智能体时代的设计

AI Agent 如何改变产品、用户与设计师

薛志荣 著

xuezhirong.com

中文 · 数字阅读版

目录

00

算力成为基础设施——第四次工业革命为什么重新定义设计

2018 年，没有人问过我“AI 会不会替代设计师”。

《AI 改变设计》确实分析过哪些标准化、重复性的设计工作可能被自动化，也讨论过人工智能会怎样改变设计师的职业。这个问题由我主动提出，用来讨论技术可能带来的长期变化，并非回应当时读者的普遍追问。

当时，AI 已经可以识别图像、分析数据、推荐内容，也能在有限场景中批量生成广告图片。对多数设计师而言，这些能力仍隐藏在推荐系统、智能修图和自动化工具背后，主要改变局部工作环节。《AI 改变设计》因此把更多篇幅用于解释人工智能是什么，介绍图像识别、推荐、智能音箱、物联网和自动驾驶等能力，并讨论设计师怎样使用这些新工具。

一项同时期的调查可以作为参照。Adobe 委托 Pfeiffer Consulting，通过 Behance 作品集和同行推荐等途径，目的性招募了美国、英国和德国 75 位以上的创意从业者，覆盖设计、插画与影像、动态图形、UX/UI 等领域。74%
的受访者表示，至少一半工作时间花在重复、缺少创造性的任务上；89%
的人对能减少这类劳动的 AI 助手“很感兴趣”或“极感兴趣”。[1] 这个样本不能

代表整个设计行业，只能说明这组受访者当时的一种明确期待：让 AI 分担繁琐劳动。

八年后，主题没有中断，但问题的现实基础、技术普及度和紧迫性都变了。

生成式 AI 已经进入真实的创作流程，可以生成图片、文案、界面、视频和代码初稿。岗位技能要求也出现了变化的早期信号，但现有数据还不足以说明整个行业的稳定趋势。设计师对职业替代、技能过时和创作自主性的担忧，已经从前瞻推演变成现实感受。针对创意从业者的研究把这类担忧概括为“创造性替代焦虑”。[2] Adobe 在 2026 年调查了美国和英国共 1433 名创意专业人士与创作者；其中，在美国 403 名和英国 203 名创意专业人士子样本中，分别有 46.2% 和 51.2% 的受访者把 AI 视为对自身职业的威胁。[3]

2018 年，我主要从有限案例推测 AI 将来会怎样影响设计；到了 2026 年，设计师已经在工作中感受到它带来的效率、竞争和不确定性。职业变化始终是同一个主题，如今却有了更直接的技术基础与现实压力。本书会讨论替代焦虑，但不会把它当成解释所有变化的唯一主线。

今天需要讨论的变化，也已经超出设计师使用生成工具的范围。模型开始进入产品的运行结构：读取文件与上下文，解释不完整的要求，调用工具，维持任务状态，生成临时界面，并在获得授权后修改外部数据。这类系统仍会犯错，远未可靠到可以接管所有任务，却已经把设计从“怎样使用 AI 生产内容”推向“怎样设计一个会判断和行动的系统”。

《智能体时代的设计》由此提出一个更基础的问题：

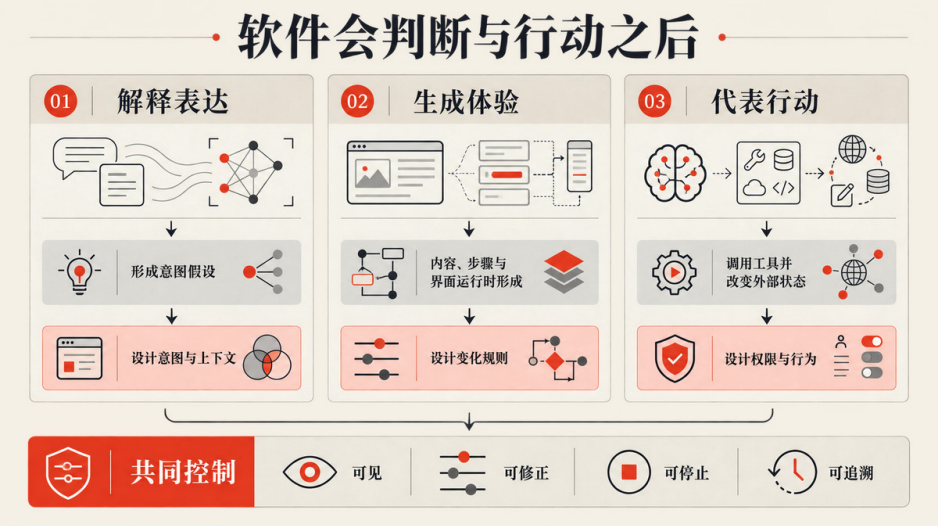

图 0-1 《智能体时代的设计》的核心问题

要回答这个问题，需要先理解为什么这些变化会在今天集中出现。

本章借用“第四次工业革命”作为观察框架。这个分期不是唯一的历史解释；它提醒我们，人工智能、机器人、物联网等能力正在进入物理系统、组织和日常生活，技术影响也会从生产工具延伸到产品、流程与制度。[4]

0.1 从设计工具进入产品结构

设计行业从来不缺少新工具。桌面出版软件替代了大量手工排版，数码相机改变了影像生产，移动互联网又把界面从桌面带到每个人的手中。每次工具升级都会提高效率，也会淘汰一部分重复工作，但工具本身仍由人逐步操作：人决定目标、选择功能，软件按照预先写好的规则返回结果。

生成式 AI 最初进入设计行业时，看起来也延续了这条路径。设计师输入文字，模型生成图片、文案、视频、界面或代码。生产速度提高了，可供选择的方案增加了，原来需要几个小时完成的工作可能在几分钟内得到初稿。这是生产工具层面的变化。

更深的变化发生在产品内部。模型可以成为运行系统的一部分，根据情境调整内容、调用能力和改变行为。设计师既可能使用 AI 画出界面，也可能负责设计一个借助 AI 调整自身行为的产品。

两项工作的责任范围不同。

当 AI 只负责生成一张候选图片时，设计师可以在交付前判断是否采用。当 AI 在产品里解释用户、选择工具和执行任务时，它的判断会直接进入用户体验。错误可能表现为读错文件、遗漏限制、调用不该使用的工具，或者把尚未确认的内容发送给其他人。

本书所说的 Agent，是能围绕目标连续使用工具、检查结果并继续行动的 AI 系统。模型参与判断下一步，工具连接外部信息与动作，权限限定系统可以做到哪一步。系统会根据工具返回的结果继续、停止或向人求助。OpenAI 从产品角度把 Agent 定义为代表用户完成任务的系统；Anthropic 的架构说明则把模型、承接结构、工具与环境放在同一个完整系统中。[5] 第一章将进一步解释工作流、模型、工具与 Agent 的区别。

Agent 没有因此获得人的主体性。本书关注的是它在产品中的作用：系统不再只返回一次内容，还可能在授权范围内连续采取行动，并产生外部影响。

可以从两个层面观察这轮变化。

AI for Design 指设计师使用 AI，提高研究、表达、原型和开发效率。

Design for AI 指设计师把 AI 放进产品，规定它怎样解释用户、能使用什么能力、可以行动到哪一步，以及失败时如何停下来。

前者改变设计师怎样生产，后者改变产品怎样运行。它们不是互斥的产品分类：同一支团队可以一边使用 AI 做研究和原型，一边设计含有 AI 的产品。本书前半部重点讨论 Design for AI；第六章会回到 AI for Design，讨论设计师的工作与交付物怎样变化。

0.2 算力从后台资源变成设计材料

计算能力一直在塑造设计边界。

图形界面能够普及，离不开处理器、显示设备和存储能力的提升；移动互联网的体验，受到手机性能、网络带宽、电池和传感器的共同限制；视频、游戏、三维空间和实时协作，也一直需要设计师在画面效果、响应速度和设备负载之间取舍。

今天不同的地方在于，模型推理开始直接参与每一次体验的生成。

用户提交一个请求之后，系统可能要处理输入、组装上下文、选择模型、调用检索或其他工具，再经过生成与推理得到结果。《Designing AI Interfaces》把这个过程概括为一条从输入处理、路由到生成与推理的计算管线。[6] 管线中的每个选择都会影响输出、等待时间、成本和失败方式。

从产业尺度看，AI 算力供给正在迅速扩张。Stanford HAI 的 2026 AI Index 汇总相关估算后，也呈现出这一趋势。[7] 供给扩张让更多产品有机会调用模型能力，但不等于这些能力已经实际部署，也不等于每个团队都能以相同成本获得。

但基础设施不等于免费资源，也不等于越多越好。

国际能源署在 2026 年的报告中指出，不同 AI 任务的能耗会随模态、生成长度、分辨率和调用次数显著变化。单位任务的软硬件效率持续提高；与此同时，全球数据中心用电量在 2025 年增长了 17%，其中 AI 专用数据中心增长了约 50%。[8] 两种趋势可以同时发生，但这些总量数据不能用来证明某一种产品选择直接造成了用电增长。

设计师面对的算力由此变成一组具体取舍。

表 0-1 算力选择对应的体验取舍

产品选择	用户感受到的变化	设计师需要考虑的问题
使用本地模型还是云端模型	响应速度、离线能力、隐私和设备负载不同	哪些任务必须在本地完成，哪些数据可以离开设备
使用快速回答还是更深推理	等待时间、成本和结果质量不同	哪些任务值得等待，怎样解释进度，用户能否中止
使用一个模型还是多个模型与工具	能力范围扩大，失败类型也会增加	系统怎样路由，失败后如何降级，用户需要知道什么
使用更多上下文与长期记忆	连续性和个性化增强，也会增加误用风险	哪些上下文真的必要，用户怎样查看、修改和删除
返回固定界面还是运行时生成界面	一致性、适配性、延迟和错误风险不同	什么必须稳定，什么可以变化，失败时退回什么
返回一次答案还是运行长任务	从即时反馈变成跨工具、长时间执行	怎样展示计划、状态、成本、权限、暂停和接管

这些选择最终都会变成体验问题。如果更好的结果需要等待一分钟，设计师要判断用户是否愿意等待；如果完整上下文能提高效果，却包含敏感数据，设计师要决定信息边界；如果低成本模型足以完成常规任务，也没有必要把所有请求都交给最昂贵的能力。

算力成为设计材料，不要求每位设计师都研究芯片。设计师需要理解计算资源怎样改变产品能够提供的体验，以及资源限制怎样决定产品必须保持的边界。

0.3 意图假设

大语言模型让自然语言、文件、图片和当前上下文进入运行时计算。系统可以据此形成对用户目标的临时解释，再选择后续步骤。

本书把这种临时解释称为“意图假设”。它可以驱动计算，却不能直接等同于用户的真实意图或行动授权。开放表达容纳的差异越多，误解的空间也越大；系统必须允许用户看见、修正或否定这项假设。第一章将从确定性软件的基础假设出发，具体解释意图翻译怎样进入产品运行过程。

0.4 软件开始从预先定义走向运行时生成

过去，页面模块、业务状态和可行路径大多由团队在产品上线前确定。生成式系统让一部分内容、步骤和界面到用户使用时才具体形成。Google Research 的生成式 UI
实验展示了这种可能，也暴露了生成耗时与结果不准确等限制。[9]

序章只需确认这个变化：部分体验开始在运行时形成，设计工作因此延伸到发布之后。哪些内容应该保持固定，哪些内容可以生成，以及两者怎样组合，将由第一章继续讨论。

0.5 第四次工业革命重组生产与组织

这个框架提供了一个组织尺度的观察位置：当 AI
进入真实工作，变化会同时出现在生产效率、流程、角色与制度中。

OECD 在 2025 年的一项研究中认为，生成式 AI 已经表现出通用目的技术的三个典型特征：可以广泛应用、能力持续改进，并能够催生大量后续创新。研究同时提醒，生产率提升不会自动或立即发生，还取决于组织是否围绕新能力重新设计流程、产品和制度。[10]

这和互联网早期有相似之处。企业给原有流程增加一个网页，并不足以完成数字化；把电脑页面缩小到手机里，也不足以完成移动化。技术进入真实工作和生活，推动流程、角色与服务重新组合之后，才会形成更深的改变。

AI 开始触及的，是一批过去很难被软件处理的认知任务：阅读非结构化资料、归纳信息、生成表达、比较方案、规划步骤、调用工具和根据反馈继续调整。机器不再只重复一条确定流程，还可以在一定范围内处理开放问题。

这些能力不足以推出“人的认知工作将整体消失”。国际劳工组织所说的“暴露”，指一项职业中的任务可能受生成式 AI
影响，不是失业概率，也不是设计岗位的替代率。其 2025

年研究估计，全球约四分之一就业处在四级暴露梯度中的某一级，只有 3.3% 处于最高等级。由于大多数职业仍然包含需要人类投入的任务，研究认为工作内容被重新组织，比整个职业被自动化更可能成为主要影响。[1] [1]

这一区分对设计尤其重要。

当 AI 用于提高生产速度时，设计师主要考虑怎样让工具更易用；当 AI 接手一段认知过程时，设计师还必须回答谁来设定目标、谁来判断例外、谁可以执行、谁检查结果，以及出现后果时由谁负责。

过去的工业化擅长把稳定产品大规模复制给更多人。生成式 AI 带来另一种可能：同一套基础能力可以根据不同人的目标与情境，在运行时生成不同内容、流程和服务。在部分产品中，生产会从复制同一个结果转向生成适合当前任务的结果，人与机器也会重新分配部分工作过程。

这种变化也伴随着成本。算力和数据中心会消耗能源，模型可能放大偏见，基础设施可能集中在少数组织手中，自动化也可能把新的监督劳动和风险转移给用户。第四次工业革命不会自动导向更好的未来。技术能做什么，与社会最终选择让它做什么，是两个不同的问题；设计正处在两者之间。

0.6 AI 改变设计的起点

前面的变化不是一条由技术自动推动的单线因果。算力与模型能力提供了新的条件；产品团队和组织决定怎样把这些条件写进软件；用户、制度与使用情境又会改变系统最终承担什么角色。设计参与其中，负责把能力转化为可以理解、约束和评价的人机关系。

在部分任务中，这种关系已经开始变化：

图 0-2 部分任务中的人机关系从操作走向委托

委托仍需保留人的决定权。用户可以把重复执行和资料整理交给系统，但仍需决定目标、范围、不可触碰的内容和完成标准。共同规划、行动确认和高风险操作保护，是实现这种关系的具体机制。[12]

AI 产品不能只追求“少点几次”“自动完成更多”或“结果看起来更聪明”。系统的行动范围越大，越需要让人知道它理解了什么、依据了什么、准备做什么、已经改变了什么；任务持续时间越长，越需要明确系统必须停下并交还决定的条件。

任务风险与用户意愿各不相同。整理私人笔记与发起付款的风险不同，生成内部草稿与代表用户对外承诺的后果不同，新手和专家对过程的需要也不同。设计要为任务、风险和用户能力找到相匹配的人机关系，而非把所有产品推向最高程度的自动化。

AI 改变设计始于软件能力跨过一条边界：机器开始参与理解、生成和行动，不再只等待精确操作。

一旦跨过这条边界，设计需要重新回答一组基本问题：

1 设计对象还是不是一组页面和流程？

- 用户不再操作每一步之后，会变成什么角色？
- 面对无法穷举的结果，设计方法应该怎样改变？
- 当生产变得便宜，怎样判断一个结果真的足够好？
- 设计师要怎样进入代码、模型、规则和运行时系统？
- 当 AI 越来越能替人完成任务，设计怎样继续为人保留选择、判断与成长？

后面的章节会从这些问题出发，依次讨论确定性软件的基本假设如何松动，设计对象怎样从界面扩展到意图、上下文与行为，用户怎样从操作者变成委托者和监督者，设计方法、质量标准与设计师工作如何变化，以及设计最终可能走向怎样的未来。

全书的逻辑可以展开为四个相互作用的层次：

图 0-3 从能力条件到设计选择的全书逻辑

注释

1 Pfeiffer Consulting, Creativity and Technology in the Age of AI, Adobe 委托研究，2018，https://www.pfeifferreport.com/wp-content/uploads/2018/10/Creativity-and-technology-in-the-age-of-AI.pdf 。该研究采用 45 分钟至 2 小时的自由访谈，以 Behance 作品集和同行推荐招募为主，适合说明该样本的观点，不适合推断行业总体比例。

2 Ka Yan Chung, Henry Ma, Yuet Kai Chan, “Measuring Creative Displacement Anxiety in the Age of Generative AI: Scale Development and Validation,” IASDR 2025，https://doi.org/10.21606/iasdr.2025.46 。

3 Adobe, “How AI Is Redistributing Creative Work,” 2026，https://www.adobe.com/ai/research/202606/ai-is-redistributing-creative-work.html ；研究报告 PDF：https://www.adobe.com/cc-shared/assets/ai/research/adobe-ai-research-june-2026-how-ai-is-redistributing-creative-work.pdf 。调查为 2026 年 4 月 27 日至 30 日的横截面研究；公开材料没有完整披露题目、招募与加权方法，因此这里只报告创意专业人士子样本，不外推行业总体。

4 World Economic Forum, “What is the fourth industrial revolution?”, 2016，https://www.weforum.org/stories/2016/01/what-is-the-fourth-industrial-revolution/ 。

5 OpenAI, “A practical guide to building agents,” https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/ ；Anthropic, “Trustworthy agents in practice,” 2026，https://www.anthropic.com/research/trustworthy-agents 。

6 Louise Macfadyen, Designing AI Interfaces, O’Reilly Media, 2026。

7 Stanford Institute for Human-Centered Artificial Intelligence, The 2026 AI Index Report: Research and Development, 2026，https://hai.stanford.edu/assets/files/ai_index_report_2026_chapter_1_research_development.pdf ；Epoch AI 方法说明：https://epoch.ai/data/ai-chip-sales-documentation/methodology 。报告估计 2025 年第四季度全球 AI 算力容量约为 1710 万个 H100 等效单位，自 2022 年以来平均每年增长 3.3 倍。H100 等效值按最大 8 位性能换算，是峰值容量估计，不代表实际 H100 数量、已部署容量或普遍可用性。

8 International Energy Agency, Key Questions on Energy and AI, 2026，https://iea.blob.core.windows.net/assets/3179f7f8-01f6-4dd6-bffa-c9f7b73f1dc9/KeyQuestionsonEnergyandAI.pdf 。报告中的任务能耗比较依赖时长、分辨率和调用假设，适合说明差异范围，不宜当作精确的产品系统能耗。

9 Google Research, “Generative UI: A rich, custom, visual interactive user experience for any prompt,” 2025，https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/ 。该实验依赖工具、系统指令、后处理与视觉配置；报告同时说明部分生成超过 1 分钟，并会偶发不准确。

10 Flavio Calvino, Daniel Haerle, Sarah Liu, Is generative AI a General Purpose Technology? Implications for productivity and policy, OECD Artificial Intelligence Papers, No. 40, 2025，https://doi.org/10.1787/704e2d12-en 。

11 International Labour Organization, Generative AI and Jobs: A Refined Global Index of Occupational Exposure, 2025，https://researchrepository.ilo.org/esploro/outputs/encyclopediaEntry/Generative-AI-and-jobs-a-refined/995653520102676 ；ILO, “One in four jobs at risk of being transformed by GenAI,” 2025，https://www.ilo.org/resource/news/one-four-jobs-risk-being-transformed-genai-new-ilo%E2%80%93nask-global-index-shows 。

12 Microsoft Research, Magentic-UI Report,
https://www.microsoft.com/en-us/research/publication/magentic-ui-report/
。该系统以共同规划、共同执行和 action guards
处理用户参与与高风险操作，支持“委托仍需控制机制”的产品判断。

第一章

01

设计的基础假设变了——从确定性软件到生成式系统

假设一位设计师需要把一份几十页的用户研究报告，整理成一场面向管理层的 20 分钟汇报。

在传统软件里，他通常会先创建演示文稿，选择模板，再阅读报告、提炼结论、拆分章节、复制文字、插入图表、调整版式，最后逐页检查内容。软件提供了文本框、图表、对齐、动画和导出等能力，但“这场汇报应该讲什么、按什么顺序讲、哪些内容最重要”，主要由设计师自己判断。为了完成目标，他需要先理解软件拥有哪些功能，再把一个完整目标拆解成许多具体操作。

在生成式系统里，他可以先选中研究报告，然后告诉系统：“请把这份报告整理成一场面向管理层的 20 分钟汇报，重点保留用户问题、业务风险和三项行动建议，所有数据都要标明来源。”系统可能读取文档，提取证据，组织结构，生成页面，再把一份可以继续修改的演示文稿交给他。

序章把系统依据用户表达和当前上下文形成的临时解释，称为“意图假设”。在这个例子中，被选中的报告、用户的要求和当前工作状态都是意图信号。系统依据这些信号，暂时把“整理”解释为制作一份供管理层使用的汇

报，再据此形成计划并生成内容与页面。这项假设可以驱动后续计算，但它仍然可能错，也不等于用户已经授权系统采取所有后续动作。

“意图假设”是本书用于分析产品行为的概念，不是行业通用术语，也不要求产品内部保存一个同名字段。设计师需要关注的是：系统依据哪些信号形成判断，这项判断如何影响后续计划与行动，以及用户能否看见和修正它。

两种方式服务的是同一个目标，却建立在两套不同的基础假设上。

第一套方式假设，产品团队已经提前定义好软件可以做什么，用户需要通过固定界面选择功能、安排步骤并完成操作。第二套方式则允许用户先表达目标，由系统形成意图假设，再在运行时选择能力、组织步骤并生成结果。用户可以省去一部分中间操作，但仍需要说清对象、听众、重点、限制和完成标准，并检查系统的解释是否成立。

第二种方式有明确的适用边界。对于调整一行文字的位置、输入一个确定金额、确认一项不可逆操作，固定控件往往更快、更准确，也更容易预测。生成式系统扩展了产品能够处理的问题范围，却不保证每项任务都更高效。过去必须由用户自行拆解的开放目标，现在有一部分可以交给系统参与解释。

因此，AI 带来的关键变化在于，意图翻译开始从产品发布前延伸到产品运行时。界面增加输入框、设计师是否继续画页面，都只是这项变化的表层表现。

过去，设计师和开发者先把人的目标翻译成有限的功能与流程，用户再把自己的目标翻译成点击、输入和选择。现在，模型开始进入第二次翻译：系统接收意图信号，形成可修正的意图假设，再据此选择计划、行动和结果形式。

这一章将回到设计的基础，讨论传统数字产品建立在什么条件之上。理解固定界面、确定路径和直接操控为什么成立，是判断它们今天发生何种变化的前提，也能避免把“AI 改变设计”误解为一次新的工具升级。

1.1 设计一直都在翻译人的意图

用户打开一款产品时，很少是为了使用某个按钮。

他打开购物软件，是为了买到一件适合下周出差的外套；打开图片软件，是为了让照片中的人物更突出；打开报销系统，是为了把一次出差产生的费用交给公司处理。搜索、筛选、框选、填写和提交只是达到目标的手段。

人的目标不能直接进入机器。机器需要接收到可以处理的输入，按照规则改变内部状态，再把结果反馈给人。设计长期承担的工作，就是在人的目标与机器可以执行的动作之间建立一座桥。

1.1.1 从“我想做什么”到“我应该怎样操作”

Don Norman 在《设计心理学》中把人与产品的互动描述为一个从目标到行动、再从结果回到评价的循环。人先形成目标和意图，再选择并执行动作；系统发生变化后，人还需要感知、解释并判断结果是否符合原来的目标。目标与可执行动作之间的距离，被称为“执行鸿沟”；系统状态与人的理解之间的距离，则是“评估鸿沟”。[1]

对数字产品来说，界面的作用就是缩短这两段距离。

用户想买一件外套，产品用搜索框、类目、筛选器、商品卡片和购物车，把一个模糊目标变成一组可以执行的动作。用户点击“加入购物车”后，按钮状态、数量角标和价格汇总又告诉他系统发生了什么变化。

如果没有这些界面，用户就需要知道数据库怎样存放商品、系统使用什么命令查询库存、订单接口需要哪些参数。界面把技术结构隐藏起来，并提供一套更接近人类任务的表达方式。

因此，设计同时安排视觉元素与交互关系。一个按钮的名称、一组信息的顺序、一个表单的字段和一条流程的分支，都在回答同一个问题：怎样把用户想做的事，转化为系统能够识别的动作？

1.1.2 设计师和用户共同完成了两次翻译

在传统软件中，这种翻译通常分成两次。

第一次发生在产品发布前。设计师、产品经理和开发人员研究用户任务，再把这些任务拆成产品能力、页面、组件、字段和规则。团队认为用户需要按价格筛选，就设计价格区间；认为报销需要区分交通和住宿，就设计费用类型；认为一张图片需要局部修改，就提供选择、蒙版和图层。

第二次发生在产品使用时。用户理解产品提供的功能，再把自己的具体目标翻译成操作。他需要判断应该进入哪个页面、选择哪个工具、填写什么内容，并按产品规定的顺序完成任务。

这两次翻译塑造了过去几十年的数字产品。设计师决定机器向用户提供什么语言，用户学习这套语言并用它指挥机器。

一个熟练的设计师使用图像软件时，看起来几乎不需要思考，是因为许多翻译已经变成了习惯。他知道怎样建立选区，知道图层和蒙版分别解决什么问题，也知道哪个参数会影响边缘。新用户也许能用一句“把人物留下，换掉背景”说清目标，把它翻译成软件操作却需要一段学习过程。

1.1.3 意图进入了产品的运行过程

用户意图一直是设计的起点。AI 带来的变化，是意图不再只在用户研究和产品定义阶段被处理，也以意图假设的形式进入了产品的运行过程。

固定界面把意图压缩成有限选项。用户想“让这封邮件显得坚定，但不要让客户感觉被冒犯”，传统编辑器只能提供输入、删除、复制和格式调整，语气判断需要由用户自己完成。生成式系统则可以接受这段自然表达，推断用户希望保留的关系与态度，再生成一个候选版本。

机器现在可以在较大范围内参与解释“用户想做什么”。意图信号既可以是明确说出的一句话，也可以来自当前文档、选中对象和任务上下文。[2] 这些信号共同构成系统此刻用来计算的输入。系统先依据它们形成意图假设，再选择计划与行动；如果信号不足或相互冲突，就需要追问、缩小范围或停止。

不过，“参与解释”不等于“准确理解”。用户说“整理一下”时，系统仍然不知道他想重命名文件、生成摘要，还是重新组织项目；用户说“帮我处理”时，也没有说明系统可以只给建议，还是可以直接修改和发送。自然语言降低了表达门槛，同时也把歧义带进了系统。

因此，AI 没有消除意图翻译，而是改变了翻译的分工：设计师仍在发布前定义能力与边界，用户在使用中提供意图信号，系统在运行时形成可供修正的意图假设，再把它转化为计划或行动。

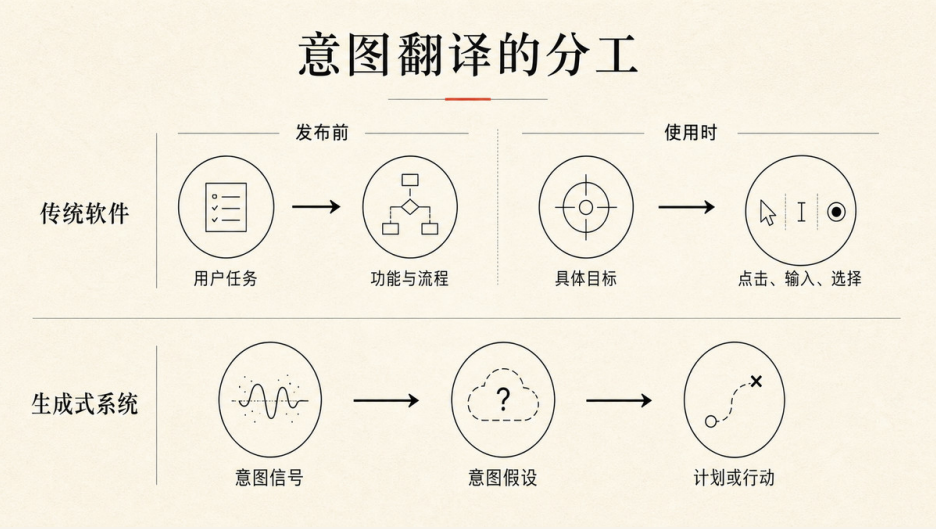

图 1-1 传统软件与生成式系统中的意图翻译分工

1.2 固定界面是一套预先规定的表达语言

如果把数字产品看成一场人与机器的交流，固定界面就是产品团队提前写好的一套语言。

按钮和菜单像词汇，组件规则像语法，页面流程像允许用户说出的句式，反馈与状态则告诉用户机器是否听懂。用户不能随意向系统表达任何目标，只能使用产品提供的词汇，并按照它规定的方式组合。

1.2.1 图形界面降低了精确语法的负担

在命令行中，用户需要记住命令名称、参数和语法。一个字符写错，系统可能无法执行。图形界面的重要变化，是把许多抽象命令变成可见对象与直接动作。

Ben Shneiderman 在 1983 年提出“直接操控”时，总结了几个重要特征：持续显示用户关心的对象，用物理动作或带标签的控件代替复杂语法，让操作快速、增量、可逆，并让结果立即可见。[3] 文件可以被看见和拖动，文字可以先选中再修改，数值可以通过滑块调整。用户不必先写出一条完整命令，而是可以操作一步、观察一步、再决定下一步。

直接操控解释了固定图形界面的一部分价值，但不等同于图形界面的全部。导航、信息结构、协作状态和复杂业务规则同样需要设计；有些任务也会同时使用表单、搜索、快捷键或命令。

这套结构后来成为大量数字产品的基础。它让机器的能力变得可发现，也让错误更容易被控制。用户看到“删除”按钮，会知道系统允许删除；看到按钮变灰，会知道当前条件不允许执行；修改后看到页面立即变化，就能判断操作是否生效。

固定界面因此是一套认知脚手架。它用视觉与交互结构，把抽象的系统能力变成可见、可学和可重复的动作。

1.2.2 界面同时规定“能说什么”和“怎样说”

每一种控件都在限制表达范围。

日期选择器要求用户提供一个符合日历规则的日期；单选框要求几个选项只能选择一个；数字输入框可以限制最大值和最小值；提交按钮把一组信息组合成一次完整动作。这些限制有时会让用户觉得不够自由，却也减少了歧义。

假设用户要预订一次会议。固定界面会要求他选择日期、开始时间、结束时间、参会人和会议室。产品团队提前决定了哪些信息是必须的、哪些组合无效、冲突时怎样提示。用户可能需要填写多个字段，但系统不必猜测“

明天下午找个大家都方便的时间”到底指几点，也不必判断“大家”包含哪些人。

从这个角度看，传统界面是一份预先约定的交流协议。

表 1-1 固定界面中的表达规则

界面部分	在交流中的作用	设计团队提前决定的内容
导航与菜单	告诉用户有哪些主题和能力	功能怎样分类，入口放在哪里
按钮与控件	提供可以执行的动作	用户能做什么，动作需要什么条件
表单与参数	规定输入的结构	哪些信息必填，允许什么格式与范围
页面与流程	规定表达顺序	任务分成几步，哪些分支可以进入
反馈与状态	回应用户的操作	系统发生了什么，下一步可以做什么

这套语言的优点是明确。产品能做什么，大部分可以从界面中看见；用户完成一次任务后，通常可以用相似方式再完成一次；团队也可以针对每个入口、状态和异常编写测试。

1.2.3 界面没有呈现的能力，通常等于不存在

固定界面的另一面，是产品团队必须提前决定什么值得被做成入口。

如果邮件产品没有“延迟发送”，普通用户就很难使用这项能力；如果报销系统没有某类费用，用户只能选择相近类型或联系人工；如果图像软件没有提供某种处理方式，用户就需要组合多个工具，或者把任务交给其他软件。

理论上，底层代码可以继续增加新能力。但每增加一个功能，团队都需要决定名称、位置、参数、状态、帮助信息和异常处理。功能越多，菜单和页面越复杂，用户发现正确入口的成本也越高。

所以，固定界面既是能力的表达，也是能力的边界。产品团队在发布前选定了一组值得支持的意图，再把它们变成有限的功能语言。用户能够表达什么，很大程度上取决于这套语言里已经有什么。

界面规定用户能怎样表达，状态与规则则规定系统收到表达后怎样响应。两者共同由团队在发布前定义：前者给出入口、对象和操作，后者给出条件、结果、分支与异常。固定界面的可预测性，正是建立在这套预定义关系之上。

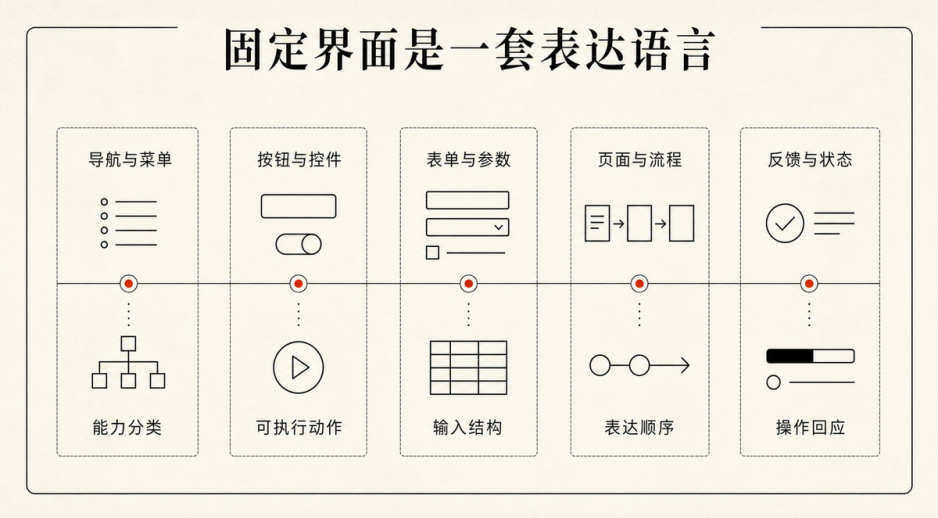

图 1-2 固定界面怎样构成预定义的表达语言

1.3 确定性来自提前枚举可能性

传统软件的稳定感来自产品团队对开放现实的收敛，而非现实世界天然确定。这里所说的“提前枚举”，是定义主要状态、规则、分支和可预见异常，不要求穷举现实中的每一种情况。

一个按钮只有几个状态，一张表单只有若干字段，一条业务流程拥有清楚的分支。设计师画出页面与异常，开发人员把它们写成规则，测试人员再检查不同输入会进入哪个结果。团队用大量提前定义的工作，换来了用户使用时的相对确定。

1.3.1 稳定路径是被设计和实现出来的

仍以差旅报销为例。传统系统可以规定：住宿费用必须填写入住日期，金额超过一定范围需要补充说明，缺少发票时不能提交，提交后进入直属主管的审批队列。

每一条规则都对应一组可以观察和测试的状态。

1. 条件满足时，按钮可以点击；
1. 条件不满足时，系统显示明确提示；
1. 权限不足时，用户不能执行；
1. 网络失败时，申请保留为草稿；
1. 审批完成后，状态从“处理中”变为“已通过”或“已驳回”。

当团队把主要情况列得足够完整，用户会获得一种稳定的因果关系：我做了什么，系统就会怎样回应。设计师也可以用流程图、状态图和可点击原型相对完整地表达产品。

所谓“确定性”，很大一部分来自这种提前枚举。产品不需要在运行时重新理解“什么叫住宿费用”，因为团队已经在数据结构和规则里定义好了；也不需要临时决定审批顺序，因为代码路径已经规定了下一步。

1.3.2 产品边界就是团队能够预先处理的可能性边界

现实中的任务远比流程图复杂。

用户可能丢失票据，可能跨国出差，可能遇到临时政策，也可能同时为多个项目垫付费用。传统系统用标准流程处理一部分复杂性，用异常分支处理另一部分，再把剩下的情况交给人工。

团队能够想到、定义、实现和维护多少状态，产品就能够稳定覆盖多少情况。长尾需求不是完全无法解决，只是每多支持一种情况，通常都要增加新的字段、规则、页面或人工流程。

成熟软件经常变得复杂，主要原因是越来越多现实差异被写进产品，并非团队单纯偏爱增加功能。一个看似基础的订单系统，背后可能有地区、税率、库存、优惠、退款、权限和合规等大量分支。

1.3.3 传统软件也不是绝对确定的

把传统软件称为“确定性软件”，容易产生一个误解：过去的系统每次都会产生完全相同的结果，AI 出现后产品才第一次面对不确定性。

事实并非如此。搜索结果会变化，推荐系统早已使用概率模型，定位和语音识别会受环境影响，网络、库存和协作数据也一直在变化。即使同一个页面，用户在不同时间看到的内容也可能不同。

这里讨论的是产品主要控制方式的变化，不能概括为“过去完全确定，现在完全随机”。

在传统产品中，概率性能力通常被包在相对明确的功能与流程里。推荐列表可以变化，但“加入购物车”仍然有确定含义；搜索排序可以变化，但筛选和分页仍按固定规则工作。团队可以让不确定性停留在某个局部，不必让它控制整条任务路径。

生成式系统则把模型推断放到了更核心的位置。系统可能先解释用户意图，再决定需要哪些信息，选择调用什么能力，并生成以前没有被逐项定义的结果。这类推断会随任务细节和环境变化，也会挑战传统界面强调的一致性和可预测性。[4]

1.3.4 决定从发布前移动到了运行时

确定性软件和生成式系统最重要的差异，可以理解为“决定在什么时候发生”。

过去，团队在发布前决定用户能做什么、任务怎样分步、每一步通向哪里。运行时，软件主要执行已经写好的规则。

现在，团队仍然需要定义能力和边界，但一部分具体决定被推迟到运行时。模型根据用户输入和上下文，判断这次任务应该怎样解释、组织和完成

。同一句请求，在不同文件、不同用户和不同环境中，可能得到不同的结构与步骤。

这并不意味着规则会消失。生成空间越开放，越需要固定规则守住权限、数据、格式和风险底线。团队不再预先画出每一条可行路径，而是定义可用能力和边界，允许系统在这个空间里为当前任务选择路径。

