Malorean Corp.

一部引发共鸣的小说

一幅帮助理解的全景图

开发者、AI 与小型企业：一段关于转型的（几乎）真实故事

当今 SaaS ERP 团队
面对 AI 时的真实经历



LAURENT DESÉCHALLIERS

Malorean Corp

> 过去，把一个想法变成原型有时需要数周。如今，一个人在人工智能的帮助下，几天内就能做到。这一变化令人瞩目。但真正的巨变也许发生在别处：发生在我们共同工作的方式之中。

Laurent Deséchalliers

翻译说明

本版由人工智能根据法文原文自动翻译而成。

译文未经人工译者审校。书中的观点、结构和示例忠实于原作；但部分表达可能略显生硬，个别段落也可能留有源语言的痕迹。

这是为了让本书不只以法文、而能以多种语言面世所接受的代价。若对某处表述存疑，请以法文版为准。

引言

AI 最令我震撼的，并不是它会写代码，而是它能如此迅速地实现一些不久前还需要数天乃至数周才能完成的事情。

关于可行性的讨论甚至还没有结束，一个想法就可能已经变成原型。开发者无需等待整个团队都有空，就能探索一种可能性。不会编程的人，有时也能亲手构建自己设想的第一个版本。这并不意味着原型就是可靠的产品，也不意味着复杂性已经消失。但仅仅是能够动手尝试，就足以改变我们看待“什么是可能的”这一问题的方式。

然而，我们的企业并不是为这样一个能够极快产出初步解决方案的世界而设计的。

企业围绕专业能力、可用时间、工作交接以及推动工作所需的各种取舍组织起来。这些机制自有其存在的理由。可是，当个人的能力发生变化时，新的问题便随之出现：我们仍然需要按照同样的顺序工作吗？仍然需要等待同样的审批吗？仍然需要把同样的活动保留给同样的岗位吗？

这些问题并不只涉及开发者。产品经理、设计师、销售人员、团队负责人以及许多其他人，也都在发现新的工作方式。有些人如今能够直接完成过去需要委托同事处理的一部分工作；另一些人则看到自己工作的性质正在改变。一个看似只是工具的问题，开始触及我们协作和作出决策的方式。

人们很容易宣告一场仿佛结局已知的革命。我不认为我们已经走到了这一步。

一些企业正在试验新的工作与责任分配方式。另一些企业推进得更加谨慎，有时也有充分的理由。某些实践看起来很有前景，但我们仍缺乏足够的时间来判断它们能否经受日常约束的考验。适合小团队的办法，也不会自动适用于规模更大、监管更严格或历史更悠久的企业。

因此，我并不想写一本关于企业应如何采用 AI 来建立“正确组织”的操作手册。本书试图理解真正发生变化的是什么：工具带来了哪些可能，企业如何回应，得到了什么，可能失去什么，又有哪些问题终究需要它们作出裁决。

为此，我选择了两条彼此交汇的路径。

第一条是观察。我们将考察真实企业、公开的实践、研究和实验。我会尽量区分已经得到证实的事实与仍然停留在假设阶段的判断，也不会把一则诱人的宣告误当成已经证明的结果。

第二条是一段故事。我们将跟随一家虚构企业 Malorean Corp.，看它的员工如何逐步发现这些新可能给工作带来的变化。他们并没有一份重塑组织的计划。他们有客户、有习惯、有紧急事项、有自己的信念，有时也会发生分歧。虚构故事不能证明什么，却能让我们从具体的人出发观察这些转变。

这两种视角都不可或缺。真实案例帮助我们避免把直觉误认为事实；故事则提醒我们，组织变化并不只发生在组织结构图上，它也会触及信任、能力、责任以及每个人的位置。

归根结底，接下来的每一页都贯穿着同一个问题：**当一家企业中的人们发现，自己能够做的事情远远超出原有岗位所允许的范围时，这家企业会发生什么？**

关于作者

我在软件行业工作了数十年，如今是一名产品经理。我的职业经历从开发和架构开始，随后扩展到团队组织，再到产品、产品运营与市场营销。

这段经历让我得以近距离观察人工智能的发展及其对我们工作的影响。我既在日常职业活动中亲身经历这种变化，也从与产品经理同事的交流中不断学习。我们分享经验、发现和疑问。这些对话常常促使我把目光投向工具之外：我们的岗位、决策以及共同工作的方式，究竟发生了什么变化？

随着观察和讨论不断积累，问题也越来越多。我希望退后一步，把我们的感受与其他企业正在发生的事情相互对照，并分享由此产生的思考。本书就这样诞生了。

写作长期伴随着我的职业生涯。我曾为专业媒体撰稿，也曾担任 M2 Édition 和 Eyrolles 的技术审校。如今，我在 malorean.net 撰写有关产品管理、AI、工作方法和信息系统的文章。

我也为独立杂志 *Proposition Marketing* 撰稿，该杂志关注战略与运营营销。面向来自市场营销和企业管理领域的读者写作，促使我用更容易理解的方式讨论产品问题，而不假定大家已经拥有共同的术语体系。

Laurent Deséchalliers

- laurent@malorean.net
- https://malorean.net

同一作者的其他作品



Le Contexte — 如何构建一种上下文，让 AI 理解真正重要的东西，连接你的知识，并把所学不断积累起来，服务于你的目标。

出版：*2026* 年 *10* 月



NO AI — 不要把理解交给 *AI* 不借助 AI 的方法，用来维护我们的记忆、深化我们的思考、锻炼我们的批判精神。这些实践帮助我们发展自身能力，更好地利用 AI，而不把理解交给它。

预计出版：*2026* 年底

更多信息请访问 https://malorean.net.

致谢

首先，我要感谢那些经常与我交流的产品经理。我们会相互比较自己对产品管理、开发岗位演变，以及更广泛地说，人工智能对工作和企业组织所产生影响的看法。这些有时颇为热烈的讨论，不断带来新的问题、反思和想法，其中一部分也出现在本书的篇章中。感谢你们的好奇心、分享的经验以及讨论的热情。

我还要感谢我的妻子 Christine。完成第一本书 *Le Contexte* 时，我曾答应她休息一阵，最后却几乎立刻开始了这一本。幸运的是，第二段旅程没有像上一次那样占用我们那么多夜晚和周末。谢谢你，Christine，感谢你的耐心和支持，即使我的思考又一次变成了一本书。

示例说明

本书的思考与示例源自三十多年的信息技术行业实践。它们来自我在不同项目和职业环境中遇到的情形，也来自我与其他企业的许多产品经理、CTO、工程经理等人的交流，横跨多个组织与我职业生涯的不同时期。

为了服务于本书的论述，我对这些情形进行了修改、概括或重新组合。使用场景、数字和指标用来说明反复出现的问题，以及我从中获得的经验。它们都是经过构建的案例：没有一个案例描述真实的客户、产品或组织，也不反映我雇主的数据、实践或立场。

本书表达的分析、方法和观点均属个人意见，仅由我本人负责。

这本书是用 AI 写的吗？

是的。整本书都是我与人工智能协作完成的。

我知道，这个问题如今具有特殊的分量。人们会问一篇文章、一部小说或一本书是否由 AI 写成。有些人甚至会把文本放进检测工具，仿佛一定要划清哪些内容来自人类、哪些来自机器。

我更愿意坦诚说明：从资料研究到写作，从整理观点、核查参考资料到寻找思维盲区，AI 参与了本书的每一个阶段。但这种协作需要进一步解释，因为它本身并不能说明究竟是谁思考了这本书。

过去，要完成如此规模的工作，人们可以请不同的人搜索资料、比较观点、审校、组织结构或调整形式。如今，AI 让我获得了其中一部分能力。它执行迅速，提供措辞建议，指出不一致之处，有时还会提出我没有想到的问题。它让我能在有限的时间里，探索远超独自工作所能覆盖的范围。

为了写作本书，我使用了一个基于 ChatGPT 技术的上下文引擎，并通过 Notion 保持信息的持久化。来源、观点、决策和工作版本因此可以在不同对话之间得到保存和重新找到。我已在第一本书 *Le Contexte* 中介绍过这种方法，并解释了这样的系统如何让 AI 成为一种超级私人秘书：它不再只是回答问题，还能持续掌握项目脉络，帮助我长期推进工作。

不过，有一个根本性的区别：**执行的能力并不等同于理解和选择的责任。**引导本书的问题、孕育这些问题的职业经验、解释、取舍，以及我最终决定捍卫或舍弃的观点，都属于我。AI 可以反驳我、挑战我，帮助我更准确地表达思想；但它既不能替代我的判断，也不能替代我作为作者的责任。

我也会区分本书与那些“写法本身”就是创作核心的作品。当我们阅读 François-Victor Hugo 翻译的莎士比亚、阅读塞万提斯或 Cizia Zykë 的原文时，声音、节奏和风格与故事内容同样重要。对于翻译作品，译者的选择也是阅读体验的一部分。人们希望让 AI 在这类文学中占据什么位置，会引出另一些问题，我无意在此作出裁决。

本书包含虚构成分，但它首先试图理解一场技术与组织变革。形式应当清晰、易读，更重要的是服务于我希望分享的观点、事实与疑问。

AI 让这个项目达到了一个仅凭我个人能力、在本职工作之外恐怕无法达到的规模。它让我能够推进本书这样的个人项目，同时不放弃这些项目所要求的思考水准。

归根结底，用这样的方式写一本探讨 AI 所引发变革的书，也颇为合乎主题。

Laurent Deséchalliers

目录

翻译说明 3

引言 4

关于作者 6

同一作者的其他作品 7

致谢 8

示例说明 9

这本书是用 AI 写的吗？ 10

Malorean Corp. 13

1 Malorean Corp.——不同寻常的一天 14
2 AI 极大提升了开发者的产出能力 18
3 Malorean Corp.——第二天 24
4 新的组织模式正在出现 34
5 Malorean Corp.——能做什么，应该做什么 41
6 角色边界日渐模糊，责任依然存在 52
7 Malorean Corp.——Alice 心中那些从未写下的知识 59
8 更多自主权需要更多框架，而不是更少 72
9 Malorean Corp.——两个合理的决定 83
10 AI 会让企业的中间层消失吗？ 94
11 Malorean Corp.——重新获得的时间 105
12 结语 117

Malorean Corp.

1 – Malorean Corp.——不同寻常的一天

Malorean Corp. 的总部位于勒阿弗尔。但自新冠疫情以来，这个地址已经无法充分说明工作究竟在哪里发生。员工分散在法国各地。大家在摄像头前见面，通过共享屏幕讨论；大多数时候，没有人需要待在同一个房间里，也能推动事情向前发展。

Malorean Corp. 开发一套在线 ERP。这类软件最终会渗透进企业几乎所有的运营活动。它的客户可能是一家只有几名员工、需求非常具体的小微企业；也可能是一个拥有复杂流程和特殊要求、对任何含糊处理都缺乏耐心的大型集团。

客户主要来自法国。德国和西班牙也有一些用户，但大多是法国现有客户集团的当地子公司。公司负责人 Hubert Félix Duca 看得更远。他希望 Malorean Corp. 真正走向欧洲市场。

困难在于，ERP 不像展示型网站那样容易跨越国界。各国的规则不同，薪资处理惯例、行政义务以及企业的工作方式也不一样。Hubert 很了解这些约束。他来自业务、管理、会计和财务领域，并非软件开发出身。在他看来，一项功能只有在客户能够使用而不陷入困境时才有价值。

公司内部的角色划分相当清晰。

两个各有五名开发者的团队负责持续改进软件。两名产品经理 Alice 和 Bob 与他们合作，理解需求、选择有用的改进，并为产品指明方向。Alice 做过开发者，因此能参与技术讨论而不至于跟不上。Bob 毕业于商学院，会从不同角度看待问题。两人的背景不同，却合作得非常顺畅。

销售团队负责寻找新客户并维护现有关系，另有一名市场人员提供支持。五名专家参与售前工作，随后在现场或远程协助部署。他们并不只是安装 ERP，还帮助企业明确需求、调整流程，并经常在销售完成后的六个月内继续提供支持。

客服团队有十二人，七人负责小客户，五人负责大客户。他们往往最先听到客户抱怨某项功能迫使自己绕开软件，而不是借助软件完成工作。两名

法务人员负责 Malorean Corp. 的法律事务，同时也为产品设计提供专业意见，尤其是当一项改动涉及薪资等敏感规则时。

这一切共同维持着公司的运转。当然，并不完美。部署在如此多客户处的软件，会积累大量请求、例外、难以改变的习惯，以及因为没有时间重做而继续保留的功能。但每个人大致都知道自己该做什么。

随后，ChatGPT 开始出现在人们的谈话中。

一开始，这并不是什么大型转型计划。几个月前，两个最好奇的人率先尝试了它：销售团队的 Julie Morel 和开发团队的 Lucas Martin。他们先从自己的用途入手，随后开始讲述自己能够做到什么。

在 Julie 身上，收益一目了然。她会准备一封难写的邮件，让工具帮忙改得更清楚；会在见客户前重新整理笔记；也会把一串想法变成演示文稿的大纲。有时，她还会准备几页幻灯片的内容。并不是 ChatGPT 决定应该对客户说什么。Julie 提供材料，修改建议，纠正措辞，最终只发送自己愿意负责的内容。

她并不声称这个工具比自己更了解客户。她只是很高兴，不必再花半个小时整理自己早已知道的事情。

同事 Philippe Garnier 在一旁观察。作为一名经验丰富、习惯与大客户打交道的销售人员，他了解合同背后的人，知道没有写进任何会议记录的分歧，也记得那些足以改变整场会议意义的只言片语。

“你应该用它写写邮件，”一天早上 Julie 对他说，“哪怕只用来整理格式也好。”

“我的邮件，我自己会写。”

他说这话并无恶意。只是他想不明白，为什么要把工作的一部分交给一个工具。在他看来，自己的价值恰恰来自对客户的了解，以及与客户说话的方式。

Julie 没有坚持。她还要准备一场会面。

Lucas 使用 ChatGPT 的方式完全不同。他是一名经验丰富、住在里昂的开发者，并不只想在重复性工作上节省几分钟。他花很多时间试验，提出越来

越难的问题，答案不满意就重新开始。Julie 首先想得到有用的结果；Lucas 则不断探索可能性的边界。

第一批用户的反馈最终传到了 Hubert 那里。每人每月只需二十欧元左右，账很容易算：哪怕每个人只是在繁琐任务上省下一点时间，这笔投入也完全值得。因此，Malorean Corp. 决定向所有员工提供 ChatGPT。

这并不是一场用大量演示文稿高调宣布的革命。谁觉得有用，谁就可以使用。有人每天打开，有人很少使用，Philippe 则完全不用。对大多数人来说，AI 仍然只是一个助手：人们说明自己的需求，它给出建议，人们核查，然后作出决定。

公司的生活照常继续。

星期三，产品团队与开发团队讨论时，一个老问题再次被摆上桌面：休假管理。

没有人真正喜欢这个模块。它年代久远、僵硬，选项很少。它还算能用，所以大家继续用着；但又没有好用到让客户不再抱怨。

员工要申请休假，必须通过办公电脑操作。星期天晚上想到假期时，无法直接用手机轻松提交。缺勤天数的计算还有一个更尴尬的问题：模块无法正确考虑法定节假日。如果申请周一到周五休假，而周三恰好是假日，它可能扣除五天而不是四天。

客服知道这个问题，销售也知道。在一些客户那里，这个模块已经成为反复引发不满的话题。它之所以还在使用，是因为已经存在，而不是因为真正符合期待。

“应该把它彻底重做一遍。”有人总结道。

没有人反对。但“彻底重做”意味着重新设计界面、处理计算规则、让整个流程能够在手机上使用，并替换一部分已经随时间老化的内容。

大家估计需要大约两个月。公司还有其他优先事项。这个议题又一次回到了“将来应该做”的清单中。

星期五下午，一场视频会议即将结束时，Lucas 请求大家再留几分钟。

“我有东西想给大家看看。”

他没有预告主题。没有人知道，自星期三的讨论以后，他一直在开发休假模块。

会议中出现了他的屏幕。一个清晰、现代的界面打开了，重要信息都出现在人们预期的位置。

Lucas 演示了如何提交休假申请，随后把窗口缩到手机宽度，界面随之自动适配。用户仍然可以继续操作、选择日期并确认申请，不必回到办公电脑前。

他又演示了第二个例子：从周一到周五的一周，周三是法定节假日。

系统显示扣除四天。

他修改日期，结果随即重新计算。他又回到移动端视图，滚动几个页面，连续进行多项操作。这并不是一组静态的界面模型：表单会响应，计算会执行，流程确实能够运行。

起初，几位同事还提出一些普通问题：某个选项在哪里？如果改动日期会怎样？随后，问题渐渐停了下来。

视频会议的小窗口里，人们的表情发生了变化。

星期三，他们还在谈两个月的工作。星期五，Lucas 已经展示了第一个可用版本。他没有向任何人求助，也没有提前通知谁。至少到这时，他还没有解释自己究竟是怎么做到的。

“等一下，”终于有人开口了，“这真是我们前天讨论的那个模块？”

Lucas 确认了。

一阵沉默。

屏幕前的每个人都看着同一件事：一项原以为需要数周规划的工作，两天后已经出现在眼前。

没有人明白这怎么可能。

2 – AI 极大提升了开发者的产出能力

几十年来，软件开发组织方式的很大一部分，都围绕着一个简单的约束建立：**生产软件成本高昂**。

把一个想法变成产品，需要时间、稀缺的技能以及大量协调。一项功能必须经过界定、拆分、排序、开发、测试、审查和部署。可用开发者的人数、他们的专业水平，以及他们能投入某个议题的时间，直接限制着企业能够开展的工作。

这种稀缺性塑造了组织。我们学会保护开发者的时间，提前准备他们的工作，减少打断，管理待办列表，并在启动开发前进行长时间取舍。现代产品运作方式有相当一部分正是在这一背景下形成的：必须避免把昂贵的工程资源浪费在一个错误的想法上。

代码助手的出现，以及随后能够执行更广泛任务的智能体，正在开始改变这个等式。

变化已不再只是更好的自动补全

最初面向开发者的 AI 工具，可以被理解为新一代生产力工具：补全一个函数、解释一段代码、编写测试、提出重构方案。

新一代工具则彻底改变了工作的性质。一个智能体可以接收任务、探索代码库、修改多个文件、运行测试、修正部分错误，并交付一项可供审查的变更。因此，一名开发者可以把执行工作中的很大一部分交给智能体，更重要的是，可以同时推动比过去更多的工作。

其结果并不只是“开发者写代码更快”。

一个人能够发起并推进的软件工作量大幅增加。

一些企业已经开始公布引人注目的成果。AMPECO 在介绍自身转向 AI 原生工程体系时，称其开发速度提高了四倍，每个用户故事的缺陷减少了 50%。这个数字不应被视为市场平均水平：它是一家公司在特定背景下自行公布的结果。但它说明了人们所追求的变化量级：目标已不只是为编写一个函数节省几分钟，而是大幅提升团队的执行能力。[1]

一家企业宣称取得的亮眼成果，并不是现有的唯一迹象。我自己也在使用 Claude Code 时有过亲身体验。四天之内，通过 180 多次提交，我构建出了一个包含界面、API 和测试的首个产品版本。按我过去的工作方式估算，这项工作原本需要一到两个月。2026 年 7 月，我在 Malorean.net 上讲述了这段经历。它当然不能用于概括普遍的生产力提升幅度：它展示了一位有开发经验的产品经理如今能够做到什么，同时也暴露了生成代码在质量、可维护性和掌控方面的困难。[7]

这让问题的量级发生了变化。

如果 AI 只能让编码效率提升几个百分点，组织或许可以保持不变，直接吸收这部分收益。但当产出能力的增长快得多时，其他约束的重要性就会相对上升。

加快编码并不足以加快企业

软件团队并不是一条只有“编写代码”这一个环节的流水线。

在开始开发之前，必须理解问题，决定它是否值得投入，选择方向，并澄清某些约束。完成开发之后，还必须审查、测试、保障安全、部署、观察和维护。而在整个过程中，必须不断做出取舍。

