

复杂度不会消失，只会搬家

AI 抹平表面复杂度之后，人该把力气花在哪

竹小竹8167

费曼式写法 · 由多个 AI 代理撰写与互相审校

导读

AI 抹平表面复杂度之后，人该把力气花在哪

先别急着学新工具。

每次技术换代，大多数人的第一反应是报课：学 Python、学框架、学提示词工程。这本书想劝你慢一步——因为 AI 时代的分工变化，藏在一个比「学什么」更根本的问题里：**你手里的事情，哪些复杂度是真问题，哪些只是过路费？**

1986 年，软件教父布鲁克斯把软件的难度劈成两半：问题自带的「必要复杂度」，和工具强加的「表面复杂度」。他断言没有银弹，因为过去所有进步都在清扫后者，前者纹丝不动。四十年后，大模型第一次把「意图到代码」的翻译劳动整个接了过去——中介还在，但过路费归零了。这是本书上半部分要讲的历史。

但故事没有停在「从此变简单」。复杂度守恒定律告诉我们：复杂度像水，只能搬家，不能蒸发。AI 把复杂度从「工具与语法」搬到了「需求与验收」——被搬回来的这部分，一个都不能少。于是本书下半部分回答那个真正重要的问题：**当「做」变得廉价，「想清楚要什么、验收做得对不对」成了唯一还在收费的环节，普通人该怎么练这门手艺？**

你会读到 vibe coding 为什么让花店老板第一次摸到了「自己做工具」的门槛，也会读到为什么请假审批小工具里藏着五类绕不过去的复杂度；你会拿到一张「必要复杂度五面孔」的盘点清单，也会看到律师、分析师、设计师的工作如何被同一条定律重写。

全书只有一条主线：分清两类复杂度，把省下来的力气，全部押到真问题上。带上你手头的一件烦心事，从第一章开始。

第一章 · 两种复杂度

去年我搬了一次家。搬完那天晚上，坐在纸箱堆里，我突然意识到：这一天里我做的事情，可以干净地分成两类。

第一类，只有我能做：这套书哪些留着、哪些送人；新家的柜子怎么分区，锅放在伸手就能够到的地方；孩子的书桌靠窗还是靠墙。这些事没人能替我决断，因为答案藏在我自己的需求里——我的生活习惯、我的家庭结构、我未来十年的打算。

第二类，我其实不该做：研究哪种纸箱最结实、比价三家搬家平台、搞清楚货拉拉的高速费怎么算、跟物业预约货梯、把书一本本捆好防止散架。这些事烦琐、耗神，但和「我要过什么样的生活」毫无关系。它们存在，只是因为搬家这件事的中间环节太多。

用一个词来称呼这两类事：

必要复杂度，英文叫 essential complexity——问题本身自带的、绕不过去的复杂性。只要你要搬家，「哪些东西值得带进新生活」这个复杂性就永远在那儿，谁也消灭不了它。

表面复杂度，英文叫 accidental complexity——因为工具、流程、中间环节而额外长出来的复杂性。它不该是问题的一部分，只是你恰好在用笨拙的方式解决问题。

大白话

大白话：必要复杂度是「搬家就得决定扔什么」，表面复杂度是「跟物业抢货梯」。前者躲不掉，后者纯属过路费。

这个区分，是软件行业最重要的一次「想明白」

1986 年，IBM 大型机操作系统 OS/360 的项目经理 弗雷德·布鲁克斯（Fred Brooks）写了一篇著名的论文《没有银弹》。在此之前，他已经在《人月神话》里记录过一个惨痛事实：给一个延期的软件项目加人，只会让它更延期。

《没有银弹》的核心论证，就是我们刚才搬家悟出的东西。Brooks 说，软件开发的难度来自两个完全不同的源头：

- **本质的困难**：问题域本身的复杂性。一个银行账户系统之所以复杂，不是因为编程语言难，而是因为「转账」背后有一整套业务规则——透支怎么办、并发转账怎么办、退款走哪条路径、月底对账怎么对。这些规则是银行这个现实世界自带的，你把程序员全换成天才，规则一条也不会少。
- **偶然的困难**：工具带来的复杂性。得自己管理内存、得记住某个框架的配置文件怎么写、得搞定环境依赖、得等编译。这些困难不是「转账」这个问题要求的，是「用这台机器、这门语言、这个框架」这个选择带来的。

Brooks 的断言是：过去几十年软件变快，几乎全部来自偶然困难的下降；而本质困难几乎没被触碰。所以「没有银弹」——不存在一种技术能让软件生产力十倍提升，因为它瞄准的只能是偶然那一半。