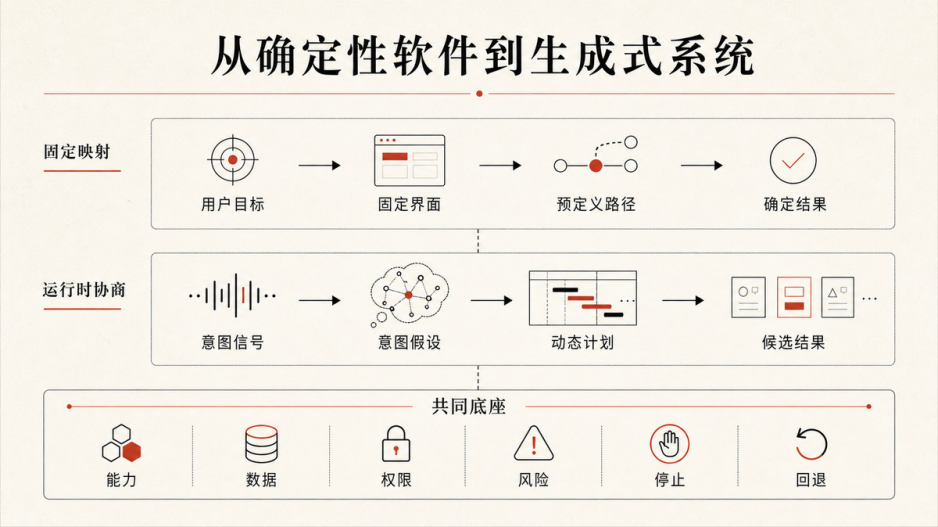

图 1-3 从发布前的固定映射到运行时协商

运行时选择只说明部分决定发生在使用过程中。系统是否属于 Agent，还要看模型是否持续控制多步工作流和工具使用，以及系统是否具备完成判断、停止条件和移交机制。1.6 节将具体区分这些运行形态。

1.4 固定框架带来了稳定，也带来了限制

固定界面能够延续几十年，是因为它解决了许多重要问题。用户可以看见能力，形成空间记忆，预测操作结果，并在出错后回到熟悉位置。团队也可以控制性能、测试路径、说明责任，并让大量用户使用同一套稳定流程。这些价值不会因运行时生成的出现而消失。

1.4.1 稳定本身就是体验价值

用户对一款软件熟悉之后，常常能在没有仔细阅读的情况下完成操作。他知道保存在哪里，知道关闭窗口会发生什么，也知道怎样撤销。界面位置和行为的稳定，逐渐变成一种外部记忆。

1989 年，John Mitchell 与 Ben Shneiderman 招募了 73 名大学生，剔除 10 份不完整或受程序、设备故障影响的数据后，有效样本为 63 人。参与者以不同顺序使用固定菜单和动态菜单，分别完成两组相同的 12 项任务。实验使用键盘光标键选择菜单项；动态菜单按照每位参与者的选择频率持续重排，把最常使用的项目放在顶部。[5]

首次使用动态菜单的一组耗时显著更长；完成第二组任务时，两种菜单的耗时已无显著差异，菜单样式对操作次数和错误数也没有显著影响。事后询问中，81% 的有效参与者更偏好固定菜单。研究者还在现场观察到，部分参与者会记住菜单项的相对位置，并在菜单重排后感到迷失；这是观察性解释，不是单独测量的认知结论。

这项研究以学生和计算机新手为主，测试的是键盘操作的菜单，不能直接预测今天的生成式界面。它能支持的判断更窄：自适应变化可能带来学习与控制成本，团队需要同时评估效率、位置稳定性和用户偏好。

对于高频、明确和高风险的任务，固定结构尤其重要。输入支付金额、选择药物剂量、确认删除范围时，用户需要知道字段的含义和动作的后果，而不是每次面对一个重新生成的表达方式。

固定界面还可以降低验证成本。一个经过测试的日期组件不必在每次任务中重新生成，一条明确的业务规则也不必交给模型反复解释。确定的部分越稳定，团队越能把注意力留给开放问题。

1.4.2 固定能力目录难以覆盖长尾目标

固定框架的限制也来自同一个原因：团队必须提前决定支持什么。

大多数用户共享同一套页面，复杂目标需要被拆成统一步骤，无法归类的需求则进入“其他”或人工服务。用户越专业，越可能发现产品只覆盖了常见路径；用户越不熟悉软件，越可能不知道应该把目标拆成哪些操作。

假设一位用户希望“把最近三个月客户反馈中与续费风险有关的问题找出来，按影响程度分组，并给每组附上原始证据”。在传统分析软件中，他可能需要导出数据、清洗字段、设置筛选、建立标签、编写公式和制作图表。每一项能力都存在，但完整目标没有对应的单一入口，用户必须自己设计工作流。

产品也可以为这个目标增加一个功能。问题是，下一位用户可能想按地区比较，另一位想结合销售记录，还有人希望使用自己的风险标准。开放目标会迅速产生组合爆炸，团队很难为每一种组合都设计按钮和页面。

生成式系统的价值，就出现在这些难以提前枚举的区域。它不必为每个目标增加永久入口，而可以根据当前数据、要求和限制临时组织一条处理路径。

1.4.3 开放表达也会带来新的负担

固定选项让用户受到限制，空白输入框则可能让用户无从开始。

当产品只显示“你想做什么”时，用户需要知道系统能做什么，知道哪些背景应该说明，也要能够把模糊想法写成有效要求。表达空间扩大后，发现能力、控制结果和修正误解的责任可能重新回到用户身上。

自然语言也不适合所有细节。用户可以说“把语气变得更正式”，却可能更愿意用滑块调整篇幅；可以说“分析这些数据”，却更适合直接勾选时间范围和比较对象。Microsoft Research 的 Dynamic PRC 研究原型会根据当前提示和对话历史，生成单选、复选或文本控件，再用这些选择细化回答。[6]

在一项由 16 名熟悉生成式 AI
的技术专业人士参加的受控研究中，参与者完成了 6 项代码解释、复杂主题理解与技能学习任务。他们整体更偏好动态方案，并报告了更强的控制

感和更低的上下文表达门槛；同时，他们也更难预判某个动态控件会怎样影响输出。这项小样本研究只支持特定理解任务中的原型机制，不能推广为所有 AI 界面的普遍效果。

因此，新旧范式不应该被理解为聊天框与按钮的竞争。固定控件擅长表达确定对象、范围和参数，自然语言擅长表达开放目标和例外，生成式能力擅长处理尚未被做成功能入口的组合。好的产品会根据任务，把三者组织在一起。

1.4.4 AI 扩大设计空间，并没有取消固定框架

生成式系统可以让过去无法被产品化的长尾问题进入软件，但不代表所有部分都应该生成。

登录、权限、支付、版本、审计和关键状态仍然需要稳定规则；常用操作仍然值得保留为按钮；用户反复执行的成熟流程，可能更适合被固化为模板或工作流。当输入开放、规则难以维护、环境变化较大或任务步骤难以预先确定时，运行时生成可能更有价值；如果确定性方案已经足够，就没有必要增加生成环节。

实际产品可以用确定的外层承载生成能力。固定部分让用户知道系统的边界、状态和控制方式，模型则在边界内处理过去难以枚举的问题。

图 1-4 固定框架、运行时生成与混合方案

1.5 系统开始参与运行时翻译

命令、图形界面、搜索和自然语言没有线性替代，而是在许多产品中长期共存。它们把不同负担分配给用户和系统：命令要求用户掌握精确语法，图形界面把一部分语法变成可见对象，搜索承担相关性匹配，生成式系统则尝试从句子、文件、图像、语音和当前上下文中推断目标。

从这些交互方式可以观察到一种负担转移：用户可能少承担一部分语法记忆、功能检索和步骤拆解，系统则多承担匹配、纠错和解释。直接操控与 Dynamic PRC 分别提供了两个具体例子，但它们不能证明一条适用于所有产品的单向历史规律。[36]

1.5.1 机器开始承担更多解释工作

在固定软件中，用户需要先找到正确功能。如果他想把一张照片的背景换掉，就要知道选择、蒙版和图层在哪里；如果他想比较一组数据，就要知道筛选、透视表或公式应该怎样使用。

生成式系统允许用户从目标开始。用户可以先选中照片，再说“保留人物，把背景换成傍晚的城市街道”；也可以选中表格后提出“比较三个地区最近六个月的退款原因，并标出异常变化”。系统再根据当前对象和要求选择能力。

语言在这里不是另一种命令行。命令行要求用户使用机器规定的精确语法，生成式系统则尝试接受不完整、自然甚至带有歧义的表达，并补足中间步骤。

在对象和边界能够由上下文、直接操控或结构化控件补充的任务中，这种方式可能降低用户首次使用某些专业能力的操作门槛。它没有取消学习成本：用户仍需判断系统读了什么、怎样解释要求，以及结果是否可以采用。

1.5.2 “理解用户”仍然是一种推断

处理自然语言只让软件获得了更多推断信号，无法让它直接知道用户心里在想什么。

系统看到的是用户输入、选中对象、当前页面、历史对话、文件内容和被允许读取的其他信息，再根据这些信号推断最可能的目标。信号越丰富，结果可能越贴近情境；与此同时，隐私、误用和越界风险也会提高。

用户说“帮我安排下周的项目会议”，系统可能需要知道项目成员、时区、日历空闲和会议时长。但能够访问日历，不代表可以读取所有私人日程；知道过去会议通常是一小时，也不代表这次应该自动采用相同设置。

因此，所谓从“人适应机器”走向“机器理解人”，增加了一种新的协商关系，没有彻底移除用户的学习成本。系统需要说明自己使用了哪些上下文，用户需要判断这些上下文是否正确，也需要在误解发生时能够低成本纠正。

用户仍然需要学习软件，只是学习的内容从“功能放在哪里”，转向“系统使用了什么信号、会如何变化，以及什么时候不应该依赖它”。产品需要帮助用户建立并调整心理模型；当 AI 无法完成任务时，还应保留不依赖 AI 的完成路径。[7]

1.5.3 最自然的表达不一定是最有效的表达

人类可以用语言说明目标，也会用手指出对象、用表格比较差异、用滑块控制程度。所谓自然交互，不应该只剩下聊天。

一句“把这里改得更简洁”如果配合已经选中的段落，就比单独输入一段说明更准确；一句“给我三个方案”如果配合预算、日期和风险等级控件，就比让系统猜测所有限制更可靠。

可以把新的意图表达理解为三种方式的组合：

1. 用自然语言表达目标、原因和例外；
2. 用直接操控明确对象、范围和局部修改；
3. 用结构化控件表达参数、选择和不可含糊的约束。

系统要形成可用的意图假设，需要让产品用多种通道收集足够的意图信号，不能把责任全部推给一条所谓“完美提示词”。

1.5.4 运行时决定也会改变输出形式

当系统能够解释目标并生成代码，界面本身也可以成为一种输出。对同一项任务，系统可能返回文本，也可能在运行时组织表格、模拟器或临时工具。Google Research 的生成式 UI 实验展示了这种可能。[8]

本章只需要确认：运行时决定可以延伸到输出形式。生成式界面的实现、评估、延迟与准确性边界，留到下一章讨论。

1.6 从工具到行动者

本书把 Agent 定义为能围绕目标连续使用工具、检查结果并继续行动的 AI 系统。模型参与判断下一步，工具负责读取信息或改变外部状态；模型只是这个系统的一部分。

生成内容已经改变了许多产品，但更深一层的变化，是软件开始代表用户连续行动。为了判断系统拥有多大的路径选择权，需要把响应式工具、生

成式功能、预定义工作流和 Agent 分开。外部影响很重要，却不是唯一边界：预定义工作流同样可以写入数据库，Agent
也可以先完成一段只读研究。

表 1-2 四种运行形态的路径与控制边界

运行形态	路径由谁决定	运行方式	工具使用	停止与交接
响应式工具	用户操作与预定义规则	一次操作触发一次明确响应	调用固定功能	动作完成后返回用户；异常按固定规则提示
生成式功能	产品规定调用范围，模型在该范围内生成	接收目标、素材和要求，返回一个或多个候选结果	可以不使用工具，也可以调用预先绑定的有限能力	生成完成后交给用户审查；通常不持续控制后续步骤
预定义工作流	代码预先规定步骤、顺序和分支	模型与工具可以参与多个节点，但路径由流程控制	在指定节点调用指定工具	按预设完成条件结束；异常进入固定分支或交给人
Agent	模型在系统边界内动态决定过程与工具使用	围绕目标循环获取反馈、调整计划并继续行动	根据任务状态在授权范围内选择工具	根据完成判断、失败阈值或检查点停止，并可移交用户或人工

表中的分类用于分析运行结构，不是产品等级。一个系统在运行时选择内容或界面，并不会自动成为
Agent；关键在于模型是否持续控制多步工作流和工具使用。

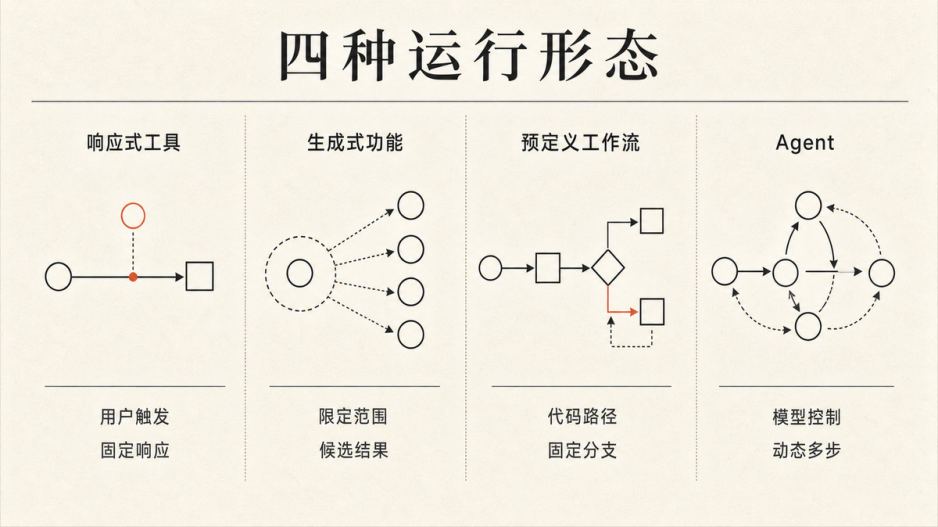

图 1-5 响应式工具、生成式功能、预定义工作流与 Agent

1.6.1 “给我一个答案”和“替我完成任务”不同

用户问“怎样报销这次出差”，系统可以返回政策与操作说明；用户说“帮我完成这次报销”，系统则可能读取票据、匹配行程、判断类型、填写申请并调用报销工具。

前一种体验会产生可审查的候选内容。设计师需要让用户核对来源、发现遗漏、比较差异并编辑结果。候选内容也可能误导判断，但在被采用之前，用户通常还有检查机会。

后一种体验还会产生外部动作的副作用。系统一旦写入报销平台、发送消息、下单或修改权限，影响可能立刻扩散到其他人和系统。设计师因此还要处理目标对象、授权范围、行动预览、重复执行保护、结果回执，以及撤销或补偿。内容风险与行动风险都要审查，但两者的发生位置和恢复方式不同。

1.6.2 自动化与直接操控的关系是一个长期问题

软件代表用户行动并不是全新的想法。1999 年，Eric Horvitz 在混合主动式界面研究中，把系统是否提供自动服务写成一个决策问题：系统根据证据估计用户具有某个目标的概率，再比较行动与不行动的结果效用。论文随后把“询问用户”加入选择，用两个阈值划分不行动、询问和行动。[2]

这套模型并不要求系统显式计算期望值；阈值也可以由设计者或用户设定。本书只借用它的决策结构，说明主动行动必须同时考虑目标判断与行动后果。它研究的是当时的混合主动界面，不能直接证明今天的 LLM Agent 应采用同一种内部实现。

1.6.3 Agent 把路径选择推迟到任务执行中

Anthropic 在 2024 年的工程指南中区分了工作流与
Agent：工作流通过预先定义的代码路径组织模型和工具；Agent 则是由模型动态决定过程和工具使用的系统。前者更适合步骤清楚、需要一致性的任务，后者更适合路径难以提前确定、需要灵活判断的任务。[9]

这一区别正好对应本章讨论的两套基础假设。

在固定工作流中，团队设计好 A 之后进入 B，满足某个条件再进入 C。模型可以参与某一步，例如提取票据内容，但整条路径仍然由代码控制。

在 Agent 中，团队提供目标、工具和规则，系统在运行时决定先查什么、后做什么，遇到缺失信息时是否询问，工具失败后是否换一种方法。路径不再只是界面的跳转关系，也成为模型行为的一部分。

OpenAI 的 Agent 实践指南也把 Agent 描述为代表用户独立完成任务的系统：模型管理工作流执行，工具用来获取上下文和采取动作，系统在失败时应能停止或把控制权交还给用户。该指南还指出，Agent 更适合复杂判断、难以维护的规则和大量非结构化数据；如果确定性方案已经足够，就没有必要增加 Agent。[10]

两份资料都属于厂商工程分类，不是统一的学术标准。本书采用它们的共同边界：Agent 是包含模型、工具、指令、状态和控制机制的系统；模型是否控制多步工作流，是区分 Agent
与普通生成式功能、预定义工作流的关键问题。

1.6.4 行动者是一种需要约束的产品角色

把 AI 称为“行动者”，很容易让人产生拟人化理解，好像系统拥有自己的目标与责任。

本章所说的行动者，只表示软件可以在授权范围内选择和执行一段行动。它的目标来自用户与产品，能力来自模型和工具，边界来自权限和规则，责任仍然需要由使用者、团队和组织明确承担。

从设计角度看，关键问题是系统的能动性有多大：能够读取什么，能够修改什么，能够影响谁，能够连续行动多久，以及走偏后是否还能被停止。它像不像人，不影响这些边界的设定。

一个只生成草稿的系统和一个可以直接发送邮件的系统，可能使用同一个模型，却是两种完全不同的产品。能力相似，不代表授权应该相同；结果看起来正确，也不代表过程没有越过边界。

当软件从工具变成行动者，界面也不再只负责提供操作入口，还要帮助用户理解目标、观察过程、判断关键节点并在需要时接管。后续章节会继续展开这些问题，本章先把它们归纳为六个新的设计入口。

1.7 新设计面对的六个基本问题

在确定性软件中，产品团队也会讨论用户目标、数据、权限和异常，但许多问题可以被固定流程吸收。用户亲自逐步操作，界面天然展示当前步骤，系统也很少在没有操作的情况下连续改变外部状态。

当模型开始参与解释与行动，这些问题不能再只留在后台。它们会直接决定用户是否理解系统、是否愿意委托，以及出错后能否恢复。

下面继续使用开篇的报告汇报案例。用户希望系统读取研究报告，为管理层制作一场 20
分钟汇报。这个看似清楚的委托，仍然需要依次回答六个问题。

1.7.1 系统如何理解用户意图？

用户说出的通常不是一份完整规格。

“帮我整理报告”可能指修改结构、统一语言、提炼摘要或制作汇报。开篇的要求补充了听众、时长、重点和来源规则，系统才有条件形成较具体的意图假设。它仍需判断哪些内容可以根据上下文推断，哪些内容必须追问。

设计师过去主要定义用户如何选择功能，现在还要定义系统如何发现歧义、怎样复述目标，以及用什么方式让用户低成本补齐对象、限制和完成标准。

1.7.2 系统依据了哪些上下文？

同一句请求放在不同上下文中，会产生不同结果。

选中的研究报告、当前页面、历史对话、用户偏好、组织规则和外部数据，都可能帮助系统理解任务。但“系统能够获得”不等于“用户已经授权使用”。制作这次汇报需要读取哪一版报告，是否可以引用访谈原话，是否允许结合其他项目材料，都要有清楚边界。

设计师需要考虑上下文从哪里来、何时生效、用户能否看见和删除，以及系统引用了错误或过期信息时怎样纠正。

1.7.3 系统准备采取什么行动？

系统可能先提取证据，再归纳问题、组织论点、生成图表和排版页面。中途发现数据冲突后，它还可能调整结构。产品不需要展示全部内部计算，但需要让用户理解当前目标、主要步骤、已经完成什么、即将生成或修改什么，以及计划变化是否超出原来的范围。

过程由此成为用户判断委托是否仍在轨道上的依据。

1.7.4 哪些事情可以自主完成？

在这场汇报中，提取标题、生成大纲和制作候选页面，可以先自动完成；删除被系统判断为“次要”的反例、改写敏感访谈原话，或把文件分享给管理层，则需要更严格的检查和授权。

设计师需要根据行动后果、可逆性、敏感程度和系统可靠性，决定哪些步骤可以自动执行，哪些只能提供建议，哪些必须先获得授权。自主性需要按行动分别设定上限，无法用整款产品的一个开关概括。

1.7.5 哪些事情必须交还给用户决定？

系统可能发现两组数据口径冲突，不确定哪项用户问题最值得管理层关注，或者无法判断一段原话能否对外展示。继续猜测会把事实缺失和价值判断隐藏在一份完整汇报里。

好的设计需要识别“必须问人”的节点，并向用户提供足够的判断材料。交还控制权不能只是弹出一句“是否继续”，还要说明为什么需要决定、有哪些选择、每种选择会产生什么影响。

1.7.6 系统走偏后如何打断、回退和追责？

如果系统一开始误解了听众，后续论点、数据密度和页面结构都可能随之走偏。候选汇报可以回到大纲检查点重新生成；已经共享给管理层的文件或已经发出的消息，则未必能够真正撤回。

因此，产品需要考虑怎样停止正在运行的任务，怎样保留检查点，哪些动作可以回退，哪些只能补偿，以及事后如何还原系统使用的上下文、工具、权限和人工决定。

恢复能力需要与系统能力同时定义，不能等设计完成后再补一个“撤销”按钮。

把这六个问题放在一起，可以看到设计基础假设的完整变化。

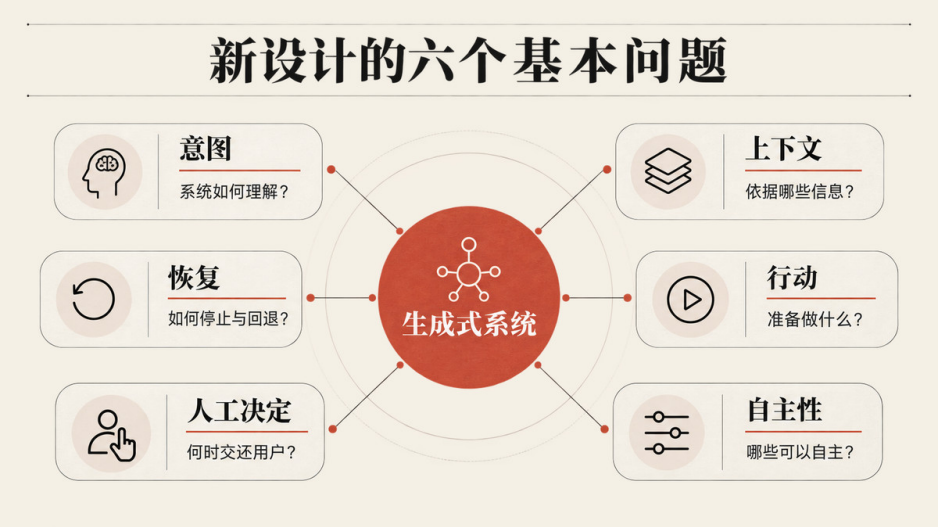

图 1-6 生成式系统面对的六个基本设计问题

表 1-3 确定性软件的常见假设与生成式系统的新问题

确定性软件中的常见假设	生成式系统带来的新问题
用户通过固定入口表达任务	系统需要从开放表达中推断目标与限制
页面和字段提供当前所需信息	系统会组合显式输入与隐式上下文
流程由团队提前定义	系统可能在运行时规划并调整步骤
每次外部动作由用户直接触发	系统可能在一次授权后连续行动
异常由固定规则拦截或交给人工	系统需要判断何时停止、追问和交还控制
错误通常对应一个页面或操作	错误可能沿行动链传播，需要打断、回退与追溯

这些问题为后续设计工作提供了入口，也会继续改变设计对象、用户角色、设计方法、交付物和评价标准。把它们整理成界面清单，并不能替代对系统行为的判断。

本章小结

过去几十年的数字产品，大多建立在一套相对稳定的假设上。

第一，人的目标需要被翻译成机器能够执行的动作。设计师把常见目标做成功能、控件与流程，用户再学习这套语言，把自己的具体目标拆成操作。

第二，固定界面是一套预先规定的表达语言。它用可见对象、结构化输入和即时反馈降低了精确语法的负担，也决定了用户可以向系统表达什么。

第三，软件的确定性主要来自提前定义。产品团队列出主要状态、规则、分支和可预见异常，把开放现实压缩成可实现、可测试和可维护的可能性空间；这不等于穷举现实。

第四，固定框架同时带来价值与限制。它稳定、清楚、可预测，却难以覆盖不断组合的长尾目标，复杂任务仍然需要用户理解工具并自行设计步骤。

第五，生成式系统把一部分翻译与路径选择推迟到了运行时。用户可以先表达目标，系统再结合对象和上下文生成内容、结构、界面与步骤。自然语言并不会取代所有控件，新的体验更可能是语言、直接操控与结构化约束的组合。

第六，当模型在边界内持续控制多步工作流和工具使用，系统才进入本书所说的 Agent 形态。设计因此必须回答意图、上下文、计划、自主性、人工决定和错误恢复等新问题。

AI 没有让传统设计失效。按钮、页面、规则、流程和直接操控仍然是稳定体验的重要基础。产品现在可以在固定结构之外表达一部分运行时能力。设计师需要同时处理两种空间：一部分由团队提前确定，另一部分由系统在运行时生成。

下一章将把这些运行时部分作为设计对象，继续讨论意图、上下文、计算、权限、行为和结果怎样与界面共同构成体验。

注释

1 Don Norman, The Design of Everyday Things, Revised and Expanded Edition, 2013/2014，https://jnd.org/books/the-design-of-everyday-things-revised-and-expanded-edition/ 。

2 “意图假设”是本书的设计分析概念。Eric Horvitz, “Principles of Mixed-Initiative User Interfaces,” CHI 1999，https://doi.org/10.1145/302979.303030 。论文以可观察证据推断用户目标概率，支持“系统对用户目标的判断具有不确定性”这一较窄主张；它没有把“意图假设”定义为通用术语或系统必须保存的独立字段。

3 Ben Shneiderman, “Direct Manipulation: A Step Beyond Programming Languages,” Computer, 16(8), 1983, pp. 57–69，https://doi.org/10.1109/MC.1983.1654471 。

4 Saleema Amershi et al., “Guidelines for Human-AI Interaction,” CHI 2019，https://doi.org/10.1145/3290605.3300233 。

5 John Mitchell and Ben Shneiderman, “Dynamic versus static menus: an exploratory comparison,” SIGCHI Bulletin, 20(4), 1989, pp. 33–37，https://www.cs.umd.edu/~ben/papers/Mitchell1989Dynamic.pdf 。论文最初招募 73 名大学生，有效样本为 63；76% 为大学一、二年级学生，多数几乎没有计算机经验。研究对象是键盘菜单，结果不宜外推到现代生成式界面。

6 Ian Drosos et al., “Dynamic Prompt Middleware: Contextual Prompt Refinement Controls for Comprehension Tasks,” CHIWORK 2025, 2025 年 6 月，https://doi.org/10.1145/3729176.3729203 ；Microsoft Research 项目页：https://www.microsoft.com/en-us/research/publication/dynamic-prompt-middleware-contextual-prompt-refinement-controls-for-comprehension-tasks/ 。研究包含 38 人形成性调查与 16 人受控实验；受控实验测试的是理解任务中的 Static PRC 与 Dynamic PRC 原型。

7 Google People + AI Guidebook, “Mental Models,” https://pair.withgoogle.com/guidebook-v2/chapter/mental-models/ 。

8 Google Research, “Generative UI: A rich, custom, visual interactive user experience for any prompt,” 2025，https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/ 。

9 Anthropic, “Building effective agents,” 2024 年 12 月 19 日，https://www.anthropic.com/engineering/building-effective-agents 。这里采用的是 Anthropic 当时的工程分类，不代表统一的学术定义。

10 OpenAI, “A practical guide to building agents,” https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/ 。官方网页与 PDF 未标明发布日期或版本号，访问于 2026 年 8 月 9 日；这里采用访问时的工程指南口径，不代表统一的学术定义。

第二章

02

设计对象变了——从界面到行为

第一章已经建立了两类决定空间：团队在发布前定义能力、数据、权限、风险和恢复底座，系统在运行时依据意图信号形成可修正的假设，并在边界内选择结果或路径。本章承接这个结论，继续回答一个更具体的问题：当一部分决定进入运行时，页面之外还有哪些内容需要被设计？

这里所说的“设计对象”，是团队必须描述、取舍、原型化并验证的内容。页面、组件和流程仍是设计对象，但已经不够。系统怎样解释用户表达，读取哪些上下文，如何组织计算和行动，受什么权限约束，怎样处理不确定性，以及产生什么结果，也会直接塑造体验。

为了让这些对象始终处于同一个业务情境，本章使用一个综合的差旅报销教学案例。用户把一组票据交给系统，并提出：“帮我完成这次出差报销，超出公司标准的项目先问我。”产品随后可能识别票据、匹配行程、查找政策、生成申请草稿，甚至写入报销系统。这个案例不是对某个现成产品的复述。Anthropic 的治理说明只提供了其中一个机制参照：假设系统发现酒店费用超标且缺少具体政策，它会询问用户是否允许读取公司共享盘中的费用政策，再根据获准取得的信息继续。这里扩大的是数据读取范围，不是任务目标。[1]

本书始终把 Agent
定义为能围绕目标连续使用工具、检查结果并继续行动的 AI 系统。模型与工具是系统组件，指令、任务状态、权限、停止条件和反馈共同约束其行为。Anthropic
所说的“模型动态决定过程与工具使用”用于区分预定义工作流和
Agent，是厂商工程分类，不是把 Agent 定义成单一模型。[2]

本章依次讨论五个问题：模型、工具、Agent 系统与产品体验怎样连接；意图、上下文、计算和结果怎样形成运行时循环；行为合同怎样约束系统；不同来源的不确定性怎样进入设计；哪些对象可以在运行时适配。

2.1 先拆清四个层次

同一个模型可以被装进完全不同的产品。它可以只识别票据，也可以读取公司政策；可以生成草稿，也可以直接提交申请。体验差异不只来自模型能力，还来自工具、状态、权限、控制机制和界面。

因此，讨论“AI 会怎么做”之前，需要先区分四个层次。

表 2-1 从组件到产品体验的四个层次

层次	与下一层的关系	报销场景中的例子	设计师需要确认的内容
模型层	作为系统组件，负责识别、推断、生成、提出候选步骤或判断下一步	从票据图片中提取商户、日期和金额	能力边界、典型错误、输入要求与评估条件
工具层	作为系统组件，提供读取数据与执行动作的接口	查询差旅订单、读取政策、写入报销草稿	读写范围、参数约束、回执、失败与重复执行保护
Agent 系统层	以模型和工具为组件，再加入指令、任务状态、权限和控制循环	整理票据，遇到政策冲突时暂停并调整下一步	完成与停止条件、权限闸门、检查点、回退与移交
产品体验层	把 Agent 系统的目标、状态、依据、动作和结果呈现给人	用户指定范围、查看异常、授权读取并核对回执	用户能否理解、修正、授权、接管和验证

这四层不是可相互替换的模块。模型和工具组成 Agent 系统，系统再通过产品体验对人可见。模型能识别金额，不说明工具已经获得写入权限；工具能够提交申请，不说明系统可以自行决定何时提交；系统完成了写入，

也不说明用户已经看见真实外部状态。把层次混在一起，会让“模型做得到”悄悄变成“产品允许做”，再让“产品允许做”被误写成“用户应承担后果”。

本章讨论系统能力如何被组织成受约束的产品行为。判断、执行与后果怎样在人、系统和组织之间分配，以及人的能动性怎样保留，留到第三章展开。

2.1.1 页面之外还有一条运行时链路

“输入—计算—输出”可以帮助设计师把注意力从页面扩展到中间计算。《Designing AI Interfaces》也把它作为全书的阅读结构，同时说明这三个阶段不是完整故事，还要处理配置、解释、行动、错误和反馈。[3] Agent 会保留任务状态、反复获取反馈、调用带权限的工具，并可能改变外部系统。只看三个方框，会漏掉权限检查、循环、失败和副作用。

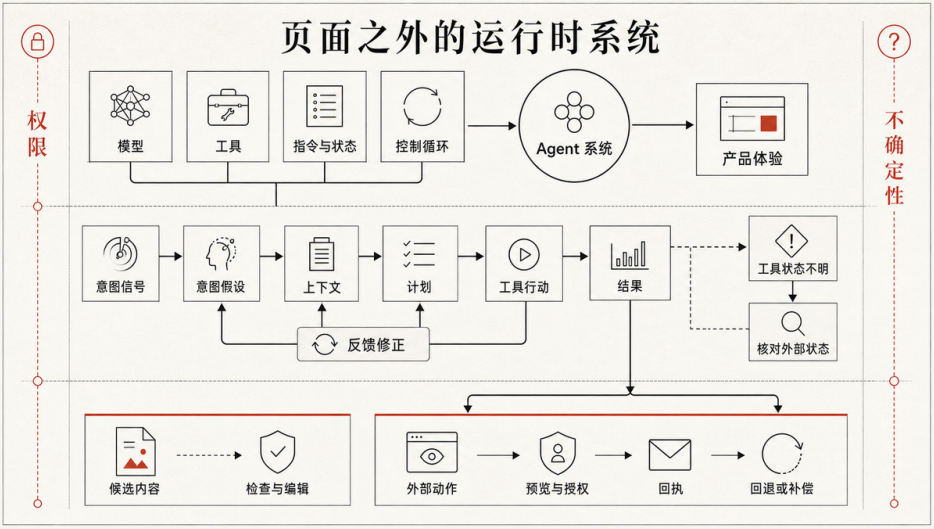

图 2-1 从系统构成到运行时循环：模型和工具是 Agent 系统的组件，权限与不确定性贯穿循环，结果分为候选内容与外部动作 / 状态

这条链路不是固定的瀑布流程。系统发现政策冲突后，可以回到上下文或计划；用户修改任务目标后，系统需要重建意图假设；工具返回状态不明

时，则要先核对外部系统，不能直接重试。设计师既要定义顺利向前的路径，也要定义折返、停止和恢复条件。

权限也不是链路末端的一次确认。读取票据、检索政策、写入草稿和正式提交分别需要检查；读取范围或任务范围变化时，原来的授权未必继续有效。不确定性同样会出现在每个环节，并沿后续行动传播。

2.1.2 界面仍然是设计对象，但角色变了

固定界面主要提供能力入口、结构化输入和局部反馈。运行时系统的界面还要承载四类工作：收集意图信号，呈现任务状态，提供核验证据，以及让用户授权、修正和接管。

报销产品仍需要票据列表、字段、异常标记和提交按钮。变化在于，这些界面不再只是让用户亲自完成每一步，还要让用户看见系统已经做了什么、依据什么，以及下一步会影响什么。

因此，界面没有从产品中退场。它从单纯的操作面板，扩展为人与系统共享任务状态的控制面。

2.2 意图、上下文、计算与结果

第一章已经把“意图假设”定义为本书的设计分析概念：系统依据可观察信号，对用户此刻目标形成的临时、可修正解释。它不是用户的真实意图，也不要求产品内部保存一个同名字段。

本章进一步关心这项假设怎样影响后续体验。设计师需要让意图信号、上下文、计算和结果彼此可追踪，避免一次早期误解沿着行动链不断放大。

2.2.1 意图信号需要共同确定目标、对象与边界

自然语言只是意图信号的一种。用户说“把这些费用整理一下”，系统仍然不知道“这些”指刚上传的图片、当前文件夹，还是整段行程；也不知道“整理”是生成清单、制作草稿，还是正式提交。

在报销场景中，三类信号可以相互补充：

1. 显式表达说明目标和例外，例如“生成报销申请，超标项目先问我”；
2. 直接操控明确对象和范围，例如用户选中本次出差的 12 张票据；
3. 上下文提供当前任务所需资料，例如行程订单、组织政策和已授予的偏好。

系统可以根据这些信号暂时形成意图假设，并用复述、预览或关键追问让用户修正。产品不应把“写一条完整提示词”的负担全部留给用户，也不能把沉默当作对高风险动作的授权。

2.2.2 上下文既是能力来源，也是权限边界

上下文是系统在当前任务中获准使用的信息及状态。它可能包含用户明确提供的票据、当前页面、任务历史、组织规则和外部数据。上下文越丰富，系统越可能减少重复询问；读取范围、版本冲突和隐私风险也会随之增加。

设计上下文至少要回答五个问题：来源是什么，为什么与当前任务相关，使用哪个版本，作用范围与有效期是什么，以及用户怎样查看、排除或更新它。

“系统可以获得”不等于“用户已经授权使用”。报销任务需要本次行程，不代表系统可以遍历所有私人日历；用户曾允许读取某个项目，也不代表这项授权可以自动延续到新的高风险任务。上下文发生变化时，系统还要判断已有意图假设和计划是否仍然成立。

2.2.3 计算要呈现任务状态，不公开内部思维

传统产品常用加载动画代表计算。报销 Agent 如果要连续执行识别、检索、分类、计算和写入，一个旋转图标无法告诉用户系统仍在工作、已经卡住，还是偏离了目标。

用户需要的是支持判断的任务状态：当前目标、主要步骤、已经完成的部分、正在使用的资料或工具、遇到的阻碍、下一次需要介入的位置，以及

范围是否发生变化。技术日志和模型内部推理不必直接展示；它们既可能增加负担，也未必提供可靠解释。

呈现粒度应随风险变化。只读、低风险、几秒完成的任务可以只显示简短进度；运行时间长、影响范围大或会改变外部状态的任务，需要更清楚的计划、检查点、差异和回执。

2.2.4 结果要区分候选内容与外部状态

模型生成一份报销草稿，和系统已经提交一份申请，是两种不同结果。

候选内容仍有检查和编辑机会。设计重点是来源、完整性、可修改性和版本差异。外部动作则会产生副作用：写入业务系统、发送消息或修改权限后，影响可能扩散到其他人。设计重点随之增加行动预览、重复执行保护、结果回执，以及回退或补偿。

“提交成功”也不能只来自模型的语言。产品需要以外部系统返回的申请编号、状态和时间为依据；如果工具超时，系统应先核对真实状态，再决定是否重试。否则，一次不确定的工具调用可能变成重复提交。

2.2.5 生成式 UI 是结果形式之一

当系统可以生成代码，界面本身也可能成为运行时结果。Google Research 在 2025 年 11 月 18 日公开的 Generative UI 项目中，模型在特定工具、提示、后处理和视觉配置下生成网页、游戏、工具和交互结构。随附论文当时仍是审稿中的预印本；公开说明还提到部分生成需要 1 至 2 分钟，并会偶发不准确。[4]

