什么重要？因为 FDE 的工作是在前线发现问题和解决问题。他对行业的理解停留在过去，发现的问题就是过去的问题；他对技术的掌握停留在过去，给出的方案就是过去的方案。第四章讲过，AI 编程工具是 FDE 模式在现阶段才成立的核心原因。一个不主动跟进 AI 工具发展的 FDE，很快就会掉队。

7.2 外部招聘：三阶段筛选

FDE 从哪里找

素材里列了几个常见来源：大厂的客户工程师（AWS Solutions Architect、GCP Customer Engineer）、Palantir 和 Scale AI 等公司的前 FDE、有创业经验的工程师。

在中国市场，这些来源需要调整。Palantir 的前 FDE 基本找不到，但有两个群体值得关注：一是有现场交付经验的 IT 咨询顾问，他们已经具备了在客户现场工作的能力和业务理解力；二是 SaaS 公司的解决方案架构师，他们理解产品能力边界，也接触过大量的客户需求。

但不管从哪个来源招人，候选人进了面试流程之后，都要过三道关。

第一阶段：战壕测试——评估业务管理和客户管理能力

战壕测试看的是候选人能不能在极端混乱、信息不透明、政治阻碍的环境下活下来。

测试方式是把候选人推入一个虚拟的复杂客户场景。素材里的例子是：核心数据存在一个四十年前的、没有文档的 DB2 数据库里，相关部门的高管对 AI 技术抱有敌意，担心岗位被替代。看候选人在这种环境下怎么推进。

这一阶段评估的是候选人的业务管理能力和客户管理能力。他能不能在没有完美需求文档的情况下自己找到问题？能不能在面对敌意的利益相关方时保持推进？能不能在信息极度不透明的环境里做出合理的判断？

否决红线：候选人流露出对完美 PRD 的依赖——“你得把需求文档给我写清楚我才能做”——或者表现出传统 IT 咨询人员一味顺从、不敢直言现场真实技术赤字的倾向。这种人到了前线，要么等着别人把问题定义好了才动手，要么客户说什么就做什么，两种都是 FDE 的反面。

第二阶段：开放式情境评估——评估可直接落地的咨询能力

这一阶段看的是候选人有没有真实的、可直接落地的咨询能力。

面试官向候选人详细讲解一个冷门的业务规则——比如“特定证券市场的内幕交易判定方法”或“医院重症监护室床位流转限制”——然后要求候选人现场设计数据流、提出向客户提取关键元数据的沟通策略，并画出决策链路。

测试的核心是看他能不能把模糊的业务描述转化成具体的技术方案。注意，是转化成可执行的数据流和伪代码，不是转化成架构 PPT。

否决红线：候选人只给出宽泛的架构图，无法现场写出关键的伪代码或具体的数据清洗逻辑。这种人能做方案但不能落地，而 FDE 必须能落地。

第三阶段：教导型技术分享——评估外循环能力

这一阶段看的是候选人能不能从具体的交付经验中抽象出可复用的模式——也就是第五章讲的外循环能力的核心要求。

要求候选人分享他过去最引以为傲的复杂交付项目。重点不在于“做了什么”，在于他如何在无现成文档与工具的前提下自己解决现场难题。更重要的是，他有没有从这次经历中提炼出可复用的东西——一个通用的模式、一个标准化的方法、一个可以教给别人的经验。

如果候选人通过了前两个阶段但没有通过第三个，他可能是一个合格的现场工程师或咨询顾问，但他不具备把前线经验沉淀为平台能力的素质。7.1 讲的第四个维度——平台反哺意识——就是这个阶段要测的东西。

否决红线：候选人展示的项目都是成熟体系内流水线式的固定任务，缺乏在无序环境中独立破局的经历，也没有从中提炼过可复用的经验。

三个阶段的逻辑

三个阶段是递进的。第一阶段筛掉的是“到了前线就懵了”的人——他可能技术不错，但受不了现场的混乱和压力。第二阶段筛掉的是“能沟通但不能落地”的人——他能聊、能做 PPT，但写不出可执行的方案。第三阶段筛掉的是“能干活但不会沉淀”的人——他能解决眼前的问题，但从思考解决方案的通用性。

通过三个阶段的人，就是 7.1 说的那种“工程师外交官”——既能在混乱现场把事情做成，又能把做成的经验变成别人可用的东西。

7.3 内部培育：两条不同的路，同一个终点

业务侧：先培训再上前线

业务人员学习 AI 编程，有一条可以独立走的结构化路径。不用等到上了项目才开始学，可以在进入 FDE 项目之前就完成基础培训。

第一步：建立信心。让业务人员亲眼看到 AI 能帮他做什么。生成一个单页网页、做一个浏览器插件、搭一个桌面小工具——先让他相信“这些东西我能做”。信心建立之后，学习阻力会大幅降低。