为什么这个区分值得你花三分钟

因为它直接决定你该把力气花在哪。表面复杂度的正确应对是**外包、自动化、消灭**——能交给工具的绝不自己动手。必要复杂度的正确应对是**直面、想清楚、亲自决策**——能外包的恰恰是你最不该外包的部分。

搬家时，你可以雇一个搬家公司，把纸箱、货梯、货车这些表面复杂度整个买走。但你没法雇一个人替你决定「什么值得留下」——那是你的必要复杂度，请了人它也不会消失，只会被胡乱决定。

软件世界也一样，只是尺度更大。接下来的问题是：既然这个区分 1986 年就被讲清楚了，为什么四十年里程序员们的主要精力，仍然耗在表面复杂度上？AI 这次又到底改变了哪一半？

别急，我们一层一层来。下一章，先看看表面复杂度是怎么一层一层长出来的。

第二章 · 中间商是怎么长出来的

1946 年，世界上第一批「程序员」这个行当还不存在。那时候的计算机 ENIAC 没有键盘，想让机器算一道题，工程师要抱着图纸，钻进机器内部，把几百根电缆重新插拔一遍——用物理连线来表达计算步骤。写完一个程序，满身大汗，机器还得停机。

你今天写 `print("hello")` 的三个词，背后是八十年的「翻译层」演化史。这段历史值得讲一遍，因为它解释了程序员的时间都花在了哪。

一层层的翻译，一层层的中介

计算机唯一听得懂的语言是机器码——一串 0 和 1。每个 0 和 1 对应电路上一个开关的通断。问题在于，人类不擅长记 0 和 1，就像人类不擅长用莫尔斯电码聊天。于是第一家中介开张了：

1940 年代末，汇编语言。把机器码变成人能读的缩写：`ADD` 代表加法的二进制，`MOV` 代表搬运数据。程序员写缩写，一个翻译程序把它变回 0 和 1。这是第一次「没有中间商」的尝试，但很遗憾，汇编还是围着机器转，不围着人转。

1957 年，Fortran。IBM 的约翰·巴克斯带着团队做出第一门高级语言，允许人写接近数学公式的语句。当时很多程序员嗤之以鼻：让机器帮你翻译，效率肯定低。结果 Fortran 程序跑起来够快，这些人转而看不起不用 Fortran 的人。这是软件史上第一次「离机器远一点」的大胜利。

1970 年代，C 语言和 Unix。操作系统不再需要每个机器厂商各写一遍。但 C 语言管内存管指针，程序员得亲自给每个字节安排住处——这就是第一章说的表面复杂度，换了个形式回来了。

1990 年代到 2000 年代，框架与云服务。人们不再直接操作数据库，而是用 框架——一套替你写好了常见套路的半成品软件；不再自己买服务器，而是租亚马逊的云计算。每一层新中介都宣布：「以前的苦都不用吃了。」

可惜，中介的承诺从来没完全兑现过

2002 年，程序员出身的作家乔尔·斯波尔斯基（Joel Spolsky）写了一篇流传极广的文章，提出抽象泄漏定律：所有重要的抽象，都会在某些情况下失效。你用框架写网页，岁月静好，直到有一天网页在某些手机上白屏——你得钻进框架底层，搞明白是 HTTP 缓存头出了问题。你以为远离了机器，其实只是把机器藏起来了；藏起来的机器，会在最要命的时刻探头出来。

大白话

大白话：翻译层像中介，能帮你谈下大部分生意，但每一笔大单，最后都得你亲自出面。你以为省掉的中介费，会以「应急协调费」的形式收回来。

更说明问题的是 2008 年 Stack Overflow——一个程序员互相回答问题的网站——的爆红。翻开它早期最热门的问题：怎么配环境变量、某个依赖库装不上、字符编码乱码。你会发现一个行业里最聪明的人，把最聪明的年华，大量消耗在与「我要解决的问题」无关的地方。这就是表面复杂度的总量：它大到能撑起一个日活千万的网站。

每一层翻译都有存在的理由——它们真的降低了门槛，真的让更多的人做出了更多的软件。但同时，每一层都在收过路费：你得学它的语法、背它的约定、处理它的泄漏。到 2020 年前后，一个想做点小工具的普通人，面对的不是「我的问题有多难」，而是「我要穿过多少层中介才能碰到我的问题」。

那有没有可能，这一层层翻译，本身就是可以被绕过去的？绕过去需要什么条件？这就是下一章要回答的——先回到 1986 年，看看布鲁克斯为什么断定「没有银弹」，再看看他的论证里，藏着什么被时代漏掉的假设。

第三章 · 银弹为什么没有来

1986 年那篇《没有银弹》，其实是一篇论证极其克制文章。重读它，你会发现布鲁克斯没有否认技术进步——他列举了当时所有的时髦工具：高级语言、分时系统、统一编程环境，承认它们都带来了可观的进步。他否认的只是：这些进步都是一次性的「偶然困难的局部清缴」，而不是生产力数量级的跃迁。