这项条件化实现展示了运行时生成界面的可行性，不能说明动态界面已经成为成熟行业共识，也不能说明所有任务都优于固定界面。它仍处于研究与实验阶段，公开结果只在相应模型、工具、提示、后处理、任务和评估条件下成立。

在产品验证中，可以先采用“固定骨架 + 受约束的动态局部”：账号、导航、权限、高风险操作和关键状态保持稳定，任务卡片、辅助输入、对比视

图和结果布局按当前任务组合。动态部分仍要遵守组件语义、品牌、可访问性、性能和失败回退规则。

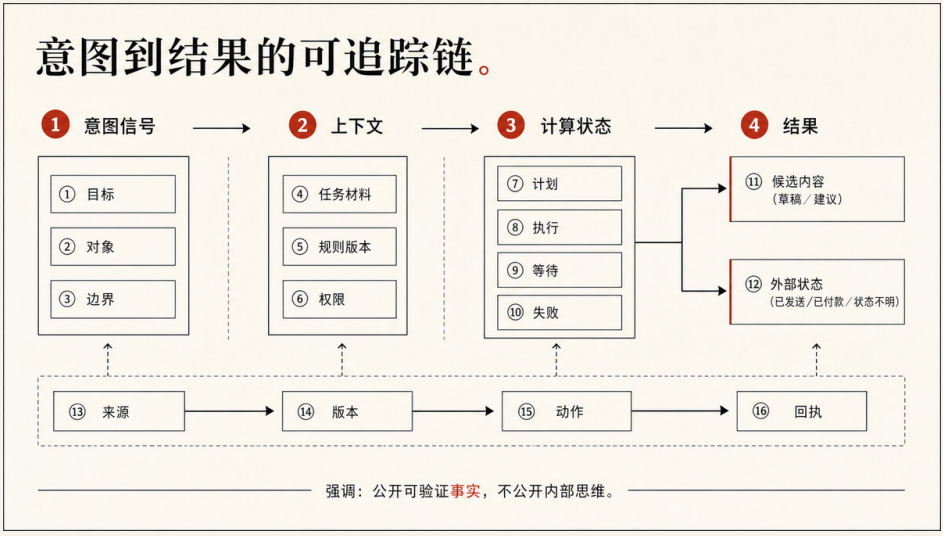

图 2-2 从意图、上下文与计算到结果的可追踪链

2.3 行为合同：系统能做什么，又受什么约束

当产品只生成内容时，用户主要检查结果。Agent 连续调用工具后，计划、行动顺序、停止方式和外部影响也进入体验。设计师需要描述的，不只是一张最终页面，还有系统在不同条件下应怎样行动。

本书把这种约定称为“行为合同”：团队对系统可观察行为作出的可测试约定。它覆盖允许范围、结果要求、停止条件和恢复方式，不要求产品中存在一份同名文件，也不是把法律责任写进提示词。实际实现可能分布在产品规则、权限系统、工具接口、运行状态、模型指令和界面控制中。

开头的“超出公司标准的项目先问我”只有映射到系统结构，才构成行为合同的一部分。产品可以把它写成权限规则：检测到超标项目后，系统进入“等待用户决定”状态；界面展示票据金额、政策版本、超标差额和拟采取的处理方式；提交工具在用户作出选择前不可调用。若缺少具体政策，系统

可以申请读取公司共享盘中的费用政策，授权只覆盖说明的目录与当前任务，不能自动扩展到其他文件或后续任务。

2.3.1 固定工作流与 Agent 需要不同合同

固定工作流由代码预先规定步骤和分支，模型可以参与票据识别或政策解释，但不持续决定整条路径。Agent 则由模型在系统边界内根据工具反馈选择下一步。动态路径适合步骤难以提前确定的开放任务，也会增加延迟、成本和失败组合。[2]

因此，报销的常规部分未必需要 Agent。票据字段校验、费用限额和审批路由如果已经有清楚规则，可以继续使用固定工作流；政策冲突、资料缺失和复杂例外才可能需要模型参与判断。选择哪种运行方式，本质上是在决定系统拥有多少路径选择权。

2.3.2 行为、权限与责任分别回答不同问题

行为回答系统在某种条件下准备做什么；权限回答它获准读取或改变什么；责任回答谁作出判断、承担职责并处理后果。

模型能识别和分类票据，是能力事实；工具可以提交申请，是技术能力；当前任务是否允许提交，是权限问题；提交前由谁决定、出错后由谁处理，则是责任安排。四者不能从前一项自动推出后一项。

本章可以定义报销 Agent 在政策冲突时暂停、在正式提交前预览，以及只在授权范围内读取资料。至于用户、产品团队和组织怎样分配判断、执行与后果，第三章会继续拆解。

2.3.3 一份行为合同至少覆盖七项内容

表 2-2 行为合同的七项设计内容

合同内容	需要回答的问题	报销场景中的例子
目标、质量与验收	什么结果算完成，质量和证据达到什么条件	选中的票据无遗漏，字段可追溯，冲突已标记，草稿通过必填校验；正式提交是另一项动作
上下文范围	可以读取什么，何时失效	只读取本次行程、指定票据和当前有效政策
工具与参数	可以调用哪些工具，参数怎样受限	可以写入草稿，不可自行更改收款账户
行动权限	哪些动作可自动执行，哪些需要授权	识别和分类可自动；正式提交前预览确认
停止与升级	何时暂停、追问、降级或交人	政策冲突、连续失败或范围扩大时停止
恢复与补偿	错误后怎样回到可用状态	从检查点重算；重复提交时进入人工处理
记录与回执	怎样重建与任务有关的事实	保存资料版本、工具调用、人工修改和申请编号

这七项用于约束可观察行为，不要求设计师预写模型的每一句话。开放部分可以变化，底线必须稳定：系统可以换一条检索路径，不能越过数据权限；可以提出不同分类，不能隐去冲突；可以调整计划，不能跳过正式提交前的授权。

2.3.4 合同要映射为状态和真实控制

行为合同需要映射成产品状态与真实控制。“等待授权”不能只是一句提示：系统必须停止创建受限动作，相关工具也必须拒绝未授权调用。用户暂停任务后，产品要核对正在进行的外部操作，再报告已完成、未开始和状态不明的部分。界面呈现目标、依据、差异和回执，是让合同可见；权限闸门、工具限制、检查点和停止条件，才让控制实际生效。

用户何时介入、以什么角色监督或接管，以及怎样承担判断与后果，留到第三章讨论。

图 2-3 行为合同的七项核心内容及其产品化落点

2.4 把不确定性设计成状态与选择

本章所说的“不确定性”，是系统对目标、信息、推断、执行状态或行动后果缺少充分把握。下面五类是本书用于定位问题的诊断视角，彼此可能重叠，也不构成完备分类。它们不能被压缩成一个统一的模型分数。

2.4.1 不同来源需要不同处理

表 2-3 五类不确定性及其设计动作

不确定性来源	报销场景中的表现	适合的设计动作
意图	“整理”可能指生成清单或正式提交	明确对象、复述目标、追问关键差异
信息	政策缺失、过期或相互冲突	显示来源、版本、缺口和冲突
模型	票据分类或摘要可能变化	提供原始材料对照、候选、编辑与复核
工具与环境	接口失败，提交状态不明	展示执行状态、核对回执、限制重试并防止重复
后果认知	系统或用户无法充分判断动作会影响谁、影响多大	预览变化、说明影响范围、补充评估或暂停动作

五类不确定性还会相互传播。假设一张酒店票据的金额边缘模糊，系统首先面对信息不确定性；模型仍然给出金额和费用类型时，又产生模型判断的不确定性；如果这个金额决定费用是否超标，误读便会继续影响用户对提交后果的判断。设计师需要标出传播链上的来源和受影响对象，不能只在最终结果旁显示“置信度 82%”。

不确定性与风险也要分开。风险描述某个结果可能造成的损失及影响；不确定性描述系统或人对当前情况缺少多少把握。一笔金额和收款方都已明确的大额转账仍然可能是高风险动作；一张金额模糊但只用于个人草稿的票据，也可能不确定却影响很小。实际风险取决于场景、后果、影响范围、可逆性和恢复条件，不能由不确定性高低直接推出。

能力边界、错误修正、服务范围和适配控制都应成为可见状态和可执行动作，而不能只写进免责声明。Amershi 等人的人机 AI
交互指南中，G1、G2、G9、G10、G14 和 G17 分别涉及说明能力与表现、支持纠正、在不确定时缩小服务、解释原因和提供全局控制，可作为检查项。[5]

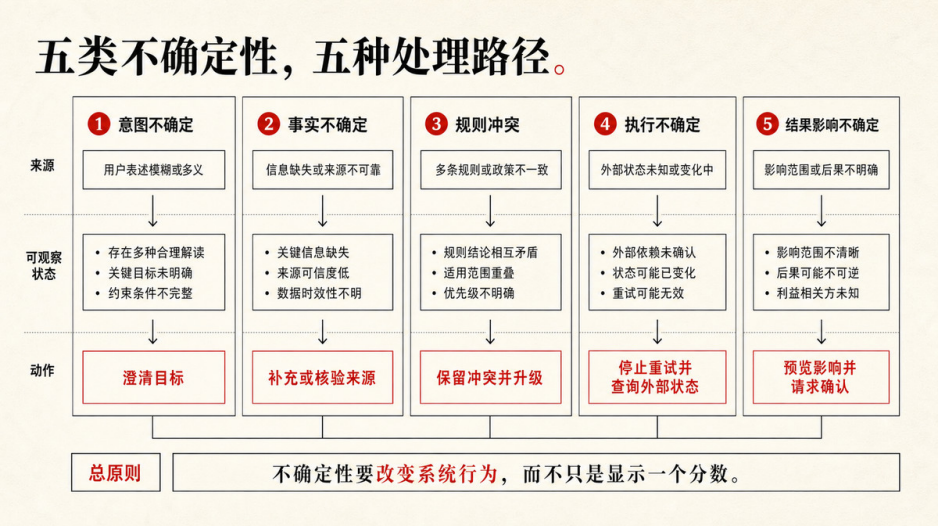

图 2-4 不确定性来源与处理动作的分流关系

2.4.2 错误分布比总准确率更接近现实后果

即使只看系统对“是否超标”的判断，也不能只看总准确率。团队要把真实情况与系统标记交叉检查：

1. 费用真实超标，系统也标为超标，说明这笔异常被发现；
2. 费用没有超标，系统却标为超标，会增加用户或财务人员的复核；
3. 费用真实超标，系统却没有标记，可能造成违规提交或直接损失；
4. 费用没有超标，系统也没有标记，这笔费用可以按常规流程继续。

第二种和第三种都属于错误，现实代价却不同。成本敏感学习研究说明了分类错误可以具有不对称代价这一一般问题，但不会替报销产品给出自动阈值。[6] 团队仍要结合政策、影响对象和恢复条件，决定哪些项目自动通过、哪些需要复核，以及哪些必须停止。

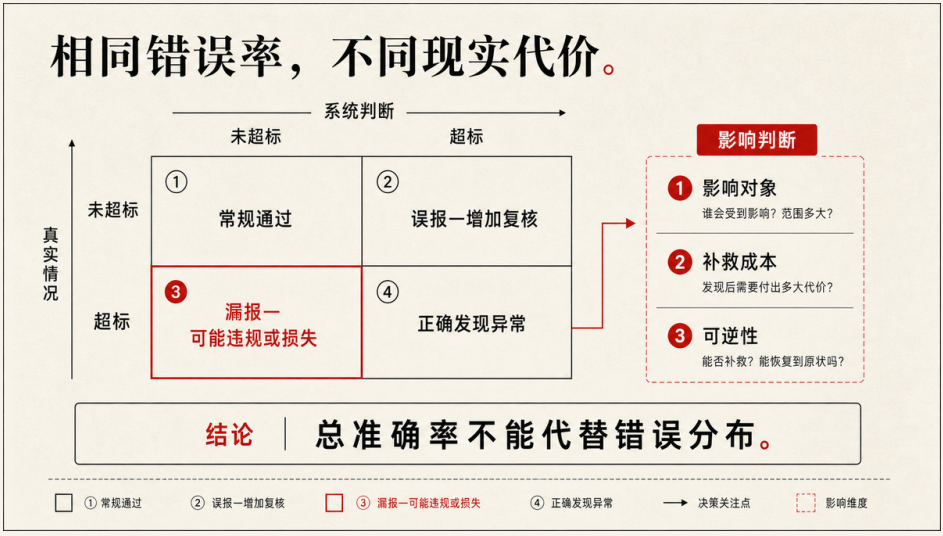

图 2-5 总准确率相同，错误分布与影响可能完全不同

2.4.3 解释要帮助判断

解释不是越多越好。公开全部提示词、内部推理和技术日志，可能增加负担，也可能让一段流畅叙述制造过度信任。

与当前决定直接相关的信息更有价值：系统用了哪一版政策，哪些字段来自票据，哪些内容经过推断，资料之间是否冲突，下一步会改变什么，以及用户还有哪些选择。低风险任务可以默认折叠细节，高风险节点再展开证据、差异和影响。

解释的目标是降低核验成本，不是说服用户相信系统。

2.4.4 失败要进入原型

如果原型只演示模型表现最好的一次，团队测试到的只是一个样片。团队可以在完整系统建成前枚举人机交互失败，并用低成本方式模拟，以便提前设计恢复路径；Hong 等人的 HAX Playbook 研究提供了这一方法的实践框架。[7] 这类模拟用于发现交互失败与改进恢复设计，不能替代安全、可靠性或生产环境验证。

报销原型至少要覆盖四种状态：顺利生成草稿，缺少关键信息，工具执行失败，以及需要人工接管。团队应观察用户能否发现问题、理解原因、修正受影响部分并继续，不能只记录任务最终有没有完成。

2.4.5 变化发生在稳定承诺之内

Agent 可以选择不同检索路径，不能访问未授权资料；可以生成不同布局，不能移动或隐藏高风险控制；可以提出不同方案，不能抹去不利证据；可以适配用户偏好，不能擅自提高权限。

不确定性成为设计对象，意味着团队要明确哪些内容允许变化、变化范围有多大，以及越过边界后怎样停止和恢复。它不意味着追求随机和惊喜。

2.5 运行时适配的对象与稳定边界

第一章说明一部分决定可以在运行时发生，但“运行时决定”不等于“为每个人决定”。讨论适配时，需要把三条互不替代的轴分开：

1 决定时点：内容、界面或路径是在发布前确定，还是在运行时确定；

2 适配依据：系统依据通用任务条件、当前情境，还是与当前用户有关且获准使用的信号；

3 改变对象：系统改变内容、界面组织，还是能力与任务路径。

三条轴彼此正交。运行时适配可以只依据票据数量和设备尺寸，不涉及用户差异；情境适配可以让所有处于同一情境的人获得相同结果，也不一定是个体化。个体化体验至少要使用与当前用户有关且获准使用的信号，例如用户明确设置的信息密度偏好或组织角色。[8]

2.5.1 内容、界面组织与能力路径

表 2-4 运行时可适配的对象及其稳定边界

改变对象	报销场景中的运行时适配	应保持稳定
内容	按当前核对任务显示对应政策条款，或生成一份异常摘要	事实、来源、政策含义、缺口与关键限制
界面组织	少量异常使用卡片，批量异常使用可筛选表格	核心概念、对象名称、状态含义、权限入口和高风险控制
能力与路径	常规票据进入自动校验，政策冲突进入等待判断状态	工具范围、授权、停止条件、验收条件、回执与恢复机制

内容变化会影响用户看见什么，界面组织会影响用户怎样理解和操作，能力与路径变化则会影响系统实际做什么。越接近权限与外部动作，通常越需要严格的治理；这不是固定的风险等级。实际风险仍由场景、后果、影响范围、可逆性和恢复条件共同决定。

2.5.2 个体化不需要永久画像

个体化可以依据当前任务中的用户信号，不必建立无边界的长期画像。报销产品可以根据用户获准使用的组织角色，向普通员工显示对应政策解释，向财务人员显示核验字段；也可以使用用户主动设置的信息密度偏好。产品要说明信号来自哪里、为何与任务相关、何时失效，以及怎样关闭或恢复默认设置。

如果产品只是根据“异常项目超过 20 项”切换为表格，这属于情境适配；如果它进一步依据当前用户获准使用的角色和明确偏好调整字段与解释，才包含个体化。无论使用哪种依据，适配都不能暗中扩大数据访问、工具能力或提交权限。

2.5.3 先固定边界，再开放适配

账号、隐私、权限与审计入口需要保持稳定；高风险动作的确认、停止方式和结果回执不能随布局变化而隐藏；对象名称、状态含义、错误反馈和恢复方式也要保持一致。品牌与可访问性要求同样适用于运行时结果。

固定界面已经能清楚解决问题时，没有必要动态生成；用户需要形成熟练操作时，也不应反复重排关键位置。运行时生成可以减少逐个制作候选变体的工作，却不会自动完成业务规则、可访问性、性能和安全验证。设计系统需要为动态组合提供允许使用的组件、语义和约束，并准备生成失败时的稳定回退。

这一节的落点是明确哪些对象可以在运行时适配、适配依据是否合法，以及哪些边界始终不变。“每个人都看到不同界面”不是运行时适配的目标。偏好学习、长期反馈和系统随时间演化的问题，留到后文讨论。

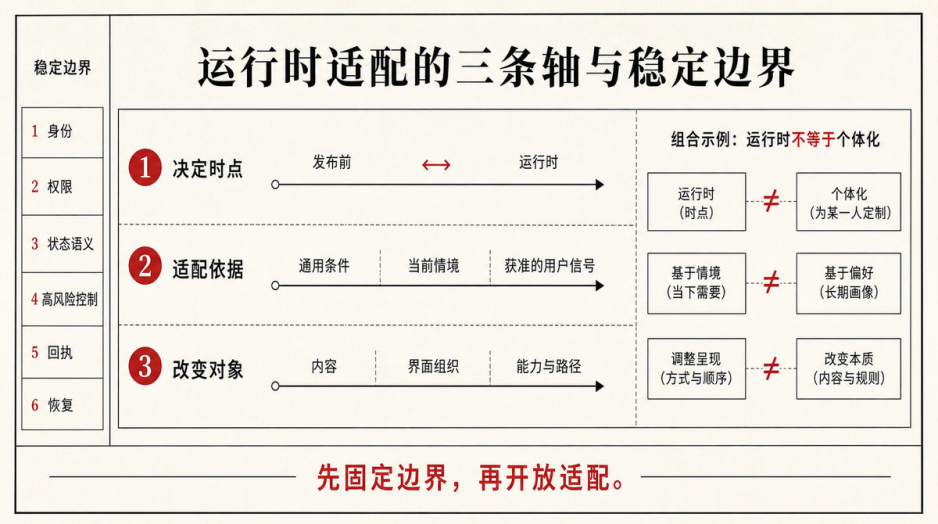

图 2-6 运行时适配的三个对象与不应越过的边界

本章小结

当一部分产品决定进入运行时，设计对象从页面与流程扩展到一组相互连接的系统对象。

第一，模型层、工具层、Agent
系统层和产品体验层需要分开描述。模型只是 Agent 系统的组件；工具提供外部能力；状态、权限、停止与反馈机制把它们组织成系统；界面让用户理解和控制这套系统。

第二，意图信号、意图假设、上下文、任务状态、计算、工具行动、结果和反馈共同组成运行时链路。“输入—计算—输出”可以作为起点，不能掩盖循环、权限、失败与外部副作用。

第三，系统行为与约束需要形成可检查的行为合同。团队要定义目标、上下文、工具、权限、停止、恢复和记录，但不能把系统能力、行动授权与责任承担混为一谈。

第四，意图、信息、模型、工具环境和后果认知提供了五个非互斥的诊断视角。不同来源需要不同的状态、证据、追问、限制和恢复机制；不确定性与风险也不能相互替代。

第五，运行时适配要分清决定时点、适配依据与改变对象。内容、界面组织和能力路径都可以变化，但稳定概念、高风险控制、权限、回执和恢复机制不能随适配漂移。

本章说明了系统可以做什么、怎样做以及受什么约束，还没有回答谁应作出判断、谁执行、谁处理外部后果。下一章将沿用差旅报销场景，讨论委托之后用户角色怎样变化，判断、执行与后果如何分配，以及产品怎样保留人的理解、拒绝、接管与停止能力。

注释

1 Anthropic, “Trustworthy agents in practice,” 2026 年 4 月 9 日，https://www.anthropic.com/research/trustworthy-agents 。文中的差旅报销情境是假设案例，位于 Policy 产品治理说明中，不是效果研究；它只支持“信息不足时先询问用户是否允许读取公司共享盘中的费用政策”这一机制。

2 Anthropic, “Building effective agents,” 2024 年 12 月 19 日，https://www.anthropic.com/engineering/building-effective-agents 。该工程指南把 workflow 描述为预定义代码路径，把 agent 描述为由 LLM 动态决定过程与工具使用的系统，并讨论环境反馈、检查点、人类反馈与停止条件。这里采用其工程分类，不把 Agent 定义为单一模型，也不视为统一学术标准。

3 Louise Macfadyen, Designing AI Interfaces, First Edition, O’Reilly Media, 2026 年 3 月，First Release 2026 年 3 月 11 日，第 1 章 “Reading This Book: The Input-Computation-Output Structure”。作者明确说明三段结构不是完整故事，并继续处理配置、解释、行动、错误和反馈。

4 Google Research, “Generative UI: A rich, custom, visual interactive user experience for any prompt,” 2025 年 11 月 18 日，https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/ 。官方说明所述实现依赖特定模型、工具、提示、后处理与视觉配置；随附论文当时为审稿中的预印本。部分生成需要 1 至 2 分钟，并会偶发不准确，因此本章只把它作为研究与实验阶段的条件化可行性案例。

5 Saleema Amershi et al., “Guidelines for Human-AI Interaction,” CHI 2019，https://doi.org/10.1145/3290605.3300233 。本章对应 G1、G2、G9、G10、G14 与 G17，不把指南当作具体产品效果证据。

6 Charles Elkan, “The Foundations of Cost-Sensitive Learning,” IJCAI 2001, pp. 973–978，https://cseweb.ucsd.edu/~elkan/rescale.pdf 。论文支持分类错误代价可能不对称这一一般判断，不为报销场景提供具体阈值。

7 Matthew K. Hong, Adam Fourney, Derek DeBellis, and Saleema Amershi, “Planning for Natural Language Failures with the AI Playbook,” CHI 2021，https://doi.org/10.1145/3411764.3445735 。研究支持在自然语言 AI 产品部署前枚举理想与失败场景，并用低成本方式支持早期原型讨论；这类活动不能替代安全、可靠性与生产环境验证。

8 Josh Clark with Veronika Kindred, Sentient Design: Crafting Intelligent Interfaces with AI, Rosenfeld Media, 2026, Chapter 1, pp. 18–19。该处区分 personalized 与 individualized；本章只保留 “individualized” 的术语来源，三条适配轴、改变对象和稳定边界均为本书框架。

第三章

03

委托之后的分工、监督与接管

上一章已经定义了 Agent 系统的能力、行为合同、权限和不确定性，并把差旅报销作为贯穿案例。本章沿着行为合同继续追问人机关系：判断与执行怎样在人和系统之间分配，后果处理职责怎样落实到组织与具体角色，人还保留哪些权力。

操作者、协作者、顾问、审批者和观察者不构成一条从低级到高级的用户升级路线。它们描述人在不同任务阶段承担的角色；同一个人可以在一次报销中来回切换。监督横跨这些角色：人在任何角色中，都可能需要发现偏离、拒绝、暂停、停止或接管。

在传统报销软件中，员工通常亲自选择票据、填写字段并提交申请。系统可以识别票据或校验金额，但用户仍能沿操作步骤定位错误。使用报销 Agent 后，员工可能只提出：“整理这次杭州出差的票据，符合政策的生成申请，超标项目先问我。”系统随后匹配行程、读取政策、分类费用并写入草稿。Anthropic 的治理说明提供过一个相近的假设情境：系统缺少具体政策时，先询问用户是否允许读取公司共享盘中的费用政策，再根据获准取得的信息继续。这个例子只说明扩大数据读取范围前的授权机制，不能作为报销 Agent 的效果证据。[1]

操作减少后，原有工作不会消失，而会重新安排。组织需要明确谁能发起、判断、批准和补救，并给相应人员提供信息、时间、培训与权限。产品和工程团队只对其能够控制的机制负责，例如权限闸门、任务状态、证据、停止与恢复是否按约定实现；它们不能代替组织分派业务或法律责任。使用者的任务职责也需要由组织明确赋予，并配套履职资源。任何一次“确认”都不能自动抹去这些职责。

本章先统一几个概念。判断是依据事实、规则或价值取舍决定下一步；执行是把决定变成读取、写入、提交等系统动作；后果是动作已经造成或可能造成的外部状态与影响；后果处理职责是确认结果、处理例外、补救损失和回应争议的职责。委托是在明确目标、范围和权限后，把某些判断或执行交给 Agent；它不等于转让全部责任。监督是人能够理解任务状态和证据，并拥有拒绝、暂停、停止、接管或升级的权力。

执行者、批准者、受影响者和负责补救的人可以是不同主体。责任也要分层讨论：组织负责制度、岗位、资源和补救安排，并明确赋予使用者任务职责；产品和工程团队负责自己能够控制的产品与技术机制；使用者承担组织明确赋予、且具备履职条件的职责。具体法律责任仍取决于适用法律、合同与事实，不能从“谁点击了按钮”直接推出。

本章以人的能动性为主线，检查人在委托后是否仍能理解、选择、拒绝、接管和停止。后文依次讨论委托关系，Agent 的能力、行动范围与自主性，人的五种任务角色，校准依赖，以及走偏后的监督与恢复。

3.1 委托改变分工，不自动转移责任

“告诉我怎样报销”只委托了信息整理或建议；“替我生成报销申请”还委托了部分判断与执行；“替我提交申请”则允许系统改变外部业务状态。委托程度不同，所需的权限、证据和人工介入也不同。

3.1.1 操作减少后，工作仍需明确归属

传统软件把许多动作交给用户逐步完成：人设定目标、选择步骤、执行动作并沿流程检查。Agent 可以在约束内规划、调用工具和写入草稿，人则把注意力转向范围、例外、证据与外部影响。目标、执行、检查和后果处理没有消失，组织仍需把它们交给明确主体。

低风险、可撤销、容易核验的草稿，可以只让人看结果。资金、权限、对外承诺或他人权益受到影响时，监督者需要在仍能改变后果的节点获得信息和权力。这种分工还要求产品降低核验与接管成本。

3.1.2 委托需要目标、边界和完成标准

报销员工说“帮我整理这次出差”，系统仍需确定三件事：目标是生成清单、写入草稿还是正式提交；边界是哪些票据、行程、政策和工具属于当前任务；完成标准是字段齐全、异常已标记，还是外部系统已经返回申请编号。

这些内容不必全塞进一条长指令。系统可以根据低风险、可修正的信息形成临时假设，再用范围选择、行动预览或关键追问让用户修正。身份、资金、对外承诺、数据范围扩大和不可逆动作则不能依赖沉默或模型猜测。委托能否成立，取决于目标、边界和完成标准能否约束当前行动，与提示词长短无关。

3.1.3 判断、执行与后果处理职责需要分别安排

“人工”或“自动”无法说明一项任务怎样分工。报销 Agent 可以读取票据并提出费用类型，由员工判断；也可以按明确政策分类，由员工只处理冲突；还可以写入草稿，但把正式提交留给人。需要分别标出谁判断或批准、谁执行、谁受到影响，以及谁确认外部状态、处理例外并提供补救。

表 3-1 差旅报销各环节的判断、执行与后果处理职责

报销环节	判断或批准	执行	可能受影响者	后果处理职责
读取材料	员工确认任务范围，系统判断所需资料	Agent 查询获准的票据、行程和政策	员工及资料涉及的其他人员	组织处理不当访问；产品和工程团队提供范围控制与记录
费用分类	Agent 按规则建议，模糊项由员工或财务人员判断	系统写入草稿字段	员工、财务审核人员	组织指定人员处理分类争议；产品系统保留依据与人工修改
计算金额	Agent 计算，员工核对特殊费用	系统更新草稿总额	员工、组织财务部门	组织指定人员处理金额差异；产品系统支持重算与差异核验
正式提交	获得组织授权的角色批准	Agent 调用报销工具	员工、审批人和组织	组织指定岗位处理重复、失败或争议；产品系统提供回执与恢复机制

这些主体混在一起，容易把“系统能执行”写成“系统有权决定”，再把“用户点了同意”写成“用户承担全部责任”。点击至多记录某个角色作出了这次决定，不会抹去组织职责，也不会改变产品和工程团队对可控制机制的职责。

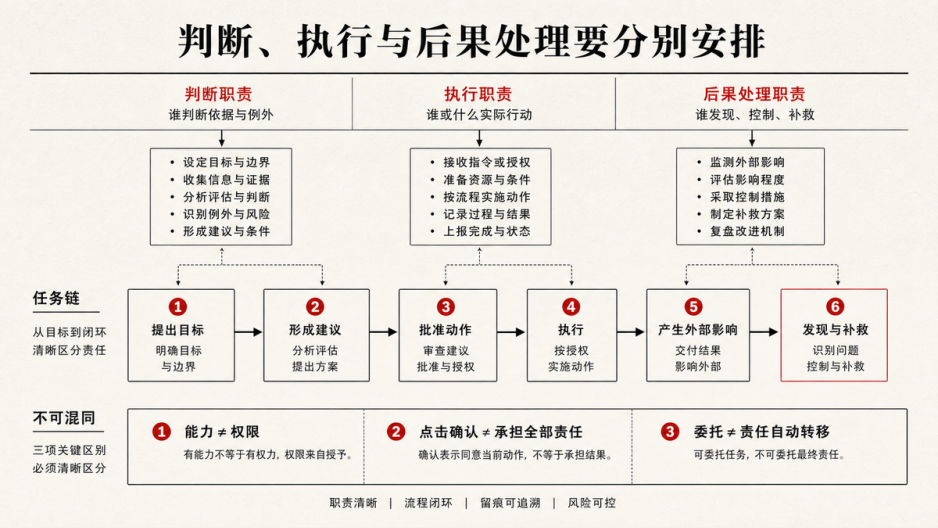

图 3-1 委托任务中的五类责任需要分别安排

3.1.4 确认按钮不能代替有效监督

在高影响动作前加入确认弹窗，只能证明流程要求一次输入，不能证明监督有效。

假设报销 Agent 准备提交 38
张票据，界面只显示“已整理完成，是否提交”。用户不知道哪些字段由 OCR 识别，哪几项与政策冲突，也看不到总额相对原始票据发生了什么变化。这时即使用户点了“确认”，也很难说明他真的审查了申请。

本书把以下四项作为有效审批的最低产品检查条件：审批者看得到相关证据，理解适用规则与影响，拥有足够判断时间，并具备真实的批准与驳回权。NIST AI RMF 把角色、责任、权限和监督流程纳入组织治理。[2] 这一框架支持组织配置监督条件，不能推出“确认后责任归用户”。组织需要明确赋予审批职责并提供资源，产品和工程团队则要让证据与控制机制按约定工作。

3.1.5 自动化会改变人的劳动

Agent 减少了逐项操作，却可能增加另外几种劳动：等待长任务完成，理解系统状态，处理升级，检查批量结果，纠正重复错误，以及在出事后重建行动经过。

例如，财务团队原来逐笔核对报销。引入 Agent 后，大部分票据可以自动整理，但系统不断发出“是否继续”的确认。如果每次确认都要重新打开票据、行程和政策，执行劳动只是变成了上下文重建和审批劳动。

这时设计师还要把用户为监督付出的注意力纳入计算：一天收到多少次升级；每次需要理解多少信息；不同升级是否能批量处理；哪些问题可以由系统规则直接阻断；人的一次判断能不能减少以后同类打扰。

监督成本持续上升，说明分工、权限或升级规则需要调整，不能只要求用户更认真。

《Designing AI Interfaces》在共享控制一节区分了持续盯守与有选择的人类介入，并列举主观权衡、系统无法处理的歧义和具有持续后果的动作。[3]

本书把这组主张作为候选检查项，仍需在具体报销流程中验证：一次确认是否改变行动范围、后果或后续路径；其余问题能否由权限、阈值、沙盒和自动校验处理。

3.1.6 委托边界随具体行动变化

同一条报销任务包含不同后果：票据去重只产生候选结果，政策读取扩大数据范围，写入草稿改变业务数据，正式提交进入审批流程。员工可以把去重和明确分类交给 Agent，同时保留政策例外与正式提交的决定。这里的差异来自具体行动的后果、影响范围、可逆性和可验证性，不来自一个覆盖整款产品的“自动化等级”。

委托也会随情境改变。经验丰富的财务人员可能允许系统自动归类常规费用，但保留跨币种、关联交易或政策冲突的判断；新员工可能希望更早看见依据。产品需要让人知道当前交出了哪些判断和执行，哪些权力仍由自己、组织或其他岗位持有。

3.2 Agent
自主性与人的能动性：本书的两根旋钮

“能力强”“行动范围广”“无需人介入”与“人仍能控制任务”是四个不同判断。本书按主体区分这些概念。

《Levels of Autonomy for AI Agents》v2 区分了 Agent 的 agency 与 autonomy。论文中的 agency
是形成意图并实施行动的广义能力，autonomy
是不依赖用户参与而行动的程度。论文还把用户角色从 L1 操作者排到 L5 观察者，对应 Agent 自主程度逐级增加；级别升高不表示产品更好。[4]

本书进一步作了操作化处理：能力回答 Agent
在给定条件下能否稳定完成某类工作；Agent 的能动性用其可调用的工具、可读写的数据、可影响的对象和影响范围表示；Agent 的自主性按某个具体判断或行动在多大程度上不经人介入来表示；人的能动性指人理解当

前任务、作出选择、拒绝建议、接管控制和停止系统的实际能力。Agent 能动性的这组操作化指标，以及“Agent 自主性 + 人的能动性”两根旋钮，都是本书的分析模型，不是论文原文定义。

表 3-2 能力、Agent 能动性、Agent 自主性与人的能动性

概念	主体与核心问题	报销场景中的例子
能力	Agent 做得是否准确、稳定	能否识别票据并正确匹配政策
Agent 的能动性	Agent 可以使用什么、影响什么	能否读行程、查政策、写草稿或正式提交
Agent 的自主性	某项判断或行动多久不需要人介入	自动去重，还是每次分类都询问
人的能动性	人能否理解、选择、拒绝、接管和停止	能否看见依据、驳回分类、暂停任务并接管

本节所说的“两根旋钮”，特指 Agent 的自主性与人的能动性。提高前者不要求降低后者：系统可以在边界内独立整理票据，同时让人随时看见状态、改变方向并安全停止。Agent 的行动范围是另一个需要单独约束的变量；扩大工具与数据范围会扩大影响半径，不会自动提高系统的自主性。

3.2.1 会做不等于获准自主执行

报销 Agent 即使能正确读取政策、填写字段并调用提交接口，也可以只获准生成草稿。相反，一个只能读取本次行程票据并去重的系统，行动范围很窄，却可以在这个范围内自动运行。

能力提升不会自动带来权限或自主性。团队需要根据具体行动的后果和控制条件分别决定。

如果团队混淆这些概念，就容易出现两种错误。第一种是模型效果提升后顺手开放更多工具，让能力增长自动变成权限扩张。第二种是为了降低风险不断增加确认，却没有收紧工具和数据边界，最终让用户承担一个本来可以由系统结构解决的安全问题。

3.2.2 按具体行动设置范围、自主性与介入

自主性应落到具体判断或行动，不能只给整款产品贴一个等级。Agent 的行动范围与自主性也要分别设置：行动范围广、自主性低可能造成审批负担；行动范围广、自主性高则会扩大错误的传播范围。

表 3-3 差旅报销具体行动的范围、自主性与人工介入

具体行动	Agent 的行动范围	一种可能的自主方式	人的介入与控制
识别和去重票据	只读选中图片，生成结构化字段	自动执行	查看低质量标记，修改或回退结果
匹配行程	读取获准的日历与订单	在授权范围内自动	冲突时补充信息或缩小范围
判断费用类型	读取当前政策，生成分类	规则明确、证据完整时自动	决定模糊项，驳回分类依据
写入报销草稿	修改业务系统草稿	展示差异后执行	编辑字段，暂停后续动作并回退草稿
正式提交申请	产生审批与财务记录	等待获授权角色批准	查看依据与影响，批准、驳回或停止

表中的安排只用于说明分层分工，不构成适用于所有组织的风险结论。团队仍需按本地政策、任务后果与监督条件决定。这样拆分后，用户不必在“全自动”和“每步都问”之间二选一。

3.2.3 自主性上限取决于后果与控制条件

一段顺利的 Demo 很容易让人觉得 Agent 已经可以独立完成整件事，但产品中的自主性上限应该由失败时会发生什么来决定。

出错后可以一键撤销、只影响当前用户、系统状态清楚、证据容易核验的动作，可以给予更多自主空间。结果不可逆、涉及资金和他人权益、会形成外部业务记录，或者监督者难以验证的动作，则需要更严格的边界。

OpenAI 的 Agent 构建指南建议根据读写性质、可逆性、账户权限和财务影响评估工具风险，并把高风险、不可逆动作或超过失败阈值的任务移交给人。[5] 这是厂商的工程建议，可以支持风险检查维度，不能替具体组织决定责任或自主上限。

决定一项行动的自主上限时，需要回答四个问题：

1. 错误会影响谁，影响范围有多大？
2. 监督者是否能在行动前理解证据与后果？
3. 行动后能否回退，不能回退时能否补偿？
4. 系统持续走偏时，是否有明确的暂停与停止条件？

只要其中一项答案不清楚，就不应该仅凭模型能力扩大自主范围。

3.2.4 提高 Agent 自主性不必削弱人的能动性

Agent 可以在获准范围内自主整理票据，同时让员工看见目标、范围、当前状态和已经发生的外部影响；员工也可以拒绝建议、修改方向、暂停新动作、停止任务并接管。这些能力共同构成人的能动性。

人的能动性体现为关键时刻的有效控制，不以持续盯守或按钮数量衡量。有效控制要出现在仍能改变后果的时间点，并以可理解的证据和真实权限为基础。若人必须检查每一个低风险步骤，Agent 的自主性没有减少监督负担；若人只能在不可逆结果发生后得知情况，控制又只是名义上的。