第二步：从最简单的开始。在云端做 HTML 单页应用。这是最没有门槛的起点——不需要搭环境、不需要装软件，打开浏览器就能做。

第三步：云端 Agent 工具搭建。用云端的 Agent 平台搭建简单的工作流工具。这一步开始接触“把业务逻辑变成可运行的东西”。

第四步：学会和 AI 对话。学习 Prompt 的技巧，做一些 AI Skill，把自己常用的操作封装成可复用的工具。

第五步：抽象工作流。把自己的日常工作流抽象成 Agentic Workflow——把重复性高的步骤自动化，让 AI 帮着跑。这一步是最高阶的，需要业务人员对自己的工作有足够的元认知，能清晰地描述出“我每天都在做什么、哪些步骤可以交给 AI”。

完整走完这五步大概需要三个月。但大部分业务场景只需要走到第二步就能搭出可用的 Demo。第四步和第五步是在做项目的过程中逐步掌握的，不需要一开始就学完。

技术侧：在前线上学

技术侧的人学业务理解，没有办法像业务侧那样拆成一门课程。理解业务这件事，只能在真实的客户场景里发生。

第四章讲过 FDE 团队的搭建方式：从业务和产研各拉人出来，配对作战。技术侧的人学业务，就是在这个过程中发生的。

跟着业务跑一遍。技术人员到了现场，第一件事不是写代码，是坐到业务人员旁边，看他怎么工作。打开什么系统、填什么字段、卡在哪个环节、找谁审批。这个过程像产品经理做用户调研，只不过 FDE 的技术人员是亲自动手跟着走一遍，不是在旁边看。

在系统里抽象出业务流程。跟着做完之后，把观察到的东西画成流程图。这一步的技术含量不高，但要求技术人员从“这个系统怎么实现”的视角切换到“这个人为什么这么做”的视角。

学着沟通和收集需求。业务人员描述需求的方式和技术人员理解需求的方式之间有巨大的鸿沟。业务说“这个地方太慢了”，技术人员需要追问“慢在哪里？是等审批慢、还是填信息慢、还是数据不通导致的重复录入？”这种追问的能力只能通过大量的实际对话来建立。

和用户一起把需求做出来。根据收集到的需求，动手做一个粗糙但能跑的原型。做出来给业务人员用，用完反馈，立刻改。这个循环跑几遍，技术人员就理解了业务侧的人在什么上下文里工作、什么需求是真实的、什么需求是想象出来的。

两条路为什么不对称

业务侧的学习路径是一条线，可以独立走，有明确的阶段和时间预估。技术侧的学习路径没法独立走——它嵌入在第四章讲的配对作战里面，节奏由项目决定，不由课程表决定。

这个不对称是合理的。业务人员学的是工具的使用方法，工具是确定的，输出是可以验证的。技术人员学的是对业务的理解，业务是活的，每个客户都不一样，不可能用一套固定的课程去覆盖。

两条路的终点是同一个：7.1 描述的那个人——同时具备业务理解力、系统设计能力、组织洞察力、平台反哺意识和行业前沿意识。业务侧的人通过学技术补齐了技术维度，技术侧的人通过做项目补齐了业务维度。他们各自回到团队之后，就变成了第四章讲的“FDE 种子”——可以在团队内部培养更多的 FDE。

7.4 常见误区与红线

三个招人的坑

第一个坑：只看技术，不看业务翻译能力。

技术团队招人最容易犯这个错。面试考算法、考系统设计、考代码质量，全过了就发 offer。但这个人到了客户现场，听不懂业务人员在说什么，没法把模糊的业务描述转化成具体的技术方案，做出来的东西技术很漂亮但没人用。

7.2 讲的三阶段筛选就是为了避免这个坑。如果只用传统的技术面试流程，你招到的是优秀的工程师，但可能不是合格的 FDE。

第二个坑：只看沟通，不看动手能力。

和第一个坑相反。候选人很会聊天、很会做 PPT、能和客户谈笑风生。面试的时候大家觉得“这个人太合适了”。但到了现场，他写不出能跑的代码。他能发现问题、能设计方案，但做不出来。

这种人在组织里可以是一个很好的解决方案架构师或客户成功经理，但做不了 FDE。7.2 第二阶段的开放式情境评估就是筛这种人的——能画出架构图但写不出伪代码，一律不通过。