如果只有其中一个环节大幅提速，它只会把更多工作推向其他环节。

代码可以更快地产生，但对客户的理解未必能以同样速度加深。原型可以成倍增加，但判断哪个原型值得继续的能力并不会同步增长。更多变更可以进入审查流程，但可用于检查它们的人类注意力并不会按同样比例增加。

问题并不在于 AI 突然让其他活动失去了意义。

而在于它转移了瓶颈。

过去被执行缓慢所掩盖的因素变得更加显眼：所选问题的质量、意图是否清晰、取舍、审核、一致性，以及交付后的学习能力。

产出更多不等于创造更多价值

这一区分变得至关重要。

当生产本身很困难时，交付的软件数量似乎可以近似代表组织创造价值的能力。这个简化原本就不准确。到了 AI 时代，它变得危险。

如今，一个团队可以生成更多代码、更多 pull request、更多原型和更多功能，却没有改善产品。如果额外的产出增加了缺陷、返工、成本或维护复杂度，团队甚至可能降低自身的整体绩效。

换句话说：

制造的能力，不等于选择什么值得制造的能力。

因此，活动指标变得更加不足。变更数量、代码量，或只是使用了某个 AI 工具，都无法说明团队是否创造了更多价值。

Canva 提供了一个很有意思的例子，说明重心如何转向影响：该公司开发了一套系统，让产品、工程和数据团队能够估算实验的商业影响，并更直接地把局部决策与业务后果联系起来。[2]

于是，问题的性质发生了改变。

过去：

我们能够开发多少？

现在：

在如今能够开发的一切之中，究竟什么真正值得开发？

> 软件生产力问题已有四十多年历史
>
> *1986* 年，*Fred Brooks* 首次提出了 No Silver Bullet，该文于 *1987* 年刊登在 IEEE Computer。他区分了软件制造中的偶然困难——语言、工具、把设计转译为代码——与其本质复杂性：理解、设计、表达并持续演进一个系统。他认为，没有任何一项单独的技术或管理进步，能够保证在十年内将开发的生产力、可靠性和简洁性提高十倍。*[3]*
>
> 这篇文章后来收录于 *1995* 年出版的 The Mythical Man-Month 周年纪念版；该书初版于 *1975* 年。*[4]* 生成式 *AI* 或许会改变执行能力的量级，却不会自动保证所选问题是正确的。这是借助 *Brooks* 观点所作的当代解读，并不是他对 ***AI*** 智能体的预测。

新的瓶颈是判断力

随着执行成本下降，若干资源的相对价值会上升：客户知识、对问题的理解、战略背景、清晰表达意图的能力、风险评估、产品品味、说“不”的能力，以及对决策负责。

2026 年 3 月，我在 Malorean.net 发表的一篇文章中提出了这一观点：随着技术工具越来越容易获得，竞争优势正在转向对上下文的理解——业务、用户、客户、技术约束、竞争和监管。这是一项战略分析，而不是实验结果。它进一步说明了运用判断力意味着什么：对情境理解得足够深入，从而选择正确的开发对象，而不只是知道如何把它做出来。[8]

AI 让“怎么做”变得更容易。

它并不会自动让“为什么做”变得显而易见。

在组织尚未充分理解问题之前，它就能产出解决方案。它可以迅速实现一项客户请求，哪怕企业本应判断该请求并不符合自身市场。它也能生成一个功能的众多变体，却无法告诉我们哪个变体真正带来了更好的结果。

开发越容易，决定什么不做的纪律就越重要。

Ramp 很好地体现了这一原则：其首席产品官强调，必须明确优先服务的理想客户画像（ICP）、企业希望特别擅长解决的问题，并且在某些请求把产品拉向其他方向时，愿意拒绝它们——即使这些请求可以带来可观收入。[5]

2026 年 7 月，Marty Cagan 将这种错位称为“AI 生产力悖论”。他的判断是：更快地开发，并不能纠正那些原本就不值得开发的想法。他由此区分了为学习而制造的东西与为取得结果而交付的东西。这是一位产品从业者的分析，并非证明所有企业都以相同方式受到影响的测量结果。[6]

因此，AI 的悖论在于：

它大幅增强了我们的行动能力，也因此让行动的选择变得比以往更加重要。

一场系统变革，而不只是工具变革

面对这种变化，我们可以选择只是给每位开发者配发一个代码助手，然后要求团队照旧工作。

这样会带来局部收益。

但随着这些收益扩大，组织会碰到其他极限：理解、决策、协调、核验，以及消化新增产出的能力。

到了这一步，问题就不再只是采用一种工具。

它变成了组织问题。

一个在工程能力最为稀缺时建立起来的组织，当这种稀缺性下降后，没有理由仍然保持最优。

于是，一个问题变得不可回避：

当一种稀缺资源——工程能力——变得充裕得多时，一个原本为保护它而设计的组织会变成什么样？

参考文献

[1] How we built an AI-native engineering system – AMPECO

- URL：https://www.ampeco.com/blog/how-we-built-an-ai-native-engineering-system/
- 一家围绕智能体重建工程体系的组织所分享的经验；该企业称其能力获得了大幅提升。

[2] Measuring Commercial Impact at Scale – Canva

- URL：https://www.canva.dev/blog/engineering/measuring-commerical-impact-at-scale/
- 介绍 Canva 如何把产品与工程团队的实验，连接到一种易于获取的商业影响估算。

[3] No Silver Bullet: Essence and Accidents of Software Engineering – Fred Brooks

- URL：https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet
- 这篇文章于 1986 年提出、1987 年发表，区分了软件构建的本质复杂性与偶然困难。

[4] The Mythical Man-Month，周年纪念版，第 16 章 – Fred Brooks

- URL： https://www.oreilly.com/library/view/mythical-man-month-the/0201835959/ch16.xhtml
- 1995 年版的编辑目录，证明 *No Silver Bullet* 后来被收入书中；链接仅提供章节预览。

[5] 5 insights from Ramp's CPO on building for hypergrowth – Ramp

- URL：https://ramp.com/blog/5-insights-from-ramps-cpo-reddit-ama
- Ramp 首席产品官强调聚焦、优先 ICP，以及拒绝会分散方向的请求之必要性。

[6] The AI Productivity Paradox – Marty Cagan，Silicon Valley Product Group（2026）

- URL：https://www.svpg.com/the-ai-productivity-paradox/
- 2026 年 7 月的一篇分析，区分 AI 带来的速度收益与真正创造的结果；这是从业者的思考，而非因果证明。

[7] Quatre jours de vibe coding dans la peau d'un PM rouillé – Laurent Deséchalliers，Malorean.net（2026）

- URL： https://malorean.net/articles/2026-07-28-quatre-jours-de-vibe-coding-dans-la-peau-d-un-pm-rouille.html
- 使用 Claude Code 的个人经验：构建产品的速度、遇到的技术困难、测试与质量掌控；既不是受控研究，也不是可普遍推广的证据。

[8] Dans le logiciel, l'avantage ne sera plus la technologie. Ce sera la compréhension du contexte – Laurent Deséchalliers，Malorean.net（2026）

- URL： https://malorean.net/articles/2026-03-23-dans-le-logiciel-lavantage-ne-sera-plus-la-technologie-ce-sera-la-comprehension-du-contexte.html
- 作者关于稀缺性转向客户、业务、监管与技术上下文理解的战略文章，并非量化证明。

3 – Malorean Corp.––第二天

“我用了 AI。”

Lucas 说得很自然，仿佛这一句话就足以解释大家刚才看到的一切。

他仍在共享屏幕。新的休假模块占据了窗口的大部分。几分钟前，他演示了如何通过手机申请休假，以及软件如何在计算时考虑法定节假日。

“等等，”Alice 接着说，“你说用了 AI，是指让它帮你写代码吗？”

“是的。不过不只是写代码。设计模块、理解现有代码、开发界面、进行计算、编写测试……实际上，差不多什么都用到了。”

“而这些都是从周三开始做的？”

“周三下午开始。最后一点东西是今天早上完成的。”

Alice 看着屏幕上一张张面孔。有几个人在微笑。另一些人则像是在寻找自己漏掉了什么。

“但你没有规格说明。”她指出。

“没有。我们周三聊过这件事。我了解这个模块和它的问题。我先让 AI 分析现有内容，然后我们一点一点往前推进。”

“所有东西都能正常运行吗？”

Lucas 轻笑了一声。

“所有东西，我不知道。刚才演示的那些可以。肯定还有一些地方需要检查。”

和他同属一个团队的 Thomas 第一次开口。

“那背后的代码怎么样？”

“你可以看看。我做了不少测试。”

“我会看的，”Thomas 回答。

他看起来并没有特别吃惊。也可能是吃惊了，只是不想表现出来。

Hubert Félix Duca 一直没有说话。

他参加过足够多的演示，知道一套能顺利运行十分钟的软件，并不一定已经准备好供数百家企业使用。他了解各种例外、大客户的特殊需求，以及那些只有到了客户现场才会暴露的问题。

但他同样记得周三听到的答案。

两个月。

而眼前的东西，已经非常接近他们当时设想的模样。

“Lucas，”他问，“你能再告诉我，你在这上面花了多长时间吗？”

“大约两天。”

Hubert 微微凑近屏幕。

“好。”

他不知道还能补充什么。

几分钟后，会议结束了。每个人都回去继续工作，但谁也没有真正重新谈起自己下午的任务。

周六早晨，Alice 一边煮咖啡，一边回想那场演示。

让她不安的不只是 Lucas 的速度，而是他不需要任何人。

她非常熟悉产品经理这份工作。在启动开发前，她会努力理解问题，对照不同需求，与客户交流，并同开发者一起寻找合理的解决方案。

像休假模块这样的议题，她大概会组织好几轮讨论。她会帮助澄清规则、权衡特殊情况，然后决定首先开发什么。

这些事，Lucas 一样也没做。

他听过一次讨论，主动采取行动，然后构建了一个解决方案。

她一方面感到钦佩，另一方面却在想：如果所有开发者都开始这样工作，会发生什么？

然而，过去几个月里，她自己的日常工作也发生了巨大变化。

起初，Alice 用 AI 总结会议记录或改写文档。后来，她开始把更重要的任务交给它。她自动化了部分客户反馈处理、某些摘要的准备、相似需求的归类，以及过去会占用她数小时的若干行政操作。

如今，有些过去会占去大半个上午的工作，她几分钟就能完成。

她一点也不怀念这些任务。

恰恰相反。

她终于有更多时间与客户交流，理解他们的请求背后隐藏的问题，并思考究竟应该开发什么。

AI 并没有让她的工作变得更无趣。

它替她卸下了那些妨碍她把工作做好的事务。

既然如此，为什么开发者却应该继续完全照旧工作?

Alice 放下杯子。

这个问题在她看来再明显不过。

答案却远非如此。

Hubert 也花了周末的大部分时间思考。

他的公司一直有太多想法，却没有足够能力实现。许多功能已经等待了好几个月。客户要求的一些改进总是一再延期。还有欧洲扩张项目，它仍然意味着软件方面的大量工作。

他再熟悉不过那个障碍：他们没有足够的时间把所有事情做完。

周五，Lucas 或许已经表明，这个障碍不再那么难以逾越。

如果一个预计需要两个月的项目能在两天内产出首个版本，还有多少项目会突然变得可行?

Malorean Corp. 有十名开发者。Hubert 试着想象，如果每个人都能以同样速度推进，会发生什么。

接着，他又试着想象十名开发者各自启动自己的功能，彼此毫无协调。

第二幅画面就远没那么让他高兴了。

他周六就想过给 Alice 打电话，但忍住了。

周日晚上，他只是给她发了一条消息。

明天早上有时间吗？我想再谈谈周五的事。

几分钟后，回复来了。

有。我也想谈。

周一上午九点，Alice 和 Hubert 通过视频会议见面。

“所以呢？”Hubert 问。

“什么所以呢？”

“你弄清楚周五发生了什么吗？”

Alice 笑了。

“技术上，大致清楚。Lucas 用 AI 开发，速度远超我们的想象。”

“其他方面呢？”

“问题恰恰就在这里。”

Hubert 点点头。

“我整个周末都在想这件事。多年来，我们一直缺少产能。我们拒绝需求、推迟项目，把时间都花在决定哪些事情不做上。结果 Lucas 两天就拿出了一个我们估算要做两个月的东西。”

“只是首个版本，”Alice 指出，“我们还不知道它是否能上生产环境。”

“是，我知道。但即使还要两周才能完成，它也会改变很多事情。”

Alice 无法反驳。

“让我不舒服的是，”Hubert 接着说，“他做这一切都没有经过你。”

Alice 没有立刻回答。

“我也觉得不舒服。”她承认。

“你认为应该提醒他我们的工作方式吗？”

“我想过。”

“然后呢？”

“我觉得那会是个错误。”

Hubert 皱起眉头。

“可如果他开始自行决定开发什么，我们可能会遇到不少意外。”

“当然。但看看实际发生了什么。周三，我们说应该重做休假模块。如果按照平常的工作方式，我会先收集需求，我们会组织会议、准备议题、拆分工作……可能要过好几个星期，才能展示任何东西。”

“那不一定是浪费时间。”

“不是。我也不是说我们应该停止理解需求。但 Lucas 向我们证明，他不必等待这些工作完成，也能向前推进。”

“所以你建议让开发者自己做决定？”

“不，恰恰不是。”

Alice 想了几秒钟。

“我觉得我们混淆了两件事：决定应该做什么，以及决定怎么做。”

“两者相互关联。”

“是。但这不意味着我必须介入每一个决定。”

她稍稍坐直了身体。

“你知道吗，过去几个月里，我的工作发生了巨大变化。我每天都用 AI。我把许多过去手工完成的事情自动化了。现在，我花在准备文档或整理信息上的时间少了很多。”

“我知道，你跟我提过。”

“但你或许没意识到，它把我的工作改变到了什么程度。现在，我有多得多的时间去做真正重要的事：理解客户、识别问题，并决定应该把精力投入哪里。”

Hubert 听着。

“过去，我的部分工作是推动议题在组织中向前流转。准备工单、澄清需求、回答问题、协调沟通。假如明天开发者在这些事情上不再那么需要我呢？”

“你会有更多时间。”

“没错。他们也一样。”

她停顿了一下。

“我们接受 AI 改变我的工作。为什么却希望开发者的工作保持不变？”

Hubert 靠上椅背。

“好吧。但总得有人决定要开发什么。”

“当然。更重要的是，为什么要开发。”

“那你怎么看？”

Alice 犹豫了。

她没有准备演示文稿。她提不出任何组织模型，也没有可供部署的新流程。她只有一个直觉。

“我在想，我们是不是应该尝试一些不同的做法。”

Hubert 等着她继续。

“我们可以把一个议题交给 Lucas。一个真正的客户问题。然后给他更大自由，让他自己解决。”

“没有规格说明？”

“不把现成解决方案交给他。我们向他解释问题，让他接触了解问题的人，然后让他自己探索。”

“那你呢？”

“我会随时提供帮助。我帮助他理解需求、质疑他的想法，避免他仅仅因为能开发就去开发某个东西。但我不会规定每一步。”

Hubert 思考着。

“所以你想让他承担一部分产品经理的工作？”

“也许。或者说，我们只是停止一味地在彼此之间切割工作。”

“如果失败呢？”

“我们会看到的。”

Hubert 笑了。

“这个方法听起来有点单薄。”

“我知道。”

Alice 也笑了。

“但我宁愿在一个有限的议题上试一试，也不愿花三个月讨论一种新的组织方式。”

Hubert 没有立刻回答。

接着，他像是想起了什么。

“正好，一些食品行业客户这几周一直在反映一个问题。”

Alice 点点头。她很清楚他说的是哪一个。

Malorean Corp. 的 ERP 可以管理库存和生产作业，却无法准确追踪制造成品时使用的原材料批次。

在食品制造企业里，一旦供应商报告某批原材料存在缺陷，就必须能够迅速找出用这批原材料生产的产品。

目前，Malorean Corp. 的客户只能自行重建这些信息。

“上周我又收到了一次反馈，”Hubert 解释道，“一位客户告诉我，因为供应商发出警报，他和质量负责人花了整整一天才找出受影响的产品。”

“是。客服向我们反馈过好几个类似案例。”

“你觉得 Lucas 能处理这个问题吗？”

Alice 想了想。

这个议题比休假模块更棘手。它不是做几个界面就够了。必须理解车间里实际发生了什么、原材料如何被使用，以及客户怎样识别自己的产品。

但她感兴趣的恰恰就是这一点。

“我觉得可以把这个问题交给他。”

“你认为他对这个行业了解得够多吗？”

“大概不够。”

Hubert 扬起眉毛。

“真让人放心。”

“我也不了解所有细节。但我知道该和谁谈。我们的客户非常了解自己的约束。客服和实施人员每天都在看到这些困难。Lucas 可以去找他们。”

“你想让他直接和客户谈？”

“为什么不呢？”

“通常，那更像是你的工作。”

“正因为如此。”

Hubert 看起来并没有完全信服。

“如果他做出的东西无法满足需求呢？”

“我的职责就是帮助他避免这种情况。但不是给他写一份五十页的规格说明。”

“如果他想把自己的方案投入生产呢？”

“它必须像软件的任何其他改动一样，经过检查和验证。给他更多工作自由，并不意味着允许他承担所有风险。”

Hubert 沉默了片刻。

“你知道我担心什么吗？”他终于问道，“我担心的不只是他可能犯错，而是我们会再也搞不清谁负责什么。”

“我也担心。”

“你还是想试？”

“是。”

她斟酌片刻，补充道：

“因为另一个选择，就是继续完全照旧，只要求开发者写代码更快。我不确定这是不是 Lucas 周五真正向我们展示的东西。”

他们又讨论了大约二十分钟。

Hubert 不想干扰正在进行的项目。Alice 不希望这次实验还没开始，就变成一项向全公司高调宣布的计划。

于是，他们决定一切从简。

Lucas 将处理批次追溯问题。他可以自由探索解决方案，并直接接触那些能够帮助他理解需求的人。

Alice 将担任他的产品对接人。她不会为他准备详细的待办列表，但会继续参与重要决策。

他们将从有限的范围开始，不向任何客户承诺交付日期。

最重要的是，他们会花时间观察接下来发生什么。

“不过，我们还是应该定义怎样才算实验成功，”Hubert 指出。

“是。而且我不希望只用他开发了多少功能来衡量。”

“那你想衡量什么？”

“首先，看他能不能理解问题并提出有用的方案。然后，看在不需要我们替他组织一切的情况下，他能推进多远。”

“还要看这件事占用我们多少时间。”

“也对。”

Hubert 看了看时间。

“好。我们试试。”

Alice 笑了，但心里并没有完全踏实。

“试试吧。”

“你今天和他说？”

“是。”

视频会议结束了。

Alice 在屏幕前又坐了一会儿。她打开通讯软件，找到 Lucas。

她开始输入：

你好，*Lucas*。今天有时间吗？我想给你提一个和我们平常做法稍有不同的议题。

发送前，她又读了一遍消息。

这是很长时间以来，她第一次把一个问题交给开发者，而自己还没有开始具体设想解决方案。

她还不知道这是不是个好主意。

而她也不再确信，自己的职责是否就是比他更早知道答案。

4 – 新的组织模式正在出现

AI 让更多执行成为可能。但企业会如何重新组织团队与决策？它们给出的答案并不相同。

有些企业加强开发者周围的产品管理力量。另一些则扩大开发者的职责，让非技术业务人员拥有更多手段，或围绕由智能体辅助的小团队重组工作。

历史形成的分工不再理所当然。

传统模式原本有其逻辑：产品管理集中掌握大量客户与业务上下文，设计把体验具体化，工程则把这些意图转化为软件。工单、规格说明、工作交接和会议，让这些职能能够围绕有限的开发能力保持同步。

这种组织方式并不会突然变得糟糕。

但随着执行能力提升，一些企业开始以不同方式重新安排人员、上下文和工作。

因此，眼下还不是寻找“正确模式”的时候。

我们首先要观察正在出现的主要应对方式。

加强开发者周围的产品力量

最直观的应对方式，是保留产品与工程之间相对传统的分工，同时增加围绕开发者配置的产品能力。

Gamma 提供了一个有趣案例。该公司称，其比例约为每四名工程师配备一名产品经理，而人们经常提到的历史比例更接近一比十。[1]