为了讲清楚为什么本质困难无法被工具触碰，他给出了软件的四条固有属性。这四条今天读来依然锋利：

- **复杂性**：软件是人类建造过的最复杂的物体。一座桥梁有几十万个零件，但同类零件大量重复；一个大型软件系统里没有两个完全相同的模块，规模一大，复杂度按组合数爆炸。
- **随附性** (conformity)：软件没有自己的物理定律，它必须去迁就人类社会任意的、历史形成的规则——银行系统要符合会计惯例，税务系统要符合税法，而税法明年可能就变。这部分规则毫无道理可讲，但一行都不能错。
- **易变性**：桥建好了就没人想着「把桥重构一下，把第三根柱子改成可插拔的」。软件被使用就必然被提新需求，而改动会引发连锁改动。
- **不可见性**：建筑有图纸，电路有板图，软件没有天然的直观形态。它看不见摸不着，你没法绕着它走一圈获得整体感。

这四条属性界定了「必要复杂度」的领土：它们是问题自带的，不是工具带来的。布鲁克斯由此断言：任何瞄准本质困难的银弹都不存在，未来的进步只能继续来自偶然困难的削减。

四十年里，人们真的信过无数次「这次是银弹」

布鲁克斯的冷水没能浇灭行业的热情，因为「让不会编程的人也能指挥计算机」这块蛋糕实在太诱人。历史上一串闪亮的名字，今天听来都像银弹宣言：

1959 年，COBOL。这门语言被刻意设计成像英语：`ADD TAX TO PRICE`。设计者的梦想是让经理们直接写代码。结果 COBOL 确实统治了商业计算半个世纪，但写它的依然是专业程序员——英语式的语法，骗过了外行，骗不过机器：写 `ADD TAX TO PRICE` 之前，你还是得想清楚变量、类型和控制流，这些才是难点，而这些难点被漂亮地伪装掉了。

1980 年代，第四代语言和 CASE 工具。 厂商承诺「画流程图自动生成系统，从此不需要程序员」。大批企业买了单，大批项目死在了交付前。问题不在工具好坏，在于需求本身会生长：图能表达正常流程，表达不了「对账差一分钱时该怎么追溯」这种长在业务褶皱里的规则。

二十一世纪初，Scratch。 麻省理工学院的团队 2003 年前后开始研发图形化编程，2007 年正式公开：拖积木代替写语句。这次是真实成功——但请注意它的成功范围：教孩子表达逻辑。它降低的是「键盘语法」这个最浅的偶然困难，而孩子们用它写的，依然是他们自己构思的程序。一旦涉及复杂规则，积木图照样会乱成一团。

注意这三十年潮起潮落的共同模式：每一次，人们都试图用「更友好的表达形式」绕过翻译层，每一次都失败了。失败的原因不是形式不够友好，而是缺了翻译层下游的那个东西——**一个能接住模糊意图、自己补齐细节的翻译者**。形式再友好，「把需求精确化」这活儿还是得人来干。

大白话

大白话：以前的「自然语言编程」，是给了你一支英文笔，作文还得自己写。缺的不是笔，是代笔的人。

所以布鲁克斯是对的，而且比他自己知道的更对：他论证的是「意图到代码的这段路必须有专人翻译」。四十年里这个论证无懈可击——直到翻译这件事本身，被证明可以不由人来做。

下一章，看看 2020 年代到底什么东西变了。

第四章 · 这次不一样

2020 年，一件当时没多少人意识到其分量的事发生了：OpenAI 发布 GPT-3，展示了一个语言模型可以只看你给的几个例子，就学会完成你交给它的文字任务。同一时期，工程师们发现了一件更惊人的事——这个模型居然也会写代码。

接下来的节奏快得不像传统行业：**2021 年**，GitHub 推出 Copilot——一个住在你编辑器里的 AI 补全工具，由 OpenAI 的 Codex 模型驱动，你写下注释，它替你补代码续完；**2022 年 11 月**，ChatGPT 上线，两个月涌进一亿用户，其中一大半问题不是聊天，是「帮我写」「帮我改」「帮我看看为什么报错」；**2023 年**，GPT-4 通过模拟律师资格考试的消息传遍全网，而程序员们更在意的是另一件事：它能读懂一整个项目的代码，回答「这段逻辑是谁写的、想干什么」。

为什么说这一次不一样？回到上一章的结论：过去四十年所有尝试都卡在同一个地方——「把需求精确化」这个翻译工作必须由人来完成，因为机器接不住模糊的人类意图。COBOL 给你英语语法，但作文得自己写。