第三个坑：把 FDE 当更贵的咨询顾问来招。

有些公司看到 FDE 的薪酬比普通工程师高 10-25%，就觉得“我愿意出这个钱，招一个更厉害的驻场顾问”。招进来之后，给他排满客户会议、让他写解决方案文档、让他做技术评审。慢慢地，这个人变成了一个会写代码的高级顾问，而不是一个在客户现场直接交付的工程师。

FDE 的核心价值是“在现场把东西做出来”，不是“在现场写文档和开会”。如果招进来的 FDE 每周有 80% 的时间在开会和写文档，只有 20% 的时间在写代码，这个岗位设计就有问题。

两个用人的坑

第四个坑：英雄工程师依赖。

一两个特别强的 FDE 承担了几乎所有重要的项目。他们效率高、客户信任、什么都能搞定。但知识全在他们脑子里，没有文档、没有沉淀、没有可复用的模式。一旦他们离职或者转岗，所有项目都停摆。

素材里把这种情况叫做“个人英雄模式”——过度依赖少数明星 FDE，知识与关系未组织化沉淀。解决方法在第四章讲过：每个 FDE 项目必须产出至少一个核心产品改进，必须有结构化的知识转移。

第五个坑：把 FDE 用成了接需求的工具人。

第六章讲过最典型的失败：FDE 到了现场，客户说什么就做什么。不问为什么，不想能不能标准化，不考虑做出来的东西和产品的关系。完全变成了接需求、做需求的执行者。

从人才角度看，这个问题可能出在招人的时候就没选对——7.2 第一阶段的战壕测试本来应该筛掉那些“等着别人把问题定义好才动手”的人。也可能出在用人上——即使招到了有判断力的 FDE，如果考核的是计费工时而不是业务价值，他也会退化成工具人。

一条红线

第六章讲过整本蓝皮书的红线：前线优先。在人才层面，这条红线有一个具体的表现——FDE 的考核和激励必须指向业务结果，不能指向工时。

一旦你开始用计费工时考核 FDE，他的行为就会变——待在现场的时间越长越好，问题解决得越快反而越吃亏。一个被工时考核的 FDE，和一个被业务结果考核的 FDE，是两个完全不同的人。

第三章讲过三种汇报模型，其中 GTM 主导（向 CRO 汇报）的结局是“18 个月内你会拥有一个咨询公司，产品会越来越腐烂”。这个结局在人才层面的表现就是：好的 FDE 会走，因为他们不想做咨询顾问；留下来的都是愿意按天卖时间的人。最终你的 FDE 团队变成了一个更贵的外包团队。

A. FDE 人才画像速查卡

五维硬指标

维度	核心要求	具体表现
业务逻辑理解 + 系统设计能力	知道业务怎么运转，能把它变成系统	看懂报销流程为什么设计五个审批节点；判断哪些环节可以自动化、哪些必须保留人工
组织洞察与协调能力	识别部门墙、政治问题、被隐瞒的低效流程，并且能处理	能观察到“这个流程一直在用但没人觉得有问题”；找到谁能拍板、谁的利益需要照顾
结果驱动的执行力	“我一定要把这个事情做成”的驱动力	遇到阻力想办法绕过去或推过去，碰到技术问题自己解决
平台反哺意识	判断眼前的问题其他客户也可能遇到，推动沉淀	在解决一个客户问题时主动思考“这个模式能复用吗”
行业前沿意识	主动让自己站在业务和技术的前沿	持续关注新工具、新方法、新的行业趋势

三阶段面试红线

阶段	测什么	否决红线
战壕测试	业务管理和客户管理能力	流露出对完美 PRD 的依赖，或一味顺从客户、不敢直言技术赤字
开放式情境评估	可直接落地的咨询能力	只给出宽泛的架构图，无法现场写出伪代码或数据清洗逻辑
教导型技术分享	外循环能力（抽象和沉淀）	展示的项目都是流水线式的固定任务，缺乏独立破局经历和经验提炼

B. 最小可行 Pod 组建模板

角色配置

角色	人数	来源	核心职责
业务侧成员	1-2 人	从业务部门拉出	理解业务需求、做 Demo、验证方案、持续使用和反馈
技术侧成员	1-2 人	从产研部门拉出	把 Demo 变成可运行的系统、和现有系统对接、沉淀可复用模式