在这种模式中，工程能力的提升伴随着产品管理密度的增加。

其选择在原理上很简单：既然能够开发更多东西，就投入更多能力去理解、排序并引导这些工作。

让开发者承担更大范围的议题

另一种策略是扩大工程师的职责范围。

开发者不必再等待一个问题被完整转译成解决方案，再拆成工单。他可以更多地参与理解需求、制作原型、进行局部取舍和追踪结果。

Linear 在 Product Engineer 的招聘启事中这样描述岗位职责：从想法到交付产品，全程负责一项功能，并参与问题定义，而不是不断依赖产品管理。[2]

这种模式试图通过让同一个人掌握更多上下文和行动能力，减少职能之间的某些交接。

它并不意味着每个开发者都变成产品经理。

它描述的是在某些议题上扩展工程角色。

让不同职能跨越过去的边界

这场变化并不只涉及开发者。

Figma 在一项关于角色演变的研究中指出，64% 的受访者认为自己至少承担两种角色，超过三分之一的人表示自己的职责横跨多个职能。这个结果不能只归因于 AI，但它确实呈现了更不僵化的角色边界。[3]

Atlassian 也记录了一些工作流：设计师直接在真实产品代码中工作，并在工程审查前自行发起 pull request。[4]

Cursor 则描述了一种让产品、设计与工程都更加靠近实际构建的组织方式。[5]

因此，真实情况与其说是“所有人什么都做”，不如说是**更多人开始在自己历史形成的职责范围之外采取行动**。

这种边界渗透随即带来另一个问题：如果多个职能都能在同一领域行动，如何保持责任清晰？

我们稍后会回到这个问题。

把更多工作交给对结果负责的小团队

还有一类做法，是让更少的人承担更大的范围。

AMPECO 描述了一种“对齐同步会”：一名工程师和一名 Product Owner 先定义意图，随后让智能体承担很大一部分分析、开发与测试工作。[6]

Taito 描述的工作方式则是，团队先批准一个计划，再由一两个人借助智能体推动构建。[7]

在这些模式中，一个小型人类单元可以推动比过去更多的工作。

因此，可以观察到的组织变化，是职能间的交接可能减少，以及议题会更加集中到一个人或一个小组周围。

尚待定义的是：这些人可以自行决定什么，以及何时必须征询领域主管。

将部分检查自动化

生产加速也会给审核机制带来压力。

一些企业通过自动化更多检查，或根据风险对变更采取不同处理方式来应对。

Meta 使用 Diff Risk Score 来估算一次变更引发事故的风险。[8]

Cloudflare 把工程标准明确化，使其能够在合并前自动检查。[9]

Taito 同样结合了确定性规则、审查智能体和风险触发条件。[7]

现阶段需要注意的只是：**审核不一定仍要以统一方式组织。**

如何组合自动化、人类责任与上报阈值，将在后文详细讨论。

提供上下文，而不是规定每一步

另一些企业走得更远：它们质疑是否还应把详细工单作为协调工作的主要手段。

Linear 明确批评这样一种模式：产品经理界定问题，把它写进系统，然后等待工程团队执行。该公司强调，应该让人类与智能体都能获取客户反馈、想法、战略方向、决策和代码。[10]

Cursor 同样强调，目标必须足够清晰，才能支持更多局部主动性。[5]

因此，可以观察到的变化并不是所有协调都消失了。

而是部分协调从详细规定转向了**更丰富、也更易直接获取的上下文。**

这些上下文应该包含什么、如何维护，以及如何定义自主权的边界，将由专门章节讨论。

让了解需求的人亲手构建

有些企业甚至允许了解需求的团队自行制作小型工具，不再总是进入开发者的排队序列。

Shopify 创建了 Quick，这是一个内部平台，让更多员工能够快速发布网站和小型工具。[11]

Vercel 描述了一种模式：非技术员工在安全环境中构建内部工具，同时不会让开发者偏离核心产品的优先事项。[12]

Atlassian 也展示了设计师先修改代码、再接受技术审查的做法。[4]

在这些案例中，AI 不只是让开发者变得更快。

有时，它让执行能够转移到**本就掌握需求上下文的人手中**。

一个比 AI 更古老的原则

早在 2018 年，Marty Cagan 就提出了他所谓的“赋权产品团队”（*empowered product teams*）：管理层交给跨职能团队的是需要解决的问题，而不是需要交付的功能清单。他要求团队结合产品、设计与工程能力寻找解决方案，并对结果负责，而不是对产出数量负责。[13]

2020 年，他与 Chris Jones 在《EMPOWERED》一书中进一步阐述了这种方法。这种模式既没有预言各职能可以彼此互换，也没有宣告管理者的终结；恰恰相反，它以共享愿景、扎实能力和管理者对团队的支持为前提。[14]

因此，这一理念并不是等到智能体出现才诞生。AI 可能改变的是，自主团队在选定探索方案后，能够以更快、更直接的方式执行。**产品自主权是一项更早的追求；AI 改变了实现它的手段与约束**。

尚未趋同的选择

这些应对方式并不是同一场转型的不同阶段。加强产品力量与让开发者承担更多产品责任，看起来甚至彼此矛盾。然而，它们可以在不同企业中有效，也可能在同一组织内共存。

它们的共同点在别处：**当执行变得更容易获得，如何分配上下文、决策和责任，就重新成为一种选择**。

接下来还要看看，这种转移会给相关人员的日常工作带来什么，而不只是观察企业公告和组织架构图。

我们很快会回到 Malorean Corp.，观察这些模式如何改变一家法国小企业的生活。随后，我们将研究这种自主权所需的责任和规则。

参考文献

[1] The bottleneck keeps shifting: What AI is changing about how we build – Atlassian / Gamma

- URL：https://www.atlassian.com/blog/how-we-build/the-bottleneck-keeps-shifting-what-ai-is-changing-about-how-we-build
- 描述当工程产能提升时，Gamma 如何提高产品管理的配置密度。

[2] Product Engineer – Linear Careers

- URL：https://linear.app/careers/0c7c2e26-0a98-42cf-a47c-9a3999fb513b
- 一则招聘启事，介绍 Product Engineer 从问题定义到交付所承担的职责。

[3] Are Roles and Responsibilities a Thing of the Past? – Figma

- URL：https://www.figma.com/blog/2025-shifting-roles-report/
- 记录角色边界日益渗透，以及职责如何扩展到传统职能边界之外。

[4] Designers' workflow for shipping code – Atlassian

- URL：https://www.atlassian.com/blog/ai-at-work/designers-workflow-for-shipping-code
- 描述设计师如何在工程审核前直接修改产品代码。

[5] How Cursor Is Navigating the New Shape of Product Craft – Atlassian / Cursor

- URL：https://www.atlassian.com/blog/how-we-build/how-cursor-is-navigating-the-new-shape-of-product-craft
- 展示一种让产品、设计与工程更接近实际构建的工作方式。

[6] How we built an AI-native engineering system – AMPECO

- URL：https://www.ampeco.com/blog/how-we-built-an-ai-native-engineering-system/

- 描述一种先由少数人界定意图，再把大部分执行工作交给智能体的组织方式。

[7] Inside an AI-native engineering organization – Swarmia / Taito.ai

- URL：https://www.swarmia.com/blog/inside-an-ai-native-engineering-organization/
- 描述一个小团队驾驭智能体、并把大量执行集中在一两个人身上的组织。

[8] Diff Risk Score: AI-driven risk-aware software development – Meta

- URL：https://engineering.fb.com/2025/08/06/developer-tools/diff-risk-score-drs-ai-risk-aware-software-development-meta/
- 介绍一种用于按照变更潜在关键程度区分审核工作流的风险评分。

[9] How Cloudflare enforces engineering standards using AI – Cloudflare

- URL：https://blog.cloudflare.com/engineering-standards-enforcement/
- 展示明确的工程标准如何转化为开发工作流中的自动检查。

[10] Issue tracking is dead – Linear

- URL：https://linear.app/next
- 主张一种超越孤立工单、让反馈、战略、决策与代码共享更多上下文的模式。

[11] Quick: An internal hosting platform for the AI era – Shopify

- URL：https://shopify.engineering/quick
- 介绍一个让更多员工直接构建并发布内容的内部平台。

[12] Stopping the slow death of internal tools – Vercel

- URL：https://vercel.com/blog/stopping-the-slow-death-of-internal-tools
- 展示业务用户如何构建内部工具，同时不让工程团队偏离核心路线图。

[13] Empowered Product Teams – Marty Cagan，Silicon Valley Product Group（2018）

- URL：https://www.svpg.com/empowered-product-teams/
- 一篇奠基文章，区分了对解决问题负责的团队与仅限于交付功能的团队。

[14] EMPOWERED: Ordinary People, Extraordinary Products – Marty Cagan 与 Chris Jones（2020）

- URL：https://www.svpg.com/books/empowered-ordinary-people-extraordinary-products/
- 介绍关于自主产品团队、所承接目标、战略上下文以及管理者培养团队职责的一本书。

5 – Malorean Corp.——能做什么，应该做什么

周二早上，Lucas 先问了一个让 Alice 意外的问题。

“我可以和客户谈谈吗？”

前一天，他收到了客服保存的几份有关批次追溯问题的记录。消息里提到生产编号、原材料、Excel 文件和供应商警报。他借助 AI 花了一晚上整理这些内容。

“当然可以，”Alice 回答，“为什么？”

“因为我还不完全明白他们缺少什么。”

Alice 笑了。若是在一周前，她大概会先亲自负责这次会谈，再把结论转交给开发团队。如今，她已经答应让 Lucas 主导这个议题。

“我会把负责这个客户的客服同事介绍给你。她可以安排交流。”

于是到了周三，Lucas 发现真实情况远比“管理批次”这句话复杂。

客户的质量负责人向他们展示了一封供应商来信。某种原材料必须被隔离。供应商知道它的批次号，制造商也知道收到的是哪些订单。但若要精确找出受影响的成品，团队必须交叉核对多份文档。

“上一次，”质量负责人解释，“我们四个人核对文件，花了大半天。即使这样，后来还得回到车间，才能确认无误。”

Lucas 要求看看生产时如何记录信息。客户向他展示了 ERP 中的一个界面，接着又展示了一张电子表格。

“可为什么还要把这个编号记在 Excel 里？”

“因为你们的软件查不出我们需要的东西。”

回答没有攻击性。正因如此，反而更加刺耳。

通话结束时，Lucas 记了三页笔记，心里的确定性却比刚开始时更少。

“怎么样？”他回拨后，Alice 问。

“我原本已经开始设想一个用于录入批次号的界面。”

“现在呢？”

“现在我觉得，最重要的是先弄清他们如何记录实际消耗的原材料。否则，我们会得到一个漂亮界面，却仍然无法回答他们的问题。”

Alice 点点头。她周一就可以告诉他这一点，但她并不遗憾他亲自发现了它。

下一个周五，Lucas 在演示数据库上向她展示了首个版本。

用户可以把一个原材料批次关联到一次生产，再找出由它制造的成品。他还添加了一个小示例，用来追踪某个批次，直至找到受供应商警报影响的产品。

Alice 点击了几次。

“这些是你这周做的？”

“是。还有很多错误或缺失，但至少有东西可以拿给客户看了。”

“如果再加上反向追溯呢？”

“已经有了。”

他打开另一个视图。

Alice 笑出了声。

“果然。”

自从休假模块演示以来，她设想项目的尺度已经改变。过去会被她推迟到下个季度的事情，如今似乎近在咫尺。

“还可以允许按供应商搜索。并直接针对相关订单发出警报。为什么不连召回文件也一起准备好呢？”

Lucas 没有立刻回答。

“这些想法可以记下来，”他最后说道，“但我想先带着现有内容再去见一次客户。”

“难道还要等十五天，才给他看一个更完整的版本？”

“我担心的不是演示。”

“那你担心什么？”

这一次，Lucas 共享了一个不那么赏心悦目的画面。陈旧的数据表、名称完全看不出含义的字段，以及生产与库存之间的多个关联。

“我们想做多少界面都可以。问题是，我们的 ERP 从来就不是为了保存完整的批次历史而设计的。有些库存变动已经存在很多年，还可能被手工修正过。有些信息根本就没有记录。”

“AI 可以帮我们把这些整理好，不是吗？”

“它可以帮我编写迁移程序，但无法凭空创造我们从未记录过的信息。”

Alice 沉默了。

“如果想把这件事做好，”Lucas 接着说，“就必须修改部分数据模型，并检查这会给其他模块带来什么变化。我们不能为了省下两周，就冒险让客户的库存数据失真。”

“我没让你破坏软件。”

“我知道。但从上周五起，每次我向你展示一样东西，你都会再向我要三样。”

这句话没有攻击性，却仍然刺痛了 Alice。

“确实如此。”她承认。

“我也想加快速度，你知道的。但我能在一小时内生成代码，并不意味着 ERP 十五年的运行历史就变简单了。”

他犹豫一下，又补充道：

“Thomas 看过休假模块的一部分代码，发现了两个我们处理错误的案例。他是对的。我们已经修正了，但这提醒了我一件事：我们的构建速度可以快很多，却依然需要熟悉那些陷阱的人。”

Alice 想起演示那天的 Thomas，他只是简单地要求查看代码。

“那你建议怎么办？”

“继续用虚拟数据演示，确认我们正确理解了需求。但要动真实库存之前，我希望先花时间理解现有系统，并完成必要检查。”

Alice 看着另一个屏幕上仍然打开的首个版本。

前一周，她还曾在 Hubert 面前为 Lucas 的自主权辩护。如今她忽然发现，这份自主权还包括对她说“不”的权利，而这就没那么令人愉快了。

“好，”她最终说道，“暂时不碰真实数据。”

接下来的周一，Hubert 用一个如今已很熟悉的问题开始每周例会。

“我们的实验怎么样？”

“在推进，”Alice 回答，“有些方面比预期更快，另一些更慢。”

“什么意思？”

Alice 向他展示 Lucas 的演示，随后概述了周五的讨论。

Hubert 特别留意那些技术保留意见。

“他小心一点是对的，”他说，“我可不想我们解决了追溯问题，却给所有客户制造库存问题。”

“是。我想我有点催他了。”

“你？”

“这两个星期，我感觉一切都变得可能。每当一个想法出现，我就觉得可以立刻试一试。”

Hubert 笑了。

“看完他的第一次演示后，我差不多也是这种心态。”

“问题是，我们很容易把在屏幕上得到一个东西的速度，和得到一件可靠东西的速度混为一谈。”

“至少，”他说，“Lucas 知道什么时候应该慢下来。”

“是。我需要在这一点上信任他。”

两天后，Julie Morel 问 Alice 能不能给她五分钟。

Alice 很熟悉这种说法。对 Julie 而言，五分钟很容易变成半小时。

“上周我见了一个相当重要的潜在客户，”Julie 开始说道，“他们有好几家工厂，正在寻找一套能适应自身工作方式的 ERP。看看我准备了什么。”

她共享了一组非常精致的原型图。不同订单类别有不同审批步骤，每个经营场所有自己的规则，界面也会根据用户而改变。

“这是你做的？”Alice 问。

“和 ChatGPT 一起做的。我把会谈笔记整理出来，请它设计一套流程，之后我们又一起修改。我花了一个晚上。”

Alice 立刻看出这些画面的价值。短短几小时内，Julie 就把一个原本困在销售记录里的需求变成了具体形态。

“客户看过这些原型吗？”

“还没有。我想先给你看。但他明确说过，如果我们能做这种东西，可能会改变谈判的走向。”

Alice 翻看着各个界面。

“这个想法很有意思。你应该直接和 Lucas 谈。”

Julie 惊讶地看着她。

“直接谈？我以为你会先准备这个议题。”

“为什么？你了解客户，和他交流过，甚至还准备了原型。你比我更能解释他在寻找什么。”

“Lucas 可以看看是否可行？”

“可以。不过 Julie，要注意：评估是否可行，并不意味着我们承诺开发。”

“当然。”

Alice 熟悉这个“当然”。它不是谎言，而是一位销售面对可能成交时的热情。

第二天，她们和 Lucas 开会。Julie 逐页展示原型，解释潜在客户的要求，并回答问题。

Lucas 立刻对不同经营场所之间规则的差异产生了兴趣。

“ERP 中已经有一部分这样的机制，”他说，“虽然并不是专门用来做这件事，但它确实存在。我们或许可以准备一些东西来测试。”

“需要多长时间？”Julie 问。

“只做演示？不一定很久。”

Julie 看向 Alice。她的笑容几乎像是已经拿下了这笔订单。

“如果下周就能给他看，真的可能带来决定性影响。”

Alice 感觉自己也想立刻推进，差点不假思索地回答。她想到了正在进行的追溯项目、尚未处理的需求、这位潜在客户的特殊规则，以及其他那些可能完全用不到这些功能的客户。

“不。”她说。

另外两人都转向她。

“什么不？”Julie 问。

“不能为了赢得这份合同，就开始在产品中开发这个功能。”

“可你刚才还说它很有意思。”

“确实很有意思。但我们不知道，这位客户需要的是所有客户都应该拥有的功能，还是只适用于他几家工厂的特殊运作方式。”

“我们可能会丢掉这笔生意。”

“也许。”

Julie 显然没有料到这个回答。

“我们终于有能力快速响应这类要求，”她坚持道，“不用岂不可惜。”

“我不是不想用这种能力。我想先弄清我们在提供什么。如果把每一项特殊要求都加进 ERP 核心，最后软件会变得无法演进。”

“可 Lucas 说可以做。”

“Lucas 说他可以准备一场演示。这不是一回事。”

一阵沉默。Julie 最后又看了一眼自己的原型。

“那我该怎么告诉他？”

“告诉他，我们理解了他的问题，想向他展示一种可能的方向。不要说我们已经决定把这个功能加入产品。”

Julie 点点头。她看起来有些失望，却并未泄气。

“好。我可以安排一次会谈，让他更具体地解释几家工厂之间的差异。”

“很好。原型也保留着，它们会帮助我们提出正确的问题。”

Julie 离开通话后，Alice 与 Lucas 又单独待了几秒钟。

“你真的能很快做出来？”

“首个版本的话，可以。”

“你想做吗？”

“说实话？想。看起来挺有趣。”

Alice 笑了。

“我也很想说‘好’。”

“是什么拦住了你？”

“我们两个当时谈的几乎全是能做什么，却不再谈应该做什么。”

下一周，Lucas 给 Alice 发来一个链接和一条消息。

我昨天随手做了个东西。你能看看吗?