大模型把这个前提拆掉了。它不是一个新语法、不是一个新框架、不是又一个「更友好的形式」。它做的事情是：把人类公开写出过的天文数字量级的代码和解释读进去，在内部形成了一张从「人话」到「代码」的映射网。你给它一句模糊的话——「帮我写个脚本，把这个文件夹里的发票按月份归类」——它能接住，因为它「见过」几万次类似的话对应什么样的代码。

大白话

大白话：以前每学一门语言，是你去考翻译证。这一次，是翻译自己找上门来了——而且它读过整个图书馆。

先看它吃掉的是哪一半

用第一章的两分法来验收。大模型带来的这些能力——写胶水代码、配环境、查报错、给陌生框架当说明书、把「这里应该有循环」补全成几十行实现——它们全部落在**表面复杂度的**

领土上。它们没有一个在触碰「你的业务规则到底是什么」这个问题。

这正是布鲁克斯框架里「偶然困难清缴」的延续。从这个角度看，AI 不是什么颠覆者，它是抽象演化史的下一层中介。但有两个关键的不同，让它和历史上所有中介都不一样：

第一，它收的是意图，不是形式。以往每一层中介都发明一套新形式让你学：汇编的助记符、Fortran 的语句、框架的约定、云服务的控制台。大模型是第一个不要求你学它的形式的中介——它迁就你的形式。你用破碎的中文、用大白话、用一个糟糕的例子，它都接得住。中介史上第一次，过路费降成了零。

第二，它处理泄漏。还记得抽象泄漏定律吗？以前的翻译层 leaked，你就得亲自下潜。现在漏了怎么办？你把报错原样贴给它。HTTP 缓存头出了问题，它看一眼就告诉你该改哪一行；某个依赖装不上，它认得这个错误的「长相」。以前中介跑了，摊子是你的；现在这个中介帮你收拾烂摊子。

所以「这次不一样」的准确表述是：以前所有中介降低了表达成本，但保留了「必须由人精确化」的底线；大模型第一次把「精确化」这个劳动本身也接了过去。第二章末尾那个问题——能不能把翻译层整个绕过去——答案是可以：绕过去的办法不是找到更好的形式，而是让翻译者变得无处不在且不要钱。

但是，先别急着庆祝。请回忆第一章搬家故事的另一半：雇了搬家公司之后，「哪些东西值得带走」这个复杂度消失了吗？没有，它还在原地等你。下一章，我们给这个「不消失」找一个更严格的表述——一条守恒定律。

第五章 · 复杂度守恒

上一章结尾留了个悬念：表面复杂度被 AI 吃掉之后，事情是不是就变简单了？在回答之前，先认识一条和布鲁克斯同样著名、但更「物理学」的定律。

上世纪八十年代中期，拉里·特斯勒（Larry Tesler）提出了一个观察。特斯勒是图形界面历史上的关键人物，先后在施乐 PARC 和苹果工作——你熟悉的复制粘贴快捷键、双击选中、拖拽操作，都和他主持过的项目有关。他的观察被后人称为复杂度守恒定律，也叫特斯勒定律：

每一个应用都自带一份不可消除的复杂度。它只能被移动——要么由系统内部消化，要么被推给用户承担。

特斯勒是在产品设计的语境下说这句话的：你在设计软件时以为「我把这个功能简化了」，多数时候你只是把复杂度从界面上挪走了，挪进了设置菜单、挪进了异常情况、或者挪进了用户的脑子。界面变干净了，复杂度并没有死，它只是搬了家。

大白话

大白话：复杂度像水，装在任何容器里就是什么形状。你可以决定它待在哪个容器，但不能让它凭空蒸发。

生活中到处是守恒的例子

汽车的自动变速箱是最好的例子。开手动挡，你要管理离合器、听发动机声音换挡——复杂度在驾驶员手里。自动挡把这团复杂度整体搬进了变速箱的液压系统里，司机的世界变简单了。注意：那份复杂度没有消失，它变成了工程师设计变速逻辑时必须处理的一整套判断规则，只是由几百万人共摊给了几十个工程师。

特斯勒定律的精髓就在「共摊」二字：**复杂度守恒，但不规定谁承担**。好的设计不是消灭复杂度，而是把复杂度搬到最能消化它的那一层——搬给系统、搬给流程、搬给专业的少数人

去共摊，让不专业的大多数享清福。

布鲁克斯和特斯勒，一个从软件工程角度、一个从人机交互角度，说的是同一件事的两个侧面：本质的复杂度是守恒的，能被削减的从来只有偶然的那部分。

现在把 AI 放进这个框架

大模型消化掉的是什么？写胶水代码、配环境、查文档、翻译报错——表面复杂度，被搬进模型内部共摊了：一次训练，亿万人受益。这正是特斯勒说的「搬到最能消化它的一层」，只是这一层不是变速箱，而是一个读过整个图书馆的翻译者。