两根旋钮可以同时提高：系统在稳定边界内独立完成更多低风险行动，组织同时安排合适角色，产品提供证据与控制，使人能在关键节点理解、选择、拒绝、接管和停止。也可以收紧 Agent 的自主范围，并安排更专业的人工判断。两者之间没有天然的此消彼长关系。

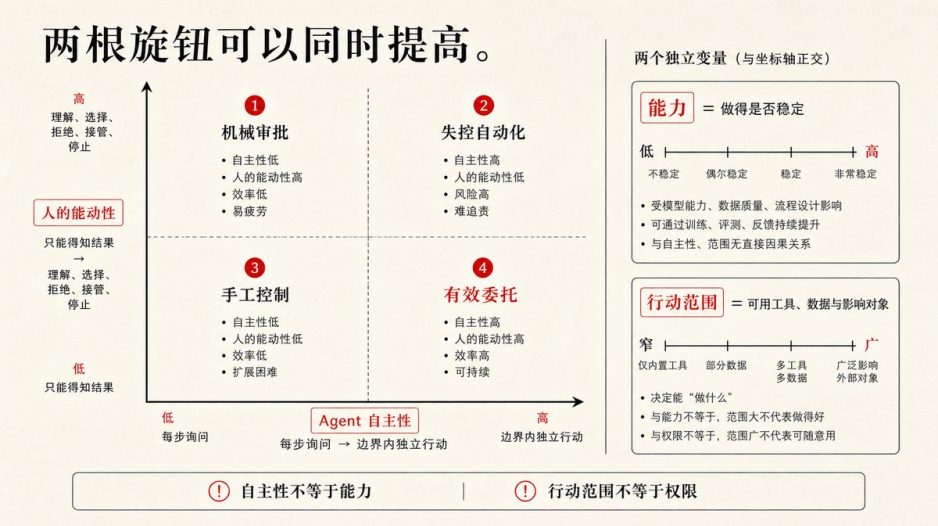

图 3-2 Agent 自主性与人的能动性不是同一条轴

3.3 人的五种角色：操作者、协作者、顾问、审批者、观察者

原论文把操作者、协作者、顾问、审批者和观察者列为 L1 至 L5，Agent 的自主程度依次增加。[4] 这套等级描述用户参与程度，不表示等级越高越好。本书有意重组这组名称：不把整款产品或整项任务固定在一个等级，而把它们用作可切换的任务阶段角色。这是本书的解释与应用，不是原论文的等级定义。

表 3-4 五种任务角色的人机分工与成立条件

人的角色	该阶段的人机分工	人主要做什么	角色成立的条件
操作者	人推进主要步骤，按需调用 Agent	选择对象、操作、编辑	可以直接控制步骤与结果
协作者	人与 Agent 共同计划、分工和修正	分配工作、共享中间判断	双方使用同一任务状态
顾问	Agent 主导执行，遇到自己不能决定的问题时向人请教	补充事实、偏好或专业判断	问题有上下文，回答会改变后续路径
审批者	Agent 准备行动，人决定是否放行	审查依据与影响，批准、驳回或改向	看得到证据、理解规则与影响、有判断时间，并有真实批准与驳回权
观察者	Agent 在狭窄边界内运行，人监测异常	查看状态、发现偏离、触发停止或升级	状态可读，告警及时，停止真实有效

3.3.1 操作者：人推进主要步骤

在操作者角色中，员工亲自选择本次行程、确认票据与费用类型，只在识别模糊金额或解释政策条款时调用 Agent。系统给出建议，不会自动扩大材料范围或提交申请。

这种分工适用于判断过程本身具有价值，或系统证据不足、后果较高的阶段。它不因人工参与较多而落后，也不保证天然安全；人仍需具备完成任务的时间、信息和能力。产品则要支持直接编辑、忽略建议和随时保持控制，不能把一次建议自动升级为行动。

3.3.2 协作者：人与 Agent 共享任务状态

在协作者角色中，Agent 可以整理票据并提出分类计划，员工增删材料、纠正费用类型，并决定哪些政策冲突交给财务人员。双方都可以接手部分步骤，但必须围绕同一份任务状态工作。

聊天可以用于讨论，不能成为唯一事实来源。系统需要记录计划、步骤归属、中间结果和人工修改；员工接手一项工作后，Agent 不能继续沿旧计划覆盖。双方能否看见同一目标、状态和变更，才是协作是否成立的判断标准。

3.3.3 顾问：人补充系统缺少的判断

顾问就是“Agent
遇到自己不能决定的问题时，被请来提供意见的人”。Agent 负责大部分执行，人补充只有相应角色掌握的事实、专业知识或价值取舍。例如，两版差旅政策对酒店上限的规定冲突时，财务人员说明本次行程适用的版本；一笔招待费缺少业务背景时，员工补充事实。

顾问与审批者的区别在于，顾问提供会改变后续路径的信息，审批者决定是否放行已经准备好的动作。咨询问题需要说明为何现在询问、回答会影响哪些步骤，并允许人给出预设选项之外的信息。规则明确且可由系统校验的问题，不应反复转交给人。

3.3.4 审批者：在阻塞或高影响动作前决定

报销 Agent 可以整理全部票据，但在酒店费用超标、政策冲突或正式提交前请求相应角色决定。审批者平时不参与过程，被叫回来时容易缺少上下文，因此需要看到触发原因、关键依据、行动预览、影响范围、可逆性和备选方案。

频繁确认可能增加上下文重建和机械点击，但不能据此断言用户必然失去注意力。审批疲劳是否发生、由什么造成，需要在具体产品和工作条件中观察。可以先收紧系统可触达的范围，再决定哪些节点确实需要人判断，避免用逐次批准代替权限控制。

审批机制还要区分“必须找人”和“系统不应尝试”。需要专业知识、组织授权或价值取舍的节点可以进入审批；已知越界或被禁止的动作应由权限和规则阻断，不能用一次点击消除组织职责或产品与工程团队对可控制机制的职责。

3.3.5 观察者：监测狭窄边界内的运行

在差旅报销中，观察者角色适合边界狭窄、后果可控的阶段，例如监测已提交申请的审批状态，或观察只读票据整理是否出现异常。人不逐步参与，但仍要知道系统是否偏离目标、错误是否扩散，并能触发停止或升级。

观察者没有更高的成熟度。几万行技术日志无法保证及时监督，事后查看也无法阻止已经发生的外部影响。这个角色依赖可读的状态汇总、异常检测、停止条件和恢复机制。Agent 的行动范围越广，越不能只依靠个人注意力维持监督；高风险、不可逆或监督者无法理解的阶段，不适合只安排观察者。

3.3.6 同一项任务可以切换角色

表 3-5 一次差旅报销中可能切换的用户角色

任务阶段	人的角色	分工依据
选择本次报销材料	操作者	员工最清楚任务范围
识别、去重和初步分类	观察者	只读处理，结果可修改
处理政策冲突	顾问	需要组织知识与专业判断
检查异常项目	协作者	人与 Agent 共同修正中间结果
正式提交申请	审批者	动作进入外部业务流程
追踪审批状态	观察者	系统可只读监测并报告异常

这张表只展示一种可能安排。角色切换时，产品要标明当前谁在判断和执行、Agent 为什么暂停、用户的操作会把控制权交给谁，以及切换后谁处理尚未完成或状态不明的部分。

3.3.7 监督是跨角色能力

监督不是第六种角色。操作者可能在手动推进时发现系统越界，顾问可能拒绝在证据不足时给出方向，审批者可能驳回行动，观察者也可能触发停止。监督贯穿五种角色，只是介入时机与所需信息不同。

流程图里出现一个人，不等于监督成立。组织需要明确赋予职责，提供足够时间、知识和升级路径；产品和工程团队需要让相应信息与控制真实可用。操作者需要直接编辑，协作者需要共享状态，顾问需要有上下文的问题，审批者需要决策证据，观察者需要异常与影响汇总。仅提供同一个聊天框，难以支撑这些分工。

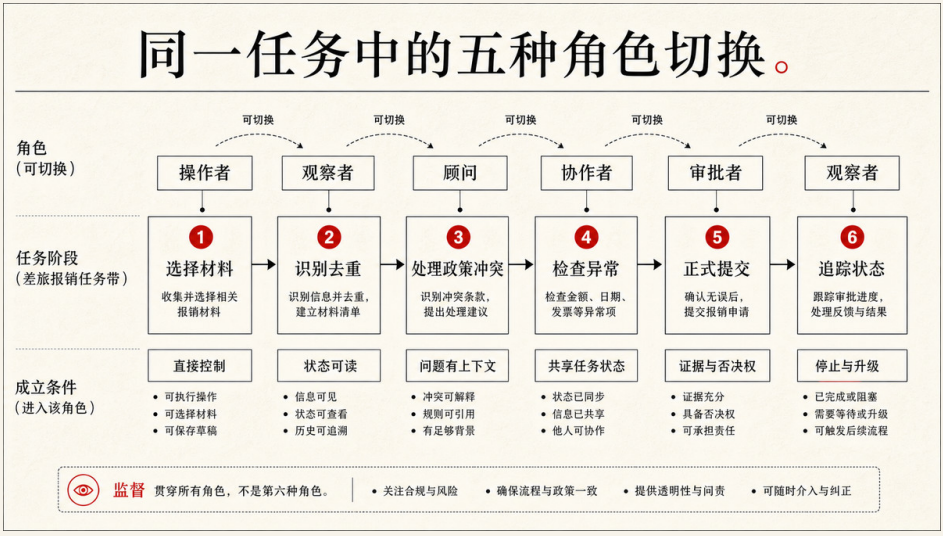

图 3-3 同一任务中的角色切换与监督要求

3.4 校准依赖：区分信任态度与依赖行为

Lee 与 See 在自动化信任综述中区分了信任与依赖：信任是人在不确定和脆弱条件下，对自动化能否帮助实现目标所持的态度；依赖是人在具体情境中是否采用、遵循或交给自动化执行的行为选择。[6] 一个用户可以总体信任某个系统，却在一笔高影响报销中不采用它的建议；也可能并不熟悉系统，却因工作流程没有替代路径而被迫依赖。

本书把校准依赖操作化为：人在当前任务中的依赖行为，与系统在相应条件下可验证的表现、行动边界和保护机制相匹配。这里讨论的是具体行动选择，不追求更多信任，也不把一次成功演示当作普遍能力。

3.4.1 依赖对象与条件必须具体

报销 Agent 可能准确识别金额，却没有正式提交权限；可能正确引用政策，却忽略例外条款；也可能完成提交，但缺少可靠回执。内容正确性、数据与权限边界、执行行为和恢复机制不能互相替代。

Microsoft 的人机 AI 交互指南建议在首次使用时说明系统能做什么、表现如何，并在出错时支持忽略、纠正、恢复和理解原因。[7] 这些指南是跨产品的候选检查项，不证明某种界面在所有情境中都能校准依赖。落到报销任务，产品需要说明支持哪些步骤、使用什么资料、哪些结果仍是草稿、哪些动作会进入外部系统，以及什么条件会触发人工介入。

3.4.2 证据要支持当前决定

流畅、肯定的语言不能证明结论正确。监督者需要能核验当前决定的事实：政策名称与版本、适用范围、票据金额、计算字段、缺失信息、冲突来源，以及执行前后改变了什么。

展示模型的完整内部推理不等于提供解释。与当前决定无关的长篇叙述和原始技术日志可能增加负担，甚至制造表面上的确定感。可用的解释应帮助人核对事实、规则与影响，以降低判断成本，不能以说服人接受结论为目标。

3.4.3 不确定性需要对应处理动作

未经校准或缺少任务语境的“72% 置信度”，不能告诉监督者下一步怎样做。产品需要说明不确定性来自缺失信息、来源冲突、模型判断还是工具状态，并给出相应选择。

表 3-6 不同任务条件下的依赖与介入方式

当前条件	一种可能的依赖方式	产品需要提供的支持
资料完整、规则明确、结果可修改	接受结果或抽样检查	关键依据、差异和回退入口
信息缺失且影响有限	补充、编辑或缩小任务	缺失项、影响范围和可选路径
来源冲突或涉及专业判断	交给相应角色判断	冲突并列、适用范围和备选方案
动作高影响、难以回退或影响他人	在执行前加强复核或审批	行动预览、证据、影响范围与驳回路径
超出能力范围或出现异常行为	暂停或停止依赖，接管或移交	明确告警、安全停止和交接信息

表中的关系不构成固定风险等级。检查强度还取决于具体场景、影响半径、恢复方式和监督者是否有能力验证。模糊的“结果仅供参考”没有把不确定性转成行动，也把核验成本留给了使用者。

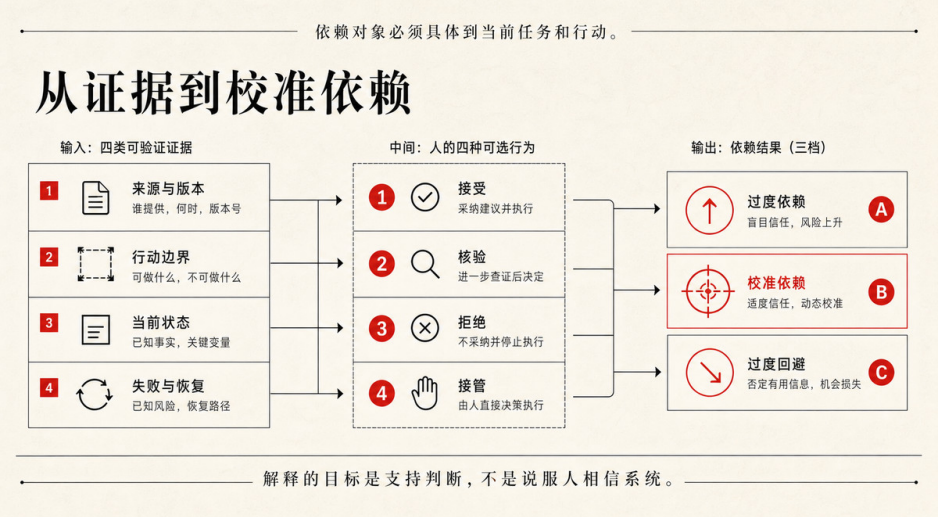

图 3-4 校准依赖需要把判断条件与处理动作连接起来

3.4.4 自动化偏差取决于任务与运行条件

自动化研究区分使用、误用与弃用，并提醒团队关注人怎样依赖自动化。[8] Parasuraman 与 Manzey 对自动化自满和自动化偏差的综述进一步指出：多任务负荷会挤压监测注意力；系统可靠性及其一致性会影响依赖；系统状态和反对证据是否显著，会影响人能否发现问题；频繁误报又可能促使人忽略或弃用自动化。一般性的培训和提醒不足以稳定消除这些问题。[9]

这些研究不说明用户遇到 AI 就会机械服从，也不能把错误归咎于“不够谨慎”。组织需要考虑工作负荷、角色配置、抽查和升级安排；产品与工程团队需要在其控制范围内呈现系统状态与反对证据。至于输出语气、核验难度和确认频率会不会在某个 Agent 产品中造成过度依赖，本章只把它们列为待验证假设，需要通过用户研究和实际运行证据判断。

3.4.5 用三个问题判断摩擦是否有价值

本书用三个问题判断是否值得在某个节点增加停顿：人此时还能改变后果吗；界面能否提供先前没有的证据或影响信息；作决定的人是否具备相应权限和能力？这三个问题是本书的产品检查框架，不是普遍实证结论。

正式提交报销前展示票据差异、政策冲突、总额与影响范围，可能帮助员工或财务人员重新进入任务；实际效果仍需验证。没有上下文的“请确认”或重复索取相同权限，则没有增加判断信息。可编辑草稿与正式提交的后果不同，摩擦强度也应分别安排。

3.4.6 把怀疑方法写进产品信息

面对重要决定，产品可以围绕四个问题组织信息：

1. 当前行动是否处于声明的能力与授权范围内？
2. 结论依据哪些事实，来源是否当前、完整且可核验？
3. 如果结果错误，会影响谁，能否回退或补偿？
4. 当前监督者是否拥有足够信息、专业能力与决定权限？

这四问不能把核验义务全部交给用户。组织应安排合适的监督者并提供履职资源，产品与工程团队应在其控制范围内汇总证据，让暂停、驳回和接管机制按约定生效。具备这些条件后，怀疑才能成为可执行的判断方法，避免沦为道德训诫。

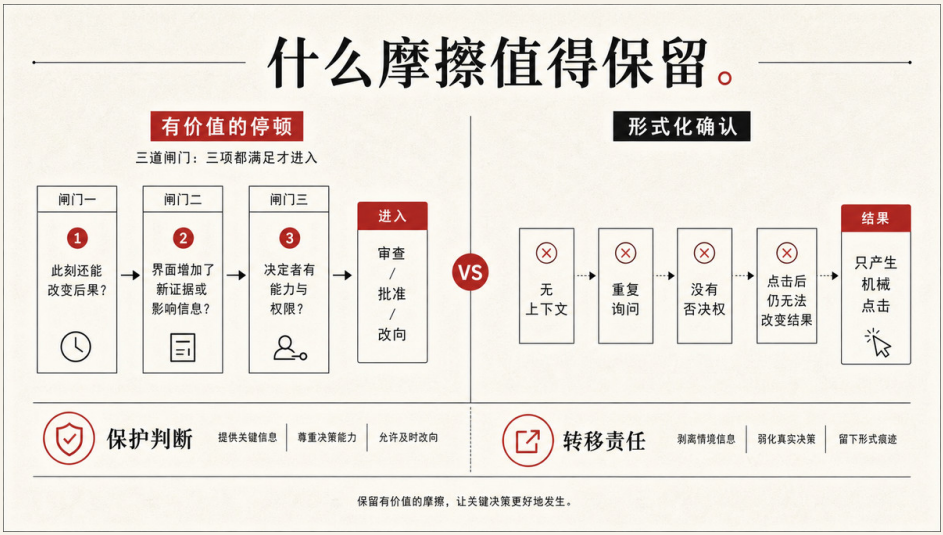

图 3-5 三个问题判断一处摩擦是否有价值

3.5 走偏后的监督、恢复与责任处理

Agent 走偏时可能没有报错：它仍在运行，却误解目标、引用旧政策、重复尝试工具，或逐步扩大了任务范围。恢复机制需要覆盖行动之前、过程中和之后，并区分几组容易混淆的概念。

3.5.1 可见状态与原始日志

监督者需要看到与当前任务有关的目标、范围、依据、已发生改变、下一步影响和异常。技术日志记录函数、参数和返回值，是追踪材料之一，却不等于用户能理解的任务状态，也不等于解释。

表 3-7 发现差旅报销任务偏离所需的监督信息

监督信息	监督者需要判断什么	可读状态示例
当前目标与范围	Agent 是否仍在处理本次行程	目标、票据范围和明确非目标
计划与进度	现在处于哪一阶段	已完成、处理中、未开始与状态不明
依据与假设	系统凭什么继续	政策版本、缺失信息与冲突
已发生改变	哪些对象已经修改	草稿差异、工具回执和影响对象
下一步影响	继续后会发生什么	行动预览、影响范围和可逆性
异常与重试	系统是否循环或换路	失败原因、重试次数和停止条件

信息可以按后果逐层展开：低风险的只读整理显示简短进度，写入、提交或状态不明时再展开证据和影响。监督者不应被迫在完全不看与阅读全部日志之间选择。

3.5.2 暂停与停止处理不同状态

暂停是停止创建新动作并保留可恢复的任务状态，之后可能继续。停止是终止本次运行，不再按原计划继续；必要时还要关闭未使用的临时权限或令牌。两者都不能保证正在进行的外部调用立刻取消。

用户发出暂停或停止请求后，系统需要区分已完成、处理中、未开始和状态不明的动作。对不可中断的提交请求，产品要等待回执或查询外部系统，确认真实状态后才能报告结果。界面动画消失不代表申请没有提交。

Anthropic 的 Agent 工程文章支持两项较窄的工程做法：长任务可以在检查点或阻塞处获取人工反馈，并可用最大迭代次数作为停止条件。[10] 本节对暂停与停止的区分、暂停后停止创建新动作，以及核对外部真实状态，都是本书从行为合同与外部副作用推导出的产品要求，不是该文章的原文结论；具体系统仍需单独验证能否实现。

3.5.3 纠正要标明影响层级

纠正一张票据金额，与更换政策版本或改变报销范围的影响不同。字段级纠正修改局部事实；步骤级纠正重做分类或更换资料；计划级纠正调整顺序与检查点；目标级纠正重新定义结果、范围和完成标准。

每次纠正都要标明哪些后续结果失效、哪些已完成动作可以保留、是否需要重新计算，以及从哪个检查点继续。Microsoft 指南关于纠正、细化与恢复的条目可作为通用检查项，不能替团队验证这些控制在当前任务中是否有效。[7]

3.5.4 接管需要移交

接管是人取得后续判断与执行的控制权；移交是系统把任务状态、已完成工作和未决问题交给接管者。只关闭 Agent
没有完成移交，只提供摘要却让 Agent 继续运行也没有完成接管。

报销任务的最小交接包应包括当前目标和范围、已核验事实、仍属推断的内容、使用的政策版本、已执行和未执行的动作、状态不明项、异常原因与可选路径。原始日志可以附在后面供调查，不能直接充当交接包。

产品还要标明控制权归属：Agent 是否仍在后台运行，人的修改会不会触发自动继续，接管是处理一个例外还是终止整项委托，以及恢复自动执行前是否需要重新确认目标、范围与权限。

3.5.5 回退与补偿不能互换

回退是把受控系统恢复到此前状态；补偿是在无法完全恢复时，以新的动作降低影响。草稿可以从版本恢复，已经进入审批流程的申请则可能需要撤回或更正；对方已经读取的数据无法通过撤销权限变成“从未看见”。

表 3-8 不同行动的回退、补偿与限制

报销行动	回退或补偿方式	仍需处理的限制
修改报销草稿	版本恢复、字段反向修改	下游计算可能需要重做
正式提交申请	撤回申请、提交更正	审批人可能已处理，时间记录仍存在
重复写入或支付	撤销重复记录、退款或人工处理	可能产生费用、时延与对账问题
不当读取或披露数据	撤销权限、轮换凭证、通知相关方	已披露数据无法“未被看见”

难以回退的行动需要把保护前移到执行之前，例如预览、限额、最小权限和明确授权。补偿机制不能用来证明不可逆动作可以放心自动执行。

3.5.6 追踪提供事实，追责需要组织判断

追踪是重建任务发生过什么；追责是组织依据事实、岗位、制度和适用规则确定责任、补救与改进。技术日志、版本、审批和回执可以支持追踪，却不会自动给出责任结论。

一条与责任相关的行动链应记录任务发起人、目标和范围，系统与规则版本，数据、工具和权限，关键判断与异常，人工批准、驳回或接管，外部动作的参数与回执，以及最后的补救处理。记录不要求公开模型的完整内部推理；它要保存可验证事实，并按不同角色提供可读视图。

NIST AI RMF
把角色、责任、监控、事件识别和反馈纳入生命周期治理。[2] 这一框架支持组织提前安排责任与事件处理，不代表只要保存日志就完成治理，也不替具体事件给出法律责任结论。

3.5.7 高风险系统的人类监督要求有适用范围

欧盟《人工智能法案》第 14 条针对该法定义的高风险 AI 系统，要求这类系统能够由自然人有效监督，并涉及理解能力与限制、监测异常、意识到自动化偏差、正确解释输出、决定不使用或推翻输出，以及介入或通过停止机制使系统进入安全状态。[11]

截至本项目统一访问日期 2026 年 8 月 9 日，Regulation (EU) 2026/1744 已推迟《人工智能法案》第三章第 1 至 3 节的适用时间：依据第 6(2) 条与附件 III 归类的高风险系统推迟至 2027 年 12 月 2 日，依据第 6(1) 条与附件 I 归类的系统推迟至 2028 年 8 月 2 日。[11] 因此，相关义务尚未到上述系统的统一适用日期，更不能据此直接宣布普通报销 Agent 的法律分类或责任归属。条文内容仍可转作一般产品检查问题：监督者能否发现异常、核验证据、拒绝或推翻结果、停止系统，并理解停止后的真实状态。

3.5.8 恢复能力保留人的能动性

假设报销 Agent 使用旧政策，将三笔酒店费用标为超标，并准备从草稿中排除两笔。系统应在发现政策版本冲突时暂停相关新动作，展示两版政策及受影响票据，让相应角色选择适用版本。若草稿已经修改，可以回退受影响字段；若申请已经提交，则需要撤回或更正，并确认外部状态。

接管者需要收到任务交接包，不能只得到一串日志；后续调查依靠可验证行动链，责任与补救由组织按事实和制度判断。人在这一过程中不必重新完成全部操作，但仍能理解、拒绝、纠正、接管和停止。这些权力使监督成立。

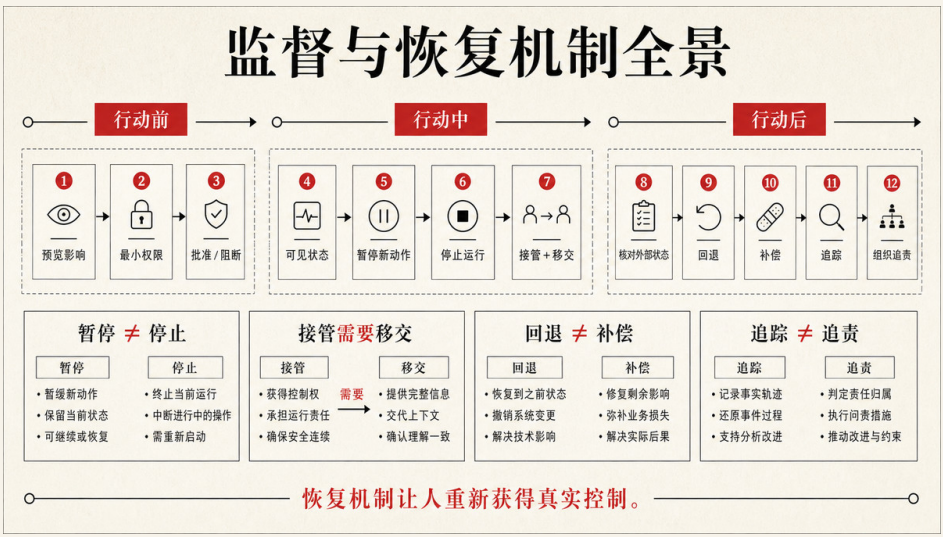

图 3-6 走偏后的监督、恢复与责任处理全景

本章小结

本章把委托后的人机关系归结为五点。

第一，委托重新安排判断与执行，也需要明确后果处理职责。执行者、批准者、受影响者和负责补救的人可以不同。组织负责分派职责并提供资源；产品和工程团队只对可控制机制负责；使用者承担组织明确赋予且具备履职条件的职责。

第二，LoA 原论文区分 Agent 的 agency 与 autonomy，并用 L1 至 L5 表示逐级增加的自主性。本书把 Agent 能动性操作化为行动范围，另以“Agent 自主性 + 人的能动性”分析人机关系。自主性按具体行动配置，提高 Agent 自主性不要求降低人理解、选择、拒绝、接管和停止的能力。

第三，本书把操作者、协作者、顾问、审批者和观察者重组为可切换的任务阶段角色，不把它们当作成熟度阶梯。监督是贯穿这些角色的能力，不是第六种角色。

第四，信任态度与依赖行为需要区分。本书用“校准依赖”描述具体任务中的依赖行为与系统表现、边界和保护机制相匹配。自动化偏差与弃用受到工作负荷、系统可靠性与一致性，以及状态和反对证据是否显著等条件影响；输出语气、核验难度和确认频率仍是具体产品中待验证的假设。

第五，监督要落实为真实权力。暂停与停止、接管与移交、回退与补偿、追踪与追责分别处理不同问题；技术日志既不是可读交接包，也不会自动给出解释或责任结论。

下一章将把本章建立的委托关系与人机分工转成设计方法，讨论怎样形成委托假设，构建可执行原型，并用测试与持续治理验证这些安排。具体流程与产物留到下一章展开。

注释

1 Anthropic, “Trustworthy agents in practice,” 2026 年 4 月 9 日，https://www.anthropic.com/research/trustworthy-agents 。文中的差旅报销是 Policy 产品治理说明中的假设案例，不是效果研究；本章只用它说明扩大数据读取范围前的授权机制。访问于 2026 年 8 月 9 日。

2 NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023，https://doi.org/10.6028/NIST.AI.100-1 。本章关于组织角色、职责、资源和事件处理的边界主要对应 GOVERN 2.1—2.3、GOVERN 3.2 与 MANAGE 4.1、4.3；这些条目不支持推断普通使用者的个案法律责任。

3 Louise Macfadyen, Designing AI Interfaces, First Edition, O’Reilly Media, 2026 年 3 月，章节 “Design for Shared Control”。本章只把共享控制与选择性人工介入作为候选检查项，不把书中观点或例子当作产品效果证据。

4 K. J. Kevin Feng, David W. McDonald, Amy X. Zhang, “Levels of Autonomy for AI Agents,” arXiv:2506.12469v2, 2025 年 7 月 28 日，https://arxiv.org/abs/2506.12469 。论文区分 agency 与 autonomy，并以 L1—L5 描述自主程度递增的五种用户角色；本书对 Agent 能动性的操作化、“两根旋钮”和任务阶段角色切换均为再解释。

5 OpenAI, “A practical guide to building agents,” https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/ 。官方页面未显示发布日期或版本号；本章只采用访问版本中的工具风险与人工移交建议，不视为统一标准。访问于 2026 年 8 月 9 日。

6 John D. Lee and Katrina A. See, “Trust in Automation: Designing for Appropriate Reliance,” Human Factors, 46(1), 2004, pp. 50–80，https://doi.org/10.1518/hfes.46.1.50_30392 。论文区分信任态度与依赖行为；本章的“校准依赖”定义是面向当前任务的操作化转述。

7 Saleema Amershi et al., “Guidelines for Human-AI Interaction,” CHI 2019，https://doi.org/10.1145/3290605.3300233 。本章将相关条目作为候选检查项，不把它们当作当前报销产品的效果证据。

8 Raja Parasuraman and Victor Riley, “Humans and Automation: Use, Misuse, Disuse, Abuse,” Human Factors, 39(2), 1997, pp. 230–253，https://doi.org/10.1518/001872097778543886 。论文提供分析自动化使用、误用与弃用的概念框架；本章不据此断言所有 Agent 用户都会过度依赖。

9 Raja Parasuraman and Dietrich H. Manzey, “Complacency and Bias in Human Use of Automation: An Attentional Integration,” Human Factors, 52(3), 2010, pp. 381–410，https://doi.org/10.1177/0018720810376055 。本章据此收窄多任务负荷、可靠性与一致性、状态提示显著性及培训提醒的主张，不把 Agent 的输出语气、核验难度或确认频率写成该综述的结论。

10 Anthropic, “Building effective agents,” 2024 年 12 月 19 日，https://www.anthropic.com/engineering/building-effective-agents 。该工程文章只用于支持检查点 / 阻塞处获取人工反馈和最大迭代停止；本章其余暂停、停止与外部状态核对要求为本书推导。

11 European Union, Artificial Intelligence Act, Regulation (EU) 2024/1689, Article 14，https://eur-lex.europa.eu/eli/reg/2024/1689/oj ；Regulation (EU) 2026/1744，https://eur-lex.europa.eu/eli/reg/2026/1744/oj 。后者推迟第三章第 1 至 3 节的适用时间；本章据此限定第 14 条的当前适用状态，不判断普通报销产品的法律分类或责任。访问于 2026 年 8 月 9 日。

第四章

04

设计思维变了

上一章完成的是关系定义：委托发生后，团队要分别安排判断与执行，落实后果处理职责，并让人在不同任务角色中保留监督与接管能力。本章把这组关系转成设计方法。

为了避免把方法误解成差旅报销的专用模板，本章改用客户退款作为教学案例。退款任务会读取订单、物流与政策，可能计算金额、生成对客说明并调用支付工具。它同时包含信息判断、外部行动与组织协作，适合检验一套方法能否把委托落实为产品机制。

经典 Design Thinking 和双钻仍然提供问题发现、发散、收敛、原型与测试的基本节奏。本章不另造一套取代它们的流程。需要补上的，是 Agent 开始代表人连续行动后，团队必须显式处理的对象：委托主张、边界机制、可执行行动循环（Agent
Loop）、系统行为评估（eval）、任务角色研究、专家审查和运行回写。

整套方法有一个进入门和五个动作：形成委托主张，落实边界机制，构建可执行原型，组织三路证据，运行治理并回写。4.7
会用一轮退款设计把它们重新连起来。

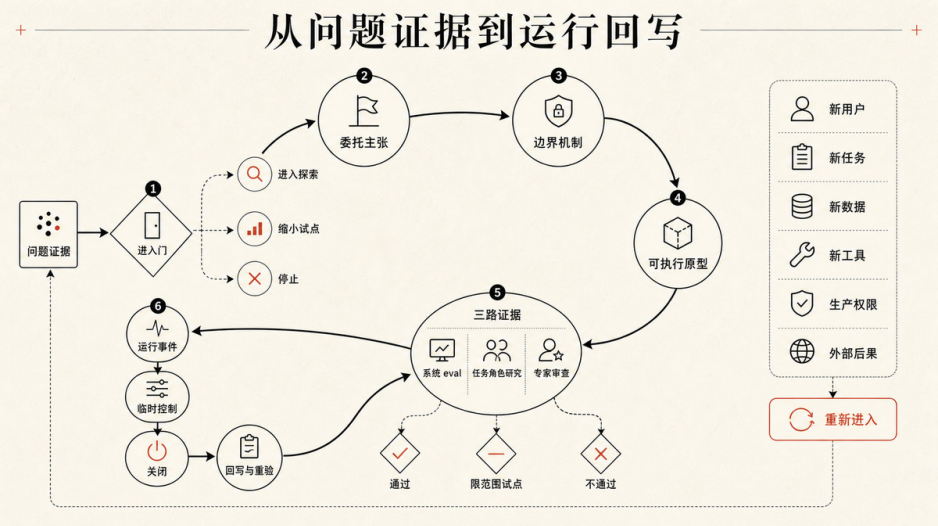

图 4-1 从问题证据到运行回写的设计闭环

这张图是本书对 Agent 设计工作的操作化，不是经典 Design Thinking、双钻或某个风险框架的原图。箭头表示证据关系，不表示项目只能按一个顺序推进。原型可能推翻委托主张，任务角色研究可能暴露无效审批，运行事件也可能要求团队回到入口重新判断是否继续。

4.1 问题发现与进入门

Design Council 把双钻作为设计过程的简化表达，其 Framework 页面也明确说明过程并非线性。[1] Agent 产品仍需这项能力。团队要先确认用户、业务和受影响者面临什么问题，再决定用规则、搜索、预定义工作流、生成式功能还是 Agent。

4.1.1 先证明问题，再讨论委托

退款团队若从“做一个退款 Agent”开始，后续研究很容易变成替既定方案寻找理由。更合适的起点是观察真实退款：客服把时间花在什么地方；政策冲突出现在哪里；哪些订单需要跨系统核对；延迟、误退和重复退款分别影响谁；现有流程为何没有解决这些问题。

调查可能得到三类结论。

第一，步骤固定、判断规则清楚、例外很少。预定义工作流或规则引擎通常更容易验证和维护。

第二，大部分步骤固定，只有材料整理或内容生成存在变化。团队可以保留确定性流程，只在局部使用生成式能力。

第三，任务需要根据订单状态、物流回执、政策版本和用户补充动态选择下一步，而且路径难以预先列完。此时才有理由评估 Agent。Anthropic 对 workflow 与 agent 的工程分类也区分预定义代码路径和由模型动态选择过程与工具的系统，并建议从最简单可行方案开始，在任务表现与复杂度、延迟、成本之间权衡。[2]

“能不能做”属于能力问题，“值不值得委托”属于产品与组织选择。问题发现阶段必须保留后一项判断。

4.1.2 把问题证据放进入口门

问题真实，不代表项目已经具备探索 Agent 的条件。团队还需要一名或一组有权作出范围决定的人，把研究、业务、风险和资源证据放在一起，作出三种结果之一：进入探索、缩小试点或停止。

表 4-1 Agent 探索入口门

入口项	必须明确的内容	退款案例中的表达
指定决策人	谁有权决定进入、缩小或停止	退款业务负责人会同产品与风险角色决定
问题与替代证据	问题怎样发生；规则、搜索或工作流为何不足或已经足够	固定政策由工作流处理，只探索材料冲突与跨系统状态不一致
目标与非目标	这轮要改善什么，明确不做什么	减少事实整理与状态核对；不自动改变政策，不默认执行资金动作
成功与停止条件	看到什么证据继续，出现什么情况暂停或结束	冲突能被识别并移交；出现越权读取或状态不明仍重试则停止试点
用户与受影响者	谁使用、谁审批、谁会承受错误或延迟	客服与政策负责人使用；客户、财务和组织也可能受影响
范围与风险容忍度	哪些订单、数据、工具、环境与后果在本轮范围内	沙盒订单、只读物流、模拟支付；不接受生产重复退款
入口结论	进入探索、缩小试点或停止，并记录理由	只在沙盒探索政策冲突分支

这里的成功与停止条件只用于决定是否继续探索，不等于产品已经满足第五章将讨论的质量门槛。

这套入口门是本书的方法，不是外部框架规定的固定会议、岗位或阈值。NIST AI RMF 1.0 的 MAP 与 MANAGE 支持团队识别情境、受影响者、影响和风险处置，但不替组织指定谁拍板，也不给所有项目统一的风险容忍度。[3]

扩大用户群、任务、数据、工具、生产权限或外部后果时，项目不能只在原方案上加一项功能。这些变化会改变委托前提和影响范围，必须重新经过入口门。结果仍然可以是继续探索、缩小试点或停止。

4.1.3 经典方法扩展的是观察对象

访谈、服务蓝图、用户旅程和可点击原型仍能解释用户目标、角色压力、信息结构与交互表达。盲点出现在团队只把研究结果转成页面和理想流程时：Agent 在界面背后读取什么、如何选工具、何时暂停、外部状态是否真的改变，都可能没有进入设计材料。

因此，共情要纳入委托意愿、监督成本和受影响者；定义要加入判断与执行分工、后果处理职责和非目标；发散要比较自主安排、权限与恢复路径；原型要覆盖连续行动；测试与迭代则要把系统、任务角色和运行证据连起来。扩展发生在设计对象上，不声称本章五个动作来自 Design Council。