Alice 打开页面。她认出了 ERP 的界面，只是旁边多了一个小面板，可以修改某些规则。她可以选择经营场所、改变审批阈值、显示或隐藏某个步骤，然后重新运行演示。

她回拨给 Lucas。

“这是什么？”

“一个小工具，用不同规则准备演示。我复用了旧的配置机制，只部署在测试实例上。”

“花了多长时间？”

“也许两个小时。回想 Julie 的原型时，我有了这个想法。”

“可这根本不是你正在做的议题！”

“不是。我只是想看看，能否避免每次测试一个变体都要修改代码。”

Alice 开始更改参数。界面随之响应。她又试了第二种配置，然后是第三种。

“等等。你的意思是，我可以自己准备几个不同的演示版本？”

“对于我们开放出来的规则，可以。不能随便改什么，尤其不能在客户环境中这样做。”

“但我可以给 Julie 看两条不同流程，问她哪一条真正符合潜在客户的情况。”

“完全正确。”

Alice 在页面前停留了片刻。第一个休假模块让 Lucas 变得更快。这个工具则赋予了她一种昨天还没有的能力。

她可以尝试一个想法，而不必要求开发者专门为她实现。

“你能想象我们用它可以做什么吗？”

Lucas 笑了。

“我想可以。不过，在你让我把它做成完整产品之前，我先提醒你，这只是花两小时随手拼出来的东西。”

“保证不会。”

她挂断电话，又继续摆弄那些参数。

接下来的周一，Hubert 注意到 Alice 没有打开平时使用的跟踪文档。

“今天什么都没准备？”

“准备了。但我想给你看一样东西。”

她向他展示 Lucas 的工具，并当场制作了同一个演示的两个变体。

Hubert 很快就明白了。

“这些修改不是你让开发者做的？”

“不是。Lucas 做了一个小工具。我可以自己测试某些规则。”

“能让销售使用吗？”

“或许以后可以。目前还不够稳健。但 Julie 已经可以和我一起处理她那位潜在客户的案例。”

Hubert 饶有兴趣地看着屏幕。

“批次项目呢？”

“Lucas 在推进。他带着首个版本再次见了客户。我们对需要做的事情理解得更清楚了。但在确保处理方式安全之前，他拒绝触碰历史数据。”

“听起来很合理。”

“是。周五我还为此恼火。现在，我很庆幸他坚持了。”

Alice 用手指转动着笔。

“有件奇怪的事正在发生，Hubert。Lucas 和客户讨论，并对解决方案作出决策。Julie 带来了几乎可以使用的原型。现在，我也能在不经过专门开发的情况下，测试 ERP 的某些变体。”

“你觉得担忧吗？”

“有时候会。我在想，如果每个人都开始承担一部分别人的工作，我们的职业会变成什么。”

“找到答案了吗？”

“还没有。”

她想了想。

“我只知道，我不想成为每个想法都必须经过的人。Julie 没有理由等我把她的原型转化成工单。Lucas 也不需要我告诉他每个界面该怎么构建。”

“但你拒绝了开发那位潜在客户的需求。”

“是。因为那是关于我们产品的决策。我可能判断错，也仍然可以改变主意，但这个选择应该由我承担。”

“而 Lucas 拒绝在库存问题上走捷径。”

“因为他知道我们可能破坏什么。在这件事上，他有一种我必须听取的判断力。即使我很着急。”

Hubert 看了看小窗口里的 Alice。

“所以，我们可以让人们做更多事情，却不能让所有决策都漂浮在他们之间。”

“我开始相信是这样。”

她犹豫片刻，又补充道：

“过去，我有时觉得自己的工作很重要，因为许多议题都要经过我。现在，我在想，也许最重要的是最后得到什么。我们解决的是正确问题吗？构建的东西可靠吗？这件事值得做吗？”

Hubert 点点头。

“如果没人知道谁应该回答这些问题，我们就有麻烦了。”

“是。”

他们沉默了几秒。两个人都不想把这场谈话变成一份新组织模式的公告。

他们的实验才刚刚开始。它带来了更多可能，也带来了更多需要作出的决策。

Alice 收到一条通知。Julie 发来消息：潜在客户愿意查看不同的演示版本，同时也想知道它们何时能供自己的团队使用。

Alice 没有立刻回复。

软件现在可以更快地制造答案。

但仍然需要有人决定，Malorean Corp. 准备承诺哪些答案——并为之负责。

6 – 角色边界日渐模糊，责任依然存在

长期以来，产品、设计和工程之间的边界一直充当着简单明确的坐标。

产品团队理解问题，决定开发什么；设计团队构思体验；工程团队把这些意图转化为软件。现实中的分工从未如此整齐，但它仍然塑造着团队、招聘、流程与责任。

这种分工在 AI 出现前就已受到质疑：第 1.4 章描述的自主产品团队，会让产品、设计和工程围绕一个共同问题协作。如今，AI 又在加速跨越另一道边界——执行的边界，因为每个人都能完成更多过去专属于某一专业岗位的任务。

第 1.4 章的组织全景已经展示了这种演变，尤其是 Figma 的研究。[1] 我也曾在 Malorean 上描述过开发者走向问题理解、产品经理走向方案实现的双向运动。[2]

人们很容易由此推断角色将会消失。但我认为，这场变化更需要我们重新表述每个人的责任：**角色边界越具有渗透性，责任就越必须明确**。

行动能力不再只属于某一种专业岗位

AI 首先重新分配的，不一定是决策权，而是行动能力。

以设计师自行修改代码为例，Atlassian 已记录过这种做法：技术团队仍然会审查质量以及投入生产的条件。[3] 这一步并不是回归旧有分工；它说明，修改代码的能力与对代码可靠性负责并不是一回事。

Linear 的 Product Engineer 招聘启事也体现了同样的区别：公司希望这个岗位能够从初始问题一直负责到结果交付。[4] 但这并不会自动赋予他决定所有产品战略问题的权力。

因此，我们可以减少交接，同时保留曾使某些交接有必要的专业能力。问题不再是统计有多少专业岗位参与，而是要明确：**当多个专业岗位在同一个领域采取行动时，谁对什么负责**。

能够贡献，并不意味着对该领域负责

必须区分组织经常混淆的两件事：行动许可与责任。

销售人员可以构建内部工具，但这并不会让他成为平台安全负责人。

产品经理可以修改界面，但这并不意味着他要对设计系统的一致性负责。

开发者可以决定局部的产品行为，但这不会自动让他承担战略、定位或整体路线图的责任。

许多团队中本已存在这种区别，但 AI 迫使我们将其正式化。

在高度专业化的运作方式中，专业岗位与责任往往相互重合。能够行动的人，通常正是该领域的负责人。一旦行动能力扩散，这种重合就会消失。

此时的风险，是把自主权与责任稀释混为一谈。

关于质量，我曾提出过一个类似原则：开发者不能在交付之后，就认为缺陷属于 QA 团队、缺陷轮值人员，或下一位有空的开发者。对所交付内容技术质量的责任，始终应该能够明确到人。[5]

角色边界的渗透性并不会改变这个原则，只会让它更加重要。

守护人的概念

为了解决这个问题，我使用了**守护人**这一概念。

每个重要领域，都有一个人或一个团队负责守护其一致性与质量。

守护人不一定亲自完成该领域的全部工作。

他也不是必须批准每一个微小决策的人。

当一个重要选择涉及该领域、当某条边界被跨越，或者当事情出错时，组织会转而向他求助。

在一个简单的组织中，至少可以设想三项明确的守护责任：

- **产品守护人**：产品一致性、结构性选择、目标客群、价值主张、重大取舍，以及与战略的关系。
- **设计守护人**：设计系统、体验一致性、无障碍、新的交互模式和用户体验的质量标准。
- **开发守护人**：架构、安全、性能、可维护性、技术质量和投入生产的条件。

原则很简单：

每个人都可以为多个领域作出贡献，但每个领域仍然由某个人负责。

这种模式允许角色边界相互渗透，同时不让责任消失。

销售人员可以构建——但不会因此成为技术守护人

来看一个刻意简化的例子。

一名销售人员识别到某个需求，使用 AI 辅助工具，制作出一个小应用或一项有用的改动。只要企业拥有足够安全的环境，就没有理由认为只有工程团队才有实际构建的权利。

但这种自由并不意味着技术质量变成销售人员的责任。

如果变更涉及安全、权限、敏感数据或架构，开发守护人必须能够要求审查、强制修正，或阻止变更。

销售人员仍然对自己的需求负责。

工程团队仍然守护企业愿意在技术上长期维护的东西。

同样的逻辑也适用于设计。

产品经理可以使用已有组件组合一个界面。只要他遵循设计系统和成熟的交互模式，一律要求设计审核只会无谓地削弱自主权。

相反，如果这位产品经理脱离设计系统，创造新的交互模式，或设计一项结构性且复杂的交互，设计守护人就必须能够介入。

因此，边界不再是：“谁有权碰什么？”

而变成：**“风险或新颖性达到什么程度时，该领域的守护责任必须重新接管？”**

议题 owner 与领域守护人不是同一个人

这或许是新组织模式中最重要的区别之一。

owner 负责一个议题。

他必须理解问题、推动工作、对局部决策作出取舍、协调必要事项、核验结果，并为议题最终完成承担责任。

守护人承担的则是一项更为横向的责任。

他不一定拥有这个议题，而是守护某个领域的一致性、质量或风险水平。

因此，Product Engineer 可以是一项功能的 owner，却不必成为公司整个架构的守护人。

产品经理可以是一次发布的 owner，却不必成为所有设计的守护人。

销售人员可以是一个业务工具的 owner，却不必成为内部应用安全的守护人。

这种区别让结果责任更加清晰，也无须要求一个人精通所有领域。

AMPECO 展示了这种变化的一种版本：一名 Product Owner 与一名工程师先对齐“做什么”和“为什么做”，再由智能体承担很大一部分分析、开发与测试。执行被分散出去，但最终审核仍由人类负责。[6]

因此，AI 可以减少执行一个议题所需的人数，同时进一步提高明确**谁对它负责**的必要性。

角色或许不再是正确的观察单位

这种演变还带来一个更广泛的问题：我们是否仍应主要按照职位名称来组织企业？

围绕 Product Owner 的争论很能说明问题。我曾在 Malorean 上提出，PO 首先是一项职能——一种决策与承担责任的授权——而不是一个凭维护待办列表来证明价值的职业。[7]

到了 AI 时代，这一区别格外有用。

如果我们用一个角色执行的任务清单来定义它，AI 会让这种定义变得不稳定。越来越多的人都能完成许多任务。

如果我们根据一个人或团队必须为何事负责来定义责任，它就稳固得多。

产品经理可以少写规格说明，却仍然对产品选择的质量负责。

开发者可以少写一些手工代码，却仍然对技术质量负责。

设计师可以不再是唯一移动像素的人，却仍然守护体验的一致性。

在 Cursor，产品、设计与开发之间的边界同样不那么僵化，但产品负责人提醒人们，核心依然不变：选择什么值得构建，并提供足够明确的方向，避免团队偏离。[8]

因此，一项职业的变化，与其说是它失去了所有责任，不如说是它失去了对某些执行形式的垄断。

守护人不应成为新的官僚机构

然而，这里有一个显而易见的陷阱。

如果产品经理每一次修改代码都必须经过工程审核，开发者每一个局部决定都必须重新交给产品经理，任何非设计师制作的界面都必须等待设计委员会批准，那么我们什么也没有解决。

我们只是用一个新名称，重新制造了旧有的必经关卡。

守护人不应变成系统性的审批者。

设置守护人恰恰是为了实现更多自主权：每个人都可以在已知框架内行动，无须不断请求许可，因为组织既知道责任在哪里，也知道在哪些情况下必须重新让这些责任介入。

而且在现实中，这种模式从来不会完美运行。

有些东西会进入生产环境，而守护人原本不会完全以这种方式批准它。局部决策有时不会引起注意。团队有时会比预期自行扩大行动空间。反过来，也有人会出于谨慎而过早上报。

这并不一定意味着模式失败。

一个真正自主的组织接受部分决策在没有事先检查的情况下作出。否则，那就不是自主权。

所以，要解决的问题不是确保永远不会出现任何偏差。

而是定义：**哪些偏差可以接受，哪些应该由系统检测，哪些则必须在代价变高前由守护人介入。**

问题由此发生了变化。

我们已经区分了两种责任：议题 owner 的责任，与领域守护人的责任。

现在还要解决另一个问题：

什么时候应该让 owner 独立决定，什么时候应该让守护人介入？

在 Malorean Corp.，主动性与责任之间的这道边界已经开始引发摩擦。下一段故事中，我们会再次看到他们，随后再研究必要的护栏与上报规则。

参考文献

[1] Are Roles and Responsibilities a Thing of the Past? – Figma

- URL：https://www.figma.com/blog/2025-shifting-roles-report/

- 研究现代产品开发团队中责任范围的扩大与职能边界日益增强的渗透性。

[2] PM, développeurs et IA : les rôles se brouillent, les responsabilités restent – Malorean

- URL：https://malorean.net/articles/2026-08-01-pm-developpeurs-et-ia-les-roles-se-brouillent-les-responsabilites-restent.html
- 阐述作者的观察：AI 扩大了能够采取行动的人群，却没有消除产品、设计与工程责任。

[3] Designers' workflow for shipping code – Atlassian

- URL：https://www.atlassian.com/blog/ai-at-work/designers-workflow-for-shipping-code
- 描述设计师直接修改产品代码，同时仍由工程团队验证质量、性能与生产条件。

[4] Product Engineer – Linear Careers

- URL：https://linear.app/careers/0c7c2e26-0a98-42cf-a47c-9a3999fb513b
- 一则招聘启事，说明 Linear 对 Product Engineer 从问题定义到功能交付的职责要求。

[5] La qualité appartient à ceux qui livrent – Malorean

- URL：https://malorean.net/articles/2026-06-03-la-qualite-appartient-a-ceux-qui-livrent.html
- 主张明确的技术责任：质量不能在交付后转给一个专门负责修复问题的团队。

[6] How we built an AI-native engineering system – AMPECO

- URL：https://www.ampeco.com/blog/how-we-built-an-ai-native-engineering-system/

- 展示在把大量执行交给智能体的同时，如何仍让人类对意图、架构和验证负责。

[7] Le PO n'est pas un métier, c'est une fonction – et l'IA n'y change rien, elle accélère – Malorean

- URL: https://malorean.net/articles/2026-08-03-le-po-nest-pas-un-metier-cest-une-fonction.html
- 区分决策与 ownership 的授权，以及主要由管理制品或待办列表所定义的职业。

[8] How Cursor Is Navigating the New Shape of Product Craft – Atlassian / Cursor

- URL: https://www.atlassian.com/blog/how-we-build/how-cursor-is-navigating-the-new-shape-of-product-craft
- 展示角色边界更加开放的同时，产品判断、目标与护栏仍然保持为核心责任。

7 – Malorean Corp.——Alice 心中那些从未写下的知识

周四上午，Lucas 本周第三次给 Alice 打来电话。

“只占用两分钟。”他保证。

Alice 已经学会警惕这句话。Lucas 从不要求开很长的会议。他会带着一个具体问题来，两人讨论一下，他便回去继续工作。正因如此，这些打断很难拒绝。

“说吧。”

“关于批次追溯，客户希望能按供应商搜索。这个我可以加进演示。但如果同一个供应商交付了几个内部编号相同的批次，我们是把它们放在一起显示，还是让用户选择？”

Alice 想了几秒钟。

“如果有多个匹配项，绝对不能让人以为我们已经确认了某个批次。把结果分开显示。之后可以和客户一起看看，他们怎样区分这些批次。”

“我也是这么想的，只是不确定。”

“那为什么还给我打电话？”

Lucas 尴尬地笑了。

“因为 Julie 的设计稿那件事之后，我觉得凡是看起来像产品规则的决定，最好先确认一下。”

上周，Alice 要求他不要把一个潜在客户的需求变成标准功能。如今她有些遗憾，Lucas 从那次讨论中得到的教训竟然是，只要稍有疑问就应该征询她的意见。

“好吧。但这个问题你不需要问我。”

“听你亲口这么说，我就放心了。”

他们挂断了电话。通话持续了七分钟。Alice 回到正在写的文档，重读上一句话，却已经想不起原本还要补充什么。

十分钟后，Lucas 发来一条消息。

另一个问题：对于没有批次信息的早期生产记录，我们显示“未知”，还是隐藏这一列？

她开始输入回复，随后停了下来。

问题虽不同，背后的推理却相同：绝不能把 ERP 并不掌握的信息表现得像是确定无误。她本可以直接告诉他这条原则，而不是分别裁定两个案例。

尽管如此，她还是回复了。

明确显示信息缺失，不要掩盖问题。

Lucas 回了一个竖起拇指的表情。

Alice 看了看日程。与客服的讨论被改期了，下午还有一场客户会议。两者之间，她必须审阅 Julie 为那位多经营场所潜在客户提出的方案。

她喜欢 Lucas 直接和她交流。把议题交给他时，她想要的正是如此。但他的推进速度太快，每出现一种新可能，就会出现一个新问题。他越频繁地征询她，她就越没有时间去做自己最适合做的事：理解客户，思考产品方向。

他也在浪费时间。有时，他必须等她回答后才能继续工作。

一个令人不快的念头闪过：如果继续这样，他们或许最终会重新把所有决定塞回工单。那样更容易组织。她负责规定，Lucas 负责执行，谁也不必再猜测他的自主权究竟延伸到哪里。

而这正是他们试图超越的东西。

8

中午，Alice 与 Bob 通过视频见面。他们已经习惯了没有议程地聊聊，只为对照彼此正在经历的事情。

“和 Lucas 合作得怎么样？”Bob 问。

“从某种意义上说，太好了。”

“听起来不像夸奖。”

“他推进得非常快，看到很多问题，也和客户交流。但他一直给我打电话，问一些原本可以自己决定的事情。”

“你回答他吗？”

“当然。”

“那他给你打电话就是对的。”

Alice 翻了个白眼。

“谢谢你，Bob。”

“我不全是在开玩笑。如果五分钟就能得到你的答案，而不必承担犯错的风险，我为什么不用呢？”

“因为照这样下去，我们根本没法工作。”

Bob 让她举个例子。她讲了早上的两个问题，又讲了与 Julie 的讨论，以及 Lucas 最近的一次犹豫：如果某个客户要求的设置大概只对这一家有用，是否应该添加？

“你意识到了吗？”Alice 说，“这些始终是同一个问题的变体。”

“那你的答案是什么？”

“要看情况。”

Bob 笑了。

“正是这种回答，让别人不得不继续给你打电话。”

Alice 正要反驳，但他说得没错。

“并不是毫无规律地‘看情况’，”她接着说，“我要看它是不是在解决一个反复出现的问题，是否服务于我们的优先客户，是否会让所有人的产品变得臃肿，以及我们能否维护这个方案……”

她停住了。

“看到了吧？”Bob 说，“你刚刚已经说出了部分规则。”

“这不是规则，更像是一种思考方式。”

“那就更好了。也许这才是应该传递给 Lucas 的东西。”

Alice 沉默了。

多年来，她一直在写工单。她会描述预期行为、特殊情况和验收标准，却几乎从不写下自己如何在多种可能行为之间作出选择。这套推理一直留在她脑中，或者随着议题交付，消失在某次会议里。

“像这样的东西，我大概有几百条。”她说。

“很可能。我也一样。”

“而我们从来没有理由把它们写下来。”

“理由一直都有，只是没有时间。况且，也没人会创建一张标题叫‘解释我们为什么有时会拒绝一个好想法’的工单。”

两人都笑了。

通话结束后，Alice 打开一个新文档，写下：产品决策原则。

她写出第一句话，又删掉，换了一种说法。十五分钟后，她得到了一段像内部政策的文字。内容没有错，却完全不像她刚才向 Bob 解释的那些话。

她关掉文档。

然后打开 ChatGPT。

“帮我把脑中对我们产品的认识提取出来，”她口述道，“我不要一套理论流程。请向我提问，帮助我回忆案例，并区分什么是真正规则，什么只是我的习惯。”

第一条回复建议她从近期决策开始。

Alice 说了起来。

她讲到 Julie 的潜在客户，讲到那些漂亮诱人的设计稿，也讲到把一个特殊需求变成面向所有客户的承诺所带来的风险。她解释说，一个大客户可能意味着很多收入，却不一定自动代表正确的产品方向。

“等等，我得解释原因，”她大声补充，“我们的 ERP 服务很多不同类型的企业。如果把每条特殊规则都加入软件核心，最后每个人都会用得更复杂。

但有时，一个特殊需求也会揭示一个普遍得多的问题，所以也不能一概拒绝。”

ChatGPT 问她如何区分两者。

Alice 描述了与客服的交流、在多家客户那里观察到的案例、销售提出的需求，以及实施方面的约束。她回忆起一些从未留下会议记录的旧讨论。她解释为什么某些方案被放弃，为什么选择增加一项设置而不是新功能，以及为什么工资计算中的例外不能像界面偏好那样处理。

有时，她惊讶地发现自己正在回答一个从未想过要明确提出的问题。

AI 会改写、归类，再提出一条原则。Alice 则进行修正。

“不，不能说‘我们从不开发定制功能’。在某些条件下可以，但必须有意识地决定它会带来什么成本。尤其不能把一项销售假设伪装成产品决策。”