那么守恒的部分去哪了？

答案可能让你不太舒服：**它被搬回了你这里**。过去，一个模糊的需求要先经过产品经理细化、程序员猜测、反复扯皮才变成代码——这段「把意图精确化」的路上挤满了人，表面复杂度遮蔽了需求的模糊。现在路被铲平了，你对着 AI 说出那句话的那一刻，你需求里的每一处模糊，都会原样落在生成的结果里。AI 不会问「对账差一分钱怎么办」，除非你告诉它对账差一分钱怎么办。你没说的，它不会替你想到——它只会替你吧「没说」的后果忠实地实现出来。

于是局面变成了：实现端几乎免费，而「想清楚要什么」成了整条流水线上唯一还在收费的环节。这就是守恒定律在 AI 时代的具体形态——复杂度没有减少，它从「工具与语法」搬到了「需求与验收」。

这就引出了这本书最实用的一个问题：既然被搬回来的这部分复杂度一个都不能少，那普通人——不会写代码的人——该怎么接住它？2025 年，一位著名 AI 研究者用一个新词回答了这个问题的一半。下一章，我们讲 vibe coding。

第六章 · vibe coding：需求即代码

2025 年 2 月，[安德烈·卡帕西](#)（Andrej Karpathy）发了一条几十字的推文，给一种早已在悄悄蔓延的写代码方式命了名。Karpathy 是 OpenAI 的创始成员、特斯拉前 AI 总监，在程序员世界里他说一句话，比大部分公司发一篇白皮书更有分量。他把这种方式叫做 [vibe coding](#)——「氛围编程」：

有一种新的编程方式，我称之为 vibe coding：你完全沉浸在氛围里，拥抱指数增长，甚至忘记代码的存在。看到报错，就把报错粘回去；出错就让它自己改。这不是真正的编程，但我只是看、说、运行、复制粘贴，而且大部分都能跑。

这段话之所以病毒式传播，是因为它精准描述了几百万人正在做的事：不写代码，只描述想要什么，看着结果，用「不对，我要的是……」继续推进。工具也在这时候齐刷刷就位——Cursor 这样的 AI 编辑器、Claude Code 这样的命令行代理、Replit Agent 和 Lovable 这样的「说一句话就给你一个网站」的平台，都在把「描述→生成→验收」这个循环压缩到分钟级。

真正的新闻不是程序员变快了

如果只把 vibe coding 理解成「程序员提效」，就看漏了重点。Karpathy 原话里最重要的是四个字：**忘记代码的存在**。这意味着写软件这件事的第一入口，第一次从「会语法的人」变成了「知道自己要什么的人」。

过去，一个花店老板想做一个「会员生日前三天提醒我送一朵花」的小工具，她的路径是这样的：找人做（贵、排队、讲不清需求）、买现成软件（不完全合身、按年付费）、或者自己学编程（三个月起步，学到的东西大半用不上）。她的需求本身极简单——记住日子、提前提醒——难的是穿过中间商丛林。现在她的路径是：打开一个 AI 编程工具，把那句话原样说出去，十分钟后工具在她手机上了。

大白话：以前「会做」和「想要」是两个人，中间隔着学费和中介费。vibe coding 把这两个人合成了一个——前提是你能把「想要」说清楚。

这就是种子那句话的落地：**没有中间商赚差价**。翻译层第一次对用户隐身，意图直接抵达执行。第四章说的「中介收意图不收形式」，在这里变成了普通用户的日常体验。

但请注意 Karpathy 说的适用边界

原推文里还有一个常被转述时刻意忽略的细节：他紧接着补充，这种方式适合「周末随手做的小东西、一次性的项目」，而不适合严肃的、要维护的生产系统。这不是谦虚，是守恒定律在敲桌子。

为什么小工具适合 vibe coding？因为它的必要复杂度小：规则少、用户少、出错代价低，「说清楚要什么」这件事几分钟就能完成。而一个大系统的必要复杂度大到什么程度，第三章的 Brooks 已经替我们说过了——几百条业务规则、随附的法规、随时生长的需求。这部分复杂度不会因为你说得含糊就消失，它只会在你没意识到的地方变成线上事故。

所以对 vibe coding 最准确的定性是：**它是必要复杂度的显影液**。需求简单的人，第一次可以不交表面复杂度的过路费，直接兑现想法；需求复杂的人，会发现模糊的需求立刻以 bug 的形式显形——AI 把你没说清楚的每一处，都忠实而快速地做了出来，包括你错的地方。