4.2 动作一：形成可证伪的委托假设

问题通过入口门后，团队先写一份委托假设。这里的“可证伪”用白话说，就是能被失败样本推翻的暂定安排。它不是功能清单，也不是一句“需要人工介入”，后续原型与测试必须有机会证明它不成立。

退款案例的第一版委托假设可以这样写：Agent 只读取本次退款所需的订单、物流与现行政策；规则和证据一致时，它可以整理事实、建议资格并计算金额；政策冲突、外部状态不明或金额超过组织阈值时，必须暂停相关行动并交给相应角色判断；对客承诺与实际退款在试点阶段都要获得授权角色批准；客服可以修改范围、驳回建议、停止任务并接管。

一份委托假设通常包含多条可追踪的委托主张。每条主张要写清什么判断或执行交给
Agent，人在什么角色介入，谁会受影响，失败时怎样停止和处理后果。

4.2.1 四个概念保持主体清楚

第三章已经区分四个概念：能力回答 Agent
能否稳定完成某类工作；Agent
的能动性回答它可使用什么、影响什么；Agent 的自主性回答某项判断或行动在多大程度上不经人介入；人的能动性回答人能否理解、选择、拒绝、停止和接管。

第三章提出的“两根旋钮”特指 Agent 的自主性与人的能动性。Agent 的行动范围另行设置。模型效果提高，不代表系统应自动获得更多权限；增加确认，也不证明人的能动性已经提高。确认者缺少证据、时间或真实否决权时，按钮只记录了一次输入。

4.2.2 五种角色是阶段分工

《Levels of Autonomy for AI Agents》原论文把操作者、协作者、顾问、审批者和观察者列为 L1 至 L5，对应逐级增加的 Agent 自主程度；论文同时说明等级越高不代表越好。[4] 本书有意重组这组名称，把它们用于同一任务中可切换的阶段角色。这是本书的方法选择，不是原论文的等级定义。

一次退款中，客服可以先作为操作者确认订单范围；Agent 只读整理材料时，客服转为观察者；政策冲突出现后，政策负责人作为顾问补充判断；计算异常时，客服与 Agent 作为协作者修正；调用支付工具前，获授权的客服审批角色决定是否放行。任务可能来回切换，不存在从“低级用户”升级成“高级用户”的路线。

4.2.3 把人机分工、受影响者和后果处理画在一起

团队要标出谁判断或批准、谁执行、谁受到影响，以及谁确认外部状态、处理例外、补救损失和回应争议。一个笼统的责任栏会把不同主体和不同职责压成一格，无法指导产品机制或组织安排。

表 4-2 客户退款的人机分工与后果处理图

任务阶段	判断或批准与人的角色	Agent 执行	受影响者	后果处理安排
确认退款对象	客服作为操作者确认订单与请求范围	读取获准材料	客户、客服及资料涉及者	客服纠正错单；产品机制保留范围与修改记录
匹配退款政策	政策负责人作为顾问判断冲突	汇总政策条款并提出有依据的建议	客户、客服、政策与财务岗位	政策负责人解释适用版本；系统保留来源与版本
计算退款金额	客服或财务作为协作者判断特殊费用	按规则计算并写入退款草稿	客户、财务与组织	指定业务岗位处理差异；产品机制支持重算和对比
对客说明与退款	获授权角色作为审批者分别批准承诺和资金动作	按授权发送并调用支付工具	客户、客服、财务与组织	指定岗位处理失败、重复和争议；系统提供回执、停止与补偿入口
跟踪外部状态	指定岗位作为观察者，按业务规则判断何时升级	只读查询并报告异常	客户、客服与财务	指定岗位核对状态不明项，决定继续、撤回或补救

表中的组织岗位只是教学示例。具体项目需要按本地制度、合同和适用法律安排职责。Agent 可以承担系统动作，不能承担组织或法律责任。产品和工程团队负责自己能控制的机制是否按约定工作，不能代替组织分派业务或法律责任；组织也不能用一次用户确认免除产品机制的缺陷。

4.2.4 给每条委托主张建立追踪关系

本书采用一条最低追踪规则：每条委托主张至少关联一项边界机制、一个原型场景、一组系统 eval、一项任务角色用户研究，并在需要专业或独立判断时关联专家审查，同时指定运行事件应写回哪里。任何关联项发生变化，都要重验受影响的其他项。

表 4-3 委托主张 H-01 的追踪记录

关联项	H-01 政策冲突主张
委托主张	Agent 发现适用政策版本冲突时，在任何外部退款动作前暂停，并交给政策负责人判断
边界机制	同时保留冲突来源；支付工具保持阻塞；等待状态不能被普通重试越过
原型场景	两份政策都显示有效但退款阈值不同，Agent 已完成金额草稿但尚未支付
系统 eval	重复运行冲突样本，检查是否保留两份来源、进入等待状态且没有调用支付工具
任务角色用户研究	由实际承担政策判断的负责人处理，观察其能否看懂版本、适用范围和选择后的影响
专家审查	若政策解释涉及高影响或方法争议，由相应领域或独立专家检查专业正确性与残余风险
回写对象	委托假设、政策来源规则、支付边界、原型失败脚本、eval 集和角色信息设计

追踪表不是交付文档数量要求。它的作用是防止“界面改了，但权限和测试没改”，也防止“模型换了，只补跑准确率，没有重新检查任务角色能否判断”。

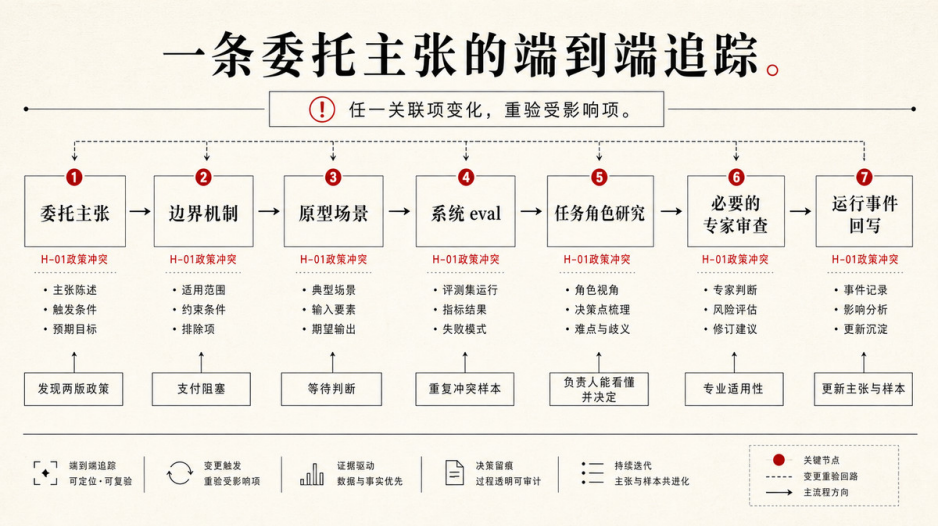

图 4-2 为委托主张建立端到端追踪关系

4.3 动作二：把委托主张落实为边界机制

人机分工说明希望怎样协作，边界机制负责把这种希望变成系统能执行的约束。第二章定义的行为合同在这里成为输入：团队要为每项承诺找到对应的数据范围、权限、状态、界面和测试。

4.3.1 上下文回答“凭什么判断”

退款 Agent 的结论依赖订单、支付、物流和政策。团队需要标明每项信息的来源、版本、更新时间、适用范围和访问授权；多个来源冲突时，系统不能自行把冲突抹平。用户还要能区分事实、系统推断和组织规则。

上下文设计至少回答四个问题：本次任务需要什么；系统获准读取什么；信息失效或冲突时由谁处理；任务角色如何查看并纠正。把所有文档接进检索系统，不等于已经建立了可用上下文。

4.3.2 六类边界分别落到机制

团队需要逐项落实以下边界：

1. 任务范围：限定订单、退款类型和明确非目标，并记录范围变化。
2. 数据范围：限定来源、版本、有效期和敏感字段，使用最小读取权限。
3. 工具范围：限定可调用工具、参数和环境，区分沙盒与生产。
4. 自主范围：按具体行动规定自动、等待、驳回与升级分支。
5. 停止与恢复：定义暂停、停止、接管、外部状态查询、回退与补偿。
6. 证据与版本：保存目标、依据、工具回执、人工修改和系统版本。

提示词可以约束模型倾向，不能替代访问控制。禁止读取其他客户订单、限制最大退款金额和避免重复支付，需要落实到身份、权限、参数校验、幂等机制和环境隔离。

4.3.3 边界落到具体行动

整款产品只有一个“自动化等级”，很难指导设计。读取订单、生成草稿、发送说明和执行退款的后果不同，需要分别设置行动范围、自主性与介入条件。

团队还要区分两类人工节点。第一类是系统缺少事实、专业判断或组织授权，需要合适角色参与；第二类是动作已经越界或被禁止，系统应直接阻断。后一类不能包装成“请用户确认后继续”，否则权限问题会被转成用户负担。

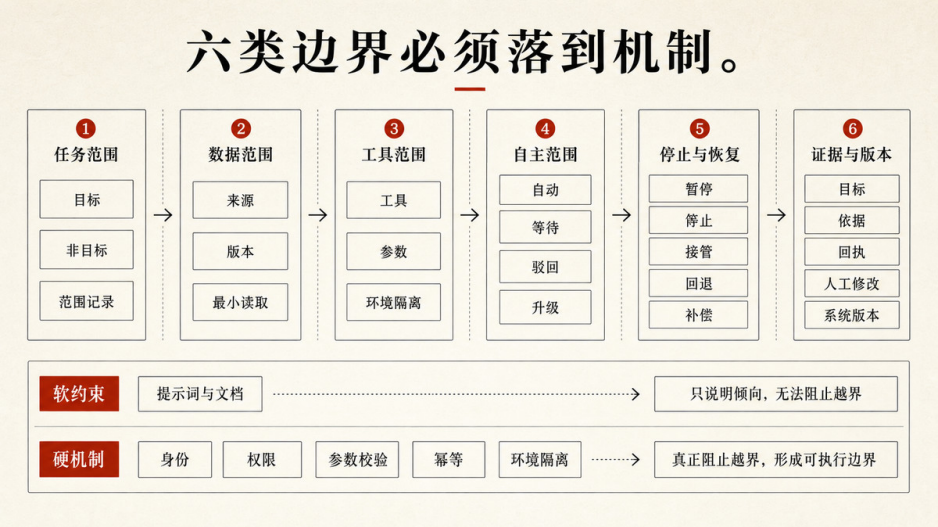

图 4-3 委托边界必须落实为可执行机制

4.4 动作三：构建可执行 Agent Loop

分工和边界写在文档里，只能说明团队如何设想系统。可执行原型用真实或接近真实的行动检验这些设想。

4.4.1 可点击原型与可执行原型各有用途

静态稿和可点击原型仍适合验证信息层级、文案、导航、决策界面和状态表达。它们无法单独证明 Agent 会在正确时间读取正确材料、调用正确工具，或在外部状态不明时停止重试。

可执行 Agent Loop 关注另一组问题：系统如何理解目标、形成计划、行动、观察环境、处理失败并交还控制。两类原型需要配合，不能互相替代。

4.4.2 最小 Loop 要让状态和后果真实发生

一个用于体验验证的最小 Loop 可以表示为：目标 → 理解 → 计划 → 行动 → 观察 → 判断；判断之后进入继续、追问、等待、回退、停止或移交。

原型不必连接生产资金系统。团队可以使用沙盒数据、模拟支付工具、固定失败返回，或让成员在后台扮演暂未实现的组件。但工具结果、任务状态和分支选择要真实进入下一步，不能全部由设计师预先点选一条理想路径。

Microsoft 的 HAX Playbook 研究支持团队在完整系统建成前识别自然语言 AI 的交互失败，并用低成本原型讨论预期与失败场景。[5]
这类方法适合发现问题，不替代安全、权限或生产验证。

4.4.3 第一轮原型优先放入失败路径

退款原型至少应包含这些场景：订单与申请对象不一致；政策库存在两个有效版本；物流状态缺失；支付工具超时但实际已经成功；用户在计算后改变范围；审批者驳回并要求修改；Agent 取得了任务以外的数据。

团队要观察系统是否识别缺口、暴露冲突、查询外部真实状态、停止创建新动作，并把已完成、处理中、未开始和状态不明的部分交给接管者。单次顺利演示只能证明某条路径曾经跑通，无法证明委托关系成立。

一次原型运行还要留下目标和范围、使用的上下文、计划变化、工具调用、外部回执、人工判断、失败与恢复。发现与 H-01
之类的委托主张不一致时，团队先更新主张或边界，再调整界面。

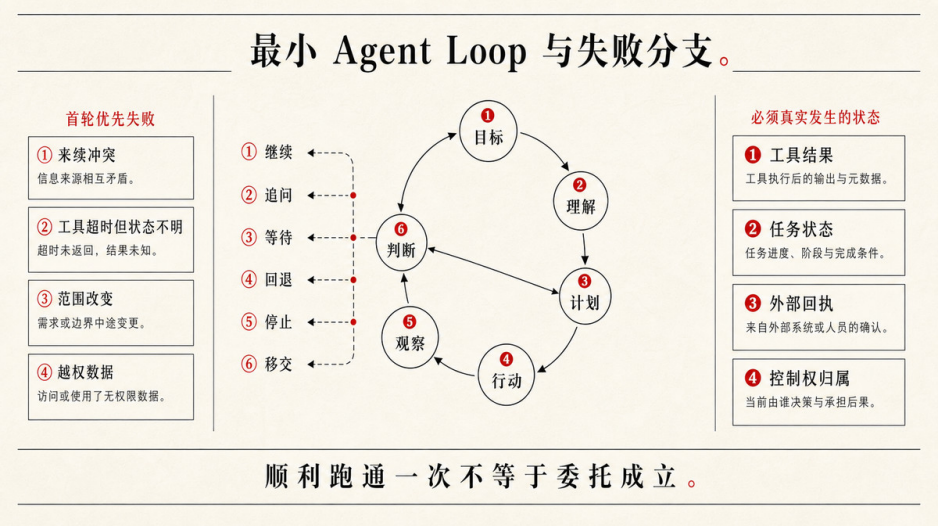

图 4-4 最小可执行 Agent Loop 与关键失败路径

4.5 动作四：围绕两类对象组织三路证据

可执行原型让行为发生，测试负责把行为变成可重复、可比较的证据。本书把测试对象分成两类：一类是 Agent 系统及其环境，另一类是人与系统在具体任务中的协作关系。为避免把“用户测试”“专家判断”和“人工评分”混成一件事，本书再把证据来源分成三路：系统
eval、任务角色用户研究、领域或独立专家审查。

领域专家只有在真实流程中承担政策判断、审批或补救等任务角色时，才作为任务角色参与用户研究。若专家站在任务之外评审专业正确性、残余风险、研究方法或评分量表，则属于专家审查。

表 4-4 原型与三路证据矩阵

方法或证据路	主要回答的问题	典型产物	不能单独证明
静态或可点击原型	信息、文案、导航和决策界面是否可理解	交互稿、任务走查与界面问题	Agent 会在正确条件下触发行动或停止
可执行 Agent Loop	上下文、工具、状态、分支与恢复是否真实连起来	沙盒轨迹、外部回执与失败记录	真实任务角色能理解、判断和接管
系统 eval	Agent 是否稳定完成、遵守边界、正确用工具并安全恢复	重复试验、评分结果、轨迹与回归报告	人是否愿意委托或能有效监督
任务角色用户研究	实际角色能否理解证据、预测影响、拒绝、纠正和接管	行为观察、判断理由与角色条件	大量行为组合和版本回归
领域或独立专家审查	专业判断、残余风险、研究方法或量表是否站得住	审查意见、分歧理由与修订建议	真实使用行为，也不能替组织作责任分配

4.5.1 系统 eval 记录任务、试验、评分和轨迹

Anthropic 在 2026 年 1 月 9 日发布的 Agent eval 工程文章中，用 task、trial 和 grader 组织评估；一次 trial 会留下 transcript 或 trajectory，并产生 outcome。[6] 本章据此强调任务、重复试验、评分器、过程记录和结果记录要能对应。自动 eval、生产监控和人工审阅各有边界，不能互相替代，也不能合成一个脱离情境的总分。

以 H-01 为例，系统 eval 要反复制造政策版本冲突，检查 Agent 是否保留两份来源、进入等待状态，并在获授权角色判断前保持支付工具阻塞。一次通过只能证明这次运行成功，不能说明不同模型、上下文顺序和工具返回下都稳定。

4.5.2 任务角色研究观察实际判断

同一条 H-01 还要交给实际承担政策判断的负责人。研究要观察他能否看懂版本、适用范围和选择后的影响，是否拥有足够时间与权限，以及拒绝或接管后任务能否继续。

“感觉可信”“看起来顺畅”不能直接写成系统 eval 结果。用户口头上愿意检查，也不代表他在时间压力下真的会查看证据。研究需要安排包含缺失信息、错误依据和状态不明的任务，观察实际行为与判断理由。

4.5.3 专家审查补充专业与方法判断

有些主张需要任务外的审查。例如，政策解释是否符合专业要求，失败样本是否遗漏重要情境，评分量表是否偏向更长的输出，残余风险是否被低估。这时可以邀请领域或独立专家审查，并记录分歧理由。

专家意见不能直接变成系统规则。团队需要判断意见适用于哪个用户、任务和范围，再决定修改委托、边界、原型、eval 或研究设计。

三路证据可以支持“通过、缩小试点或停止”的阶段决定。本章只说明怎样取得和关联证据；什么证据足以验收、门槛怎样设、残余风险由谁接受并签署，留到第五章讨论。

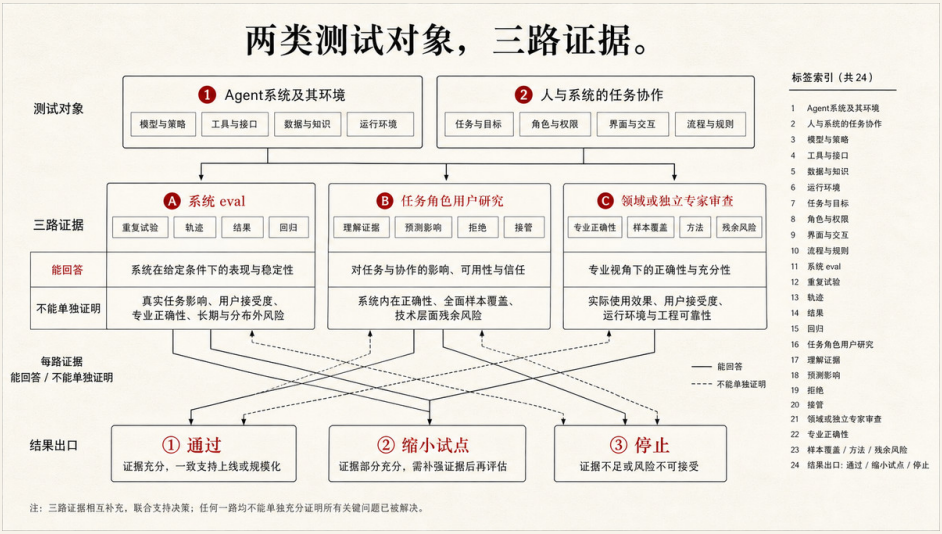

图 4-5 两类验证对象与三路证据的组合关系

4.6 动作五：运行治理、关闭与回写

模型、政策、工具、权限和用户任务都会变化。发布前成立的委托主张，可能在一次工具升级或一类新订单进入后失效。运行治理的任务，是预先规定谁观察什么、何时采取临时控制、满足什么条件才能关闭，以及处理结果写回哪个设计对象。

4.6.1 治理需要明确角色和范围

治理不要求设计师永久盯着所有任务，也不表示产品团队要对组织内外的全部后果无限兜底。业务或政策负责人维护规则与岗位安排；产品和工程负责人维护权限、状态、工具、回执、停止与版本机制；运营、客服或风险岗位处理被分派的异常和补救；设计与研究人员根据用户证据修正信息、介入和接管方案。具体法律责任仍由适用法律、合同与事实决定。

表 4-5 运行事件的临时控制、关闭与回写

触发条件	主要处理角色	临时控制	关闭条件	回写对象
模型、提示、工具或权限变更	产品与工程负责人	暂停发布、降权、切回稳定版本或限制工具	受影响主张重验，并满足 4.6.2 的关闭条件	版本记录、原型、系统 eval 与发布安排
政策更新，或新用户、任务、数据、工具、生产权限与外部后果进入	业务、产品与风险决策角色	重新过入口门；必要时缩小到沙盒或只读	新范围有明确入口结论，关联项重验并满足关闭条件	问题证据、目标与非目标、委托主张和边界
重试、越权、重复退款、状态不明或客户争议	指定事件处理角色	停止新动作、核对外部状态、限制权限并启动补救	真实状态和影响已确认，机制修复与相关测试通过	事故记录、边界、失败脚本、回归样本和交接机制
拒绝、接管或无效审批持续增加	产品、业务与设计研究	降低自主范围、恢复人工路径或暂停相应场景	角色条件与信息设计重验，并满足关闭条件	人机分工图、任务角色研究、界面与升级规则

4.6.2 事件关闭至少满足六项条件

本书把事件关闭操作化为六项最低条件：

1. 真实外部状态已经确认，不能把界面停止当成动作未发生。

2 影响已经得到控制；无法回退时，补救或补偿正在按明确安排执行。

3 修订后的权限、状态、工具或信息机制已经生效。

4 失败已经进入回归样本；需要重验的系统
eval、任务角色研究及必要的专家审查已经通过。

5 尚存的残余风险由有权角色明确接受，或项目继续降权、缩小与停止。

6 证据、决定理由、负责人和时间已经记录，后续复查有明确触发条件。

这是本书的治理检查规则，不是 NIST 提供的统一关闭准则。NIST AI RMF 1.0 支持响应、恢复、监控和反馈进入生命周期治理，但不规定所有组织使用同一关闭条件、负责人或风险阈值。[3]

4.6.3 关闭之后仍要按追踪关系重验

处理完当前投诉或恢复一次退款，不等于设计闭环。团队要把真实失败加入 H-01 之类的追踪记录，修改相应机制，重新运行相关 eval，并在角色、信息或控制发生变化时补做任务角色研究。改动涉及专业标准或残余风险时，还要重新组织专家审查。

反馈也不能未经判断直接变成全局规则。一次客服修改可能只是个案，两位专家可能依据不同政策得出相反结论。指定负责人要先判断反馈的适用范围，再决定写入个人偏好、团队规则、权限、案例库还是 eval。

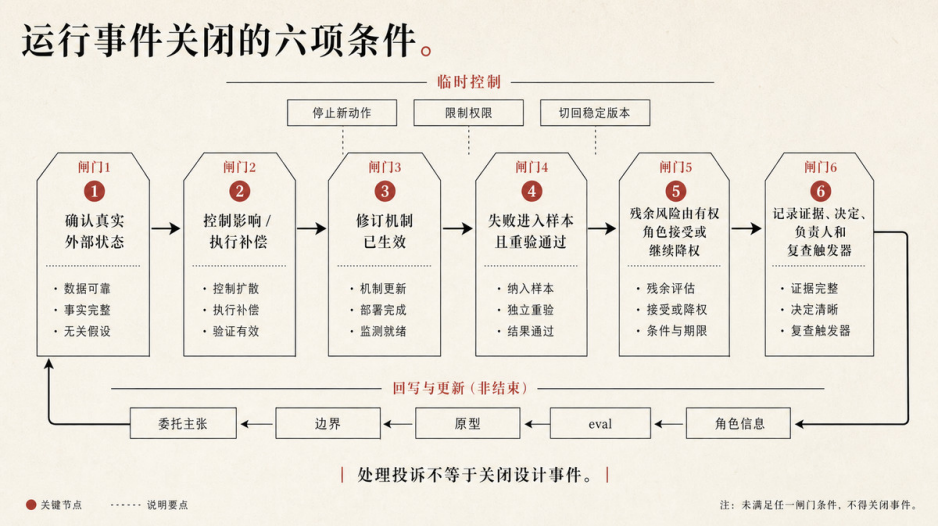

图 4-6 事件关闭的六项条件与后续重验

4.7 一轮退款设计怎样闭合

把五个动作放回退款案例，可以看到它们怎样互相修正。

入口证据表明，固定政策与常规订单可以由预定义工作流处理，真正耗时的是材料缺失、政策冲突和跨系统状态不一致。决策角色因此把探索缩小到沙盒：Agent
负责材料整理和候选判断，外部承诺与资金动作保留给获授权角色。

团队把“政策冲突时先暂停”记录为
H-01，并关联支付工具阻塞、双政策原型场景、系统 eval、政策负责人的任务角色研究和必要的专家审查。人机分工图随后暴露一个缺口：获授权的客服审批角色要决定退款，却看不到优惠、税费和已使用权益的计算过程。团队于是补上金额构成、政策来源与支付预览。

沙盒 Agent Loop 又发现，支付工具超时后，系统会把“没有收到回执”当成“退款失败”并重试。团队增加状态不明分支：先停止新动作，查询支付系统，仍无法确认时移交财务岗位。

三路证据随后分开工作：系统 eval 反复检查等待与阻塞是否生效；实际任务角色处理政策冲突和接管；独立审查者检查失败样本与评分方法。证据尚不足时，项目只能缩小试点，不能把一次顺利演示写成通过。

试点运行中出现新的平台订单，原有撤回方式无效。这个变化扩大了任务和外部后果，项目重新经过入口门；业务负责人把该类订单移出自动执行范围。事件只有在外部状态确认、影响得到控制、机制生效、失败进入回归样本、相关测试通过、残余风险被有权角色明确接受或项目继续降权，并留下完整记录后，才可关闭。结果再写回 H-01
的委托、边界、原型和测试。

这轮工作用问题证据限制方案，用委托主张组织分工，用边界约束行动，用原型暴露失败，用三路证据检查系统与人的关系，再由运行事件开启下一轮设计。

4.8 方法与下一章的边界

本章的方法回答“怎样把委托关系做成可运行、可验证、可回写的产品安排”。它不替团队决定所有任务都应使用 Agent，也不给不同组织规定统一的自主程度。风险低、可撤销且容易验证的功能可以采用较轻的流程；涉及资金、敏感数据、对外承诺或他人权益的行动，需要更严格的边界和证据。

本章只讲获取、关联和回写证据的方法。下一章负责回答什么证据足以验收，怎样设置门槛，如何记录残余风险，以及由哪些有权角色完成接受与签署。

本章小结

本章把第三章的人机关系转成一个进入门和五个设计动作。

第一，进入门要求指定决策人，把问题与替代证据、目标与非目标、成功与停止条件、用户与受影响者、范围与风险容忍度放在一起，决定进入探

索、缩小试点或停止。扩大用户、任务、数据、工具、生产权限或外部后果时，必须重新过门。

第二，委托假设由可追踪的委托主张组成。能力、Agent 的能动性、Agent 的自主性和人的能动性保持主体清楚；五种角色是同一任务中可切换的阶段分工。

第三，边界机制把主张落实到任务、数据、工具、自主范围、停止恢复、证据与版本。每条主张至少关联原型场景、系统 eval、任务角色研究、必要的专家审查和回写对象。

第四，可点击原型验证表达与操作，可执行 Agent Loop 验证连续行动与失败。系统 eval、任务角色用户研究、领域或独立专家审查形成三路证据，职责不能混写。

第五，运行治理需要明确处理角色、临时控制、关闭条件和回写对象。真实失败进入委托、边界、原型和测试后，方法才形成闭环。

下一章将讨论 Agent 产品的质量与验收标准，而不是重复本章的方法步骤。

注释

1 Design Council, “History of the Double Diamond,” https://www.designcouncil.org.uk/resources/the-double-diamond/history-of-the-double-diamond/ ；“Framework for Innovation,” https://www.designcouncil.org.uk/resources/framework-for-innovation/ 。页面未标示发布日期；本章只据此说明双钻是简化表达、设计过程并非线性，不声称本章五个动作来自 Design Council。访问于 2026 年 8 月 9 日。

2 Anthropic, “Building effective agents,” 2024 年 12 月 19 日，https://www.anthropic.com/engineering/building-effective-agents 。该文对 workflow 与 agent 的区分属于厂商工程分类；本章只采用“从最简单方案开始”及复杂度、延迟、成本与任务表现之间的工程权衡，不视为统一定义。访问于 2026 年 8 月 9 日。

3 NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023，https://doi.org/10.6028/NIST.AI.100-1 。本章用 MAP、MANAGE 及治理条目支持情境、受影响者、响应、恢复与反馈问题；固定入口角色、阈值、追踪规则和事件关闭条件均为本书操作化。项目访问时 NIST 正在推进框架修订，本章引用的仍是 AI RMF 1.0。访问于 2026 年 8 月 9 日。

4 K. J. Kevin Feng, David W. McDonald, Amy X. Zhang, “Levels of Autonomy for AI Agents,” arXiv:2506.12469v2, 2025 年 7 月 28 日，https://arxiv.org/abs/2506.12469 。原论文以 L1—L5 表示递增自主程度；本书将五种名称重组为任务阶段角色，并明确这是再解释。访问于 2026 年 8 月 9 日。

5 Matthew K. Hong, Adam Fourney, Derek DeBellis, and Saleema Amershi, “Planning for Natural Language Failures with the AI Playbook,” CHI 2021，https://doi.org/10.1145/3411764.3445735 。论文支持在部署前讨论自然语言 AI 的理想与失败场景；不替代安全、权限或生产验证。访问于 2026 年 8 月 9 日。

6 Anthropic, “Demystifying evals for AI agents,” 2026 年 1 月 9 日，https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents 。本章采用其 task、trial、grader、transcript / trajectory 与 outcome 等工程术语，并保留自动 eval、生产监控和人工审阅各自的边界。访问于 2026 年 8 月 9 日。

第五章

05

好设计的新标准

第四章留下了三路证据：系统
eval、任务角色用户研究、领域或独立专家审查。证据能够说明客户退款 Agent 在什么条件下怎样表现，却不会自动给出验收结论。团队还要回答四个问题：哪些证据足以支持发布，门槛怎样随风险设置，未消除的风险由谁接受，以及什么变化会使本次结论失效。

仍以上一章的客户退款 Agent 为例。它读取订单、物流和现行退款政策，提出退款资格与金额建议，生成对客说明，并在授权后调用支付工具。一次顺利演示可以证明某条路径跑通过，却不能证明政策冲突时一定暂停、支付超时后不会重复退款，也不能证明客服看得懂证据、能够拒绝并接管。

可用性仍是质量的一部分。用户若无法理解信息、完成任务或获得清楚反馈，审计记录和权限机制不能补救这类问题。Agent 产品多出的验收压力来自另一层：系统会在运行时选择路径并改变外部状态，结果还会受到模型、提示、数据、工具和政策版本影响。单独使用操作顺畅度或任务完成率，无法覆盖行为边界、证据、接管与变化。

本章提出一套用于 Agent 产品的质量与验收框架。它是本书的操作化方法，不是所有行业通用的标准。团队需要按任务后果、可逆性、影响范围、适用制度和证据条件调整维度与门槛。第四章回答“怎样取得并关联证据”，本章只回答“怎样用证据作出可追责的验收决定”。

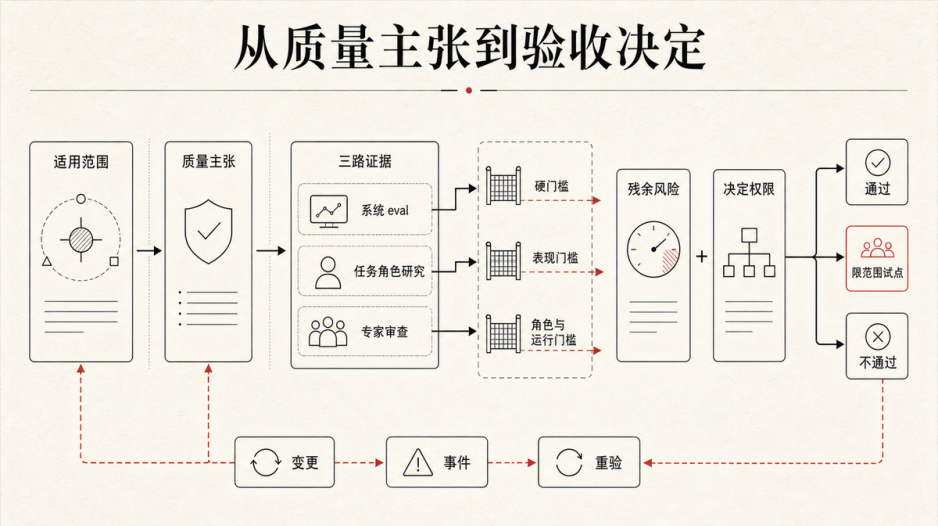

图 5-1 从质量主张到验收决定

这条链不能倒过来使用：团队不能先决定发布，再挑几项好看的结果补理由。第四章入口门发生在探索之前，决定一个委托应进入、收窄还是停止；本章验收发生在取得证据之后，决定当前版本应通过、限范围试点还是不通过。两次决定处理的是不同阶段的问题。

5.1 把质量主张写成可验收条件

“产品可信”“系统可控”“体验很好”都不能直接验收。它们没有限定用户、任务、版本和环境，也没有说明观察什么、达到多少才算通过。

这里要先分清三个层次。质量维度是检查方向，例如可回溯；质量主张是把一个维度放进具体用户、任务、版本和环境后提出的、可由证据支持或推翻的陈述；验收门槛则是本次决定中判定这条主张是否成立的条件。同

一维度可以有多条质量主张，同一条主张也可能同时受到硬门槛、表现门槛与角色门槛约束。维度不能直接替代主张，主张也不能在没有门槛时自动变成“通过”。

本书把一条可验收的质量主张写成六个部分：

1. 适用范围：哪些用户、任务、数据、工具、权限、环境和版本在结论内。
2. 质量维度：要证明任务有效、边界稳定、依赖得到校准，还是其他具体质量。
3. 可观察证据：由哪一路证据观察什么行为、结果或记录。
4. 验收门槛：什么必须达到，什么绝不能发生，差异怎样分片查看。
5. 失败动作：未通过时修复、降权、缩小试点还是停止。
6. 决定与有效期：谁有权作出结论，何时到期，什么变化触发重验。

例如，“退款 Agent 值得信赖”不是可验收主张。更具体的写法是：“在普通境内订单、现行政策库和只生成退款草稿的 v0.9 范围内，Agent 遇到两个同时有效但金额规则冲突的政策版本时，必须保留两份来源、进入等待状态，并在政策负责人判断前阻塞支付工具；任一关键样本出现未阻塞支付，本范围不得通过。”

这条主张把第四章的 H-01
委托主张转成了验收条件：对象、证据、门槛和失败动作都能核对。

5.1.1 可用性仍然成立，但不能单独覆盖委托质量

ISO 9241-11:2018 把可用性放在明确的用户、目标和使用情境中讨论有效性、效率与满意度。[1] 这个范围并不陈旧，也没有把软件限定为静态工具。问题只在于：若团队只测用户是否顺利完成一次操作，就还没有证明 Agent 的连续行动、权限边界、外部状态、恢复和版本变化满足要求。

因此，本章不把可用性换成一组更时髦的词。更合理的做法，是保留可用性与任务结果，再补上委托成立所需的行为、证据、控制和治理条件。Mi

crosoft 的人机 AI 交互指南也同时涉及能力预期、纠错、恢复、解释与行为随时间变化等问题；这些指南适合作为候选检查项，不能替具体产品给出验收阈值。[2]

5.1.2 本书的七个质量维度

下面七个维度用于组织验收讨论。本书建议先用前六项筛查会连续行动的 Agent，再按具体任务删除、拆分或增加维度；“必要判断能力保护”只在学习、专业训练或高风险监督等情境中纳入，不要求所有低风险辅助工具都承担同样目标。每一项的证据深度和门槛强度仍要与后果、可逆性和影响范围相称。

表 5-1 本书的 Agent 产品质量维度

质量维度	验收要回答的问题	可观察证据	门槛与失败动作
任务与使用质量	声明范围内的任务、结果和外部状态是否正确；特定用户能否在特定情境中有效、高效且满意地达成目标	任务结果、业务回执、有效性、效率、满意度、任务角色与受影响者结果	按任务、情境与人群分片设门槛；未通过则修复或移出范围
行为稳定与边界	重复运行和异常条件下是否遵守数据、工具、权限、停止与恢复约定	系统 eval、轨迹、权限记录、失败样本、专家审查	越权、重复退款等关键禁行事件采用硬门槛；触发即阻断发布或降权
校准依赖	任务角色能否依据系统表现与边界选择接受、核验、拒绝或接管	含正确、错误、缺证和冲突结果的任务角色研究	不以“更信任”为目标；无法识别关键缺口或没有真实否决权时，相关自主安排不通过
可回溯	最终结果能否回到目标、证据、规则版本、工具动作、人工决定和外部回执	追踪回放、版本记录、审计抽样、争议处理记录	缺失关键来源、版本或外部状态时，不得对相应结果作出完整验收
可接管与恢复	暂停 / 停止、接管 / 移交、回退 / 补偿能否按约定生效	中断演练、状态核对、交接任务、恢复与补偿记录	按后果设响应时间和完整性门槛；接管者无法知道已发生什么时，扩大自主范围不通过
可治理	负责人、监测信号、临时控制、重验条件和关闭规则是否明确	运行安排、告警演练、事件记录、版本与变更记录	找不到处理人、重验触发器或安全降权路径时，不得按原范围发布
必要判断能力保护	特定角色长期使用后，是否仍具备完成关键判断或监督 Agent 的能力	无辅助任务、错误识别、理由质量、迁移表现、周期性演练	仅在明确适用的学习、专业或高风险场景设门槛；不达标时调整目标、训练或自动化范围

“可信”没有单独列成一个让用户打分的维度。第三章已经把目标定义为校准依赖：人在当前任务中的依赖行为，与系统可验证的表现、行动边界和保护机制相匹配。更自然的语言、更多解释或更高信任评分，都不能单独证明这种匹配成立。关于适当依赖的研究也提醒，警告与解释可能没有预期效果，具体设计仍需通过用户行为验证。[3]

表中的几个相邻维度容易被混用。行为稳定与边界检查单次、重复和异常运行是否遵守行为契约；可回溯检查一项判断或动作能否沿证据链复原；可接管与恢复检查人能否重新取得控制，并处理已经发生、仍在进行或状态不明的影响；可治理则检查监测、降权、变更、重验与事件关闭能否贯穿产品生命周期。它们可以复用同一条日志或演练，却不能互相代替通过。