稍后，ChatGPT 写道，追溯功能可以保证重建所有历史批次。

“当然不行，”Alice 对着屏幕说道，“如果数据从未被记录，就不可能靠魔法重建。这是必须展示的限制，不是可以承诺的能力。”

她让它修正这段话，并要求文档区分已经决定的事项、她认为可能正确的判断，以及需要向客服、Lucas 或客户核实的内容。她没有把真实客户账户中的机密信息贴进去，所有示例都经过了改写。

一小时过去了，又过了一小时。

文档逐渐填满 Alice 早已知道的事情：Malorean 优先服务哪些客户、哪些问题反复出现、接受哪些折中、不应作出哪些承诺、旧数据有什么限制，以及为什么原型不等于可供销售的正式功能。

她还添加了指向已有信息的链接：客服反馈、仍然找得到的决策记录、会谈笔记和产品文档。没有证据的地方，文字必须明确指出。

接近第三个小时结束时，ChatGPT 给出了一份总结。

Alice 重读了一遍。有些页面太长，有些则把一般原则和一个案例混在一起。她最后修正了最武断的措辞，并加上日期和文档适用范围。

随后，她把整份内容导出为 PDF。

她滚动浏览一页页内容，惊讶于它竟然这么长。

三小时前，她主要只想向 Lucas 解释什么时候可以不经她同意自行决定。现在，她手里拿着的却是一幅并不完美、但异常丰富的画像：团队早已掌握，却从未汇总共享的产品知识。

这不是终极真理。有些段落还需要确认或更新。PDF 也不会取代客户、数据或负责人的判断。

但至少，下一个人不必再从零开始。

第二天早上，Lucas 打开了文件。

“这些都是你为我写的？”

“不是我一个人写的，”Alice 回答，“我花了三个小时和 ChatGPT 对话。它帮助我提出问题、组织答案，并找出缺失内容。我进行了审阅和修正。不过，其中大多数东西我原本就知道。”

“那为什么我以前从没看到过？”

Alice 有些尴尬地笑了。

“因为它们不属于任何一张工单。”

Lucas 浏览目录。他在其中看到原则、决策案例、业务约束、客户信息、来源链接和仍未解决的问题。

“处理追溯问题时，我可以把这个交给我的 AI 吗？”

“可以，但要在我们批准的环境中。告诉它，这只是上下文，不是所有内容都真实无误的证据。如果某条规则很重要，就回到原始来源，或者找对它负责的人确认。”

“这能为我节省大量时间。有时我会让 AI 帮我在两个方案之间选择，但它只知道我告诉过它的内容。它可以给出一个听起来非常有说服力的答案，却完全没有考虑三年前作出的某项选择。”

Alice 熟悉这种感觉。一个孤立来看合理的答案，放进他们的 ERP 里可能荒谬至极。

Lucas 停在一段文字上。

“这里说，如果业务信息缺失可能使人误以为结果是完整的，就绝不能隐藏这种缺失。”

“这就是我昨天对你说的。”

“是。如果昨天已经看到这句话，我就不会给你打电话了。”

他继续阅读，随后抬起头。

“好吧，但我还有一个问题。”

Alice 流露出泄气的神情。

“什么问题？”

“现在我更理解你的思考方式了，却仍然不知道从什么时候起，我有权自己决定。”

她放下咖啡。

他刚刚找到了这份文档中最重要的缺口。

他们一起重新研究这个问题。Alice 共享了一个空白页面。

“先试一个简单办法。给我打电话之前，你问自己三个问题：我是否理解要实现的结果？我的决定是否仍在我们已知的框架内？如果判断错误，可能发生什么？”

“那就必须定义什么叫‘在框架内’。”

“正是如此。”

他们写下三个级别。

绿：当选择保持局部、可逆、符合已知规则，且不会对客户作出新承诺时，Lucas 自行决定。他能够解释自己的推理，并核验结果。

橙：当遇到新情况、目标之间存在冲突，或对产品行为有重大疑问时，他征询 Alice。若涉及技术风险，他也要找相应的专业人员，例如涉及 ERP 历史问题时找 Thomas。

红：当需要修改结构性规则、代表 Malorean 作出商业承诺、触及敏感数据，或承担重大且难以逆转的风险时，任何人都不能独自决定。必须与相关负责人明确裁决，有时还要让 Hubert 参与。

Lucas 重读了一遍这张表。

“所以，如果我改变演示中的结果呈现方式，避免造成误解，就是绿级。”

“是。”

“如果我想增加一项可能影响所有客户的行为，就来找你。”

“我们需要讨论。尤其是在需求尚未理解清楚时。”

“如果我想修改历史数据以记录批次呢？”

“这不是我一个人能给你开绿灯的事。需要技术分析和相应检查。你已经提醒我很多次了。”

Lucas 笑了。

“我只是想确认你还记得。”

他们又增加一条规则：Alice 没有回答，绝不会让一个橙级或红级决策变成绿级。但同样，不应要求她批准每一项绿级决策。

“如果我判断错了颜色呢？”Lucas 问。

“告诉我，我们再调整框架。我宁愿一起改进规则，也不愿不断责怪你没猜到我在想什么。”

Lucas 点点头。

“我甚至可以让 AI 对照这张表检查我的选择，列出理由和缺失信息。”

“可以，前提是不能让它替我们决定某种风险是否可以接受。”

Alice 把这张表加入参考文档。她开始理解，传递信息与真正赋予决策能力之间有什么不同。

他们原本可以就此结束。但 Lucas 又回到了追溯演示。

“我们已经谈了一小时我的自主权。现在问一个关于议题本身的问题：什么能证明我们成功了？”

Alice 几乎不假思索地回答：

“客户能够找到使用某一批次生产的产品。”

“是，但我们怎么验证？它需要可靠到什么程度？”

这一次，她没有立刻寻找答案。

他们已经花了好几天探索界面、规则和数据结构，也有了一个很有希望的演示。然而，谁都还没有准确写下应该如何判断结果。

Alice 想起那位质量负责人。在供应商发出警报后，他花了大半天交叉核对文件。

“应该从这里重新出发，”她说，“拿到一个被标记的批次号后，他能否找到相关生产记录，而不必再次手工完成所有调查？需要多长时间？结果完整吗？如果我们没有某项信息，是否明确显示出来？”

“我们可以和他一起测试，而不只是问他喜不喜欢演示。”

“没错。”

Alice 打开一个新页面。

“其实，我们本该在动手前就从这里开始。”

Lucas 笑了出来。

“两周前你要是这样问，我会说你正想重新给我写工单。”

“也许。但看看我们现在写的内容。”

他们写下观察到的问题、相关用户、预期结果、衡量方式、数据限制、范围之外的内容，以及需要报批的决策。只有一页，不再多写。

Lucas 提议把它称为 *Mini PRD*。

“我们不是在告诉我应该构建哪个界面，”他指出，“而是在就什么会让工作真正有用达成共识。”

“以及不能跨越哪些边界。”

Alice 边思考边说道：

“以后处理新议题时，这应该成为起点。在问要开发什么之前，先问想取得什么结果、如何验证。然后才界定范围，决定什么值得构建。”

“如果我能用正确的信息填写这份文档，就可以开始工作，不必等你准备待办列表。”

“你可以提出界定方案。这并不意味着任何承诺都不用审查就会获批。但对，这就是目标。”

他们完成了追溯项目的 Mini PRD，同时保留了那些尚无法回答的问题。某些案例仍需向客户核实，技术约束也有待确认。

文档没有制造任何确定无疑的假象，而是说明他们正在尝试了解什么。

Alice 觉得，这比一份等待勾选的功能清单有用得多。

接下来的周一，Alice 与 Hubert 的每周例会有所不同。

Alice 没有共享平时的跟踪表，也没有展示 Lucas 已经完成的功能列表。

她展示了决策分级表，随后是 Mini PRD。

Hubert 读了开头几行。

“Lucas 现在快了这么多，你们还花这么多时间做文档？”

Alice 熟悉这个质疑。几天前，她自己也提出过。

“我恰恰认为，我们过去写得不够。我们会写应该构建什么，却很少写下支持选择的信息。结果就是 Lucas 不断让我替他裁定。有了这些，他可以不经过我，自行作出部分决定。”

“你怎么知道他没有越界？”

“我们定义了什么时候可以自行决定，什么时候必须征询我们。除此之外，我们不会检查一切。我们会核验结果，并在必要时修正框架。”

Hubert 停在有关旧数据的一行。

“从未记录的信息，你们无法保证追溯。”

“无法保证，”Alice 回答，“文档已经写明。我们必须向客户清楚展示这个限制。”

Hubert 表示赞同。

“这一点，我宁愿在销售人员承诺相反内容之前知道。”

“正是如此，”Alice 说，“上下文不只是给 Lucas 用。Julie 也可以在其中看到我们知道什么、尚未决定什么。但信息必须保持更新，而且每个人都要知道哪些部分具有权威性。”

他们决定保留一个可以修改的工作版本，PDF 只是带日期的快照。重要决策会连同理由一起记录；反复使用的原则在适当情况下可以变成智能体指令或工具中的检查。这不会取代技术测试，也不会取消投入生产前必要的审核。

“我不想六个月后才发现，AI 一直在执行一条早已无人认同的旧规则，”Hubert 说。

“我也不想，”Alice 回答。

Hubert 第一次对 Mini PRD 表现出更多兴趣。

“你们真的会观察客户能否找到自己的批次？”

“会，”Alice 回答，“我们会提供案例，让他和我们一起测试。如果结果不可靠，完成了多少界面都毫无意义。”

Hubert 合上文档。

“很好。我想知道你们学到了什么，不只是交付了什么。”

8

周三，Alice 在通讯软件中注意到一件奇怪的事。

从早上起，Lucas 一条消息也没发给她。

她查看了项目频道。那里明明有很多活动：Lucas 在演示实例上测试了几种结果呈现方式。对于其中一种，他解释说，自己选择把具有相同编号的批

次分开显示，以免暗示并不存在的关联。他引用了产品上下文中的原则，注明所使用的数据，并提议与客户一起验证他们是否能正确理解这个界面。

他没有请求许可。

Alice 读完解释，便回去继续工作。她没有任何补充。

下午临近结束时，Lucas 打来电话。

“我有一个橙级问题，可能正变成红级。”他说。

“你说。”

“客户还想搜索早期生产记录。对于一部分历史数据，我们没有可靠的原材料与成品关联。我可以显示已知内容，但如果把它呈现为完整搜索，他们可能会以为没有结果就代表没有任何产品受影响。”

Alice 坐直了身体。

“你说得对。不能留下这种歧义。”

“我可以添加警告，并区分数据可用的时期。但我希望我们共同决定究竟能向客户承诺什么。”

“这个确实要一起决定。还要和客服确认客户会如何理解这项限制。”

Lucas 共享了屏幕。他们开始处理这个问题。这不是一次按钮审批，也不是一个 Alice 早已知道答案的问题。摆在他们面前的决策，可能改变客户对软件的信任。

交流过程中，Alice 意识到自己一次也没有想到它占用了多少时间。

挂断之前，Lucas 犹豫了一下。

“你知道吗，你的 PDF 没有回答我的所有问题。”

“幸好没有，”Alice 说。

“幸好？”

“如果我真能把所有答案都写进去，那就说明你已经没有任何东西需要探索了。”

Lucas 笑了。

“好吧。不过我觉得自己更清楚什么时候该来找你了。”

通话结束后，Alice 重新打开日程。下一次客户会议前，她有一个小时的空闲。

整整一个小时。

她还不知道这场实验能否扩大规模，也不知道沿途会犯多少错误。她只知道，Lucas 已经在没有她参与的情况下作出了若干决定，然后在她的判断真正重要的时刻回来了。

她在共享文档中新增了一个需要澄清的问题，涉及如何呈现历史数据。那不是一张告诉 Lucas 应该编写什么代码的工单，而是一个需要与相关人员共同研究的决策。

他们的工作方式并没有变得不那么严格。

只不过，这种严格开始被放到了正确的地方。

8 – 更多自主权需要更多框架，而不是更少

一个组织可以决定，让更多人无须等待专家便能采取行动。

前文研究的角色边界渗透与责任分配，带来了一个新的协调问题：现在，多个人都能对同一议题采取行动，但他们不一定拥有相同的决策参照。

然而，如果每一种新的行动能力都必须接受系统性的人工审核，自主权就仍然只是理论。

我们不过是把必经关卡从一个职能转移到了另一个职能。

因此，真正的挑战是建立这样一套系统：让更多决策能够在局部作出，同时不失去一致性、安全、质量或产品方向。

这揭示了一个重要悖论：

越想分配自主权，框架就越需要明确。

自主权并不来自没有规则。

它来自这样一种能力：由于目标、边界、风险与责任足够清晰，人们无须不断请求许可，也能采取行动。

如果所有事情都要审核，就不存在自主权

设想一位对某个议题负责的 Product Engineer。

他与用户交谈、探索方案、制作原型、开发并准备发布。

从纸面上看，他是自主的。

但如果每一个微小的产品取舍都要问产品经理，每一个界面细节都要问设计师，每一个技术决定都要问 Tech Lead，那么组织并没有真正转移责任，只是要求一个人穿过更多审批。

其他职能也会遇到同样障碍：如果每一个动作仍要遵循过去的流程和审批，允许某个人动手构建并没有多大价值。

因此，问题不只是**谁能采取行动**。

还必须知道**这个人可以在哪个框架内自行决定**。

在 Cursor，这套逻辑体现为足够清晰的目标与护栏，让构建者可以行动，同时不偏离愿景或路线图。[1]

这样一来，组织就用决策框架取代了一部分详细规定。

用决策框架取代部分工单

通过 Linear 的案例，我们已经看到完全规定好的工单为何会限制主动性。[2] 现在，问题变得十分实际：团队需要拥有哪些信息，才能不等待新的规格说明，自行作出取舍？

这并不意味着要取消工单。我曾在 Malorean 上提出，待办列表是行动与协调工具，而不是产品全部记忆的存放处。[3]

因此，一个自主框架可以明确那些引导局部决策、却不提前锁死每种方案的信息。

这个框架可以包括：

- 优先客户或理想客户画像；
- 需要解决的问题；
- 预期结果；
- 范围边界；
- 重要约束；
- 预期质量水平；
- 组织明确接受不去做的事情；
- 一项决策必须上报的阈值。

Ramp 为这套逻辑提供了一个具体例子：明确核心理想客户画像、核心问题，以及企业希望特别擅长解决的事情，也使它能够拒绝那些把产品拉向另一方向的高收益机会。[4]

Figma 则提醒我们，从 0 快速走到 1 并不足够。要把快速原型变成好产品，真实案例、例外与质量水平依然至关重要。[5]

这张表体现的是一种组织选择：不只是说明应该做什么，也让必须遵守的边界清晰可见。这里并没有把它当作一种普遍适用的方法。

上下文成为基础设施

要在局部作出正确决策，就必须能够获取理解问题所需的信息。

这或许是智能体时代最重要的组织变化之一。

组织不能要求更多人作决定，却把大部分上下文继续留在少数专家脑中。Linear 强调，客户反馈、战略方向、决策与代码都应当能被人类和智能体获取。[2]

我曾在 Malorean 上把产品经理的角色描述为**上下文架构师**：把访谈、销售异议、客服信息、决策、数据、来源和微弱信号等分散知识，转化为可重复使用的组织资产。[6]

这一理念深刻改变了文档的作用。

目标不再是生产更多页面。

而是让一个人在不从零重建全部推理的情况下，获得作出决策所需的要素。

良好的上下文能让一个人理解：

- 客户真正说了什么；
- 哪些假设已经经过测试；
- 哪些约束重要；
- 已经作出哪些结构性决策；
- 已经接受哪些折中；
- 哪些指标可以用来判断结果。

自主权越大，这些上下文就越必须可靠并得到共享。

否则，企业只分配了行动能力，却没有分配理解能力。

保留“为什么”的记忆

这种重新分配又会制造另一个问题。

更多决策在局部作出，有时由不同的人作出，有时还借助没有参与过往取舍的智能体。

然而，没有记忆的决策很快就会再次变得可以争论。

人们记得结论，却忘了上下文。

人们知道某条规则存在，却忘了为何这样选择。

新的客户出现，新的管理者上任，团队发生变化，一个早已讨论过的问题便再次开始，而掌握的信息甚至比第一次更少。

这正是我赋予**产品决策记录（PDR）**的作用。[7]

PDR 不会记录每一个微小取舍。

它追踪那些足以指导多个团队、多种产品或未来多种情境的结构性产品决策。

其中尤其会保留：

- 上下文；
- 考虑过的替代方案；
- 最终决定；
- 选择理由；
- 正面与负面后果；
- 接受的风险；
- 足以证明应重新审议该决策的条件；
- 作出取舍或受过征询的人员。

目的不是增加行政负担。

恰恰相反，一份好的 PDR 应该避免同一场讨论被重新开启五十次。

由此可以作出一个有用的区别：

代码记录构建了什么；决策记忆则应该记录为什么选择了某些结构性规则。

持久的自主权需要两者兼备。

把护栏移入系统

即使拥有良好的上下文，要求每位员工记住组织的所有规则也没有效率。

因此，一些企业会把重复性规则从人工审核转移到工具中。这是一种可观察到的机制，并不能保证自动化足以确保质量。

Meta 使用 Diff Risk Score 估算一项变更导致事故的概率，并据此驱动风险敏感的工作流。[8]

Cloudflare 把明确的工程标准转化为自动检查。某些强制规则可以阻止变更获批，直至问题得到修正，或有人明确批准例外。[9]

这项变化很重要，因为它改变了专家的角色。

如果一条规则可以自动检查，技术守护人就不必在五十个 pull request 中手工重复提醒。

如果设计系统让正确选择容易、错误选择显眼，设计守护人就不必审核每一个按钮。

如果结构性决策、目标客群和目标都可以获取，产品守护人就不必每周重复解释相同约束。

框架成为基础设施。

测试、CI、设计系统、标准、安全规则、模板、智能体指令、指标与共享上下文，共同承担过去由人来完成的部分控制。

这样，人类专家便可以把注意力集中在例外上。

并非所有决策都值得接受同等控制

风险概念在这里变得至关重要。

一项文本翻译与一次权限系统修改，不应该遵循同样的审核流程。

局部且可逆的变更，无须接受数据迁移或改变企业定位的决策所需的同等审核。

理解这类组织的一种方式，是把二元问题——“需要审核吗？”——替换成一个分级问题：**这项风险值得什么程度的控制？**

以下三个级别是**本书提出的一套分析框架**，并不是所有引用企业都采用的实际做法。

局部自主

决策仍处在已知目标和标准之内。

它可逆，可能造成的后果有限，也不会改变结构性规则。

由 owner 作出决定。

定向审核

决策在某个领域引入新事物、例外或重大风险。

相应的守护人——开发、设计、产品或其他领域——介入。

他不一定接管议题，而是提供当前缺少的特定判断。

上报

决策改变了系统本身的边界。

技术方面：结构性架构、安全、敏感数据、关键迁移、监管风险。

设计方面：重要的新交互、脱离设计系统，或对其他用户路径造成重大后果。

产品方面：目标客群、价值主张、定价、重大投入、路线图或战略取舍的变化。

在这种情况下，owner 已经不只是在框架之内作决定。

他是在提议改变框架。

按照本书提出的模式，这类情况需要守护人或更高决策层级介入。

没有明确边界，就会出现两种相反的风险

如果决策边界不清晰，组织会暴露在两种相反风险之下。

第一种是官僚主义。

所有事项都向上汇报。专家变成等待队列，负责人必须批准琐碎决定，自主权随之消失。

第二种是稀释。

每个人都在局部作决定，却不知道一项决策何时已经不再局部。彼此矛盾的选择不断累积，直到为客户或企业制造出一个明显问题。

我曾在 Malorean 上介绍过几种用于保持组织一致性的工具，恰恰是为了让这些边界更加明确：共享语言、单页文档、指标层级、决策日志与上报规则。[10]