于是矛盾转移了：当「做」变得廉价，「说清楚要什么」就成了新的瓶颈工种。这门手艺有没有名字、怎么练？下一章，我们把「说清楚」拆成可操作的清单——你会看到，所谓必要复杂度，原来是一件件可以点名的东西。

第七章 · 一个都不能少

先做一个实验。打开任何一个 AI 编程工具，说：「帮我写一个请假审批的小工具，员工提交，老板批准。」几秒钟后你会得到一个界面美观、流程顺畅的应用——提交、列表、批准、拒绝，样样齐全。看起来很完美，对吧？

现在回答这几个问题，答完再回去看你的工具：

- 员工请假三天和请假三十天，走的是同一个审批吗？
- 老板三天没点批准，这件事谁负责——有没有提醒？
- 两个人同时批准同一条请假，会发生什么？
- 请假被批准之后又撤销，已经排好的班表怎么办？
- 老板能看到所有部门的请假，还是只能看到自己部门的？

你的小工具大概率一个都答不上来。不是 AI 笨——恰恰相反，它太忠实了：你只描述了「顺利的路径」，它就只实现了顺利的路径。而真实世界的复杂度，几乎全部住在「不顺利的路径」里。这部分就是必要复杂度：问题自带的、一个都不能少的复杂度。

必要复杂度的五种面孔

上面五个问题不是随口编的。几十年来，软件行业把「需求的必要复杂度」翻来覆去地踩坑，踩出的大坑基本就这五类。你可以把它们当成盘点任何一件事的清单：

一、边界与例外。数量、时长、金额走到极端时会怎样？三天假和三十天假是两种制度，一句「员工请假」把它们糊在了一起。所有「正常情况」的描述，都默认了一个未经检查的取值范围。

二、失败与时序。每一步都可能不发生、晚发生、发生两次。老板不点批准怎么办？网络断了提交一半怎么办？「超时」「重试」「并发」这些词听着技术，其实问的全是业务问题：事情不按剧本走的时候，责任归谁、状态算啥。

三、角色与权限。「谁能看到什么、谁能操作什么」从来不是实现细节，而是规则本身。请假工具里「老板」这个词，藏着至少三个角色：直属主管、部门总监、HR——他们的权限各

不相同，你含糊地说「老板批准」，AI 就给你造一个能看到全公司数据的超级角色。

四、数据的一生。数据被创建之后去哪？请假记录三年后还在吗？员工离职了记录归谁？

「对账差一分钱怎么追溯」这类问题属于此类。数据的生命周期问题平时隐形，出事时全是它在作祟。

五、验收标准。最重要的一个：你怎么知道结果是对的？「好用」「智能一点」不是标准，「请假超过三天的必须总监批准，否则主管即可」才是。没有验收标准，你和 AI 之间就没有合同，只有 vibes。

大白话

大白话：顺利的路径是公路，必要复杂度是地形。AI 能替你铺最快的路，但山在哪里、河有多宽，得你亲自指给它看——不指，它就开路开进河里。

「说清楚」不是写作技巧，是认知劳动

这份清单令人泄气的地方在于：它一个都外包不掉。你可以让 AI 写代码、配服务器、画界面，但「三十天假该不该区别对待」取决于你公司的管理制度，「数据留三年还是七年」取决于你的合规要求——这些问题的答案只存在于你的脑子里，和问题的上下文里。

过去这个劳动也存在，只是被中间商吸收了：程序员接到模糊需求时会追问、会假设、会替你补全，然后花几周把歧义磨平——你在评审会上看到的「做的不是我要的」，就是磨平的账单。AI 时代这笔账单没有消失，它只是提前了、显形了：从「开发中途扯皮」提前到了「你开口的那几分钟」。

所以 vibe coding 时代的核心技能，不是学会怎么写提示词——那些技巧三个月就过时——而是**对自己要做的事情有结构化的认识**：顺利路径是什么，五种复杂藏在哪里，验收线画在哪。这张清单就是「需求即代码」时代的语法书。

顺便说一句，这套清单不只对写软件有用。下一章你会发现，律师、分析师、设计师、产品经理的工作，正在全部变成同一件事：说清要什么，验收做出来的结果。这个趋势有个名字——vibe work。

第八章 · vibe work：不止写代码

第六章讲 vibe coding 时，我们把它框定在「写软件」里。但如果把镜头拉远，会看到一件更有意思的事：同一个循环——**说清要什么 → AI 生成 → 人来验收**——正在每一个知识工作里复制。这个趋势值得一个更大的名字：vibe work，氛围工作。它指的是：凡是产出物（文档、分析、设计、方案）的生产成本被 AI 打到接近零的工作，其工作方式都在向 vibe coding 收敛。