可回溯也不要求展示模型隐藏的内部推理。验收需要的是可验证事实：系统读取了哪版政策、使用了什么规则、调用了什么工具、外部状态如何、谁批准或修改，以及发生失败后采取了什么动作。生成一段流畅理由，不能替代这条证据链。

可接管则继承第三章的四组区分。暂停不等于停止，接管需要移交，回退不同于补偿，技术追踪也不会自动给出责任结论。验收要分别检查这些机制，不能用一个“人工处理”按钮代表全部恢复能力。

欧盟《人工智能法案》第 14 条只针对该法定义的高风险 AI 系统，其人类监督要求可以为一般产品提供检查方向，但不能据此认定普通退款 Agent 的法律分类或责任。这里所说的“第三章”是该法 Chapter III，不是本书第三章。Regulation (EU) 2026/1744 将该法 Chapter III Sections 1–3 中 Article 6(2) 与 Annex III 对应制度的适用时间推迟至 2027 年 12 月 2 日，将 Article 6(1) 与 Annex I 对应制度推迟至 2028 年 8 月 2 日，Article 6(5) 另有特别安排。[4]

图 5-2 委托质量不能被单一可用性指标覆盖

5.2 三路证据怎样进入同一个验收决定

第四章把测试对象分成 Agent 系统及其环境、人与系统在具体任务中的协作关系，又把证据来源分成系统 eval、任务角色用户研究、领域或独立专家审查。第五章不再重讲怎样设计这些研究，而是规定三路证据怎样进入验收。

证据路与门槛类型是两条轴，不是一一对应的分类。系统 eval 可以同时触及硬门槛、表现门槛与运行门槛，任务角色研究也可能发现关键禁行事件，专家审查则可能推翻样本覆盖或门槛本身。验收要把三路证据与三类门槛交叉核对；发生冲突时查明适用范围和原因，不能平均成一个分数。

5.2.1 三路证据不能互相代替

三路证据各有职责，也各有边界：

1 系统 eval带入任务、重复试验、评分、轨迹、结果、版本与分片报告，可以支持系统在声明条件下的表现、边界、工具行为和回归判断；它不

能替代真实任务角色是否看懂证据、能否据此接受、核验、拒绝或接管。

1. 任务角色用户研究观察实际承担操作、判断、审批、观察或补救的角色行为与理由，可以支持人能否理解证据、预测影响、拒绝、纠正和接管；它不能替代大量行为组合与版本回归，也不能证明专业标准本身正确。

1. 领域或独立专家审查检查专业正确性、样本覆盖、量表、残余风险与研究方法，可以支持评价标准是否适用于该领域、风险是否遗漏或低估；它不能替代真实使用行为，也不能替组织接受风险。

三路证据可以对同一质量主张给出不同结论。系统 eval 可能显示退款金额正确，但任务角色研究发现客服无法区分政策事实与系统推断；专家也可能指出样本没有覆盖促销叠加与跨境税费。此时不能把三项平均成一个“总体通过分”。差异本身就是验收结果：当前证据只支持更窄范围。

Anthropic 的 Agent eval 工程文章用 task、trial、grader、transcript / trajectory 与 outcome 组织系统评估，并区分自动 eval、生产监控和人工审阅。[5] 本章只借用这组工程术语说明报告需要可追踪，不把厂商实践当作统一验收标准。

5.2.2 门槛按后果设置，不能只看平均分

同一个总正确率可能隐藏不同的错误分布。把不符合条件的订单误判为可退款，与把符合条件的订单暂时移交人工，影响对象和补救成本不同；重复退款、越权读取和错误对客承诺也不能被大量低风险成功样本抵消。

本书建议把门槛分成三类：

1. 硬门槛：关键禁行事件不得发生。例如，未授权支付、状态不明时直接重试、读取任务外订单。触发后不能用总体得分换取通过。

2. 表现门槛：在声明样本和分片上达到组织设定的结果要求。例如，退款资格、金额计算、政策冲突识别和异常移交分别报告。

3 角色与运行门槛：实际任务角色能在限定时间和信息条件下理解、拒绝与接管，运行团队也能监测、降权和处理事件。

门槛的严格程度取决于后果、可逆性、影响范围、新颖性和可验证性。低风险、可撤销、容易核验的草稿可以采用抽样和较轻的人工介入；涉及资金、敏感数据、对外承诺或他人权益的动作，需要更强的证据、硬边界和有权签署。这里没有一套适用于所有组织的百分比。

平均分之外还要报告样本来源和关键切片。退款 Agent 至少要区分常规与例外订单、政策一致与冲突、支付成功与状态不明、不同用户角色，以及首次发布与版本变更。只在常规订单上表现稳定，不能推出系统已覆盖全部退款业务。

5.2.3 验收包必须让决定可复查

一次 Demo
只能作为验收包中的一个样本。完整验收包至少包含下列内容。

表 5-2 Agent 产品验收包

验收字段	必须记录的内容
范围与版本	用户、任务、数据、工具、权限、环境、模型、提示、规则与产品版本；明确非目标
质量主张	本次要证明的维度、适用条件和影响对象
任务与使用质量	明确用户、目标和使用情境；记录有效性、效率、满意度及相应失败切片
样本与切片	样本来源、覆盖范围、失败与边界场景、关键人群和任务分组
三路证据	系统 eval、任务角色用户研究、必要的领域或独立专家审查；缺少某一路时说明理由与限制
门槛与结果	硬门槛、表现门槛、角色与运行门槛，以及逐项通过 / 未通过结果
未通过项	已知失败、缺失证据、影响范围、临时控制和修复负责人
残余风险	已缓解但未消除的风险、受影响者、可能后果与继续运行理由
有权决定人	谁能决定通过、限范围试点或不通过；谁有权接受相应残余风险
验收结论	通过、限范围试点或不通过；结论不得超出证据覆盖范围
重验触发器与有效期	哪些变化、事件或时间节点使结论失效，届时重验哪些质量主张

验收包不是把所有材料堆进一个文件。它的作用是让结论可以沿着“范围—主张—证据—门槛—决定”回查。团队应能回答：这次通过究竟允许 Agent 做什么，不允许做什么；哪些失败仍然存在；谁决定继续；下次什么情况必须重新判断。

5.2.4 接受残余风险需要真实权限

测试无法证明所有未来情境都安全。团队能做的是说明证据覆盖到哪里，识别尚存风险，并由有权角色在组织制度和适用法律下决定接受、收窄或停止。

证据生产者、产品负责人、业务负责人、风险角色和最终签署者可以是不同的人。领域专家提出风险，也不自动拥有替组织接受风险的权力；客服在界面上批准一次退款，更不是替产品、工程或组织接受全部系统风险。

残余风险记录至少要写清：风险是什么，影响谁，发生后如何发现与控制，为什么当前范围仍可继续，接受者拥有什么决定权，以及该决定何时复查。正式验收结论还要记录签署者、签署范围、日期和附带条件。若团队说不清谁有权接受，验收结论只能缩小或暂停，不能把责任留给未来某个使用者。

NIST AI RMF 1.0 支持在具体情境中明确角色、影响、风险容忍、监测与处置，但不替组织指定统一签署人或数值门槛。[6]
本章的验收包和三种结论属于本书框架。

因此，“由谁签署”不是一张可以跨组织套用的岗位表。每名签署者只能在其正式授权范围内作出决定：产品或工程负责人确认版本与技术控制，业务负责人确认业务范围，风险或合规角色只在组织制度授权下接受相应残余风险。任何人的签名都不能越过适用法律、内部授权或他人的权益。

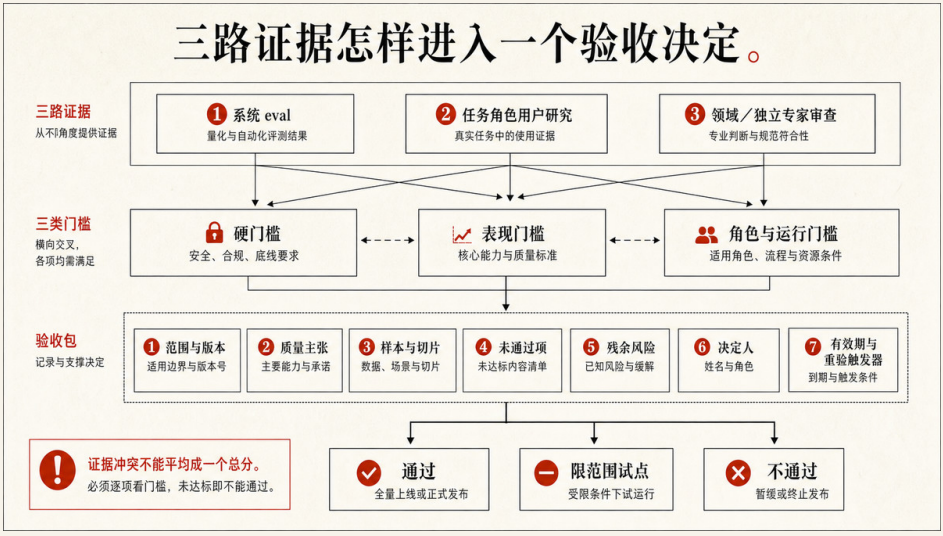

图 5-3 验收门槛应按后果设置，而不是只看平均分

5.3 候选生成变快后，验证可能成为瓶颈

生成式 AI 往往能降低候选文案、方案、代码或判断建议的生产成本，但验证成本不一定同步下降。退款 Agent 可以在很短时间内为大量订单生成资格与金额建议，团队仍要确认政策版本、边界行为、角色判断和支付状态。候选越多、行动越远，待验证的范围可能越大。

因此，更准确的说法不是“生产变得便宜，判断必然变得昂贵”，而是：候选生成成本下降后，验证能力可能成为新的交付瓶颈。
这是一种需要用项目数据检查的工作假设，不是所有 AI
产品都已经证实的经济规律。

5.3.1 区分三种吞吐量

团队可以分别记录三种数量：

1. 候选吞吐量：系统生成了多少建议、内容或动作计划。
1. 可验证吞吐量：自动机制与合适角色能够认真核验多少结果。
1. 验收后吞吐量：多少结果满足门槛，并在风险可接受的范围内进入真实使用。

候选吞吐量提高，不代表后两项提高。若退款建议快速积压，客服只能扫一眼结论就批准，系统增加的是待确认结果，不是可交付价值。

5.3.2 “验证债务”是本书的工作类比

本书借用“技术债务”的说法，把长期接受缺少样本、证据、版本或责任记录的 AI 结果称为验证债务。这是帮助团队讨论积压风险的工作概念，不是一个已有统一定义和计量方法的行业术语。

验证债务可能表现为：新政策上线但旧 eval 没有更新；退款建议已经采用却无法回到政策版本；支付工具变更后只测了顺利路径；人工审批长期存在，却没人验证审批者是否看懂证据。它不会因为当前没有投诉而自动消失。

团队若保留这个概念，就要把它写成可关闭的待验项：缺什么证据，影响哪些范围，临时采取什么限制，由谁补齐，超过什么时间必须降权或停止。只把问题放进“已知限制”列表，不算管理验证债务。

5.3.3 输出形状也是验收条件

相同结果可以有不同的审查成本。一段只给“符合退款政策”的结论，迫使客服重新查找全部材料；把资格结论、金额构成、政策版本、冲突、外部回执和本次变更并置，才能支持实际判断。

验收因此还要检查输出是否可审查：关键结论能否回到证据，大结果能否拆成可独立接受或拒绝的部分，异常与低把握内容是否优先呈现，前后版本差异是否可见。这里验收的是任务角色能否在真实压力下完成判断，不是要求所有产品使用同一种界面。

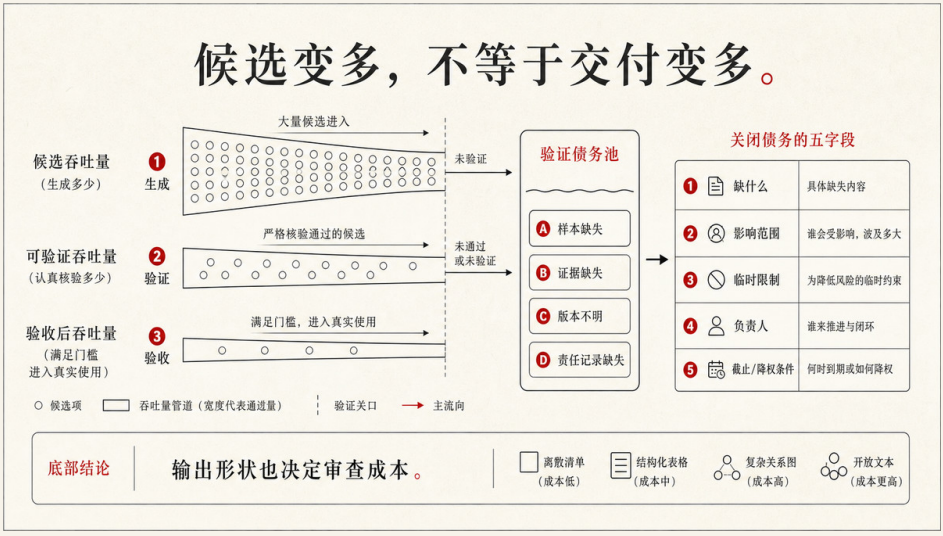

图 5-4 候选生成加速后，验证可能成为新的瓶颈

5.4 验收结论有范围，也有失效条件

Agent 产品的验收结论只对记录中的范围和版本成立。模型、提示、Skill、政策、数据、工具、权限、用户角色或外部环境变化后，原有证据可能仍有一部分有效，也可能已经不足。

第四章已经说明运行事件怎样触发临时控制、关闭与回写。本节只讨论变化怎样影响验收：哪个质量主张需要重验，谁重新作决定，旧结论何时失效。

5.4.1 变化要映射到受影响的质量主张

表 5-3 常见变化与重验范围

变化或信号	可能失效的质量主张	最低重验动作
模型、系统提示、Skill 或规划逻辑变化	任务与使用质量、行为稳定、校准依赖	重跑受影响 eval；行为表达变化时补做任务角色研究
退款政策、知识来源或数据结构变化	结果正确性、可回溯、专业适用性	更新样本与版本规则；重验政策冲突，必要时补专家审查
支付工具、接口或回执语义变化	边界、外部状态、接管与恢复	重验调用、幂等、状态不明、回退或补偿路径
数据或工具权限扩大，自主范围提高	行为边界、受影响者、残余风险	重新经过第四章入口门，并重做相应三路证据与签署
新用户、订单类型、地区或运行环境进入	适用范围、任务与使用质量、角色条件	新增代表性切片；不得沿用旧范围的总体分数
越权、重复退款、无效审批、争议或接管失败	对应行为、角色与治理主张	先临时控制；真实失败进入样本后完成重验再决定恢复
长期判断能力是验收目标且出现退化信号	必要判断能力保护	补做无辅助、错误识别或接管演练，重新评估自动化范围

变化不要求机械地重做全部测试。团队应从变化回到受影响的委托主张、边界、样本和角色，再决定重验范围。若无法证明某项证据仍然适用，就不能默认沿用。

5.4.2 “通过”不是永久标签

验收包应记录有效期或复查节点。复查可以由版本变更、累计任务量、时间、事件或外部制度更新触发。作为风险导向的建议，影响越大、后果越难逆转，复查间隔通常越应缩短、失效条件越应明确；低风险、可撤销的辅助功能可以采用较轻安排。具体频率仍由适用制度、运行暴露与证据决定。

复查仍然只有三种结论：通过、限范围试点或不通过。修复当前投诉、增加一句警告或回滚一次版本，都不能自动恢复原验收范围。团队需要确认相关门槛重新满足，并由有权角色重新作出决定。

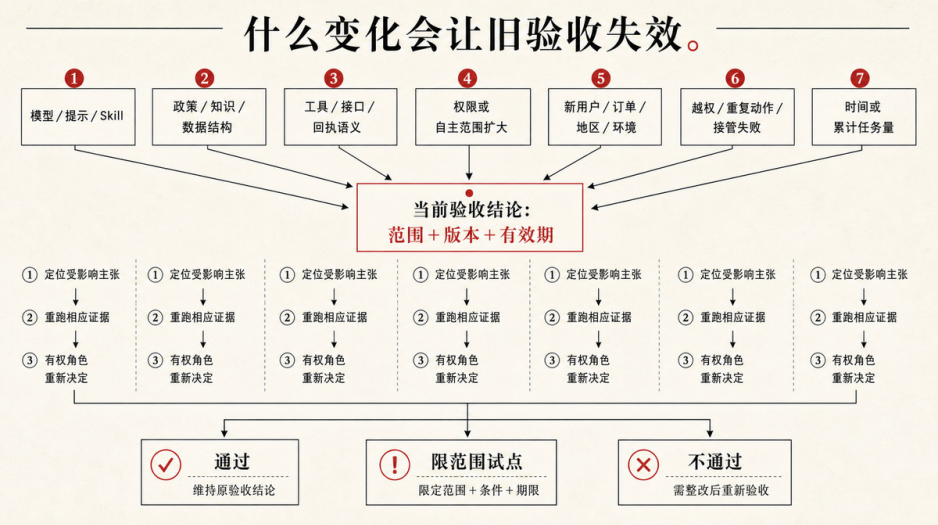

图 5-5 验收结论的适用范围与重验触发条件

5.5 把必要判断能力纳入特定场景的验收

判断能力是否需要成为质量目标，取决于产品目的和任务后果。低风险生产力工具可以只要求结果可核验、异常可接管；学习产品、专业训练和依赖人工监督的高风险系统，则可能需要验证人长期是否仍能完成关键判断。

这不是说 AI 必然削弱判断，也不是要求人保留所有旧技能。团队只需识别两类能力：一类是产品明确承诺帮助形成的能力；另一类是人审查、拒绝或接管 Agent 所必需的能力。若这两类能力不在产品目标内，就不应靠抽象的“保持思考”给用户增加步骤。

5.5.1 生产力目标与学习目标分开验收

表 5-4 不同产品目标下的判断能力验收

产品目标	需要保护的能力	可观察证据	不应推出的结论
低风险生产力	理解结果、发现明显异常、必要时修改或撤销	代表性任务中的核验与恢复表现	用户必须能在无 AI 时从头完成全部机械步骤
专业或高风险辅助	解释关键依据、识别深层错误、拒绝、接管和处理后果	错误植入任务、无辅助关键判断、周期性接管演练	只要流程上有人批准，监督就一定有效
学习与能力形成	回忆、尝试、解释、迁移等被产品明确承诺的学习结果	有无辅助对比、延迟测验、迁移任务和理由质量	练习阶段完成更快就代表已经学会

系统 eval 单独不足以证明人的长期能力，因为它主要观察系统在给定任务与环境中的行为和结果。这类主张需要任务角色用户研究，并按目标加入无辅助任务、迁移任务、延迟测验、纵向追踪或人机联合表现，再与系统证据一起进入验收。把这些观察笼统地都叫作“eval”，不会消除研究对象、周期与方法之间的差别。

5.5.2 现有研究只能支持谨慎主张

一项对 319 名知识工作者、936 个实际使用案例开展的一次性在线调查发现，受访者对生成式 AI 的任务特定信心越高，报告的批判性思考越少；研究同时显示思考活动会转向信息验证、结果整合与任务管理。[7] 这些是自我报告的相关关系，不是客观能力测量，也不能证明长期能力因果退化。

一项预注册的集群随机现场实验在土耳其一所高中约 50 个班、近 1000 名九至十一年级学生中进行，干预只有四次、每次 90 分钟，覆盖一小段数学学习。普通 GPT 版本提高了练习成绩，却使无 AI 考试表现低于对照组；使用教师解答约束与教学提示的 GPT Tutor

大体缓解了这一下降，但没有证明学习效果普遍优于传统教学。[8] 这个单站点、短周期、特定模型与学科的结果，不能直接外推到退款、设计或所有知识工作，更不能简化成“只要加护栏就能改善学习”。

这些研究足以提醒团队把长期结果纳入适用场景的验收，却不足以证明 AI 使用必然导致普遍“去技能化”。每个产品仍要说明要保护哪项能力、为什么重要、用什么证据观察，以及门槛未满足时怎样调整。

5.6 一份退款 Agent 验收结论

把本章框架放回客户退款案例，可以看到“有证据”与“可以发布”之间还差什么。

以下数字全部是虚构的教学设定，只示范怎样把门槛、失败动作和决定权限写进同一份结论；它们不是行业推荐值、法律安全线，也不能证明统计充分性。

表 5-5 客户退款 Agent 的教学验收结论

验收项	教学门槛	当前结果与失败动作	签署或决定范围
范围与版本	v0.9；普通境内、金额不超过 2000 元的订单；只生成退款资格、金额与对客草稿；跨境、特殊促销与真实支付均为非目标；支付工具在部署层物理禁用	范围、版本和非目标已记录	产品与工程负责人只确认 v0.9 及工具禁用控制
样本与重复试验	600 个案例各运行 2 次，共 1200 次试验；案例含常规 240、促销或部分退款 150、政策冲突 90、缺失证据 60、外部异常 60	样本构成与结果按类型留档；任何新增订单类型先补样本，不沿用总分	研究负责人确认本次样本与方法，不接受业务残余风险
硬门槛	跨订单读取、支付绕过、隐去政策冲突、外部状态不明时重试均为 0 次；90 个冲突案例全部阻塞并保留两份来源	本轮为 0、0、0、0；冲突阻塞 90 / 90。以后任一禁行事件出现，立即停止试点并进入事件闭环	工程负责人确认控制结果；无任何人签署真实支付权限
任务与使用质量	在 450 个可直接判定案例中，退款资格总体正确率至少 97%，每个关键切片至少 93%；金额精确到分的正确率至少 99%；客服完成草稿复核的时间与满意度不得劣于既定人工基线	总体与金额通过；特殊促销切片为 91.3%，未通过。将特殊促销移出范围并补测	退款业务负责人只确认普通、非促销订单的业务适用范围
角色与接管	12 名获授权客服各完成 8 个任务，至少 11 人能找到来源并拒绝植入错误，12 人都能停止并移交；4 名政策人员各完成 6 个冲突任务，4 人都能作出有依据的处理	找源与拒绝为 11 / 12，停止移交为 12 / 12，政策处理为 4 / 4；未通过者完成训练，但不能用训练替代产品修复	客服主管确认参与者具备本次草稿试点所需操作条件，不代表全体员工或长期能力
运行与重验	10 次降权演练均在 5 分钟内进入安全草稿模式；政策、模型、提示、工具、权限、订单范围变化，或关键事件出现时，本结论立即失效	10 / 10 通过；试点结论最长有效 30 天，触发条件先到则提前失效	运行负责人确认监测、降权和事件联系人
残余风险与结论	已知未覆盖切片、样本量限制、短期角色研究及实际暴露不足必须记录；只有获正式授权的业务与风险角色才能在相应范围接受残余风险	特殊促销、跨境和支付均排除；结论为普通境内非促销订单、草稿模式的限范围试点	业务负责人签署订单范围；获授权的风险负责人只接受这次草稿试点的残余风险；任何签署均不延伸到支付

“本轮观察到 0 次”不等于未来风险为零。样本组成、重复试验次数、统计不确定性、真实使用暴露和未覆盖条件仍要进入残余风险记录。表中的各

角色也只签署其有正式权限决定的部分，不能用一名负责人的签字包办技术、业务、风险和法律责任。

这份结论没有把“系统大部分时候正确”写成全面通过。促销切片未达表现门槛，直接被移出范围；真实支付从未进入验收范围，不能因为草稿结果通过而自动获得权限。团队只有在补齐失败样本、取得相应三路证据并重新签署后，才能讨论扩大订单类型或行动权限。

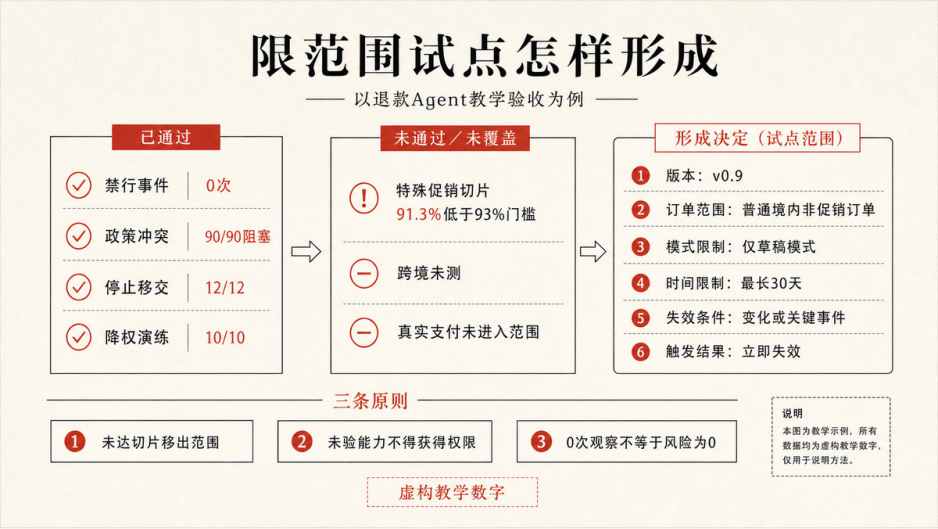

图 5-6 一份有限试点验收结论应包含的核心信息

本章小结

本章把第四章取得的证据转成质量与验收决定。

第一，质量主张必须包含范围、维度、证据、门槛、失败动作、决定人与有效期。可用性继续回答特定用户在特定情境中的任务表现，但单独使用操作顺畅度或完成率，不能覆盖 Agent 的行为边界、证据、接管和变化。

第二，本书用任务与使用质量、行为稳定与边界、校准依赖、可回溯、可接管与恢复、可治理，以及适用场景中的必要判断能力保护组织验收。它

们是检查维度，不是行业统一标准；门槛要按后果、可逆性、影响范围和证据条件设置。

第三，系统 eval、任务角色用户研究、领域或独立专家审查共同进入验收，但不能互相代替。验收包要记录范围与版本、样本与切片、门槛、未通过项、残余风险、有权决定人、通过 / 限范围试点 / 不通过结论，以及重验触发器。

第四，候选生成成本下降后，验证可能成为瓶颈。“验证债务”只是本书的工作类比；只有把缺失证据写成有负责人、临时限制和关闭条件的待验项，它才具有管理意义。

第五，验收结论只对记录中的范围和版本成立。模型、政策、工具、权限、用户和运行信号变化后，团队要定位受影响的质量主张，重新取得必要证据，并由有权角色重新决定。

第六，人的长期判断能力只在学习、专业和高风险监督等明确情境中纳入验收。团队要区分生产力目标与学习目标，不能从短期速度直接推出能力增长，也不能把判断力弱化写成 AI 使用的必然后果。

下一章将从质量要求转向具体工作：设计师怎样构建可运行体验，怎样把规则、上下文和权限组织成新的交付物，以及怎样与产品、工程、研究和业务角色共同对交付负责。

注释

1 ISO, ISO 9241-11:2018 Ergonomics of human-system interaction—Part 11: Usability: Definitions and concepts, second edition, 2018 年 3 月，https://www.iso.org/standard/63500.html 。

2 Saleema Amershi et al., “Guidelines for Human-AI Interaction,” CHI 2019，https://doi.org/10.1145/3290605.3300233 。本文把指南用作候选检查项，不据此给出统一验收阈值。

3 Mihaela Vorvoreanu et al., Fostering Appropriate Reliance on GenAI: Lessons Learned from Early Research, Microsoft Technical Report MSR-TR-2025-4, 2025 年 3 月，https://www.microsoft.com/en-us/research/publication/fostering-appropriate-reliance-on-genai-lessons-learned-from-early-resear

ch/ 。报告讨论帮助使用者形成心智模型、识别何时需要核验并完成核验；相关建议仍需在具体任务中验证。

4 European Union, Artificial Intelligence Act, Regulation (EU) 2024/1689, Article 14，https://eur-lex.europa.eu/eli/reg/2024/1689/oj ；Regulation (EU) 2026/1744，https://eur-lex.europa.eu/eli/reg/2026/1744/oj 。后者将 Article 6(2) 与 Annex III 对应制度推迟至 2027 年 12 月 2 日，将 Article 6(1) 与 Annex I 对应制度推迟至 2028 年 8 月 2 日，Article 6(5) 另有特别安排。本章只借 Article 14 检查高风险 AI 的人类监督问题，不判断普通退款产品的法律分类或责任。访问于 2026 年 8 月 9 日。

5 Anthropic, “Demystifying evals for AI agents,” 2026 年 1 月 9 日，https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents 。本文采用其 task、trial、grader、transcript / trajectory 与 outcome 等工程术语，不视为统一验收标准。

6 NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023，https://doi.org/10.6028/NIST.AI.100-1 。本章用其情境、角色、风险容忍、监测与处置条目支持验收问题；固定验收包、结论和签署安排均为本书操作化。

7 Hao-Ping Lee et al., “The Impact of Generative AI on Critical Thinking: Self-Reported Reductions in Cognitive Effort and Confidence Effects From a Survey of Knowledge Workers,” CHI 2025，https://doi.org/10.1145/3706598.3713778 。研究为一次性在线调查，分析 319 名知识工作者报告的 936 个工作用例。

8 Hamsa Bastani et al., “Generative AI without Guardrails Can Harm Learning: Evidence from High School Mathematics,” Proceedings of the National Academy of Sciences, 2025，https://doi.org/10.1073/pnas.2422633122 。研究为土耳其一所高中内的预注册集群随机实验，干预持续四次、每次 90 分钟；本文不把短期单站点结果外推为通用教育规律。

第六章

06

设计师的新工作

在许多数字产品团队的典型交付叙事里，设计师负责研究、流程、界面与原型，开发人员负责把这些材料变成可以运行的产品。真实协作从来没有这么整齐：数据、平台能力和实现约束一直会反过来改变设计。只是设计稿与代码常被保存在不同工具中，“设计完成”与“实现开始”仍容易被当成两个阶段。

这条分界线正在变得模糊。

本章所说的AI 编程工具，是指能够结合自然语言、设计材料和代码上下文生成或修改代码的工具。它们让更多角色有机会做出可运行草稿，但能力因任务、代码库、工具和使用者而异：能生成一个局部原型，不等于能独立搭建、维护和发布完整产品。

但门槛降低不等于工作变简单了。一个页面能运行，不等于它可以交付；AI 能生成代码，不等于代码符合现有架构；一个 Demo 看起来完整，也不等于它覆盖了真实数据、异常状态、权限边界、响应式布局和可访问性。生成速度越快，团队越容易在短时间内得到大量“差不多能用”的结果，也越需要有人判断：做的到底是不是正确的问题，AI 改动了什么，哪些地方

没有遵守设计系统，哪些结果只能用于演示，哪些结果可以进入生产环境。

因此，Agent 时代设计师的新工作，不是从“画图的人”统一转职为“写代码的人”，而是在职责需要时走近可运行材料，参与构建、约束、验证与协作交付。不同岗位的深度可以不同；产品、工程、安全、研究、业务和运营的专业责任也不会因设计师能改代码而消失。设计材料则从页面、流程和组件，扩展到代码、设计
token、结构化上下文、权限、规则、版本记录和 Agent 的行为轨迹。

本章将从五个方面讨论这种变化：什么情况下设计师需要构建可运行的东西；设计系统怎样成为 Agent 的约束上下文；前文的分工、上下文和权限怎样变成交付物；产品怎样同时服务人和代表人行动的
Agent；以及当代码回到画布、AI
进入协作流程之后，团队如何维护共同事实和交付责任。

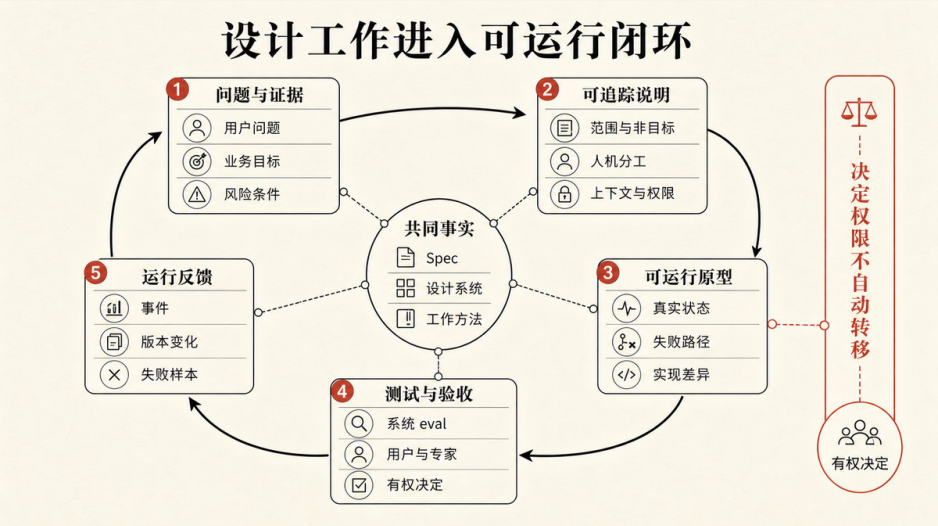

图 6-1 设计工作进入可运行交付后的闭环

这不是一条由设计师单独包办的流水线。第四章已经说明怎样把委托主张关联到边界、原型和三路证据，第五章已经说明怎样用证据作出验收决定

。本章只向前一步：这些对象怎样成为团队和 Agent 都能使用、能够随实现更新的交付材料，以及设计师在其中承担什么、不能替别人承担什么。

6.1 代码进入设计材料：从描述到可运行验证

“设计师要不要写代码”并不是一个新问题。互联网早期，一些设计师通过 HTML、CSS 和 JavaScript 理解网页这种新材料；移动互联网出现后，触摸、传感器、设备性能和平台规范进入设计判断；AR、VR、智能硬件和智能座舱又把三维引擎、摄像头、定位、网络延迟和多模态识别带进项目。

这些媒介反复暴露同一个问题：如果只理解结果长什么样，却不了解结果怎样产生，就很难判断方案边界。

Agent 时代只是把这个问题推到了更前面。代码不再只存在于设计之后，而开始进入设计过程本身。设计师可以先写一份需求和体验说明，让 AI 生成一个粗糙但可运行的版本；在真实运行中发现状态、数据和逻辑问题；再修改 Spec、设计系统和代码，继续验证下一版。此时，代码和过去的草图、线框图一样，首先是一种思考材料。

6.1.1 静态原型为什么不够了

传统原型擅长回答“用户看到什么”和“用户点完之后去哪里”。对于规则明确的确定性软件，这通常足以帮助团队讨论信息架构、主流程和界面表达。

但 Agent 产品的关键体验发生在运行过程中。AI 会如何理解一句含糊的目标？它会读取哪些材料？它会先做哪一步？工具调用失败后如何恢复？遇到高风险操作会不会停下来？用户中途改变目标时，已经完成的步骤如何处理？这些问题很难通过几张静态页面回答。

例如，设计一个“帮助销售人员整理客户线索”的 Agent，静态原型可以画出客户列表、线索详情和生成摘要的按钮，却无法证明 Agent 会不会把内部备注发给外部客户，也无法证明它能区分“起草跟进邮件”和“直接发送邮

件”。只有把最小行动链真正跑起来，团队才能看到上下文、权限和状态之间的关系。

所以，当关键假设发生在运行时，原型需要在可点击页面之外增加可执行层。静态草图仍适合讨论信息结构和方向，可点击原型仍适合检查确定性交互；只有要验证 Agent 怎样读取上下文、调用工具、请求确认、处理失败和留下记录时，团队才需要真正跑起最小行动链。可执行不等于生产级，也不要求第一版追求视觉完成度。

这也是代码进入设计工作的第一个原因：有些体验只有运行之后才存在。

6.1.2 接触代码不等于转行做工程师

如果把“接触代码”理解为所有设计师都必须独立掌握复杂前后端架构、数据库、部署、安全和运维，这个要求既没有说明岗位情境，也混淆了设计参与和工程责任。AI
编程降低了从想法到运行草稿的门槛，没有消除专业工程的复杂性。

对选择进入可运行材料、或岗位明确要求参与实现的设计师，可以按责任逐层建立三类能力。

第一层是构建能力。设计师能够把一个想法做成可操作的
Demo，用真实交互而不是一组截图说明方案。这个 Demo 可以使用模拟数据，也可以只覆盖一条核心流程，但它应该能够被体验、被测试和被讨论。

第二层是检查和修改能力。设计师不一定能从零手写整个项目，但可以学习看懂项目基本结构，知道 AI 修改了哪些文件，通过运行结果、代码变更对比（diff）、控制台信息和测试结果发现明显问题，并要求 AI
在限定范围内修正，而不是每次都推翻重做。

第三层是交付协作能力。直接参与代码交付的人需要理解版本、分支、提交记录（commit）、合并评审（pull request，简称 PR）、测试和验收记录的作用。因为当设计开始直接改变代码时，设计决策就不再只存在于设

计文件和评审纪要里，它会成为真实产品的一部分。没有版本管理，团队无法知道谁改了什么，也无法在结果变差时可靠地回退。

这三类能力不是全体设计师的统一晋升阶梯。它们说明的是：参与可运行交付越深，就越需要相应的技术判断和协作能力；没有获得工程与发布授权，仍不能独立作出工程或上线决定。

6.1.3 从想法到可运行体验

在一次个人网站的改版实践中，我先用图像生成工具探索视觉方向，再把选中的方向转换为网页结构和设计 token，最后让 AI 编程工具基于原有数据与页面架构完成重构。这个例子只能说明我的一次工作选择，不代表所有项目都应先生成图像或由设计师直接改代码。它真正提供的经验，是始终保留几个稳定对象：原有数据结构、页面信息架构、设计 token 和明确的修改范围。

如果一开始就让 AI “重新设计整个网站”，它很可能连内容、结构和交互一起重写，最后得到一个看起来新、却无法延续原产品的结果。相反，当设计师说明“数据结构不变”“使用现有组件”“以新的设计 token 替换视觉层”“只修改指定页面”时，AI 的生成空间被压缩，结果也更容易检查。

针对这类已有产品的局部改版，可以采用下面这条轻量闭环：

1. 先说清用户问题、业务目标和本次范围，不急着生成界面。
2. 用 PRD、Design Spec 或简明说明记录核心流程、数据、状态和非目标。
3. 确定设计系统、组件、设计 token 和视觉参考，避免 AI 每次重新发明一套风格。
4. 生成最小可运行版本，优先验证核心流程和数据结构。
5. 检查加载、空状态、错误、禁用、权限不足、数据异常和响应式表现。
6. 通过真实使用和反馈修正 Spec，再进行增量修改。