它们的价值不在于生产更多文档。

而在于迫使某些问题变得明确：

- 谁作决定？
- 对哪些议题作决定？
- 在什么时刻？
- 什么可以留在局部？
- 什么必须上报？

- 在哪些条件下可以重新审议一项决定？

这两种风险共同导向一个组织假设：比起增加或取消审核环节，**把决策放在更恰当的位置**可能更加重要。

决策也意味着接受损失

护栏不能只有技术层面。

框架的一个核心部分，是作出真正取舍的能力。

组织经常作出含糊到足以维持共识表象的决定。每个人都可以把自己的优先事项投射进去，直到执行阶段才发现那些约束彼此冲突。

我把这称为**软决策**。[11]

它们在高度自主的系统中格外危险。

如果目标含糊，每个团队都可能作出局部合理、整体却不一致的决策。

因此，给予更多自主权，也要求把所接受的损失清楚呈现出来。

哪类客户优先？

哪个细分市场现在不会得到服务？

什么质量水平不可妥协？

哪项约束可以暂时牺牲？

当一个目标与另一个目标冲突时，哪一个占据主导？

自主权不需要一套什么都规定的框架。

它需要让真正的取舍足够清晰，使局部决策能够趋同的框架。

衡量结果成为最后一道护栏

自主权不会在某件东西交付时结束。

如果一个人真正拥有某个议题，他还必须观察之后发生了什么。

第一章已经区分了产出数量与创造价值。此处的问题不同：**作出决策的人随后如何知道它是否恰当？**

Canva 提供了一项具体机制：团队可以把实验与其商业影响估算联系起来。[12]

对每个议题而言，这意味着需要明确：

- 预期发生什么变化；
- 用什么衡量这种变化；
- 必须保留什么质量；
- 用户实际做了什么；
- 应该继续、修正、放弃还是深入探索。

衡量让责任形成闭环。

没有衡量，自主权可能只是生产许可。

有了衡量，它才成为对结果的责任。

自主权并不意味着管理退出

第 1.4 章介绍的 Marty Cagan 赋权产品团队模式，有助于明确管理层在这里的作用。把问题交给团队、却不预先指定解决方案，需要管理层澄清目标与战略上下文。[13] Cagan 也强调能力培养、辅导与领导：让团队作决定并不等于抛下团队不管。[14]

不过，这两者的类比有一个重要限制。Cagan 描述的是拥有互补能力的产品团队所承担的责任。本书提出的守护人概念，则回答另一个问题：当执行从一个职能跨到另一个职能时，如何保留某个领域的责任？

守护人成为框架的看护者，而不是每个动作的审批者

第 1.6 章把守护人定义为一个领域一致性的负责人，即使他没有亲自完成每一项任务。到了这里，他的职责变得具体：制定可以理解的规则，将重复性检查自动化，并在风险或新颖程度超出团队可自行决定的范围时介入。

如果他必须检查一切，就会重新成为瓶颈。如果再也没有人为任何事情负责，责任就会稀释。

因此，自主权既不是审核每一个动作，也不是取消所有控制；它意味着在保留明确责任的同时，让更多局部决策成为可能。

但这场转型还带来了一个超越产品与工程职能的问题。

如果上下文可以直接流动，决策可以在更局部的层面作出，智能体也承担了部分协调工作，那么过去负责这些职能的管理中间层会怎样？

Malorean Corp. 会先向我们展示，这个问题如何落到具体的人身上。随后，我们再研究它在企业尺度上的含义。

参考文献

[1] How Cursor Is Navigating the New Shape of Product Craft – Atlassian / Cursor

- URL：https://www.atlassian.com/blog/how-we-build/how-cursor-is-navigating-the-new-shape-of-product-craft
- 展示目标、护栏与产品判断如何让边界更加开放的角色采取行动，同时不偏离选定方向。

[2] Issue tracking is dead – Linear

- URL：https://linear.app/next
- 主张一种让客户反馈、战略、决策与代码共享更多上下文，而非以工单链条为中心的系统。

[3] Le backlog n'est pas un dépotoir : c'est un outil d'action – Malorean

- URL：https://malorean.net/articles/2026-06-03-le-backlog-nest-pas-un-depotoir.html
- 区分作为临时协调工具的待办列表，以及持久的产品记忆、上下文与推理。

[4] 5 insights from Ramp's CPO on building for hypergrowth – Ramp

- URL：https://ramp.com/blog/5-insights-from-ramps-cpo-reddit-ama
- 强调核心理想客户画像与核心问题，以及拒绝会分散产品方向的机会之必要性。

[5] What It Takes to Build Great Products Now – Figma

- URL：https://www.figma.com/blog/what-it-takes-to-build-great-products-now/
- 提醒人们，快速制作原型并不足够：真实案例、例外与质量仍是构建持久产品的决定性因素。

[6] Le PM comme architecte du Contexte – Malorean

- URL: https://malorean.net/articles/2026-06-08-le-pm-comme-architecte-du-contexte.html
- 提议把分散的信号、来源、决策与取舍转化为可复用的上下文，以改善未来决策。

[7] Product Decision Record : tracer les choix produit qui structurent l'entreprise – Malorean

- URL: https://malorean.net/articles/2026-06-03-product-decision-record-tracer-les-choix-produit.html
- 将结构性决策的简短记忆正式化：上下文、替代方案、选择、后果与明确的重新评估条件。

[8] Diff Risk Score: AI-driven risk-aware software development – Meta

- URL: https://engineering.fb.com/2025/08/06/developer-tools/diff-risk-score-drs-ai-risk-aware-software-development-meta/
- 介绍一种风险评分，使控制工作流能够适应软件变更引发事故的概率。

[9] How Cloudflare enforces engineering standards using AI – Cloudflare

- URL: https://blog.cloudflare.com/engineering-standards-enforcement/
- 展示明确标准如何变成自动检查，并在投入生产前阻止某些违规行为。

[10] Les outils de cohérence organisationnelle – Malorean

- URL: https://malorean.net/articles/2026-06-02-les-outils-de-coherence-organisationnelle.html
- 把共享语言、目标、指标、决策记忆与上报规则联系起来，使组织取舍更加明确。

[11] Pourquoi les organisations préfèrent les décisions molles – Malorean

- URL：https://malorean.net/articles/2026-06-02-pourquoi-les-organisations-preferent-les-decisions-molles.html
- 分析那些维持共识表象、掩盖损失、稀释责任，并把矛盾推向执行阶段的非决策。

[12] Measuring Commercial Impact at Scale – Canva

- URL：https://www.canva.dev/blog/engineering/measuring-commerical-impact-at-scale/
- 描述一种自助式商业影响衡量方式，将局部实验、产品决策与商业价值创造联系起来。

[13] Team Objectives - Empowerment – Marty Cagan，Silicon Valley Product Group（2020）

- URL：https://www.svpg.com/team-objectives-empowerment/
- 主张把需要解决的问题交给团队，同时提供局部决策所需的战略上下文。

[14] Product Leadership Is Hard – Marty Cagan，Silicon Valley Product Group（2020）

- URL：https://www.svpg.com/product-leadership-is-hard/
- 解释为何自主产品团队需要更高要求的领导与主动辅导，而不是管理层退出。

9 – Malorean Corp.——两个合理的决定

周一早上，负责协调两个开发团队的 Vivien，用一个多年来始终相同的问题开始会议。

“谁能告诉我，本周各项议题进展到哪里了？”

他的跟踪表已经打开。工单、负责人、截止日期和阻碍因素，被整齐放进他耐心完善多年的各栏之中。他了解这两个各有五名开发者的团队，知道每个人的长处和习惯，也知道哪些议题需要催促，哪些最好让它们再成熟一些。

没有人立刻回答。

随后，Lucas 开口了。

“追溯项目方面，我周五更新了 Mini PRD。我们与客户测试了按批次搜索。下一步是验证数据不完整时的案例。我把未解决的问题写进上下文了。”

“你的交付日期呢？”Vivien 问。

“仍取决于对历史库存的检查。Thomas 要和我一起看一个问题。”

Thomas 点点头。他自己正在改进发货准备流程，也已经在工具中更新了进展。

Vivien 转向其他开发者。大家的回答彼此相似。每个人都更了解自己处理的问题，作出了一些决定，也能解释下一步。少数人仍然需要帮助拆分工作，其他人则没有等到会议才继续推进。

最后，Vivien 看了看自己的表格。

刚才听到的一切都已经写在里面，有时甚至更详细。

“好，”他说，“都讲完了。”

会议只用了十二分钟。

过去需要四十五分钟。

Vivien 不太确定自己是否应该为此高兴。

他在 Malorean 工作了很久，经历过所有开发者都集中在勒阿弗尔办公室的年代。他组织发布、处理延期、与产品团队协商优先级，也会向总监解释，为什么原定周五完成的需求要等到下周二才能上线。

新冠疫情后公司转向远程办公，他也调整了方法。走廊里的交谈已经不够。他要求把决定写下来，让阻碍清晰可见，并让每个人记录与其他部门达成的共识。

这成为他的日常工作：让那些无人能够完整掌握的信息流动起来。

他喜欢这项工作。他尤其喜欢那些时刻：一个方向不对的项目，因为有人终于让正确的人聚到一起，而重新走上正轨。

然而，自从团队开始使用 AI 工具，一些事情变了。开发者的产出更快，但更重要的是，其中一些人开始以不同方式接手议题。Lucas 直接与客户交流。Alice 共享的产品上下文解释了决策理由，而不只是待开发功能清单。Bob 也开始在自己的议题上采用同样做法。

现在，一个智能体还会替 Vivien 准备部分摘要，而这些过去都由他手工制作。

工作没有消失。只是它一点点脱离了他，或在别处完成。

他还没有和任何人谈过这件事。

接下来的周三，Lucas 向 Alice 展示了追溯演示的新版本。

“我们测试了供应商报告某个批次可能存在缺陷时会发生什么。现在可以找到我们掌握的生产记录。我还增加了一个‘核验中’状态，避免让人以为这个批次已经确认无须担忧。”

Alice 看着屏幕。

一条横幅提示该情况需要检查。对于较早的生产记录，另一条消息则说明某些信息无法获取。

“这比第一个版本更诚实。”她说。

“是。客户的质量负责人特意要求，不要把‘未发现问题’与‘我们没有必要信息’混为一谈。”

Alice 表示赞同。他们已经在上下文文档中讨论过这种区别。

“写进 Mini PRD 了吗？”

“写了，也加了测试案例。”

“那就继续。”

她无须决定横幅的位置，也不用决定消息的具体措辞。Lucas 知道他们希望实现什么结果，已经选择了一种呈现方式，并准备让客户验证。

稍后，Thomas 向 Vivien 展示了发货模块的一项改进。

这是另一个问题。对某些客户而言，即使没有任何明确禁令，一张订单也可能因为中间状态或填写不当而被卡住。物流团队会浪费时间，请人修正那些并未发现风险的档案。

因此，Thomas 澄清了准备规则：标记为“已冻结”的批次会阻止发货；其他情况保持可见，并沿用现有检查，按通常流程处理。

“你确定没有改变其他模块的行为？”Vivien 问。

“对现有案例而言，确定。我增加了回归测试，没有修改冻结规则。”

Vivien 没有理由怀疑 Thomas 的谨慎。他有时知道一些连历史文档都不再提及的例外。

“很好。继续之前，让人审查一下这项修改。”

Thomas 点点头。

这两项变更尚未被放进同一个场景中测试。

问题在周五的一套集成测试实例上出现了。

一名客服人员准备了完整流程：接收原材料、生产、供应商警报、识别相关产品，然后准备发货。

Lucas 通过视频观看演示，Thomas 也在场。Alice 要求加入，因为她想观察一个没有参与设计的人会如何理解未来的操作方式。

最初几分钟一切顺利。

批次号被找到，生产记录出现，“核验中”横幅也正常显示。

随后，客服人员进入发货模块。

送货单仍处于准备状态，没有任何消息阻止后续操作。

“等等，”Alice 说，“这里说的是同一个批次吧？”

Lucas 凑近屏幕。

“是。怎么了？”

“上面显示它在核验中，但在这里，我们却可以像没有任何特殊情况一样继续操作。”

Thomas 打开订单详情。

“它没有被冻结，而是在核验中。两者不是同一个状态。”

“在我看来，”Lucas 回答，“‘核验中’正是意味着，在有人作出决定之前，不应该把它视为已获批准。”

Thomas 转头看向摄像头。

“那你为什么不把它标记为‘已冻结’？”

“因为我不想凭空制定一条普遍禁令。有些客户必须能够在不停止所有活动的情况下调查。真正受影响的范围，应由他们的质量负责人决定。”

“而我的模块里，”Thomas 反驳，“存在信息不完整的旧批次。如果开始冻结所有未经明确确认的内容，有些订单就再也无法推进。”

Alice 看着两个屏幕。

她理解 Lucas 的理由，也理解 Thomas 的理由。

问题在于，用户可能在 ERP 的一个部分看到“核验中”，进入另一个部分时却没有遇到任何阻止操作的警告。

“演示到这里停止。”她说。

没有人反对。没有任何真实客户数据受到影响，变更也尚未投入生产。测试本来就是为了发现这种问题。

但随后出现的沉默，远不如成功演示之后那样轻松。

Vivien 在得知刚才发生的事情后加入通话。

他先让 Lucas 描述自己的决定，再让 Thomas 讲述他的决定。他问了几个有关测试和各项变更范围的问题。

“你们各自都有 Mini PRD 吗？”他问。

“有，”Lucas 回答。

“也有决策规则？”

“有，”Thomas 回答，“我没有改变现有冻结规则。这只是发货准备流程的一项局部改进。”

“我也没有，”Lucas 说，“我只是增加信息，避免掩盖疑问。我们已经明确写过，这一点很重要。”

Vivien 把两份文档并排打开。

它们都没有错，只是谈的不是同一件事。

第一份说明如何在追溯过程中标记不确定性，第二份说明发货何时必须停止。两份文档都没有回答连接它们的问题：当一个批次可疑、但尚未明确冻结时，软件应该怎么做？

“两个议题之间有一道缺口，”Vivien 说。

Thomas 点点头。

“技术上，我们可以选择多种行为。但必须知道想要哪一种。”

“还要问客户，他的质量负责人遇到这种情况时实际会做什么，”Lucas 补充，“我有一些假设，却没有一条适用于所有人的规则。”

Alice 开口了。

“我会澄清预期的产品行为，但不会独自决定。我们需要客服、Lucas、Thomas，可能还要一位客户来核实后果。”

Vivien 抬起眼睛。

“那我安排明天开会。”

Alice 犹豫了一下。

“你可以帮我们召集正确的人。但这个决定不应由你代替我们承担。”

她并不是想把他排除在外。然而，Vivien 仍感觉这句话代表了一种位置转移。

多年来，他一直是把问题带回正确流程的人。这一次，相关人员已经知道自己必须交流。他们不需要他把分歧变成一连串工单。

他们需要的是另一种东西：一项两位开发者都无法独自作出的取舍。

第二天，Alice 首先问每个人想避免什么。

Lucas 不希望质量负责人误以为，在数据缺失时仍拥有完整追溯信息。

Thomas 不希望模糊规则无谓地中断那些历史数据不完整的客户运营。

客服提醒大家，供应商警报并不总是意味着必须立即召回。有些客户首先要识别受影响范围、咨询质量团队，再决定采取什么行动。

“很好，”Alice 说，“那么我们的错误，也许是试图让同一个状态承载两种不同含义。”

她共享了一个方案。

一个批次可以被标记为**待检查**，同时 ERP 不会假装代替客户企业作出监管或质量决定。不过，任何可能使用或发出该批次的流程都必须显示警报，并要求执行预设的确认流程，而不能让警报悄然消失。

不确定的历史信息必须继续被标记为不确定。软件缺少必要数据的案例，也不能被表现成没有风险。

“这个，”Thomas 说，“可以在发货模块中验证。但旧订单也必须测试。”

“我还想和客户一起重新确认用词，”Lucas 补充，“‘待检查’对他们的质量负责人而言，意思可能不同。”

Alice 点点头。

他们尚未完成设计，却终于确定了谁应该选择什么。

她把临时决定、理由和需要验证的案例写进共享上下文，也保留了未解决的问题。她还在决策分级表中增加了一个条件：当一个局部决定改变了多个模块所使用信息的含义时，即使代码看起来可逆，也不再自动归为绿级。

“我们应该早点发现，”Lucas 说。

“是，”Alice 回答，“但测试正是为此存在。现在，我们知道下次要关注什么。”

Thomas 查看两份文档。

“还需要一个跨越多个模块的测试。我们有很多测试会分别核验每种行为，却很少验证它们相遇时会发生什么。”

已经听了几分钟的 Vivien 记下一条笔记。

自通话开始以来，他第一次重新遇到一个自己十分熟悉的问题：没有人能完整看到所有依赖关系。

但解决方案形成的方式让他印象深刻。Alice 没有规定界面。Lucas 和 Thomas 没有等待任务拆分。客服提供真实客户经验。他自己的作用，是揭示缺失，并帮助沟通。

这很有用。

但它与他每周一组织的交付会议几乎毫无关系。

下午，Thomas 收到另一支团队一名资历较浅开发者的消息。

他不明白为什么某段旧库存处理会进行两次看似相同的检查。代码智能体建议删除其中一次。

Thomas 打开文件。

“先别碰它，我们一起看过再说。”他回复。

他们共享屏幕看了半小时。第二次检查覆盖的是产品一次旧改动遗留下来的案例，代码中几乎没有解释。Thomas 已不记得它对应哪次发布，但知道该去哪里查。

“你的工具没有凭空编造一个错误答案，”他解释，“它看到两项相近检查，于是建议简化。不了解软件历史的话，我能理解它为什么这样判断。”

“你真的从来不用 AI 写代码吗？”开发者问。

“在这种议题上，我不会让它在完全不了解情况时提出修改。”

“可你刚才花了三十分钟，而 Lucas 也许五分钟就能验证。”

Thomas 想了想。

“也许。Lucas 也可能找到同样答案。重要的是，你要理解为什么这个检查存在。下次，无论问工具还是问我，你都会提出更好的问题。”

开发者问能否把解释记录下来。

“一定要记。我受够了只有我一个人还记得这些事。”

Thomas 挂断电话，自己也对最后一句话感到意外。

多年来，他对各种例外的了解使他不可或缺。他有时担心 AI 会降低这种经验的价值。这次讨论却让他看到另一种可能：他可以帮助别人获得他们所缺少的判断力，而不必把每个答案都留在自己手里。

他打开共享文档，写了几行说明。

周一会谈时，Hubert 问 Alice 实验进展如何。

她讲了两个互不兼容的决定、发现问题的测试，以及对框架的修正。

“所以，你们让两名开发者各自作决定，结果差点产生不一致？”Hubert 总结道。

“是。”

“这不太让人放心。”

“我们在任何变更投入生产前发现了问题，”Alice 回答，“问题不在于两个人作了决定，而在于我们的规则没有说明他们的选择应该如何相遇。”

Hubert 查看问题记录。

“过去，我们大概会多开一次审核会议。”

“也许。但再多一次会议不一定会让问题出现。我们需要有人测试完整流程，也需要明确当一个选择涉及多个领域时，责任在哪里。”

“现在由谁决定？”

“Lucas 负责自己的议题。Thomas 负责他的议题，并在技术风险方面帮助我们。我是产品一致性的守护人。遇到敏感案例，我们会征询相关专家。这些不能全部交给工具。”

Hubert 缓缓点头。

“那 Vivien 呢？”

Alice 想了几秒才回答。

“他帮助我们连接两个团队，也帮助我们理解问题卡在哪里。但他不需要分配任务，也不该由他决定产品行为。”

Hubert 像是在想另一件事。

“周五我见过他，”他说，“他向我展示了新的项目跟踪。报告很好，但我认为其中百分之八十的内容都能由机器准备。”

“他也知道，”Alice 回答。

“我不知道的是，我们应该从中得出什么结论。”

Alice 没有现成答案。

“不是他变得毫无用处，”她最后说道，“我们刚刚看到，他懂得让人们共同协作。只不过，这份工作未必还应该要求所有事情都经过他。”

Hubert 的目光停留在屏幕上。

“我们得和他谈谈。”

Vivien 收到了第二天的会议邀请。

他首先想到的是重组。过去几周，他的日程中出现了更多空白。开发者各自负责议题。工具自动收集了部分信息，而过去他常常整晚整理这些内容。他仍然处理人际困难、帮助新人融入，也会疏通复杂局面，但这些工作已不像从前那样填满日程。

他曾经喜欢自己对 Malorean 的交付不可或缺。

他不确定自己是否喜欢发现，自己的不可或缺程度正在降低。

然而，会议却从一个让他意外的问题开始。

“Vivien，你想继续做什么？”Hubert 问。

“我的工作。”他回答。

Hubert 笑了。

“我们正是在努力弄清它是什么。”

Vivien 看了看受邀来讨论团队运作的 Alice，又转向 Hubert。

“我不想自欺欺人。团队不再需要我每周询问进展。我也不想继续把时间花在重读一份智能体已经准备好的摘要上。”

“他们还缺少什么？”Alice 问。

“那些从未写下来的冲突。那些不敢说自己迷失的人。那些不知道该向谁求助的新人。横跨多个团队的决策。有时候，还缺少一个愿意花时间倾听的人。”

他停了一下。

“但这并不意味着这些事情今后仍然都必须经过我。”

Hubert 坐直了身体。

“没错。如果你的角色发生变化，我们必须明确谁来承担这些责任。我不希望取消一个职位名称，之后才发现再也没有人关心团队成员。”

Vivien 表示赞同。

他们讨论了开发者辅导、跨团队取舍，以及哪些事项仍需由管理层跟踪。Alice 讲到 Thomas 如何帮助同事更好地理解历史代码。

这次通话没有得出最终方案。

会议结束后，Vivien 在日历前又坐了一会儿。

他不像前一天那样感到受威胁，却也仍未安心。他的职业并没有消失，而是正在分解成多项责任，其中一些可以用别的方式承担。

那天晚上，他比平时更早关掉电脑。

难得一次，他没有需要补救的交付延期。

而他还不知道，该如何使用这段重新获得的时间。

10 – AI 会让企业的中间层消失吗?