四个行业，同一个循环

律师。过去律师助理的主要工作是在模板库和判例库之间搬运：起草一份采购合同，八成条款有成熟范本，助理的价值在于知道用哪个范本、改哪几处。AI 读完范本库之后，「起草」这一步基本免费了。但请注意守恒定律的动作：起草免费之后，「这份合同的风险条款到底该不该让步、让到什么程度」成了唯一的收费环节——那是长在商业谈判上下文里的判断，一个都不能少。

分析师。过去商业分析师一周里大半时间在做数据的搬运工：从系统里导出、在 Excel 里清洗、拼成图表。现在对着 AI 说一句「把上季度各区销售按环比拉出来，异常的点标出来」，几分钟出结果。但「哪个异常值得深挖」「这个相关性能不能当因果讲」——这些问题的答案不在数据里，在业务常识里。AI 让「做表」免费，让「读懂数字背后的生意」成了全部价值。

设计师。出图、改版式、出十几种风格稿，这些过去的「手艺活」正在快速贬值。但「这个产品应该让用户第一步做什么」没有贬值——信息架构、使用场景的取舍，是需求自带的必要复杂度。

管理者。写周报、整理会议纪要、做汇报初稿，全都可以一句话生成。但「这个季度到底该砍哪块业务」不行——它的复杂度来自你对团队、市场、现金流的全局掌握，AI 只能帮你把想法显性化，不能替你想。

大白话：各行各业过去都有一层「语法」——Excel 公式、合同模板、排版技巧、汇报套路。会语法的人拿着工资。AI 把语法层抽掉之后，工资单上剩下的只有一件事：你对业务本身的判断。

职业护城河的水位在下降

这个现象值得直说：很多职业的护城河，其实灌的是表面复杂度的水。法律文书的晦涩、财务模型的精巧、设计软件的熟练——这些「很难学会、学会了很吃香」的技能，构成了一道中介墙：客户要的是结果，你必须穿过专业人士才能拿到。

vibe work 正在抽干这道墙。不是职业消失——律师、分析师、设计师都还在——而是**职业内部的价值坐标在迁移**：从「我会做」迁移到「我看得准、判得对、敢负责」。这正是种子那句话的完整版：专注点要从工具、语法层面，转移到需求本质的必要复杂度层面。

这里有一个容易被误读的地方，必须提前挡住：「转移」不等于「工具和语法不用学了」。恰恰相反，你必须懂到**能验收**的程度——你需要看得懂 AI 写的合同里那个条款意味着什么，才能签字。学语法的目的变了：从前是为了亲手生产，现在是为了验收生产。这个区别，是下一章的主角。

第九章 · 新稀缺：判断力

上一章结尾说：学语法的目的，从亲手生产变成了验收生产。这一章把「验收」二字掰开——它是 AI 时代最贵的能力，没有之一。

先讲一个经济学上再朴素不过的道理：**当一样东西的生产成本塌下去，和它配套的那样东西就会涨价**。印刷术让抄写免费之后，校对错字成了出版社的核心工序；相机让拍照免费之后，「拍什么、怎么拍」成了摄影师的全部价值。AI 让所有文字、代码、方案的生成趋近免费，那么涨价的必然是：**判断生成物对不对**。

为什么验收变贵了：因为 AI 会一本正经地编

用大白话解释幻觉（hallucination）：大模型不是数据库，它没有「查资料」这个动作。它的工作原理是「根据读过的海量文本，生成最像正确答案的文字」。绝大多数时候，最像正确答案的文字就是正确答案；但模型分不清「我知道」和「我编得圆」——它给一个不存在的法院判例编出案号，和一个它真背过的判例给出案号，语气一模一样。

这就造成了一个阴险的不对称：生成的成本是三分钟，核查的成本是三小时。一份 AI 写的行业分析报告，读起来流畅自信、图表齐全，里面可能混着两个编造的数字和一个过时的结论。你要是直接拿去汇报，省下的三小时会在会议室里连本带利收回来。

大白话

大白话：AI 是个说得永远比做得快的实习生——热情、快、从不承认不知道。带这样的实习生，师傅的核心工作不是让他干，是检查他干的。

判断力其实是三样东西

「验收能力」听着抽象，拆开是三个可以分别练习的部件：

一、品味——知道好长什么样。验收的前提是脑子里有标准。一个写过十年合同的人扫一眼就知道条款的「味道」不对，一个看过几百份好方案的人能感到某份方案「轻飘」。品味没

法速成，但有一个加速秘诀：**认真看过大量好东西**——这是 AI 时代最不亏的投资，因为它恰恰是训练数据喂不到你脑子里的那部分。

二、核查——掌握验证的动作。核查是有套路的：让 AI 给来源，然后真的去点开；先问它几个你早知道答案的问题校准它的可信度（像老师先出几道送分题）；对「承重墙」——支撑整个结论的关键数字和引用——逐个验，装饰性的地方放过。以及永远记住那个顺序：**先抽查，再采信，顺序反了抽查就不存在了。**