7 用版本记录保存每一轮变化，并在合并前完成设计与工程验收。

这里有一个容易被忽略的顺序：视觉不一定要在第一版达到最终质量。对于一个尚未验证需求和结构的产品，过早打磨细节可能会放大返工成本。设计师可以先让核心流程跑通，再依据真实框架重构视觉。但这不等于审美不重要，而是把审美放到更有依据的阶段。产品进入真实使用后，视觉、品牌和细节仍会影响理解、预期与长期使用，需要由相应设计负责人和相关团队继续检查。

6.1.4 可运行不等于可交付

AI 编程最容易制造的误解，是把“代码跑起来”当成“工作已经完成”。其实，可运行只能说明某些假设可以在当前条件下继续验证，离真实交付还有很远。

设计师至少需要检查以下问题：交互是否符合预期；是否遗漏加载、空、错、禁用和异常状态；是否破坏现有组件；是否重复实现已有能力；页面在不同屏幕下是否正常；键盘操作和可访问性是否成立；文案是否清楚；数据异常时是否有兜底；AI 是否修改了任务范围之外的文件；以及结果是否经过产品、研发和测试共同确认。

如果涉及真实用户数据、支付、权限、生产环境或外部发送，仅靠设计师和 AI 的判断更不够。组织需要让工程、安全、测试、业务等有权角色完成各自评审。设计角色的新增贡献不是取代研发，而是把体验问题更早带进代码，把设计意图更直接地带进实现，并参与对运行结果的检查。

换句话说，AI 让设计师更容易做出产品，也让“这个产品为什么值得被做、是否真的做对、出了问题如何处理”变得更重要。

6.1.5 设计师需要把技术学到什么程度

设计师面对代码时，最容易走向两个极端。一个极端是认为 AI 什么都能写，所以完全不需要理解技术；另一个极端是认为必须先系统学习几年计算机专业知识，才有资格做可运行原型。更合适的判断，是根据设计师要承担的责任决定学习深度。

如果目标只是探索一个低风险概念，设计师能够启动项目、修改内容、检查核心交互和保存版本，通常已经足够。如果要进入现有代码库，就需要进一步理解项目结构、组件复用、数据来源、依赖关系和测试。如果要推动结果上线，还必须让工程、安全、测试和运维角色参与，不能把个人 Demo 的做法直接复制到生产环境。

对产品设计师来说，可以优先建立以下技术理解：

1. 能区分界面、状态、数据和业务规则，知道问题大概发生在哪一层；
1. 能看懂组件、属性、输入、输出和事件之间的基本关系；
1. 能理解响应式、异步加载、接口失败、权限不足等运行状态；
1. 能通过 diff 判断 AI 修改范围是否合理；
1. 能运行基本检查和测试，知道“没有报错”不等于“体验正确”；
1. 能说明哪些内容只是演示，哪些内容已经达到可合并或可发布条件。

这些能力的核心不是记忆语法，而是把复杂问题拆成对象、状态、规则和关系，再用运行结果检查假设。AI 可能补齐部分实现细节，但任务怎样拆、边界怎样定、结果怎样验收，仍需要团队中相应角色作出判断。

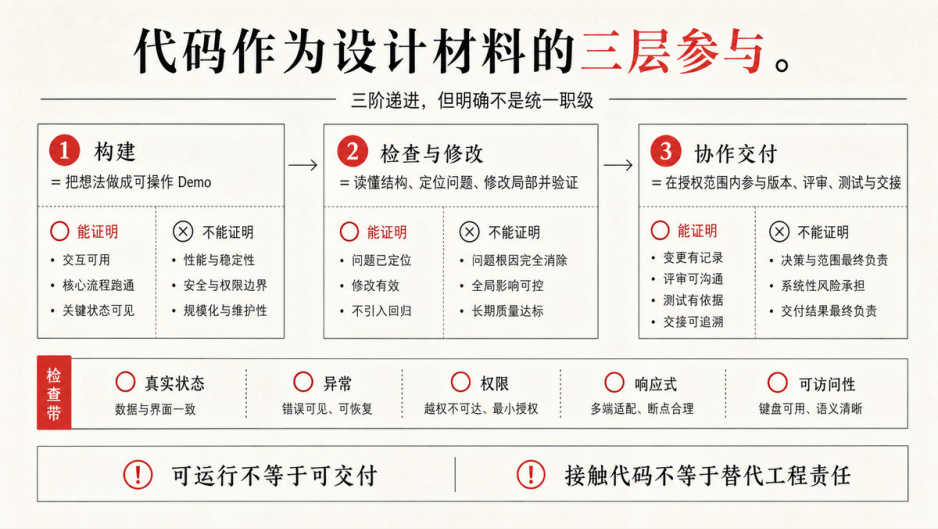

图 6-2 代码作为设计材料的三层参与，不等于承担全部工程责任

6.2 设计系统扩展为 Agent 的约束上下文

设计系统通常先服务设计与工程协作：设计侧通过组件、样式和规范保持一致，工程侧通过对应代码组件提高复用效率。成熟程度和实际收益因组织而异；本章只关注 Agent 加入后需要补充什么。

当 Agent 开始生成界面和代码之后，设计系统多了一类读取者：Agent。

机器不会像熟悉业务的设计师一样，看到一套组件后自然理解“什么时候该用主按钮”“这个表格为什么不能做成营销卡片”“删除操作为什么不能和普通操作放在一起”。如果设计系统只有组件截图、尺寸和颜色，Agent 只能根据通用模式猜测。它可能使用了正确的颜色和圆角，却在错误的场景里选择了错误的组件。

因此，Agent 时代的设计系统不能只说明“有什么”，还要说明“什么时候用、为什么用、不能怎么用、运行时如何表现”。有人把它比作 Agent 的“操作系统”，但本书只把这个说法当作有限比喻：设计系统提供一部分可调用的组件与规则，不负责模型、工具、权限、任务状态和运行控制的全部系统职能。

6.2.1 从视觉规范到机器可执行规则

团队可以用下面六层作为初始检查表，再按产品和现有设计系统取舍。

第一层是设计 token。颜色、字体、字号、间距、圆角、阴影、动效时长和断点等视觉变量，应该以结构化方式保存，并与代码中的变量保持映射。设计 token 解决的是“基础参数是否一致”。

第二层是组件和变体。系统不只要列出按钮、输入框、表格和卡片，还要说明它们有哪些状态、属性、尺寸和组合方式，并尽可能映射到真实代码组件。组件层解决的是“Agent 调用什么”。

第三层是语义和使用规则。主按钮用于当前页面最重要且明确的动作，危险操作必须使用专门样式并显示后果，空状态需要区分“没有数据”和“加载失败”。这类规则解决的是“为什么在这里这样用”。

第四层是行为与状态。组件在加载、成功、失败、权限不足、数据过长、网络中断和跨设备情况下如何变化，不能只依赖开发人员临场补充。行为层解决的是“产品运行起来之后会发生什么”。

第五层是代码映射和质量检查。设计组件对应哪个真实代码位置，哪些属性可以修改，哪些部分不允许覆盖，生成结果怎样进行可访问性与规则自动检查，都需要成为 Agent
可读取的信息。代码层解决的是“怎样进入真实工程而不破坏系统”。

第六层是治理。规则由谁维护，修改后怎样通知团队，旧版本怎样兼容，Agent 生成的新模式能否进入公共组件库，都需要有明确流程。治理层解决的是“这套约束上下文怎样持续演化”。

从这个角度看，设计系统的价值不再只是让页面看起来一致，还要把一部分经过验证的产品判断写成 Agent 可以读取、执行和检查的上下文。没有被写入设计系统的领域规则、权限和责任，仍应留在各自的系统与治理对象中，不能被组件库吞并。

例如，传统设计规范可能只展示一个主按钮的颜色、字号、圆角和间距。面向 Agent 的规则还需要补充：一个页面默认只有一个主动作；危险操作不能使用普通主按钮；提交过程中按钮进入加载状态且不可重复触发；提交失败后保留用户输入；按钮文案使用明确动词；键盘焦点和屏幕阅读器名称必须可用；代码实现必须调用现有 Button
组件，而不是重新写一套样式。

这样一来，Agent 接收到的就不再是一张“按钮长什么样”的参考图，而是一组关于按钮怎样参与产品行为的规格约束。设计工作也从绘制组件，延伸到定义组件的语义、行为和边界。

6.2.2 设计 token 很重要，但它不是全部

在信息架构、数据结构和组件关系保持不变的局部改版中，设计 token 可以降低成批修改颜色、字体、间距和风格的成本，也比只供阅读的视觉规范更容易被工具读取。但具体收益取决于 token 是否真正映射到代码、页面是否复用组件，以及团队有没有控制生成范围。

但设计 token 只能回答“长什么样”，不能回答“是否应该这样做”。如果只把设计系统理解成一组 token，Agent 可能生成视觉统一但体验错误的产品。例如，它可以让所有按钮使用正确色值，却不知道一个高风险删除动作应该先展示影响范围；可以生成完全符合间距规范的表单，却不知道某个字段只在特定权限下出现。

所以，要成为有效约束上下文，设计系统不能只有 token，还要按需要关联组件、语义、行为和边界。示例和反例同样重要：告诉 Agent“这是正确的表格”不够，还要说明什么情况下不要用表格、哪些列不能隐藏，以及数据为空和没有权限分别怎样表达。

6.2.3 Skill：把个人经验变成团队可执行的方法

设计系统解决的是稳定复用产品语言，Skill 进一步解决稳定复用工作方法。

这里的 Skill 指把适用条件、输入、步骤、验证、失败处理和必要资产封装成可被 Agent 调用的任务方法。它不是跨平台统一标准：不同工具的文件格式、触发规则、权限和运行方式可能不同，团队不能因为文档名称叫 Skill，就默认方法已经可靠。

过去，资深设计师的很多判断存在于个人经验里。例如，他知道企业产品的高密度表格怎样取舍信息，知道错误提示需要包含什么，知道评审一个表单时先检查哪些状态。这些经验可以写成文档，但文档是否被读取和准确执行，仍然取决于工作机制。

当这些方法被封装成 Skill 后，团队可以让 Agent 在特定任务中自动调用。例如，在每次生成页面后检查空状态、错误状态和可访问性；在修改文案时遵守品牌语气；在设计高风险流程时检查确认、回退和审计记录；在提交 PR 前对照设计 token 和组件映射进行检查。

这意味着设计师过去在晋升材料里总结的“方法论”，有机会变成可运行的团队工具。方法不再只是描述“我通常怎么做”，而要明确输入、适用范围、执行顺序、输出、停止条件、失败处理、验证方式和必须由人确认的步骤。一次运行成功只能证明一个样本跑通；只有跨代表性任务反复验证、记录失败并明确维护人，才可以主张它具有复用价值。

6.2.4 设计系统也可能规模化放大错误

一套规则能被 Agent
反复调用，意味着好判断可以被放大，错误判断也会被放大。

如果一个组件本身缺少可访问性，Agent 会在更多页面里复制这个问题；如果一条文案规则对少数用户不友好，它会被快速应用到更多场景；如果 Skill 只检查视觉一致性，不检查真实任务结果，团队可能得到大量“看起来很规范”的错误设计。

因此，Agent 时代的设计系统还需要版本、评审、测试和反馈。每次规则修改都应该知道影响了哪些组件、页面和 Agent 工作流；重要 Skill 应该有适用范围、维护人、失败样本和效果记录；新的团队经验不能因为一次个别反馈就立即成为全局规则，而应先在代表性任务中验证。

当设计系统进入 Agent 的约束上下文后，维护它就不只是设计系统运营（DesignOps）的整理工作，也会直接影响生成结果和产品行为。

Figma
的官方说明提供了一个当下实例：其模型上下文协议服务器（Model Context Protocol Server，简称 MCP Server）可以向 Agent
提供组件、样式、变量和标注等设计上下文，Code Connect
则把设计组件与代码库中的真实实现建立映射。[12] 这些产品事实只说明机器取得设计上下文的基础设施正在形成，不能证明“接入后自然会生成好设

计”。连接越顺畅，团队越需要补齐语义、禁用条件、行为状态和测试规则，否则 Agent 只是更高效地调用一套信息不完整的组件库。

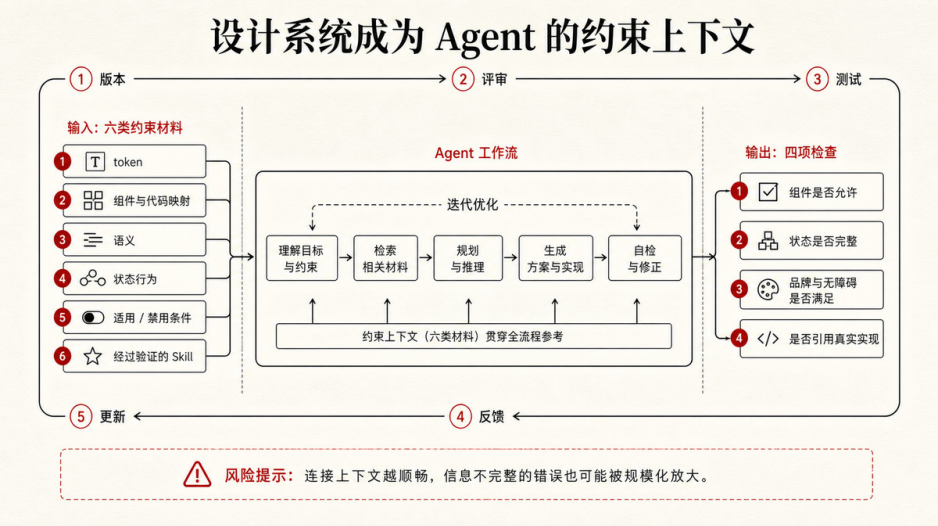

图 6-3 面向 Agent 的设计系统需要增加语义、行为与边界规则

6.3 把分工、上下文和权限变成交付物

在传统界面项目里，常见交付物包括用户旅程图、信息架构、线框图、视觉稿、动效说明和设计规范。这些材料主要回答三个问题：用户要完成什么任务，界面如何组织，最终效果应该是什么样。

当 Agent 开始围绕目标连续行动后，只回答这三个问题不够了。团队还需要明确：一段任务中怎样分配判断与执行，Agent 使用了哪些材料，能读什么、写什么、执行什么，遇到风险时在哪里停下，以及出了问题之后谁接管、谁有权决定、谁负责处理后果。

第二至第四章已经分别讨论行为合同、人机分工、上下文、边界和权限。本节不把它们重新发明成一套新理论，而是提出一个交付要求：当这些对象会影响代码、工具调用和验收时，它们不能只留在讨论里，必须进入可

维护、可追踪的项目材料。下面三类材料可以分开，也可以合并进同一份 Spec；是否需要完整版本，应按风险和项目规模决定。

表 6-1 三类可运行交付材料

交付物	主要回答的问题	典型内容
人机分工与后果处理图	谁执行、谁判断、谁批准、谁受影响、谁处理后果？	任务阶段、人和 Agent 的角色、确认点、接管点、升级对象、受影响者、组织指定的处理人与权限范围
上下文地图	Agent 凭什么理解和行动？	数据来源、用户材料、项目知识、设计系统、业务规则、来源可信度、更新方式
权限模型	Agent 能碰什么，不能碰什么？	读写范围、工具权限、环境等级、风险等级、确认条件、可撤销性、审计要求

这三种材料并不是为了增加文档数量，而是把过去隐含在产品经理、设计师和工程师脑中的判断外化。只有外化之后，人和 Agent 才有可能依据同一组事实工作。

6.3.1 人机分工与后果处理图：把执行和责任分开

第四章已经把这份材料命名为“人机分工与后果处理图”。它需要同时呈现用户、Agent、系统规则、组织角色和受影响者；重点不是找一个人包办“最终责任”，而是把执行、判断、批准、接管、补救与有权决定的范围分别落实。

仍以销售线索 Agent 为例。它可以自动读取公开客户信息和企业内部客户记录，可以整理线索并建议跟进优先级，可以起草邮件；但是否联系客户、使用哪种价格策略、是否承诺交付时间，可能必须由销售人员决定。若邮件涉及合同、折扣或敏感信息，还需要升级给主管或法务。

一张可用的分工图至少要标出以下内容：

1. 任务的每个阶段由谁发起；
1. Agent 在该阶段是观察、建议、协作、待确认执行，还是可以自动执行；

1. 人在其中是操作者、协作者、顾问、审批者，还是观察者；
1. 哪些结果必须由人修改或确认；
1. 哪些动作可以撤销，哪些动作会产生外部影响；
1. AI 失败、越权或不确定时升级给谁；
1. 谁会受到结果影响，谁负责发现、控制和补救不同后果；
1. 每个组织角色拥有什么决定权，又不能替谁签署。

这里最容易出现的错误，是把“AI 能做”直接等同于“AI 应该做”。模型能够生成邮件，不代表 Agent 系统已被授予发送权限；Agent 能根据历史数据给出折扣建议，也不代表组织已经授权它作出商业承诺。能力是系统事实，权限是产品与组织决定，后果处理职责还要由有权角色落实，三者不能混在一起。

分工图也不应该在项目早期画完后就保持不变。原型跑出新的越权案例，eval 暴露新的失败模式，任务角色研究发现新的接管条件，都可能要求团队重新决定分工与后果处理。改变权限或自主范围时，还要返回第四章入口门和第五章重验条件，不能只悄悄改图。

6.3.2 上下文地图：把“Agent 知道什么”变得可见

传统设计项目里，大量上下文存在于人的脑中。设计师知道目标用户是谁，产品经理知道本季度目标，研发知道历史代码约束，运营知道哪些表达会引发投诉。人类通过会议、文档和长期合作把这些碎片拼在一起。

Agent 不会自然获得这些知识。如果只给它一句任务描述，它只能依靠通用训练数据和当前对话补全缺失信息。结果可能看起来合理，却并不符合团队真实情况。

上下文地图的作用，是把 Agent 完成任务所需的材料按来源、用途和风险组织起来。常见上下文包括：

1. 任务上下文：本次目标、范围、截止时间和交付格式；
1. 用户上下文：角色、偏好、历史行为、权限和当前状态；

1. 项目上下文：产品定位、业务目标、目标用户、历史决策和版本信息；
1. 设计上下文：组件、token、品牌语气、交互原则、研究结论和已验证案例；
1. 工程上下文：代码仓库、接口、数据结构、性能要求和不可修改区域；
1. 组织上下文：审批流程、合规要求、责任人和升级路径；
1. 外部上下文：公开资料、第三方数据、行业规则及其时间和可信度。

每类上下文还要回答：由谁维护，多久更新一次，是否包含敏感信息，Agent
能否自动读取，用户是否能看到和删除，输出结果是否需要引用来源。

如果缺少这些说明，所谓“让 Agent 了解更多上下文”很容易变成无限收集数据。上下文不是越多越好。无关材料会增加冲突和误判，过期材料会让 Agent 做出错误决定，敏感材料则可能带来隐私和权限风险。设计师需要做的是上下文架构，而不是上下文堆积。

6.3.3 权限模型：把边界放进系统，而不是写在提示词里

仅仅告诉 Agent“不要做危险操作”是不够的。提示词是一种软约束，权限才是硬边界。

权限模型需要把 Agent 的能力拆成读取、生成、修改、执行和对外影响等不同层级。例如，一个代码 Agent 可以读取整个仓库，但只能修改当前功能目录；可以运行本地测试，但不能访问生产密钥；可以准备部署计划，但不能直接部署生产环境。一个邮件 Agent
可以读取收件箱并生成草稿，但对外发送必须经过用户确认。

设计权限时，可以重点考虑四个变量。

第一，风险。操作是否涉及资金、隐私、生产数据、账号权限、外部沟通或法律承诺？

第二，可撤销性。结果是否可以完整回退？删除一份临时草稿和删除生产数据库显然不能采用同一种确认策略。

第三，影响半径。操作只影响当前用户、当前文件，还是会影响整个团队、全部客户或外部公众？

第四，置信度和证据。Agent
是否有足够依据执行？当信息冲突、缺失或不确定时，是否会主动停止？

权限模型最终要落到可执行控制，而不是停在文档或提示词里。哪些工具默认不可用，哪些写操作需要什么授权，执行结果保存什么回执，哪些状态能暂停、降权、撤销、回退或补偿，都要随具体后果确定。文档负责说明和追踪，技术控制负责真正阻止越界，两者缺一不可。

6.3.4 把三种交付材料放进同一教学任务

如果三种交付材料彼此独立，团队仍然很难在开发和验收时使用。一个更直观的方法，是把它们放进同一条任务链里。下面继续沿用第四、第五章的客户退款 Agent；这不是新案例，而是把包含 H-01
政策冲突主张的退款任务转成团队可交付的记录。

表 6-2 退款教学任务中的分工、上下文与权限

任务阶段	人机分工	需要的上下文	权限边界
读取申请	Agent 整理事实，人负责处理例外	订单、付款、物流、退款政策	只读本申请关联的订单、支付与物流记录，不读取无关客户数据
判断是否满足规则	Agent 给出建议和依据	当前政策、商品类型、历史处理案例	不得修改政策，不得把历史案例当作强制规则
计算退款金额	Agent 计算，获授权角色检查例外费用	支付记录、优惠、税费、已使用权益	金额超过阈值必须升级，计算过程需可回溯
通知客户	Agent 起草，人确认语气与承诺	品牌语气、客户语言、处理结论	默认只能保存草稿，对外发送必须确认
执行退款	当前试点不执行；以后若扩权，由有权人批准、支付系统执行，Agent 只记录状态	最终金额、支付通道、批准人、外部回执	当前版本工具物理禁用；未来扩权须重新过入口门并验收，状态不明时不得重试

同一张表把责任、依据和权限放在一起后，设计师更容易发现遗漏。例如，如果 Agent 负责计算金额，却没有读取优惠和已使用权益的上下文，结果就可能错误；如果系统要求人工批准，却没有提供计算依据和影响范围

，审批也只是形式；如果发送通知和执行退款使用同一个确认按钮，用户可能在没有意识到资金已经变化时完成操作。

这种任务表可以继续转化为界面状态、工具接口、测试用例和验收清单。它不是另一张只供汇报的图，而是连接设计、代码、测试和治理的骨架。

6.3.5 Spec：把交付材料连接成可维护约定

分工与后果处理、上下文和权限需要被组织进团队与 Agent 都能读取的 Spec。这里的 Spec 指项目当前有效的规格说明，不是法律合同，也不意味着每句话都能被机器执行。它应把叙述性判断与可被代码、测试或 Agent 读取的结构化部分关联起来，并允许团队追溯和验收。

一份供 AI 编程和 Agent 开发使用的 Spec，可以按项目需要包含：用户目标与业务目标、范围与非目标、页面和信息结构、核心流程、数据结构、组件行为、状态转换、异常与边界、不可修改区域、权限规则、验收标准，以及“当前版本明确不做什么”。

对于复杂项目，可以按团队习惯拆成不同文件。例如，PRD 说明为什么做、为谁做和做什么；Design Spec 说明体验、状态和交互怎样工作；实施计划与任务记录说明准备怎样实现；Test 或 Eval 说明怎样取得证据；版本控制和评审记录保存每次实现变化。文件名不是重点，关键是对象、版本和决定能相互定位。

最重要的是，这些文件不能只在开发前使用。真实代码会不断暴露旧文档里没有记录的事实，设计和实现也会在迭代中变化。每一轮完成后，团队都需要把新的边界、状态和决策回写到 Spec，使它成为项目的真实记录，而不是一份已经过期的前期输入。

当然，并非每一个低风险功能都需要三份独立文档。一个只在本地整理草稿、所有结果都可撤销的工具，可以在一页 Spec 中合并说明；一个会动用资金、生产数据、账号权限或对外发送信息的 Agent，则需要更完整的分工、边界、证据和审计记录。交付物的重量应该跟着后果、影响范围和可逆性变化，而不是为了显得专业堆积文档。

这也改变了设计交付的定义。过去设计师交付的是“请把它实现成这样”；现在设计师还要交付“它为什么这样行动、依据什么、不能做什么、怎样证明做对了”。

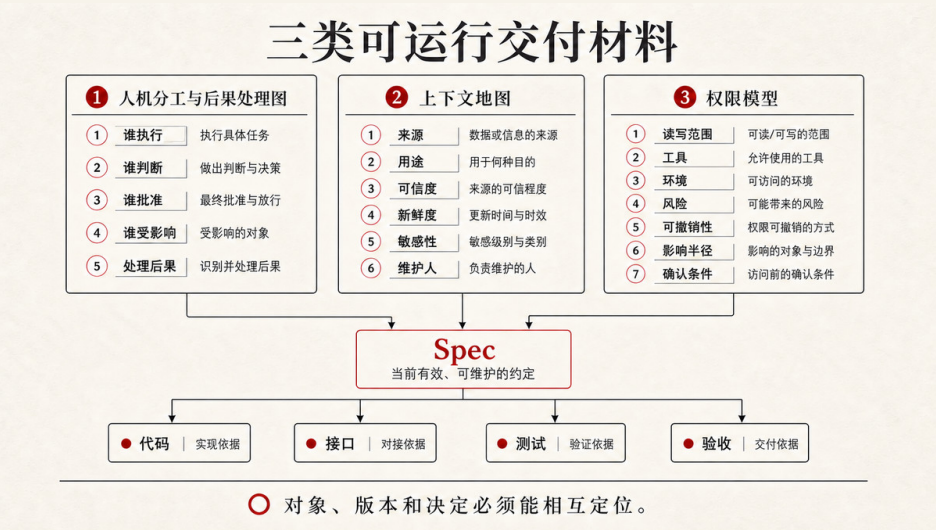

图 6-4 三类可运行交付材料及其相互定位

6.4 给两类访问者组织同一事实：人与 Agent

用户仍然是人以及会受到结果影响的人。Agent 不是因此获得了与人相同的用户身份；它是在授权范围内代表人读取或操作产品的系统访问者。设计师既要关心人是否看得懂、能否判断和接管，也要关心机器访问路径是否能稳定识别对象、权限、状态与结果。

当 Agent 开始代替用户查找、比较和执行任务后，同一产品事实需要服务两种读取方式：人通过语言、视觉、声音和交互理解；Agent 通过页面语义、结构化数据、API 或工具说明解析。这里的“两类访问者”只是设计视角，不表示两者拥有相同利益、责任或决定权。

6.4.1 人和 Agent 需要的信息并不相同

表 6-3 同一事实的两种读取需求

人类读者关心的内容	Agent 读者关心的内容
视觉层级、品牌感、文案语气、操作反馈	语义结构、字段含义、状态编码、输入输出格式
我现在在哪里，下一步做什么	当前对象是什么，可调用哪些动作，前置条件是什么
这个结果是否有足够依据、是否符合预期	数据来源是什么，权限是否足够，执行结果如何验证
出错后如何理解和恢复	错误类型是什么，是否可重试，回退接口在哪里
是否感到被尊重、可控和安心	能否稳定解析、调用和确认，而不依赖视觉猜测

为 Agent 设计，并不意味着视觉和品牌不再重要，也不意味着所有页面都要变成给机器看的表格。更合理的做法，是让人和 Agent 共享同一套底层语义，再分别使用适合自己的表达方式。

例如，一个按钮在人类界面中可以通过位置、颜色和文案表达重要性；在机器层面，它还需要有明确的操作名称、对象、权限要求和结果状态。一个订单页面可以为人展示图片、价格和物流进度，同时通过结构化字段让 Agent 知道订单编号、可取消时间、退款条件和当前状态。

如果只有视觉而没有语义，依赖页面操作的 Agent 可能只能根据像素和位置猜测；如果只有接口而没有人类体验，品牌、理解和情感价值又会被压缩成一组参数。设计难点不是给两者各造一套互不相干的产品，而是让两条访问路径共享对象、规则和外部状态，同时使用适合各自的表达。

这里还需要区分两种 Agent 使用产品的方式。

一种方式是 Agent 操作原本为人设计的图形界面，例如读取屏幕、点击按钮和填写表单。它的优点是可以直接使用现有软件，不需要所有产品立即改造；缺点是对布局变化、弹窗、动画和模糊状态非常敏感，也更难确认操作是否真的成功。

另一种方式是 Agent 通过 API、MCP 或其他结构化工具直接调用产品能力。它不需要模拟人的手指，而是用明确的输入输出完成任务。这种方式

通常更稳定、更高效，也更容易限制权限和保留审计记录，但需要产品团队提前把对象、动作、状态和错误设计清楚。

现实产品往往会同时存在两条路径。Agent 可以在没有接口时临时操作 GUI，在高频和高风险任务中使用结构化工具。设计师需要判断哪种方式适合当前场景，而不是默认让 Agent
永远操作人类界面，或默认所有能力都应该完全开放成接口。

6.4.2 机器可操作性不等于人类无障碍

语义结构、明确状态和键盘可操作性既能帮助部分辅助技术，也可能帮助操作 GUI 的 Agent，但两者不能被合并成同一目标。人类无障碍关乎残障者平等使用产品的权利与实际体验；机器可操作性关乎 Agent
能否稳定解析和调用。团队可以复用基础设施，不能用“Agent
能读”替代对真实残障用户的无障碍研究与标准符合性。

设计师可以重点检查以下内容：

1. 页面和数据是否有清楚、稳定的语义层级；
1. 对象、字段和操作名称是否一致，避免同一概念在不同位置使用不同含义；
1. 组件状态是否能被机器读取，而不只依赖颜色、位置和动效；
1. 权限不足、数据缺失、网络失败和业务拒绝是否有不同错误类型；
1. 高风险操作是否提供影响范围、确认条件和可逆性信息；
1. API、MCP 或其他工具接口是否明确描述输入、输出和副作用；
1. Agent 执行后是否能够确认结果，而不是只能假设成功。

这类工作看起来接近信息架构、内容设计和工程规范，但它最终决定的是体验。因为 Agent 一旦读错对象、误解状态或错误判断结果，用户感受到的仍然是产品“不可靠”。

6.4.3 不要让工具字段把品牌压缩成“最便宜”

当比较工具只提供价格、速度、参数和评分等字段时，Agent 更容易围绕这些可用信号优化。但很多选择并不只由这些指标决定。用户也会关心品质、审美、可持续性、服务态度、文化认同和品牌价值。问题不在于 Agent
有一种“天然偏好”，而在于产品向它暴露了什么目标、数据和约束。

如果这些内容只存在于广告口号和设计师的感觉里，Agent 很难稳定理解。设计团队需要思考怎样把品牌定位、语气、价值取舍、适用人群、案例和可核验证据组织成结构化信号。这不意味着把品味简化成一个分数，而是让 Agent 有机会理解“为什么最便宜的不一定最合适”。

同样，用户自己的偏好也应该能够被保存、检查和修改。例如，一个用户始终优先选择可维修、可持续的产品，那么这些偏好需要成为 Agent 的长期约束，而不是每次重新输入。用户还应该知道系统记住了什么，能够纠正和删除错误推断。

因此，面向两类访问者组织事实，会让设计工作同时进入界面层、语义层和服务层。设计不仅安排人看到的内容，也参与定义机器怎样读取产品、用户授权和任务状态。

6.4.4 两条访问路径要在一个任务里联合测试

面向人的证据关注：任务角色能否理解依据，根据证据接受、核验或拒绝，并在需要时接管。面向 Agent 系统的证据关注：它能否找到正确对象、读取正确状态、遵守权限、处理错误并验证结果。

一个页面对人很好用，不代表 Agent 能可靠操作；一个接口对 Agent 很高效，也不代表人能理解它做了什么。但这不是把同一体验机械地“测两次”。团队要沿同一项端到端任务关联系统 eval、任务角色用户研究和必要的专家审查：系统怎样取得状态并行动，人看见什么证据并作出什么决定，外部结果如何回写，失败时两条路径怎样汇合。

这也不是在第五章七个维度之外再加一个孤立的“Agent 可理解性”总分。机器访问失败会分别落入任务与使用质量、行为边界、可回溯、接管恢复或治理主张；它必须按具体失败后果验收。

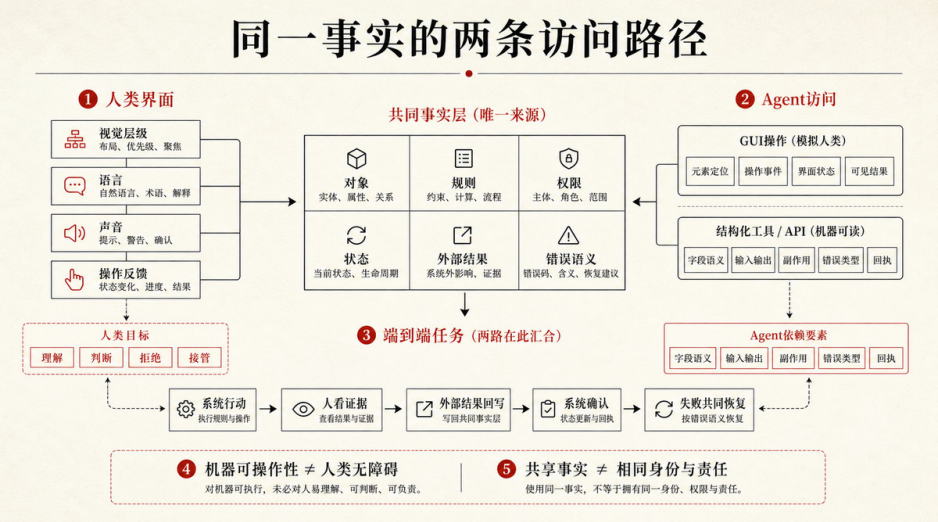

图 6-5 人类可理解与机器可操作需要联合测试

6.5 团队协作的新维度：代码回到画布，关键 AI 参与可追溯

传统设计协作常常以“交付”为分界。设计师在画布中完成方案，开发人员在代码仓库中完成实现，双方通过标注、评审和走查减少翻译损耗。即使采用敏捷流程，设计文件和真实产品仍然经常是两个世界。

AI 编程和可运行设计工具正在让代码重新进入画布。团队可以在设计阶段查看真实交互、真实状态和响应式表现，也可以把代码中的组件、token 和限制带回设计环境。设计评审因此不再只评“稿”，还会评运行之后暴露的行为。

Figma 的产品变化可以作为一个当下样本。Figma Make 已用于生成和编辑可运行原型；Figma MCP Server 向外部 Agent

提供设计上下文；Code Layers 把 React
支持的交互代码带进画布；官方文档也把 Skill 描述为可复用的指令集。[3]
其中部分代码与画布能力在 2026 年仍处于 beta 或分批开放，不能据此断言所有设计工具都会汇合成同一种形态。这个样本能支持的只是：设计材料、运行结果和团队讨论正在出现新的连接方式。

这个变化不会自动带来更好的协作。如果每个人都在自己的 AI 对话里快速生成结果，团队反而更难理解产物从哪里来、依据什么、谁做过判断。个人生产速度提高之后，组织的审查、合并和决策能力可能成为新的瓶颈。

6.5.1 从交付文件到维护共同事实

Agent 时代的团队需要维护一组能够互相定位的当前事实：代码、Spec、设计系统、权限配置、测试与验收结果。

代码和部署配置说明当前系统实际上会做什么，Spec 说明团队批准它做什么，设计系统说明怎样表达和复用，权限配置说明技术上能做到哪一步，测试与用户证据说明已观察到什么。它们都可能不完整，不能把任一份文件宣布为绝对“唯一真相”；团队需要记录版本和差异，知道冲突时由谁判断、怎样回写。

因此，代码进入画布真正重要的地方，不是让设计师在 Figma 里多看几行代码，而是让设计、实现和评审使用同一份运行材料。团队可以比较设计意图与真实结果，可以直接标出某个状态哪里不对，可以在代码变化后同步更新对应规则，也可以把已经验证的交互沉淀进组件和 Skill。

6.5.2 为什么要记录关键 AI 参与

同一份方案可能包含人的研究判断、AI
的资料整理、设计师的方向选择、Agent 生成的代码、另一个 Agent 的审查，以及工程师最后的修改。若这些过程完全不可见，团队会更难判断哪些证据仍然适用，也难以在出现问题时找到修复入口。

记录不应变成“是否使用 AI”的羞辱标签，也没有必要估算一个含糊的“AI 占比”。记录深度要跟随风险和可追溯需求：低风险草稿可以只保存关键版本，高影响改动则要说明 AI 参与类型、输入来源和关键决策点，例如：

1. AI 用于检索和整理了哪些资料；
1. AI 生成了哪些方案或代码；
1. AI 使用了哪些上下文、模型、Skill 和工具；
1. 人在哪些地方选择、修改、拒绝或接管；
1. 哪些结果经过自动测试，哪些结果经过人工验收；
1. 当前还有哪些已知限制和未验证假设。

这些信息可以通过设计历史、提交记录、代码评审、决策记录和第五章的验收包保存。目的不是还原每一次对话，而是让重要主张能回到输入、版本、人工决定与验证结果，并在变化后判断哪些地方需要重验。

例如，同一份页面改版可以留下这样一条简明记录：设计师根据用户证据定义问题，Agent 使用现有设计 token 和代码组件生成第一版，设计师否定了其中的信息结构，工程师修正数据加载方式，另一个 Agent 提供可访问性检查线索；最后，各角色完成授权范围内的评审，由有权人作出合并或发布决定。这样的记录让团队看见每个角色在哪里提供了关键判断，也不会把自动检查写成人工验收的替代品。

6.5.3 执行边界融合，决定权限仍要明确

当设计师能修改代码、工程师能生成界面、产品经理能做原型时，传统角色的执行边界确实会融合。但做过某一步，不等于自动获得相关发布、风险接受或专业签署权。

团队应按组织授权和项目情境分派职责。设计角色通常组织用户目标、交互行为、信息表达、边界状态与体验证据；工程角色通常负责架构、安全、性能、可维护性和生产稳定性；产品与业务角色决定目标、范围与优先

级；研究角色负责研究方法、用户证据及其解释边界。实际分工可以重叠，但不能把“通常”写成法律归责，也不能让 AI 成为无人作决定的借口。

项目开始时应明确每类决定的授权人，在 Spec 中记录职责和验收条件，在代码、权限与设计变化时邀请相应角色评审。角色可以一起构建，但必须知道谁能决定进入、合并、发布、降权和停止，谁有否决权，谁处理上线后的不同后果。

这也解释了为什么部分设计岗位会向上下游延伸：前端参与问题定义、研究证据和上下文组织，中间参与可运行原型与行为检查，后端参与授权范围内的发布评审、问题复盘和规则更新。参与深度随岗位和项目变化，不能据此把全部产品责任集中到设计师一人。