如今，一个团队可以直接查阅自己的决策，在已知框架内行动，并把部分跟踪工作交给智能体。它不再那么频繁地需要负责人传递信息，或组织工作的每一步。

于是，**过去承担这些职能的管理层级会怎样?**

答案似乎显而易见：如果 AI 接管协调，中层管理者就失去了作用。然而，管理者并不只是传达指令。他还会作出取舍、辅导员工、连接团队，并处理那些不属于任何流程的问题。

因此，真正的问题不是会消失多少职位，而是要理解**这些职位的职能与权力将去往哪里。**

为协调决策而建立的层级

一位重要客户报告问题。客服把它转给负责人，负责人联系产品团队。产品团队要求工程团队估算。几位负责人汇集信息、比较紧急程度，最后还可能将这项裁决继续上报。

这条路径并不一定荒谬。每位负责人掌握部分上下文，也能代表企业作出承诺。只有当这条链条成为理解问题和作出决定的唯一方式时，它才会变得沉重。

历史上，层级制度让企业能够在信息碎片化的情况下成长。中间层把管理层目标转化为优先事项，协调团队，监控依赖，并处理资源冲突。他们也参与员工评估、成长和人际困难的解决。

但这些活动的性质并不相同。准备事故报告与决定放弃一个商业机会不同。跟踪截止日期所需的判断力，也与解决冲突或帮助员工成长不同。

我曾在 Malorean 上针对 Product Owner 提出过类似区别：一个职位可能失去作为简单中转站的合理性，却不意味着它本应承担的职能不再必要。应该审视实际完成的工作，而不只是执行者的职位名称。[1]

如果中间层级减少，谁来承担它们原有的职能?

远程工作早已改变协调规则

这个问题并不是 AI 发明的。在多个国家和时区远程工作的企业，早已必须学会在不依赖走廊交谈或临时会议的情况下作出决定。

在 GitLab，“handbook-first”原则要求以书面形式提供规则和上下文。每个人都可以提议修改，由领域负责人决定是否纳入。因此，员工无须等待管理者转达，就能自行找到信息。[2]

Alan 采用了另一种互补做法。重要决策会在 GitHub 讨论串中展开：一人作出并实施决定，另一人继续对议题成功负责，相关人员则接受征询或被告知。讨论可以开放进行，却无须取得全体一致。[3]

Basecamp 背后的 37signals，则把相当一部分协调放在书面交流、阶段总结，以及与项目共同保存的信息中，而不是让上下文散落在转瞬即逝的对话里。[4]

这三家公司并非没有管理层。它们展示的是三种方式，用于**减少对某个掌握信息、或必须不断提醒大家既有决定的人的依赖**。

这项经验对 AI 智能体十分重要。只要组织把信息明确化，系统就能查阅规则、找到取舍记录，并识别负责人。留在管理者脑中的决定，对同事与智能体都同样不可获取。

不过，也不能把远程工作理想化。微软一项涵盖六万一千多名员工的研究观察到，转向远程办公后，协作网络变得更加封闭。写得更多，并不能保证上下文共享或跨团队合作。[5]

远程工作真正留下的遗产并不是办公室的消失，而是围绕可直接获取的信息与决策来组织部分工作，从而无须永久中间人的可能性。AI 可以利用这项成果，却无法独自创造它。

AI 开始接手部分工作

文档可以保存规则，仪表盘可以显示延期。智能体还能更进一步：关联多项信息、准备进度报告、发现矛盾、提出处理方案，或执行获准的行动。

这些恰恰占据了管理者的一部分时间：汇总、报告、跟踪以及为取舍作准备。它们的成本可能下降，却并不必然消失。

HEC Paris 一项以 45 次访谈等材料为基础的探索性分析认为，结构明确且数据丰富的管理任务，比依赖信任、人际关系和判断力的任务更容易自动

化。该分析既没有测量实际减少了多少管理工时，也没有测量哪些职位变得多余。[6]

宝洁公司进行的一项实验也显示，在产品创新情境中，一名借助 AI 工作的人可以取得与一个不使用 AI 的二人小组相近的结果。工具尤其有助于结合不同职能的视角。但促进一次性的协作，不等于长期取代一名负责人。[7]

另一项涵盖 66 家企业、7,137 名员工的实验发现，在研究后半段，经常使用 AI 助手的人每周大约节省两小时电子邮件处理时间。然而，研究人员在六个月内并未观察到任务构成发生显著变化。这项研究针对办公助手，而不是能够独立执行完整流程的智能体。[8]

这种差距至关重要：**个人生产力提升，还不等于组织转型**。

要让一个层级变得不再那么必要，信息必须真正抵达团队，决策权必须发生变化，异常情况也必须有地方作出裁决。这已不只是工具问题，而是组织问题。

三家企业，三条迥异的路径

各组织的反应并不相同。以下三个案例可以区分已经开始的重组、围绕 AI 建立的愿景，以及加速转型所遇到的困难。

Bayer：真正减少层级

Bayer 通过 *Dynamic Shared Ownership*（DSO）模式，把部分决策重新分配给跨职能团队。在 2025 年致股东信中，集团负责人称已取消多达六个组织层级，并减少约三分之二的管理职位。他表示，决策变快，开发周期也有所缩短。[9]

这一规模值得关注，尽管这些效果由 Bayer 自行报告。转型要求改变日常责任，而不只是修改组织架构图上的名称：团队在局部作出更多决定，负责人更专注于辅导，部分协调职能则被重新组织。

不过，有一项细节会改变我们对案例的理解：**DSO 在生成式 AI 智能体兴起前就已启动**。Bayer 证明大型集团能够大幅减少层级，却不能证明 AI 是原因。

Block：让上下文绕过整条层级链流动

Block 更进一步地描述了它希望 AI 占据的位置。在 Jack Dorsey 与投资人 Roelof Botha 联合署名的一篇文章中，公司领导者描绘了一个由共享信息系统支撑的组织。

其愿景是让员工能够理解情境并采取行动，无须等待负责人传递上下文。作者区分了贡献者、负责跨领域结果的人（DRI），以及在继续产出的同时辅导他人的负责人，后者被称为 *player-coaches*。[10]

主要承担信息中转的管理者将因此降低重要性。相反，人员培养、结果责任和方向性决策会继续存在。

这是一家公司及其投资人描述的愿景，而不是证明该模式长期成功的独立评估。

Meta：当产出速度超过控制能力

反例来自 Reuters 对 Meta 2026 年重组计划的一项调查：更小的团队、大量使用 AI，以及计划或已经开始削减某些层级。第二轮重组最终被取消。

据 Reuters 引用的内部文件，内部软件与基础设施变更的增长速度，远远超过面向用户交付的新功能。与此同时，技术事故和解决事故所花的时间都在增加。Meta 对报道的某些方面提出了反驳或补充说明。[11]

这些数字无法把事故归因于 AI 或管理者减少，因为多项转型同时发生。但它们提出了一个决定性问题：如果企业无法核验、稳定并消化自己的产出，生产更多又有什么意义？

Bayer 展示了扁平化的可能，Block 描绘了为 AI 设计的版本，Meta 则提醒人们，当产出增长快于协调能力时，重组会带来风险。其中任何一条路径都无法单独预言管理的未来。

管理职能不会随着职位持有者一同消失

来看一个例子：客户报告软件故障。只要问题已知、解决规则清晰，团队就能行动，无须等待一连串许可。借助可靠数据，团队可以调查、修正，并通知其他人。

接着，两个诊断彼此矛盾。智能体建议一种行动，开发者建议另一种。修复还可能干扰相邻服务。

此时，需要有人揭示分歧，召集必要的专业能力，并决定承担哪一种风险。自主权没有消除协调，只是改变了协调方式。

Martin Krzywdzinski 等人进行的一项实验提供了有趣线索。在模拟生产线上，87 对参与者借助一套有时会犯错的 AI 系统处理故障。自组织团队比拥有指定领导者的团队沟通更多，解决问题也更好。分析强调了沟通的决定性作用。[12]

实验室中的两个人不能代表大型企业。但这项研究提醒我们一个宝贵区别：**团队可以没有正式领导者，却不能没有协调工作。**

这项工作可以由同事、专家、获得决策授权的人或层级负责人来完成。选择取决于能力、风险与分歧。

此外，还有一些事不是任何自动报告就能完成的：欢迎新人、帮助成长、化解紧张关系、陪伴艰难决策，或认可一项工作。并非所有管理者都能很好地完成这些任务，它们也可以用其他方式分配。但仍然需要有人拥有这项使命和时间。

Gallup 观察到，使用 AI 并不会消除员工对就业的担忧，而感受到工作支持与较低的担忧相关。这并不能证明传统管理者不可或缺，却提醒我们，组织也是一个人类共同体。[13]

最后，谁来培养那些明天必须作出艰难决定的人？

我曾在 Malorean 上提到初级开发者所面临的一种风险：当 AI 立即编写代码并修正错误时，它可能减少学习调试、理解失败和建立判断力的机会。这是一项需要研究的担忧，并非普遍能力下降的证据。[14]

未来的负责人也面临同样问题。人们也会通过旁观取舍、接受反馈，以及观察团队如何走出冲突来学习。一个取消层级的企业必须设计其他路径来传递这些经验。

最终，每项职能都有多种去向：自动化、由团队承担、转交专家，或上升到管理层。**一项任务可能消失、易手，或被添加到另一个人的工作中。**

因此，要判断扁平化组织是否有效，仅仅统计删掉多少职位并不足够。还要比较减少的跟踪工作，与智能体监督、事故处理、人员辅导及培训所新增的负担。我们目前还没有关于这种重新分配的稳健净评估。

更扁平的组织真的更自主吗?

还有最后一个问题，在组织架构图上并不那么显眼：**当中间层消失时，由谁作决定?**

我们可以设想信息更充分的团队，不必把每个选择都向上汇报，就能自由解决更多问题。但也可以设想相反情况：现场的负责人更少，管理层却通过系统直接控制更多决策。

这种张力在生成式 AI 出现前就已经存在。Nicholas Bloom 等人在一项工业企业研究中指出，为员工提供更多信息的工具与更高局部自主权相关；而便于与上级沟通的技术反而可能促进集中化，因为向最高层请求指令变得更加容易。[15]

这个旧案例不能证明当代智能体会产生什么后果，却阐明了一种机制：**传播数据不等于分配权力**。

算法管理系统让这项风险更加具体。它们可以分配任务、建议优先级、衡量绩效并触发警报。研究显示，即使仍由人类管理者负责执行或处理例外，某些指导与评价职能也可以由这些工具承担。[16]

系统不会在组织架构图上显示成一位上级。然而，如果它决定每个人收到哪些任务、用什么标准评价结果，以及在何种条件下允许行动，它就塑造了部分权力。

因此，必须审视屏幕背后的决策：谁设定智能体的目标？谁选择并能修改规则？谁可以提出异议？当算法犯错时，谁真正有权撤销它触发的行动?

一个团队可能拥有惊人的执行能力，却无法改变自身优先事项。另一个团队可能仍保留管理者，却真正能够决定自己的手段和局部取舍。

我曾在 Malorean 上描述过这种“软决策”：它制造达成一致的表象，却没有明确选择、接受的损失或最终作决定的人。如果权力依然难以识别，取消一个层级也不会改变什么。[17]

因此，可以设想一个悖论：**可见的层级更少，工具中的集中控制却更多**。

这并非不可避免。同样的系统也能帮助团队在更局部的层面作决定，并让权力规则更加透明。结果取决于企业真正选择委托什么，而不取决于组织架构图上删掉了多少方框。

真正可能消失的东西

AI 会让企业的中间层消失吗?

某些层级可能变得不再那么必要。但它们不会自动消失，消失也不意味着管理终结。

那些主要用于传递信息、汇总进展或组织重复性审批的职能，可能降低重要性。其他涉及取舍、冲突、经验传递和承担责任的职能，则必须用别的方式继续承担。

Bayer 证明了大幅减少层级是可能的，却不能确定 AI 是原因。Block 提议把更多协调交给共享系统。Meta 则提醒我们，如果只加快执行，却不衡量可靠性与人力负担，组织可能变得脆弱。目前，没有任何证据证明智能体会普遍导致管理消失。

不过，这种转移仍可能十分深刻。负责人将不再频繁地作为获取信息或审批的必经关卡，而可以把更多注意力用于一致性、异常情况和人员发展。在另一些地方，这些职能会直接回到团队手中；还有一些会被写进软件，或集中到少数决策者手中。

不一定是管理本身变得无用，而是作为强制协调机制的层级制度开始受到质疑。

对开发者而言，后果十分具体。当工作经过的中间人更少时，他得到的不只是更多生产手段，也会更直接地面对必须负责的需求、折中与结果。

这正是本书第一部分一直描述的变化：行动能力在增加，与此同时，理解、选择与承担责任的必要性也在增加。

这项变化既不保证产生更好的决策，也不保证组织更加人性化。但它迫使我们以不同方式设计组织：共享上下文，让权力可以被识别，保留经验传递，并愿意为结果负责。

关键不只在于知道 AI 如今能够生产什么，更在于决定我们想围绕这种新能力建设一家什么样的企业。

在最后一段故事中，我们将回到 Malorean Corp.，看看在组织架构图之外，管理层和团队如何体验自己作出的选择。

参考文献

[1] Le PO n'est pas un métier, c'est une fonction – et l'IA n'y change rien, elle accélère – Malorean

- URL：https://malorean.net/articles/2026-08-03-le-po-nest-pas-un-metier-cest-une-fonction.html
- 作者文章，区分一项职能的有用工作，与主要充当决策和执行中转站的职位。

[2] Handbook First – GitLab

- URL：https://handbook.gitlab.com/handbook/company/culture/all-remote/handbook-first/
- 把文档作为共享参照，公开讨论修改，同时保留对决策负责的人类负责人。

[3] Comment prendre des décisions dans un environnement de travail asynchrone – Alan

- URL：https://alan.com/fr-fr/blog/healthy-business/a/decisions-asynchrones-methode-alan
- 解释如何以书面方式讨论决策，谁作决定、谁继续负责，以及谁应被征询或告知。

[4] The 37signals Guide to Internal Communication – 37signals

- URL：https://37signals.com/how-we-communicate
- 描述以异步方式为主的内部沟通，以及如何通过书面交流长期保存上下文。

[5] The effects of remote work on collaboration among information workers – Microsoft Research

- URL：https://www.microsoft.com/en-us/research/publication/the-effects-of-remote-work-on-collaboration-among-information-workers/
- 研究转向远程工作后的协作网络，指出封闭程度提高的风险。

[6] AI Automates Routine Management and Raises the Value of Leadership – HEC Paris

- URL: https://www.hec.edu/en/dare/tech-ai/ai-automates-routine-management-and-raises-value-leadership
- 基于访谈与模型评估对管理任务进行探索性分析，并未因果测量职位削减。

[7] The Cybernetic Teammate: A Field Experiment on Generative AI Reshaping Teamwork and Expertise – NBER

- URL：https://www.nber.org/papers/w33641
- 一项在宝洁公司开展的实验，研究 AI 对创新任务与跨专业知识结合的贡献。

[8] Shifting Work Patterns with Generative AI – NBER

- URL：https://www.nber.org/papers/w33795
- 在 66 家企业开展的随机实验：观察到个人时间节省，但六个月内没有发现集体任务发生显著变化。

[9] Chairman's Letter – Bayer Annual Report 2025

- URL: https://reports.bayer.com/annual-report-2025/en/to-our-stockholders/chairmans-letter.html
- Bayer 报告其 DSO 转型大幅减少管理层级与岗位；这项转型早于智能体兴起。

[10] From Hierarchy to Intelligence – Sequoia Capital / Block

- URL：https://sequoiacap.com/article/from-hierarchy-to-intelligence
- Jack Dorsey 与 Roelof Botha 提出一种通过共享系统协调、并设有 DRI 和 player-coaches 的愿景。

[11] Mark Zuckerberg had a bold plan to replace Meta staff with AI. Here's how it imploded – Reuters

- URL：https://www.timeslive.co.za/news/sci-tech/2026-08-26-special-report-mark-zuckerberg-had-a-bold-plan-to-replace-meta-staff-with-ai-heres-how-it-imploded/
- TimesLIVE 转载的 Reuters 调查，讨论 Meta 一项转型计划所带来的组织与技术张力。

[12] How team organization influences the ability to solve automation failures – AI & Society

- URL：https://link.springer.com/article/10.1007/s00146-025-02761-5
- 对面对自动化故障的自主管理与层级式二人小组进行实验研究，衡量沟通的作用。

[13] Using AI More Does Not Reassure Workers, Managers Do – Gallup

- URL： https://www.gallup.com/workplace/713231/ai-not-reassure-workers-managers-do.aspx
- 调查员工对 AI 的担忧、受到尊重的感受，以及在工作中感知到的支持。

[14] L'IA ne devrait pas seulement servir à produire dix fois plus. Elle devrait aussi nous obliger à comprendre dix fois mieux ce que nous produisons – Malorean

- URL： https://malorean.net/articles/2026-07-23-lia-produire-dix-fois-plus-comprendre-dix-fois-mieux.html
- 作者对一种风险的思考：当 AI 完成过去培养初级开发者的任务与修正时，学习机会可能消失。

[15] The distinct effects of Information Technology and Communication Technology on firm organization – NBER

- URL：https://www.nber.org/papers/w14975
- 区分信息获取与通信技术对局部决策自主权所产生的相反影响。

[16] Algorithmic Management: From Technology to Politics and Theory – Weizenbaum Journal of the Digital Society

- URL: https://ojs.weizenbaum-institut.de/index.php/wjds/article/view/4_3_9/163
- 综述算法系统承担的指导与评价职能，以及人类介入的能力。

[17] Pourquoi les organisations préfèrent les décisions molles – Malorean

- URL: https://malorean.net/articles/2026-06-02-pourquoi-les-organisations-preferent-les-decisions-molles.html
- 讨论那些延迟取舍、掩盖已接受选择，并让责任归属难以辨认的表面共识。

11 – Malorean Corp.——重新获得的时间

休假模块演示两个月后，Hubert Félix Duca 在周一会议开始时，没有询问开发进展。

“那么，上周我们学到了什么？”