时间线

节点	时间	里程碑
Day 1	进入项目	到业务人员工位旁坐下来，跟着走一遍完整流程
Day 1-3	交付第一个原型	极简但能跑，解决一个真实痛点
Day 14	首次集成完成	技术侧完成第一次和业务系统的集成
Day 30	核心功能可用	主要功能已可日常使用
Day 90	生产上线	完整系统部署到真实环境
Day 120	技术侧撤出	技术侧 FDE 回到产研团队，带走前线经验和可复用模式

配套规则

- 业务侧必须有对接人，否则项目不启动
- 每个项目至少产出一个核心产品改进
- 首次集成不超过 14 天，做不到就终止
- 生产上线不超过 90 天，做不到就终止
- 业务侧的人必须学会使用 AI 编程工具

C. 平台 Handoff 12 条 Checklist

技术侧 FDE 撤出项目时，必须完成以下 12 项知识转移：

项目上下文

1. 业务问题描述。解决了什么业务问题？为什么值得解决？原来的方式是什么样的？
2. 业务流程文档。当前工作流的流程图，标注关键节点、瓶颈、人工干预点。
3. 用户使用情况。谁在用、用哪些功能、哪些功能没用起来、反馈是什么。

技术交付物

1. 技术架构说明。系统的整体架构、数据流、各模块的职责和依赖关系。
2. 代码和配置清单。关键代码/配置的位置、说明、已知的技术债。
3. 数据集成映射。数据源的映射关系、集成方式、数据质量问题和临时补丁。
4. 安全和合规。数据分级、权限模型、是否通过安全审查、合规要求清单。

沉淀和延续

1. 已知问题清单。当前未解决的问题、临时方案、风险等级。
2. 闪光点上报。项目中发现的可复用模式——已上报的标注状态，未上报的附描述。
3. 核心产品改进建议。至少一个建议进入标准产品路线图的功能或能力，附理由。
4. 运维文档。部署方式、监控指标、故障排查指南、应急预案。
5. 业务侧自主能力评估。业务人员当前能否独立使用和维护？哪些环节仍需技术支持？

D. 参考案例

Palantir: FDE 的起源

Palantir 是 FDE 模式的创始者。2005 年，为了解决 CIA、NSA 等政府机构的数据集成问题，Palantir 摒弃了传统的软件销售模式，直接派遣具备安全资质的工程师长期驻扎在客户现场。

核心做法：- 工程师直接嵌入到 Fort Bragg、Langley 等军事基地，最长的部署能到一年 - 先建模客户的业务实体（Ontology），再做应用 - Ship on Day One：每个项目的第一天就交付可用的东西 - 现场团队拥有高度自主权，总部信任前线决策

关键结果：FDE 在现场为不同客户构建的解决方案，被持续抽象为平台通用能力，最终形成了 Palantir 的核心产品（Gotham、Foundry）。2024 年 Palantir 收入 \$2.87B，公开市场回报 640%。

对企业的启示：FDE 模式要成立，必须同时设计“现场交付”和“把现场学习反哺平台”两条线。Palantir 的成功不是因为驻场本身，是因为驻场产生的经验变成了产品。

OpenAI: 从呼叫中心到 Realtime API

OpenAI 的 FDE 在为某呼叫中心开发 AI 语音助理集成时，构建了一套实时评估框架来捕捉语音延迟和吞吐波动。这套框架是前线为了解决具体问题临时做的，但 FDE 意识到这个评估逻辑对所有语音场景都有用。

闪光点被即时上报。最终，这套前线评估框架重塑了 OpenAI 官方 Realtime API 的底层机制，同时促成了 Agent SDK 的定型。一个为单个客户做的临时工具，变成了平台的核心能力。

2026 年，OpenAI 进一步推出了 Deployment Company，补充约 150 名 FDE 和部署专家，初始资本超过 40 亿美元。FDE 从“少数高端项目的交付资源”升级为“企业 AI 规模化部署的正式产业能力层”。

对企业的启示：前线发现的价值取决于两件事——FDE 有没有抽象意识，以及组织有没有接收和标准化的机制。两个条件缺一个，前线的发现就永远留在前线。

Morgan Stanley 与 OpenAI: eval-driven 的采用飞轮

Morgan Stanley 的财富管理团队与 OpenAI 合作，用 FDE 模式部署 AI 工具。核心做法是先建立评测框架（eval framework），再从试点扩展到公司级使用。

关键结果：- 财富管理顾问团队使用率超过 98% - 文档可访问率从 20% 提升到 80% - 信息跟进速度从“数天”压缩到“数小时”

对企业的启示：FDE 成功的关键不是快速写代码，是现场发现、评测治理、渐进式扩展、组织采用四件事同时成立。做出来只是第一步，让人用起来才是终点。