三、责任——敢签字。这是最硬的一条。合同是律师用 AI 起草的，签字的是你；报告是 AI 写的，汇报的是你。责任无法随复杂度一起外包——守恒定律在人这里也成立：后果的复杂度，永远由后果的承担者持有。一个敢说「我看过、我负责」的人，和一个只能说「AI 是这么写的」的人，在 vibe work 时代是两个物种。

于是人的位置清楚了

把这一章和第五章放在一起看，图景就完整了：AI 把表面复杂度搬走，把必要复杂度送回你手上；而必要复杂度里最大的两块——「想清楚要什么」（第七章）和「验收做得对不对」（本章）——本质上都是判断力的不同侧面。

布鲁克斯四十年前担心的「没有银弹」，现在可以改写得更准确了：**银弹终于来了，但它只杀偶然，不杀本质。而本质复杂度，恰恰是人还值钱的地方。**

道理讲完了。最后一章，我们把它变成动作：一张可以今天就做的盘点清单。

第十章 · 盘点你的复杂度

最后一章，不做总结，做一件事：把整本书变成你今天下班前就能动手的四个动作。

拿出一张纸（或开一个空白文档），把你这一周做过的、占时间最多的十件事列出来。不用工整，想到就写。然后对每一件事问同一个问题：**这件事的复杂，是我问题自带的，还是工具强加的？**

第一步：贴标签，两类分家

用第一章的标准给每件事贴标签：

- **表面复杂度**（过路费）：换个工具、换个人来做，复杂度就消失或骤降的事。典型长相：整理数据、排版、写格式化的初稿、查软件用法、反复复制粘贴、配置和调试环境。
- **必要复杂度**（真问题）：谁来都得面对的复杂度。典型长相：决定做不做、定规则、排优先级、处理例外、对结果负责。

多数人的清单贴完标签会愣一下：日常时间里最大的一块，往往贴着表面复杂度的标签。这正常——过去整个软件行业都在靠这份清单吃饭。

第二步：表面复杂度，整批搬走

贴「表面」标签的事，不要一件一件优化，要**整批交给 AI**。动作很简单：对每一件事，用一句话向 AI 描述你想要的结果，然后——这是关键——用第七章的「五面孔」清单过一遍：边界例外说了吗？失败怎么办？涉及谁的数据？做完了你拿什么验收？

这五问就是 vibe coding 的全部心法。它适用于写脚本、做周报、起草邮件、整理表格——凡是 vibe work，都走同一个循环：**说清要什么 → 生成 → 拿五面孔验收 → 说清差在哪**。第一轮结果从来不好，这是设计的一部分，不是意外。

第三步：必要复杂度，认领回家

贴「必要」标签的事，一件都搬不走，但有一件你能立刻做：**把它们显性化**。规则藏在脑子里的时候，它们以「扯皮」「返工」「这做的什么玩意」的形式消耗你；写在纸上，它们变成清单、标准和决策记录——你对自己工作的掌握度会肉眼可见地上升。

同时把省下来的时间投进去。表面复杂度被搬走后，你每天多出的一两个小时，最不该的用法是刷更多需求进来，最好的用法是花在必要复杂度上：把业务的边界情况想全、把验收标准定清、把领域里的好东西看够。这是第九章说的判断力投资，也是复利最高的投资。

第四步：每周一次，重新分家

这份清单不是一次性的。AI 的能力边界每几个月就挪一次，今天的必要复杂度，明天可能变成表面复杂度——去年还需要律师判断的格式审查，今年 AI 做得比助理准。每周花十分钟，问自己一个问题：**「我上周花时间最多的那件事，换到这周还需要我做吗？」**vibe work 时代，保持竞争力的方式不是学会某个工具，而是持续重新回答这个问题。

大白话

大白话：AI 是你终于请得起的搬家公司。搬家公司进场之后，你的时间只该花在一件事上——想清楚哪些东西值得带进新生活。

写到这里，可以回到开篇那个坐在纸箱堆里的夜晚了。搬家这件事的本质从来没有变过：该处理的复杂度一个都不能少，变化的只是谁来扛。八十年的软件史，是翻译层一层层长出来又将被一层层拆掉的历史；布鲁克斯和特斯勒的两条定律一起说完了这个故事：复杂度守恒，人只选择把它放在哪、由谁承担。

过去，承担的方式是学会穿过中介丛林；现在，中介第一次安静了。省下的力气往哪放，这本书从头到尾只有一个答案：**放到只有你能放的地方——真问题上。**