Alice 共享了屏幕。

她面前是近期客户反馈摘要、正在进行的实验结果，以及几项尚未解决的决定。一个智能体连夜对信息进行了初步归类。她早上重新审阅，剔除了几项误导性的关联，又补充了两个智能体无法独立提出的问题。

“关于追溯，数据完整时，我们的试点客户可以更快找出相关生产记录，”她开始说道，“但还有一些历史案例，我们必须明确显示自身局限。我们会和他的质量负责人测试一种新的呈现方式。”

Hubert 点点头。

“其他客户呢？”

“现在下结论还太早，”Alice 回答，“正因如此，我们没有全面部署。”

Bob 接着发言。他负责另一组产品议题，刚与售前和客服团队进行了几次交流。

“我们还有三项表面相似的需求，”他说，“智能体把它们归在同一个功能想法下，但背后至少有两个不同问题。我希望先明确目标客群，再决定是否投入。”

“你需要几名开发者？”Hubert 问。

Bob 笑了。

“目前一个也不需要。”

Hubert 也笑了。

Bob 看了一眼通知，又补充道：

“顺便说一下，客服从昨天开始测试一种新的重复需求分类方式。团队里有个人用匿名副本和智能体一起做的，想看看信息能否更快抵达正确的人。”

“谁发起的？”Hubert 问。

“客服。Alice 确认范围只限内部，而且可以撤销，结果也会接受人工检查。”

“你知道这件事吗？”

“知道这个想法，但不了解测试细节。”

Hubert 本能地想问，为什么开始之前没人通知他。他忍住了。他们已经约定，有限且可逆的尝试不再需要他的批准。

“有效的话告诉我，”他最终说道，“如果需要有人维护，我想知道负责人是谁。”

他无须批准测试，但对于可能全面推广所投入的资源，仍然负有责任。

不久之前，这种回答在产品会议上还会显得奇怪。那时，人们主要决定开发者应该做什么，以及按什么顺序做。如今，Alice 和 Bob 把更多时间用在选择需要解决的问题、确保各项行动保持一致，以及在决策越过既定框架时介入。

他们的职位名称没有改变。在行政文件中，他们仍是产品经理。

但日常工作已经不太像两个月前了。

最明显的变化，也许恰恰是那些不再引人注目的事情。

一名开发者与客户交流。另一名开发者为自己模块的改进提出 Mini PRD。一位销售人员在专门环境中制作演示变体。一名客服人员直接指出产品规则中的不一致。

这些事情不再引发关于专业边界的大讨论。

Malorean 并没有宣布所有人什么都能做。公司只是停止假设，一项行动必须经过通常担任该职位的人。

两个各有五名开发者的团队依然存在。如今，多数成员会端到端负责某些议题：理解问题、提出界定方案、与相关人员讨论、开发、测试，随后跟踪结果。

有些人比另一些人更快适应这种工作方式。两名开发者仍希望在界定范围时得到密切辅导。另一名开发者非常擅长解决复杂技术问题，却完全不想增加客户会议。没人试图强迫他们变成 Lucas 的复制品。

工单也依然存在。

人们用它协调一次干预、记录一项具体任务或跟踪修复。但他们不再试图把一个议题存在的理由、所有历史决策，以及对智能体有用的全部信息都塞进工单。

取而代之的是，每位议题 owner 都会在共用文档工具中维护一个上下文空间。其中包含目标、决策、来源、风险、未解决的问题，以及指向已有内容的链接：工单、报价、客服反馈、会谈记录、测试，有时还有重新找到的旧讨论。

无须复制一切。最重要的是，任何人都能理解一项决定为何作出、到哪里核实信息，以及谁有权改变它。

Alice 的第一份文档成为了范例。

Bob 沿用了这种方法，接着是开发者，以及客服、销售和实施团队中的若干人员。许多人像 Alice 一样发现，他们可以花一小时口头向 ChatGPT 解释各种选择，而这些内容若独自正式写下，可能要花上好几天。

并非所有存在人们脑中的知识都神奇地变得可获取。但部分理由、例外和隐性知识开始显露出来。

远程工作让这项变化显得更加自然。新冠疫情后，Malorean 已经学会增加书面沟通，让分布在法国各地的同事无须等待走廊中的偶遇也能工作。AI 加快了知识的提取与组织，却没有创造共享知识的需要。

Alice 打开一个标题为*产品决策——共同规则*的页面。

“上周又发生了一次混淆，”她向 Bob 解释，“一个团队的文档用‘已验证’表示‘已经过技术审查’，客服文档却用它表示‘已与客户确认行为’。”

Bob 叹了口气。

“这个词对每个人来说从来就不是同一个意思。”

“正是如此。我增加了一个链接，指向批次问题后我们作出的决定。”

几周前，Lucas 与 Thomas 差点让追溯模块和发货模块中两种不兼容的行为同时存在。测试期间，两项变更都被叫停。此后，团队开始警惕那些在一个模块中看似显而易见、到了别处却会改变含义的词语。

上下文提供了帮助，却也制造了新问题。

有些页面不再是最新内容。临时决定被复制得像是最终规则。一次，智能体甚至翻出一项旧的销售假设，把它说得仿佛是企业承诺。

他们不得不建立纪律：每份上下文都要注明负责人、来源、已确认决定和仍不确定的事项。结构性变更必须留下记录；敏感信息不会不加区分地开放给所有工具。

“也许我们本可以避免创建这么多页面，”Bob 指出。

“也许。但过去这些矛盾同样存在，只不过藏在我们脑中，没有人能看到。”

Alice 重读同事的最后一处修改。

“原则我同意。有关德国客户的表述不能通过：我们了解得还不够。”

Bob 修改了那一行。

守护人的工作往往就是这样：不是批准每一个界面，而是阻止一项假设在不经意间变成规则。

在另一支团队中，Thomas 刚结束与一名新开发者的交流。

他花了一小时介绍 ERP 的历史代码。开发者用助手理解一套旧的工资处理机制，随后询问为什么有个例外似乎与一般规则矛盾。

Thomas 在一次旧发布中找到了这个例外的原因。

“如果只看摘要，你可能会以为代码需要清理，”他说，“但在清理之前，必须理解自己准备删掉什么。”

“你能把这些写进团队上下文吗？”

“我正在写。”

Thomas 共享了新笔记，已经添加来源、解释和两个测试案例。

他仍然不像 Lucas 那样把技术修改的产出交给智能体。他的怀疑并未消失。不过，他愿意使用新的文档空间，也更常与他人协作，把自己的经验传递出去。

稍后，一直关注新开发者入职情况的 Vivien 与 Thomas 进行了视频通话。

“怎么样？”

“很好。他好奇心很强，删除东西前会先问问题。”

“让人放心。”

“而且我终于抽时间写下某些检查存在的原因。”

Vivien 笑了。

“所以你改变了对 AI 的看法？”

“没有。我只是改变了把这些解释全留给自己的想法。”
他停顿了一下。

“但这需要时间。如果要求我辅导其他人，却仍然用开发任务填满我的每一天，那就行不通。”

“我和 Hubert 谈过。规划中已经为辅导预留时间。这不是只有等其他事情全做完才去做的活动。”

Vivien 很清楚这意味着什么。多年来，他见过许多团队宣布，一旦“稍微有点时间”就会培训新人。那个时刻从未到来。

他们又讨论了一会儿新开发者的需要，随后 Vivien 结束视频通话。
他一次也没有打开交付跟踪表。

午餐时间，Hubert 请 Vivien 加入通话。

“我有一个问题，”他说，“现在，你有多少时间还在做去年做的那些工作？”

Vivien 想了想。

“不足以支撑原来的全职职位，如果你想问的是这个。”

Hubert 没有立刻回答。

长期以来，Vivien 都是两个技术团队的必经节点。他管理交付、依赖关系、部分紧急事项，以及开发者与业务负责人之间的讨论。在组织自主程度较低时，这些工作足以填满每一天。

现在不再如此。

每个议题都有明确负责人。开发者作出大部分局部决定。信息流动不再依赖他亲自转达。工具会准备跟踪内容并提示依赖，尽管它们的判断仍需核实。

人际问题、冲突和辅导并未消失，Vivien 仍会帮助处理。但这些事情已不再需要集中在一个专门监督日常交付的职位上。

“少开没用的会议，我不介意，”他接着说，“恰恰相反。但我花了这么多年学习管理团队，不是为了变成一个检查智能体是否正确填写表格的人。”

“我不会让你做那种事，”Hubert 回答。

他们已经准备好一项过渡安排。重要技术决策由相关守护人承担。负责辅导的人会获得明确时间。Hubert 继续负责职业发展选择，以及不能交给一份简单文档处理的人际困难。Vivien 则完成最后几项协调工作的交接。

所有这些都不完美。他们曾不得不重新处理一个案例：两个人都以为对方在关注一名处境困难的员工。一支团队也曾因没有指定决策负责人，而等待裁决太久。

每一次，他们都会修正责任及其可见方式。

“有一件事我不想做，”Hubert 说，“就是让你的职位消失，然后假装你过去所做的工作毫无价值。”

“我也不想。”

“那么，我们必须找到如今可以发挥这份经验的地方。”

Vivien 看着自己的杯子。

“我想我有一个主意。”

几天前，他的妻子又和他谈起了德国。

她是德国人。两人近来一直在讨论搬到那里生活的可能性，并不急迫：他们在法国有自己的生活、习惯和朋友。但这项设想一再出现，细节也越来越具体。

Vivien 一直推迟这个问题。

他太喜欢在 Malorean 的工作。他了解团队、他们的历史、软件与客户。离开会让他觉得自己抛弃了一些亲手参与构建的东西。

但他也喜欢挑战。职业生涯走到这个阶段，他惊讶地发现自己想接受一项新的挑战。

上周六，两人浏览了德国的住房信息，却都不愿明说。随后，妻子合上电脑。

“你知道，”她说，“我们不必再谈十年。”

Vivien 笑了。

但到了周一，他仍在想着这件事。

“德国？”Hubert 重复道。

Vivien 点点头。

“我们已经在战略会议上讨论了很多年。那里有一些客户，但主要是法国集团的子公司。我们还没有真正学会向德国企业销售 ERP。”

“确实如此。”

“每个人都说，可以等产品准备好后再去。可只要不认真和当地企业交流，我们就永远不知道缺少什么。”

Hubert 专注地听着。

“你想负责市场开发？”

“是。但我不会假装自己一夜之间变成德国市场专家。我了解我们的产品、约束、团队和客户实施方式。我知道如何召集人们共同处理复杂问题。其他东西需要学习。”

“你做过销售吗？”

“不像 Julie 或 Philippe 那样。我也不打算教他们怎么工作。我想和他们、实施团队以及业务专家合作，理解德国需求，再看看我们有没有能力响应。”

Vivien 犹豫了一下。

“我妻子是德国人。我们讨论搬过去生活已经有一段时间。这会让我有机会去做真正想做的事，又不必离开 Malorean。”

Hubert 沉默片刻。

“你知道，这不只是一个翻译项目吧？”

“当然。ERP 涉及的规则和流程会因国家而异。翻译界面远远不够。必须知道真正能承诺什么，以及哪些内容需要开发、调整或拒绝。”

“我们可能会判断错误。”

“我们很可能会在某些事情上判断错。我宁愿和客户一起发现，也不愿投入一年开发之后，才知道没人需要那些功能。”

Hubert 看着自己的项目清单。

长期以来，他们一直把德国当作未来的抱负。首先要完成延期议题、招募更多开发者、腾出时间调整 ERP、准备演示并培训团队。

这些困难没有全部消失，但其中一些已经改变了量级。

准备演示所需的工作减少了。产品变体无须每次都动用开发者便能测试。客服和售前更容易制作文档、分享知识。开发者在探索有限方向时，也不再那么缺少余力。

这并不意味着他们没有工作可做。

而是意味着，他们可以选择把部分精力用在吸收法国市场需求队列以外的事情上。

“给我准备一个第一阶段方案，”Hubert 说，“不要承诺开拓市场。列出需要进行的会面、想回答的问题、所需能力，以及决定是否继续的方式。”

Vivien 笑了。

“德国项目的 Mini PRD？”

“别笑。如果技术议题能用这种方式界定，我们至少可以试着写清市场项目的假设。”

“好。但我希望保留发现我们判断错误的权利。”

“我正希望如此。”

下一周，Hubert 召集 Alice、Bob、Vivien、Julie 和 Philippe。

介绍方案时，他没有先问需要开发多少功能。

“现在，我们拥有两个月前没有的能力。我想投入其中一部分，去理解真正进入德国市场意味着什么。”

Julie 第一个回应。

“我已经能准备德语演示了，也可以很快联系潜在客户。有了 AI，几小时就能调整材料。”

“材料也许可以，”Philippe 说，“但一位购买 ERP 的德国管理者，不会只因为演示使用他的语言就选择我们。”

Julie 抱起双臂。

“我没这样说。”

“我知道。我只是说，我们必须理解他们如何选择供应商、会咨询谁，以及需要建立何种信任。我们对这个市场还不够了解。”

Vivien 在讨论变成工具争论前插了进来。

“我想了解的正是这些。Julie，我们需要你快速准备备选思路与演示的能力。Philippe，我们需要你处理大客户的经验，以及那些不能靠漂亮界面解决的谈判。实施人员也了解客户往往发现得太晚的问题。”

Alice 开口了。

“我们这边必须非常明确产品边界。可以快速探索，却不能承诺未经批准的监管功能。”

Bob 点点头。

“还必须选择首先想了解哪些企业。德国并不是一种客户类型。如果试图响应所有人，我们会重犯在这里已经学会避免的错误。”

Hubert 看着屏幕上的一张张面孔。

他请 Vivien 提出一套做法。没有人要求他立刻组建永久团队、预留所有开发能力，或承诺商业开拓日期。

然而，几个人已经开始识别自己能够提供什么。

这在 Malorean 已经成为习惯。

“很好，”Hubert 说，“那么我的决定是：我们资助第一阶段探索，由 Vivien 负责。它必须告诉我们，哪些德国市场问题已经能够解决，哪些需要适配，哪些我们不想承担。之后，再决定是否增加投入。”

他补充道：

“而且，不能仅仅因为工具替某个人节省了时间，就把他调到这个项目。我们要看每个人实际能腾出什么，同时不能抛下今天正在服务的客户。”

Vivien 记下最后一点。他管理规划多年，很清楚纸面上释放的能力，在现实中未必可用。

接下来几天，他逐一与售前和实施团队交流。

自从演示能够通过可复用配置来准备，客户交流摘要也更容易制作后，他们的部分工作已经改变。两个人告诉他，他们用于重复制作相同材料的时间减少了，希望参与最初几次德国访谈。

“我可以帮你准备实施方面的问题，”其中一人提议，“还可以远程参加几场会谈。”

“你的法国客户呢？”Vivien 问。

“我不能全部放下。但如果把它写进规划，就能腾出时间。”

另一名正忙于多个棘手上线项目的人则表示，自己完全没有空闲。

Vivien 没有坚持。

他们制定了临时分工：预留几个时段了解德国企业的期望，在有帮助时与已成为客户的德国子公司会谈；在测量真实投入前，不承诺永久调配人员。

节省出来的时间并不会自行转化为增长。仍然需要决定如何使用它，也要接受不把重新获得的每一个小时立刻填满昨天的需求。

晚上，Alice 收到 Lucas 的消息。

我看到了德国项目。我们已经有一套配置机制，可以帮助测试某些流程差异。我可以准备一个探索环境，不触碰标准产品。你们有兴趣吗？

她重读了这条消息。

两个月前，类似提议大概会引发一场讨论：开发者是否应该等待分配议题。现在，首先要问的问题简单得多：它对战略有用吗？他可以在什么框架内推进？

Alice 打开德国项目第一阶段的上下文。Vivien 已经填入假设、负责人、边界和未解决的问题。

她回复：

可以，前提是你与 *Vivien* 一起界定范围，而且未经审核，不得把业务假设变成标准行为。从我们想了解什么开始，不要从我们知道怎么构建什么开始。

Lucas 几乎立刻回复。

收到。我来准备 *Mini PRD*。

Alice 笑了。

她想起那个周五，Lucas 向他们展示了两天内开发完成的休假模块。当时，所有人都在努力理解，他怎么能如此迅速地产出那么多。

今晚，再也没有人惊讶于他能快速构建演示。

问题在于，他们为什么需要它。

一天结束时，Vivien 把最初几场德国企业会面的日期告诉妻子。

他尚未签署任何合同。Malorean 既没有开设办公室，也没有让 ERP 适配当地市场的所有要求。他们必须了解本地用法、核实法律与监管约束，并学会说服那些从未听说过他们的企业。

未知远多于确定。

“害怕吗？”妻子问。

“有一点。”

“以前协调团队时，你看起来总是很有把握。”

Vivien 笑了。

“因为我花了很多年学习那份工作。”

“现在呢？”

他看着为了第一次访问而开始撰写的文档。

一些问题，一些假设，一些要见的人名，还有一份初步列出自己尚不了解之事的清单。

“现在，”他回答，“我又有东西要学了。”

她走近屏幕。

“那我们呢？”

“可以认真讨论搬家的事了。”

她把手放在他的肩上。

Vivien 想起旧日的交付跟踪表、周一会议，以及那些让各个项目维系在一起的工作。他喜欢过那一切，也不后悔那些年。

他同样不后悔接受团队不再以同样方式需要自己。

屏幕上，第一次德国会面已经确认。

对 Malorean 而言，这还不是征服，也不是成功。

而是一次出发。

12 – 结语

职业边界不再封闭，责任更加清晰可见

职业已无法精确限定每个人能做什么。开发者可以探索需求，产品经理可以亲手构建，销售人员可以制作原型。AI 打开了过去只对其他专业开放的通道，却没有让各种专业能力变得可以互换。

我们不再只对自己亲手执行的工作负责。我们可以负责一个横跨多个职能的议题，同时由守护人确保各自领域的一致性。行动能力可以流动，责任却不能消散。

风险来自两个相反方向：继续说“这不是我的工作”，或者认为“既然我能做，就可以独自决定”。无论是捍卫职能领地，还是让它们彻底消失，都不足以组织工作。

自主权建立在共享上下文之上

只有当每个人都拥有作出判断所需的信息时，更多自由才有意义。没有共享上下文，一项决定可能在局部看似正确，却与更广泛的需求、选择或约束相互矛盾。

因此，记录文档并不是堆积会议报告，而是让需要解决的问题、目标、假设、决策理由、边界与上报规则能够被获取。把这些内容明确化，人们就无须在每一步都重新寻找那个掌握信息的人，也能采取行动。对人类和智能体而言，上下文都成为决策工具。

守护人的工作方式也由此改变。他们不检查每一项行动，而是让框架可以被理解，守护其一致性，并在一项决定超越框架时介入。没有护栏，自由可能变成混乱；如果一切都要向上汇报审批，自由就只是一句承诺。

管理同样在演变。部分跟踪与协调可以通过其他方式完成，但仍然有人需要辅导，经验需要传递，冲突需要解决，取舍需要有人承担。即使过去赋予这些责任具体形态的任务消失了，责任本身也不会消失。

把更多能力用于真正重要的事情

人们很容易用产出的功能数量或节省的工时来衡量这场转型。AI 确实可以让我们生产更多。但如果立刻用新任务填满节省出来的时间，就只是加速旧有运作方式，却没有质疑它的目的。

我们也可以把这些能力用于更深入地理解需求、验证假设、改善已有事物，并放弃那些无法创造价值的东西。这一切都不会自动发生。我们必须主动选择，把释放出来的注意力与资源投入何处。

在 Malorean Corp.，Vivien 发现的不只是过去的工作占据了更少空间。他还发现，自己可以把经验用于另一个问题，即使成功并无保证。进步不仅存在于已经自动化的事情中，也存在于这场变化所开启的可能性里。

故事开始时，所有人都惊讶于 Lucas 构建得如此迅速。到了结尾，问题已经变成：为什么要构建、为谁构建，又会带来什么后果？

AI 的关键不在于让我们永远做得更多，而在于赋予我们能力，把更多精力用于真正重要的事情。