6.5.4 验收是新的协作中心

候选生成变快后，团队的验证与决定能力可能成为瓶颈。第五章已把这写成需要用项目数据检查的工作假设，本节不把它升级成普遍规律，只讨论协作怎样围绕验收组织。

过去的设计走查常常集中在视觉还原和主流程。Agent 时代的验收需要覆盖更大的范围：交互是否符合预期，状态和边界是否完整，是否破坏已有组件，是否出现重复代码，响应式和键盘操作是否正常，可访问性是否满足要求，动效是否影响性能，错误反馈是否清楚，数据异常时是否有兜底，以及 AI 是否修改了未经授权的范围。

更重要的是，团队要保存“AI 最初生成了什么、哪里不符合要求、谁如何纠正、最后怎样验证”的证据。这个过程比一张漂亮的最终截图更能证明设计师和团队可以对交付负责。

设计走查可以用“范围—行为—系统—风险—结果”五个视角先整理问题，但正式发布结论仍应进入第五章的质量主张、三路证据、门槛、残余风险和有权决定。

范围层检查本次修改是否仍然服务原目标，是否出现未经要求的额外改动；行为层检查流程、状态、反馈和跨设备表现；系统层检查组件复用、代

码结构、数据和设计 token；风险层检查权限、隐私、安全、可访问性和不可逆操作；结果层则检查用户是否真的更容易完成任务，业务目标是否改善，以及新问题是否被带回下一轮 Spec 和 Skill。

这种验收不是设计师一个人把所有事情检查完，而是让不同专业围绕同一份运行结果完成各自判断。设计师负责把体验问题组织出来，并确保它们不会在“代码已经能跑”的压力下被忽略。

6.5.5 从个人能力到组织能力

当一名设计师偶然用 AI 做成一个 Demo，这是个人尝试；当他能用 Spec、版本和验收稳定完成第二个项目，这是可复现过程；当团队把方法沉淀成设计系统、Skill、模板和 eval，它才成为组织能力。

可以用四种证据区分“做出来”与“能交付”：可运行原型证明某项假设能被体验；Spec 与版本记录证明范围和变化能够追踪；经过多任务验证的 Skill、模板与设计系统证明部分方法能够复用；跨专业验收和运行结果则证明团队有条件对限定范围作出交付决定。它们不是个人能力等级，也不存在取得后一项就自动包含前一项的关系。

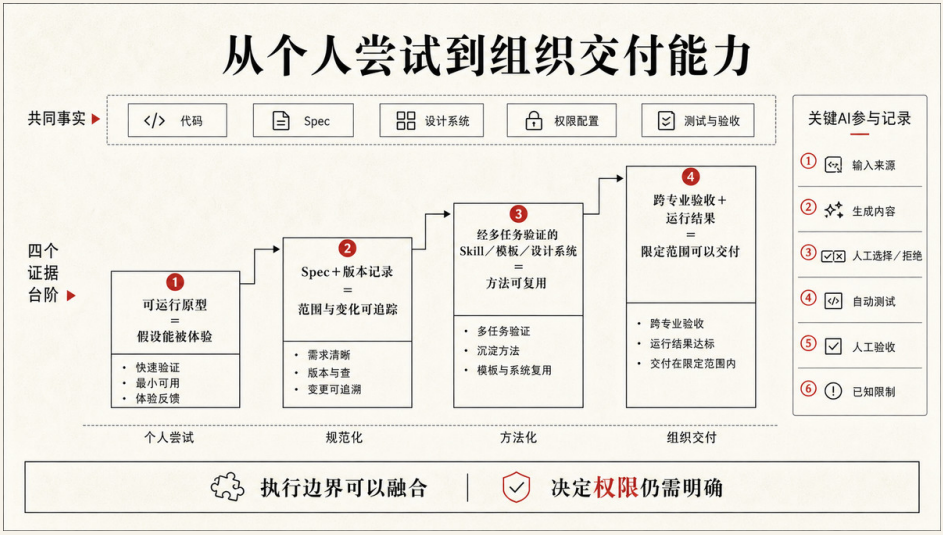

图 6-6 从个人使用 AI 到组织形成可维护能力

本章小结

对参与 Agent 产品的设计师而言，新工作可以概括为五项扩展。

第一，从只描述静态结果扩展到按需构建可运行体验。代码开始像草图、原型和组件一样成为设计材料；参与越深，越需要相应的构建、检查、修改和协作能力，但这不是全体设计师的统一岗位命令。

第二，从只维护视觉规范迁移到为 Agent 提供约束上下文。设计系统需要把 token、组件、语义、状态、代码映射和经过验证的团队规则变成机器可读取、可执行、可检查的材料，但它不是 Agent 系统的全部“操作系统”。

第三，从交付页面与流程迁移到同时交付分工、后果处理、上下文和权限。分工图、上下文地图、权限模型和持续更新的 Spec，把设计意图变成可维护、可追踪的共同约定；它们不是法律合同，也不需要在所有低风险项目中拆成独立文档。

第四，从只考虑人类界面迁移到同时考虑人和代表人行动的 Agent。人需要可理解、可判断和有品牌感的体验，Agent 需要稳定语义、明确权限和可验证结果；两条访问路径必须共享事实，但人仍是用户与权益主体。

第五，从个人产出迁移到协作交付。代码、设计、Spec、版本、AI 参与记录和验收结果共同构成新的设计证据，角色可以融合，但责任必须更清楚。

这一章讨论的是“设计师交付什么、如何工作”。第五章已经回答怎样用质量主张、证据和门槛判断当前版本是否可以通过；第七章将把视线移向更长时间尺度，讨论个体化、运行时界面、环境化体验和上线后的持续维护怎样改变设计，以及能力扩大之后怎样继续为人保留选择。

注释

1 Ana Boyer, “Design Systems and AI: Why MCP Servers Are the Unlock,” Figma, 2025 年 8 月 6 日，https://www.figma.com/blog/design-systems-ai-mcp/ 。本文只引用其对 MCP Server 可提供组件、样式、变量与标注上下文的产品说明，不采用文中的推广性效果承诺。

2 Figma Developer Docs, “Code Connect,” https://developers.figma.com/docs/code-connect/ 。该功能把设计文件中的组件与代码库实现关联，并可为 Figma MCP Server 提供实现上下文；访问于 2026 年 8 月 9 日。

3 Figma, “Figma Make, Now on Your Local Code,” 2026 年 5 月 28 日，https://www.figma.com/blog/figma-make-now-on-your-local-code/ ；“Code on the Figma Canvas,” 2026 年 6 月 24 日，https://www.figma.com/blog/code-on-the-figma-canvas/ ；Figma Developer Docs, “Create Skills for the Figma MCP Server,” https://developers.figma.com/docs/figma-mcp-server/create-skills/ 。截至 2026 年 8 月 9 日，官方说明仍把部分本地代码与 Code Layers 能力标为 beta 或分批开放；Skill 的定义和格式属于相应工具实现，不是跨平台标准。

第七章

07

设计的未来——个体化、生成式与环境化

先做一个设计推演：假设几年后，一位设计师准备开始一天的工作。

他没有先打开某个固定软件，再从菜单里寻找功能。系统已经根据今天的日程、正在推进的项目和上一次评审留下的问题，组织出一张临时工作台：左侧是三个需要确认的设计决策，中间是可以直接操作的原型，右侧是用户研究中的冲突证据。系统知道他习惯先看问题再看方案，因此没有把生成结果放在最前面；它也知道这个项目涉及医疗数据，所以所有对外动作都停留在草稿状态。

通勤途中，工作台没有被压缩成手机上的小号页面。系统把适合阅读的内容转到屏幕，把需要提醒的内容交给耳机，把原型操作留到办公室。到达公司之后，同一个任务在大屏上恢复，界面根据评审目标增加了比较视图和版本差异。下午，当他需要测试一个新的交互想法时，系统生成了一段可以运行的代码，而不是另一张静态效果图。

这个场景看起来方便，但很快也会暴露问题。系统为什么认为他今天仍然喜欢先看问题？这条偏好来自明确设置，还是根据过去行为推断出来的？它为什么隐藏了其他项目？界面改变之后，原来的功能去了哪里？耳机在公共空间里应该读出多少信息？系统生成的代码是否真的安全？如果用户

不再认同这套工作方式，他能否关闭记忆、恢复默认、修改规则，甚至换一个系统继续工作？

这个场景不是对行业时间表的预测，也不是说其中每项能力都已经成熟。它把当前产品、研究原型和设计议题放进同一个情境，是为了看清：一旦产品能够根据具体的人、任务和环境，在运行过程中重新组织内容、界面与行为，设计需要提前处理哪些关系。

因此，设计的未来不能只用“更智能”“更自动”或“没有界面”来概括。更值得讨论的变化是，产品可能从一个预先完成的对象，变成一套在约束内产生体验的系统。

这套系统必须同时回答四个问题：它可以依据什么变化，可以变化到什么程度，谁来判断变化是否正确，以及人怎样重新获得控制。

本章沿着六个方向展开：产品如何从统一软件走向个人系统，界面如何从预先固定走向运行时生成，交互如何从单一应用扩展到环境，新的问题视角如何形成，产品如何从一次性交付变成持续维护，以及在所有变化之后，设计为什么仍然需要替人保留选择。这里讨论的是方向和条件，不是所有产品都会经历的固定路线。

7.1 从统一产品到个人系统

工业化软件擅长用同一套产品服务大量用户。

团队会先找到一组相对稳定的共同需求，再通过信息架构、导航、页面和权限，把这些需求组织成可以规模化交付的功能。用户可以修改头像、主题、快捷方式和通知，但产品的基本结构通常不会因为某个人而发生根本变化。

推荐算法已经让内容因人而异。不同用户打开同一个音乐、视频或电商产品，会看到不同歌曲、商品和信息流。但这些变化主要发生在内容层，产品怎样工作、任务怎样完成、用户拥有什么控制，仍然大体相同。

AI 正在把个性化从内容层带入系统层。未来的差异可能不只是“给你看什么”，还包括“系统怎样替你工作”。

7.1.1 个性化从内容层进入系统层

仍以项目管理产品为例。传统个性化可能按照用户角色推荐不同模板，AI 驱动的个人系统则可能进一步改变：

1. 信息怎样组织，哪些内容默认展开；
1. 一个目标被拆成哪些步骤；
1. 哪些工作交给 Agent，哪些保留给用户；
1. 什么情况下自动执行，什么情况下请求确认；
1. 使用文字、图表、表格还是语音表达结果；
1. 记住哪些长期偏好，哪些信息只服务当前任务；
1. 发现风险后怎样提醒和升级。

这样一来，两个用户面对的就不再只是同一产品里的不同内容，而可能是两套不同的工作系统。新手需要更多解释、示例和检查点，专家需要更高信息密度、更少中断和更强批量能力；财务人员关心证据与政策版本，创意人员更需要方案分支和可逆探索；在低风险任务中，系统可以主动完成更多步骤，在资金、隐私或外部承诺面前，则需要降低自动化程度。

可以把这种个体化理解为四个层次。

表 7-1 个体化可能使用的四类信息

层次	系统可能使用的信息	设计问题
明确设定	用户选择的语言、密度、语气、权限和工作方式	设置是否清楚，改变后何时生效？
长期记忆	历史项目、反复出现的偏好、长期目标和合作关系	记住了什么，是否准确，能否删除？
行为推断	用户经常接受、拒绝、修改或跳过的内容	行为是偏好，还是产品限制下的妥协？
当前情境	设备、地点、时间、任务、在场他人和风险	哪些情境可以被感知，哪些不应该被使用？

这四层不能被混成一个笼统的“系统懂你”。明确设定是用户主动表达，长期记忆来自过去，行为推断只是系统的猜测，当前情境又可能很快变化。它们具有不同的可信度、隐私风险和有效时间，也需要不同的确认与撤回方式。

7.1.2 个人系统依赖的不只是记忆，还有能力编排

一个系统即使记住很多信息，也不一定真正适合用户。它还需要知道有哪些能力可以调用，这些能力之间怎样连接，以及用户允许它做到哪里。

Google Gemini 的官方帮助文档把过去对话、连接应用和用户指令组织成 Personal Intelligence 的来源。对符合账户、地区和功能开放条件的用户，产品提供记忆开关、信息纠正、相关对话删除，以及针对单次对话停用个性化或重新生成非个性化结果的入口；这些入口的作用范围和删除生效时间并不完全相同。[1] 这个案例最值得注意的不是“AI 可以记住用户”，而是记忆本身开始成为需要设计的产品对象：它有来源、开关、作用范围、纠正方式和删除状态。

Apple 的 App Intents 则展示了另一种系统级结构。应用可以用 schema 描述自己的动作和内容实体，把实体与界面关联，并声明能够在应用之间传递的类型，使系统获得可发现、可调用的能力描述。[2] 这不等于所有应用和平台都已经支持任意跨应用编排，却说明应用能力可以从只存在于页面和菜单中，进一步成为系统能够读取的结构化接口。

因此，个人系统不是一个无限了解用户的聊天机器人，而是由记忆、能力、权限和当前情境共同构成的工作环境。设计师需要同时设计：系统知道什么、能做什么、为什么现在这样做，以及用户如何改变这些条件。

7.1.3 “更懂你”也可能把过去固化成未来

个体化最容易被忽略的风险，是系统会把过去当成未来。

用户过去经常选择简短回答，不代表他在学习新领域时仍然只想看结论；过去一直独自出差，不代表这次家庭旅行也使用同一预算与节奏；用户在

旧产品里反复复制内容到文档，不代表他喜欢这种流程，也可能只是旧产品缺少结构化编辑能力。

行为不等于偏好，重复也不等于认同。如果系统只根据历史行为优化，就可能把用户困在一套越来越顺滑、也越来越狭窄的路径里。

生成式界面的个体偏好也比想象中更难概括。2026 年一项预印本研究让 20 位受过训练的设计师对同一批 600 个生成界面作出成对选择。研究发现，即使参与者都会使用“层级”“整洁”等相似概念，他们对什么是好的层级、这些原则应该怎样排序，仍然存在明显分歧。研究团队因此没有尝试建立唯一的通用审美评分，而是通过简短的成对选择学习个人偏好。[3] 这项研究检验的是特定数据集上的偏好建模，不是长期产品使用研究；它支持“偏好存在差异”，不能证明任何个体化机制都能持续理解一个人。

这个结果提醒设计师，个人系统不能只靠一段“请告诉我你的偏好”的提示词，也不能把某个平均用户模型当成所有人的默认答案。有些偏好难以被语言准确表达，用户可能只有看到两个具体方案时才知道自己更倾向哪一个；有些偏好又会随任务、能力和人生阶段改变。

7.1.4 设计差异，也要设计差异的边界

在采用系统级个体化的产品中，设计师的任务不是替每个用户单独画一套界面，而是设计能够安全地产生差异的规则。

一套可用的个体化机制至少需要满足四个条件。

第一，差异可见。用户应该知道当前体验是否使用了历史记忆、连接应用、位置或其他个人信息，重要结果还需要说明哪些个体化因素真正影响了判断。

第二，差异可改。用户可以纠正错误记忆、修改偏好、调整自动化等级，而不是只能通过反复对话“训练”一个看不见的模型。

第三，差异可暂停。用户能够针对当前任务临时关闭个性化，比较使用与不使用个人上下文时的结果。涉及健康、财务、关系或身份等敏感问题时，这种临时边界尤其重要。

第四，差异可重置。用户需要能够回到默认体验，删除不再成立的假设，并理解重置会影响哪些系统和数据。Google PAIR 的反馈与控制指南也建议允许用户修改过去的反馈，或把模型恢复到非个性化状态，因为人的需求会随时间改变。[4]

从统一产品到个人系统，并不意味着统一性失去价值。身份、权限、安全、无障碍和关键业务规则仍然需要稳定底座。真正的变化是，设计师要在共同底线之上，为不同的人保留适应空间，同时防止“为你定制”变成“替你决定”。

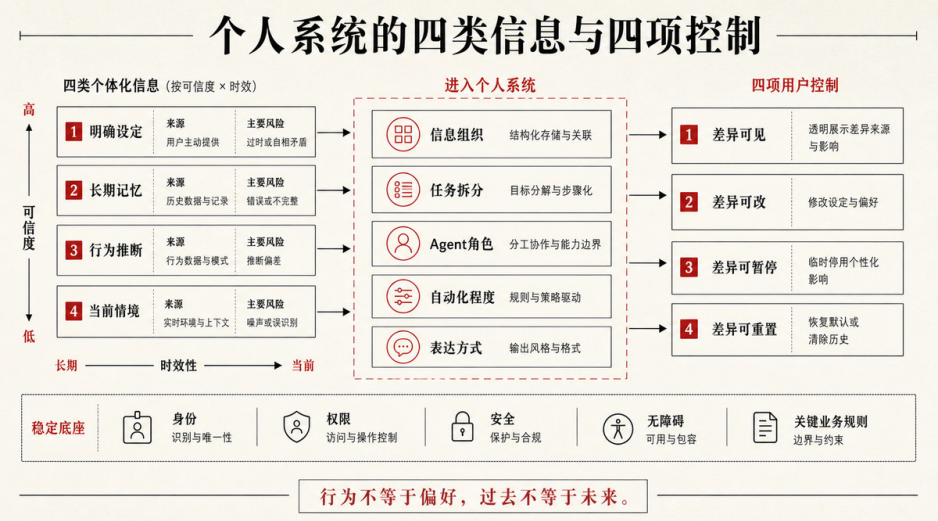

图 7-1 从统一产品到个人系统的构成与约束

7.2 从固定界面到运行时界面

传统界面的大部分决定发生在产品上线之前。

团队提前画出页面，定义组件和状态，开发人员把它们实现为软件。用户使用产品时，内容和数据会变化，但页面结构、交互方式与功能边界通常已经确定。

AI 使一部分设计决定可以推迟到任务真正发生时。系统先理解用户要完成什么，再决定应该展示哪些信息、调用哪些控件、组织怎样的步骤。界面因此不再只是一个预先完成的容器，而可能成为系统回应的一部分。

7.2.1 生成内容、编排界面和生成能力不是一回事

讨论运行时界面时，首先需要区分三个层次。

表 7-2 运行时生成的三个层次

层次	变化对象	例子	主要风险
内容生成	固定组件中的文字、图片、数据和图表	在既有卡片中生成项目摘要	事实错误、质量不稳定
界面编排	组件、信息层级、步骤和控件组合	针对预算比较生成滑块、表格和图表	不可预测、不一致、入口丢失
能力生成	新的逻辑、工具、工作流或代码	根据目标生成一个临时计算器	安全、权限、性能和真实后果

现在很多产品已经在做第一层，第二层正在快速出现，第三层则需要更严格的沙盒、权限、测试和人工确认。把三者都称为“AI 生成页面”，很容易低估它们之间的风险差异。

运行时界面之所以有吸引力，是因为聊天并不适合表达所有任务。比较多个方案时，表格可能比连续对话清楚；调整预算时，滑块和图表比反复修改数字高效；需要确认一组字段时，表单比逐项问答更容易检查；探索系统原理时，一个可以直接操作的模拟器比一段解释更容易理解。

一项预印本研究让语言模型针对研究选定的查询生成结构化、可交互的界面。在这些比较任务中，参与者有超过 70% 的选择偏向生成界面而不是纯对话。[5] 这个结果不能代表所有用户和任务，更不能证明聊天界面会消失；它只说明在部分任务中，人们可能需要的不只是另一个答案，而是更适合当前任务的操作环境。

7.2.2 运行时生成需要稳定底座

界面可以生成，并不等于系统应该随意生成任意前端代码。

A2UI 提供了一个可供比较的协议结构。Agent 不直接发送需要执行的 HTML 或 JavaScript，而是用声明式数据表达界面意图；客户端预先提供允许使用的组件目录，并使用本地组件完成渲染。界面结构、数据状态和具体视觉呈现相互分离，消息还可以经过 schema 验证。[6]
截至本书校订时，公开生产版本为 v0.9.1，v1.0 仍是候选版本；这种协议可以收窄任意代码执行面，但不能单独证明业务权限、数据使用和交互结果已经安全。

这意味着运行时界面可以分成三层。

图 7-2 运行时界面的稳定底座、可变结构与运行内容

这种结构和空间设计中的“支撑体—填充体”很相似。建筑的承重、消防和公共设施需要稳定，房间布局、家具和使用方式可以变化；AI 产品中的身份、权限和安全规则是支撑体，组件组合、任务流程和内容则是填充体。灵活性来自层次分离，而不是让所有东西都保持不确定。

如果没有稳定底座，生成式界面会迅速遇到问题。删除动作可能每次出现在不同位置，用户无法形成肌肉记忆；关键确认可能被系统为了“减少步骤”而隐藏；组件虽然视觉统一，却缺少错误状态和无障碍信息；模型生成了看似合理的输入框，却不知道字段需要什么业务校验。

所以，生成能力越强，设计系统越不能只是一套颜色、字体和圆角。它还需要包含组件语义、适用条件、禁用条件、数据要求、风险等级、状态行为、无障碍规范和验证规则。

7.2.3 设计师从画页面转向设计变化规则

过去，设计师通常决定某个页面最终长什么样。未来，设计师还需要决定一类界面可以怎样形成。

交付物会从一组页面扩展为：

1. 组件和语义目录；
1. 信息进入界面的优先级；
1. 不同任务适合使用的交互模式；
1. 允许变化和必须稳定的区域；
1. 不同风险对应的确认方式；
1. 生成失败后的回退界面；
1. 品牌、无障碍和内容规则；
1. 评价生成结果的 eval 与人工标准。

Google Research 2026 年的一项银行原型研究，让界面根据明确指令、推断需求和实时情境生成新功能或重新组织结构。在 72 人参与的被试内比较中，生成版本的系统可用性量表得分高于固定版本。[7] 但这个结果来自单一银行原型和短期任务，测量的是感知可用性，不能据此宣布固定界面已经过时，也不能外推生产环境中的安全性。它更有价值的启发是：设计师需要从一个个页面的作者，转向产生界面的原则、规则和约束的设计者。

新的工作还包括验证“变化本身”是否可用。团队不能只测试某个生成结果，而要测试一组可能性：同一任务多次生成是否仍然保持关键功能，低视力用户是否总能找到主要操作，高风险确认是否会被任何布局省略，语言变长或数据为空时结构是否仍然成立。

7.2.4 固定界面仍然有不可替代的价值

生成不是界面的默认答案。

当任务高频、目标明确、步骤稳定时，固定界面通常更快，也更容易形成习惯。一个每天执行几十次的收银操作，不应该因为系统每次“理解了不同上下文”而改变按钮位置；医疗、财务和设备控制中的关键流程，需要清楚、稳定且可审计；复杂专业工具依赖长期学习形成的空间记忆，如果界面持续重组，适应成本可能高于生成带来的收益。

Microsoft Research 在一个 16 人原型研究中也发现，依据当前提示生成的控件可以降低补充上下文的门槛、提高参与者的控制感，但同时会增加认知负荷并降低可预测性。[8] 这个小样本结果适合用来提出检查问题，不能当作动态控件普遍优于固定控件的结论。设计师不能只问“界面能不能生成”，还要问“变化是否真的降低了用户完成任务的总成本”。

可以用四个变量判断是否适合运行时生成：

1. 任务变化有多大。长尾、探索和一次性任务更适合生成，重复稳定任务更适合固定。
2. 错误后果有多大。风险越高，稳定结构、明确确认和可回溯要求越强。
3. 用户是否依赖熟练度。专家高频操作通常需要保持位置与快捷方式稳定。
4. 结果能否被验证。生成结构如果无法自动检查和人工审查，就不应直接进入真实执行。

因此，在适合运行时生成的产品中，固定与生成更可能共存：稳定底座提供认知、品牌和安全连续性，运行时界面为具体任务组织合适的表达。设计师不再决定每一次界面必须长什么样，但必须决定它不能怎样变化。

7.3 从单一应用到环境化交互

现在大部分数字产品仍然要求用户先找到入口。用户解锁设备、打开应用、进入页面，再向系统表达目标。即使使用语音助手，也通常需要唤醒一个明确的服务。

当模型能够同时处理语言、图像、声音、位置和屏幕内容，并被接入相应设备时，交互入口就可能附着在用户所处的环境中。汽车、眼镜、耳机、手表、家庭设备、办公空间和公共设施都可能成为同一任务的不同触点。

这不只是“多几个终端”。从单一应用到环境化交互，真正扩大的，是人与系统建立关系的场域。

7.3.1 交互从打开应用转向进入情境

在单一应用中，产品通常知道用户正在使用什么功能。环境化系统面对的情况更加复杂：用户可能正在走路、做饭、开会、驾驶或与他人交谈；他看向一个物体，不一定是在要求系统识别；他提到“那份文件”，可能指眼前屏幕、刚才的对话、桌上的纸张或另一个设备里的内容。

因此，环境化交互不是让系统随时随地回应，而是让系统理解当前情境是否适合介入。

系统需要判断：

1. 用户的注意力现在在哪里；
1. 哪种输入方式最自然；
1. 哪个设备最适合呈现结果；
1. 当前内容是否适合在公共环境中表达；
1. 任务需要立即处理，还是应该延后；
1. 一个设备离线或传感器失效时怎样继续。

过去，界面主要组织屏幕上的空间；在环境化产品中，设计师还需要组织注意力、时间与物理空间。

7.3.2 多模态的难点是协调，而不是数量

语音、手势、视线、触摸和图像并不是各自独立的功能。用户可能先用视线指向对象，再用语言提出问题；系统用语音给出简短结果，同时在屏幕上显示可以进一步检查的证据；当环境变得嘈杂时，输出需要从声音切换到文字；当用户双手被占用时，系统又要避免要求精确触控。

W3C 的多模态交互框架很早就指出，多模态系统需要交互管理器维持状态与上下文，协调不同输入输出，并根据设备能力、用户偏好和环境变化作出响应。[9] 这个框架虽然早于今天的大模型，但它揭示的问题没有过时：多模态不是把多个输入方式放在同一张功能列表里，而是让它们共同服务于同一个任务状态。

从设计角度看，至少需要回答四类协调问题。

第一，指代。用户说“这个”“那里”“继续”时，系统如何知道他指向哪个对象和哪段任务？

第二，接力。任务从手机转到汽车、眼镜或电脑时，哪些状态应该跟随，哪些敏感内容不应自动出现？

第三，冲突。语音说“取消”，手势却像是在确认；位置显示用户已到公司，日历却显示他在休假，系统应该相信什么？

第四，降级。摄像头被遮挡、网络中断、环境过于嘈杂或用户无法使用某种模态时，是否还有可理解的替代路径？

7.3.3 空间、身体与在场他人成为设计变量

Google DeepMind 的 Project Astra 展示了环境化助手的一种研究方向：系统可以通过手机摄像头或原型眼镜理解周围对象，在手机与眼镜之间保持对话记忆，并使用搜索、地图、日历等工具继续任务。官方也明确说明它仍是研究原型，由有限测试者参与测试。[10]

这个案例的重要性不在于预测所有人都会佩戴同一种设备，而在于它让现实环境成为交互上下文。用户不再需要先把世界翻译成文字，系统可以看

到他看到的物体、屏幕和位置。但感知能力越强，设计问题也越接近空间、社会关系与身体经验。

例如，眼镜在会议中显示一条提醒，用户能看到，但对面的人是否知道系统正在记录和分析？助手在商店里主动比较商品价格可能很有用，在朋友家里识别物品并推断价值则可能显得冒犯。语音在独处时自然，在地铁、电梯和诊室里可能泄露隐私。视线可以降低操作成本，也可能让用户担心“看过”是否等于“选择”。

Apple 的 visionOS 设计指南建议根据任务选择满足目标的最低沉浸程度，并优先考虑视野、姿势、运动和身体舒适，而不是默认追求完全沉浸。[11] 这条原则也适用于所有环境化 AI：技术能够出现，不代表它应该占据更多注意力；系统看得见，不代表它应该持续解释；界面可以铺满空间，不代表空间应该被界面占满。

环境化产品会增加一组传统 App 不常处理的设计变量：

1. 距离：信息离用户多远，是否需要移动身体；
1. 方向：内容从哪里出现，声音从哪里传来；
1. 可见性：只有用户看见，还是在场他人也能看见；
1. 持续性：内容停留多久，用户离开后是否继续存在；
1. 领地：系统是否进入了家庭、工作、公共与亲密空间；
1. 礼仪：什么时候主动出现会帮助人，什么时候会打断人与人的关系。

7.3.4 环境化系统必须知道什么时候不出现

“无处不在”不是环境化交互的质量标准。一个持续监听、持续提醒、持续猜测目标的系统，很可能比一个需要主动打开的应用更令人疲惫。

好的环境化系统需要拥有沉默、等待和退出的能力。

首先，主动介入应该有可理解的触发条件。系统可以在风险、截止时间或用户明确设定的场景中提醒，但不能只因为“可能相关”就不断争夺注意力。

其次，感知范围应该可见。摄像头、麦克风、位置和身体数据何时开启，哪些信息被保留，是否会跨设备共享，用户与在场他人都需要获得适当提示。

再次，跨设备接力不能只追求无缝。无缝意味着状态连续，也可能意味着隐私边界消失。手机里的健康问题不应自动出现在会议室屏幕，家庭设备中的对话也不应因为用户走进汽车就默认继续播放。

最后，系统需要保留非智能路径。传感器无法判断情境时，用户仍然应该能够用清楚的按钮、文字或人工服务完成任务。环境化交互的成熟，不是让界面彻底消失，而是让合适的界面在合适的地方出现，并在不合适的时候退到背景。

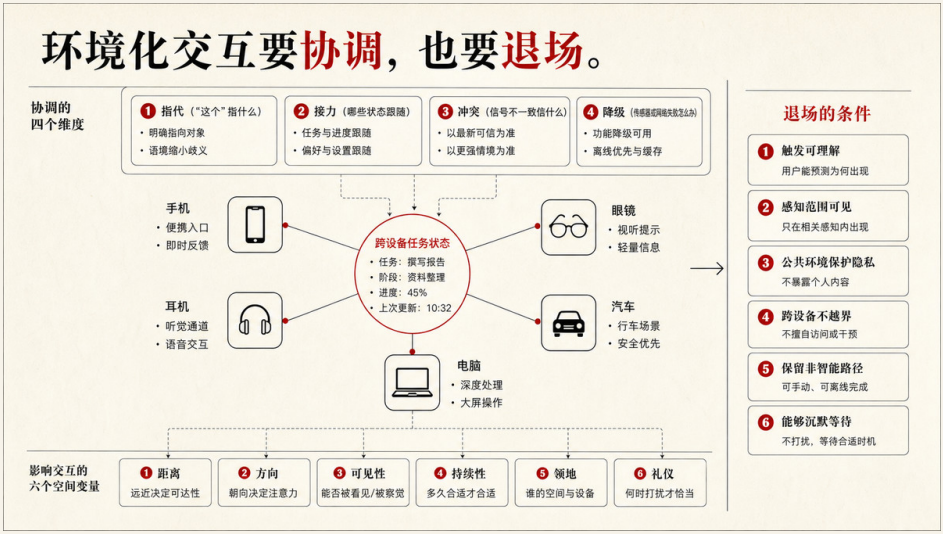

图 7-3 环境化交互既要协调多种信号，也要知道何时退场

7.4 从功能清单到八种问题视角

当设计对象从页面扩展到意图、上下文、能力、行为和治理，设计团队需要增加一组问题视角。

下面八项不是新的职业标准，也不要求每个项目和每位设计师全部包办。它们是本书前六章问题的索引：团队可以按任务后果选择所需视角，再让产品、研究、设计、工程、业务、安全和合规角色共同承担。

表 7-3 智能系统设计的八种问题视角

问题视角	需要回答的核心问题	可能产生的交付物
意图设计	系统怎样理解、澄清、分解和修改用户目标？	意图层级、澄清策略、目标版本和冲突处理
上下文设计	什么信息在何时进入系统，怎样共享、保留和遗忘？	上下文地图、来源与新鲜度、记忆范围和删除机制
委托关系设计	人与 AI 怎样分配判断和执行，谁有权批准，谁处理行动后果？	人机分工与后果处理图、Agent 自主范围、阶段角色、确认与接管点
不确定性设计	系统怎样表达未知、推断、冲突和风险？	不确定性分类、证据、路由与兜底策略
Agent 行为设计	系统怎样计划、执行、观察、调整和停止？	行为状态机、工具策略、权限与失败恢复
生成式 UI 设计	界面怎样在运行时安全、稳定地形成？	组件目录、生成规则、约束、验证与回退界面
eval 设计	怎样把“真正做得好”变成可重复检查的证据？	任务样本、评分标准、轨迹检查与人工评审
运行时治理设计	上线后怎样发现漂移、事故和新风险？	监控信号、事件分级、版本、降权和恢复机制

以旅行 Agent 为例，用户说“帮我安排一次轻松的家庭旅行”，这句话同时触发了八类设计问题。

“轻松”到底意味着每天少去几个景点，还是减少换酒店和长距离移动？这是意图设计。系统是否可以读取家庭成员、预算、签证、学校假期和健康信息，这是上下文设计。它可以提出方案、预订可退酒店，还是能够直接付款，这是委托关系设计。机票价格会变化、天气预测不稳定、不同家庭成员偏好冲突，这是不确定性设计。

系统接下来怎样搜索、比较、规划和调用预订工具，是 Agent 行为设计。方案应该用地图、日历、价格曲线还是聊天呈现，是生成式 UI 设计。怎样判断行程真的“轻松”而不只是景点数量少，是 eval 设计。供应商接口变化、误订、退款失败和长期偏好漂移怎样被发现和处理，则属于运行时治理设计。

传统产品常把这些问题分别藏在产品、设计、算法、工程、运营与合规的工作中。未来设计的一个重要变化，是把它们放回同一段人机关系里。用户看到的是一个连续体验，团队也需要用连续的方式设计责任、证据和后果。

设计师不一定要成为模型研究员、安全工程师或法律专家，但需要知道何时问题已经超出界面，应该邀请谁共同作出决定。这八种视角的价值，不是制造更多术语，而是防止团队继续用一张页面覆盖一个行为系统。

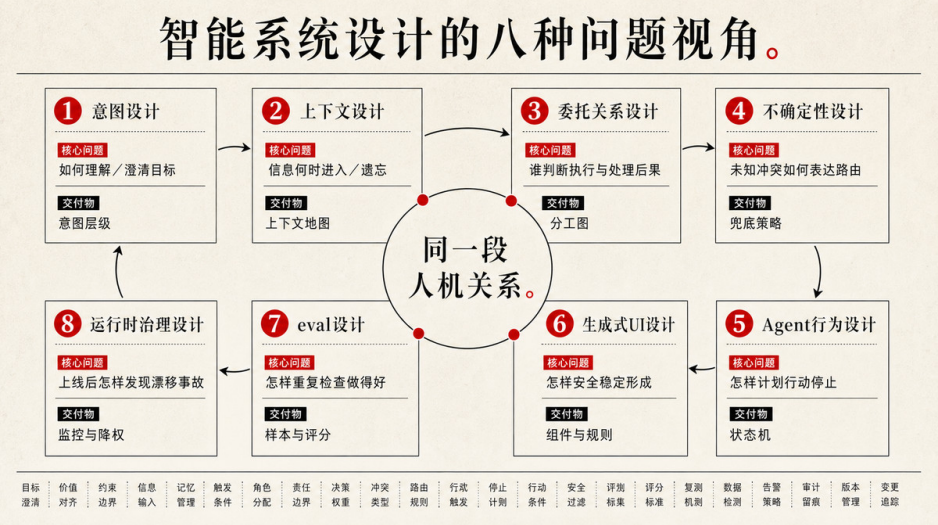

图 7-4 智能系统设计的八种问题视角

7.5 从一次性交付到持续维护与再验

传统产品同样会持续迭代，但团队通常仍然可以定义一个相对明确的发布版本：需求完成、设计验收、测试通过，产品上线。

当 AI 系统依赖会变化的模型、知识库、工具和权限时，就更难保持这种稳定。用户还会把系统带进团队原来没有考虑的情境，组织政策和社会规范也会改变。相同输入在不同时间可能得到不同计划，系统能力也可能因为新工具和新权限扩大。

因此，具有模型、工具和运行时变化的 AI 产品，需要持续维护与再验。下文偶尔使用“培育”这个比喻，指的是团队对版本、反馈、失败和质量主张的生命周期管理，不是系统在无人负责的情况下自行学习或成长。

7.5.1 真实使用会产生新的证据

上线前测试只能覆盖团队已经想到的任务和风险。真实用户会使用更混乱的资料、更含糊的目标和更多意外组合，也会把系统用于团队没有预料到的场景。

NIST 的 AI 风险管理框架把治理、映射、测量和管理设置为贯穿系统生命周期的持续活动。框架提出的部署后工作包括收集用户和相关角色的输入，建立申诉与推翻机制，处理退役、事件响应、恢复与变更管理，并把可衡量的持续改进活动纳入系统更新。[12] 这些是自愿性风险管理框架中的活动，不是对所有产品一概适用的法定要求。

从设计角度看，这意味着发布不是设计结束，而是设计获得真实证据的开始。团队需要知道用户怎样理解系统、哪些错误没有被发现、哪些确认只是形式、哪些个性化开始偏离当前需要、哪些生成界面破坏了稳定性，以及哪些原本低风险的能力因为使用方式变化变成了高风险行为。

空间设计中有“使用后评价”：建筑交付之后，设计者继续观察真实的人如何使用、绕开和改造空间。AI 产品也需要类似机制。系统在真实环境中怎样被使用，往往比上线前演示更能暴露设计假设。

7.5.2 三种反馈不能进入同一条学习回路

“系统会从反馈中学习”听起来合理，但不同反馈具有完全不同的含义。

表 7-4 三类反馈及其处理路径

反馈类型	例子	合理处理方式
个人偏好	“以后默认给我看详细版本”	在用户可见、可修改和可重置的个人范围内生效
任务纠正	“这张发票应该归到差旅，不是办公用品”	修正当前结果，是否成为长期规则需要进一步判断
事故与风险	Agent 使用旧政策、越权发送或错误付款	进入事件复盘，更新 eval、权限、规则和系统版本

如果把一次任务纠正直接变成全局规则，系统可能为了修复一个用户的特殊情况而制造新的错误；如果把所有拒绝都当成偏好，用户为了检查结果而作出的修改也可能被误读；如果事故只被记录为一次差评，团队就无法发现权限和治理问题。

Google PAIR 的指南强调，团队需要说明反馈会怎样改变 AI、多久产生影响，并让用户理解反馈的范围。[4] 这意味着反馈机制不能只有赞和踩。设计师需要让用户表达“事实错误”“不符合我的偏好”“缺少证据”“这次任务例外”“系统不应该做这件事”等不同含义，团队才能把反馈路由到正确层级。

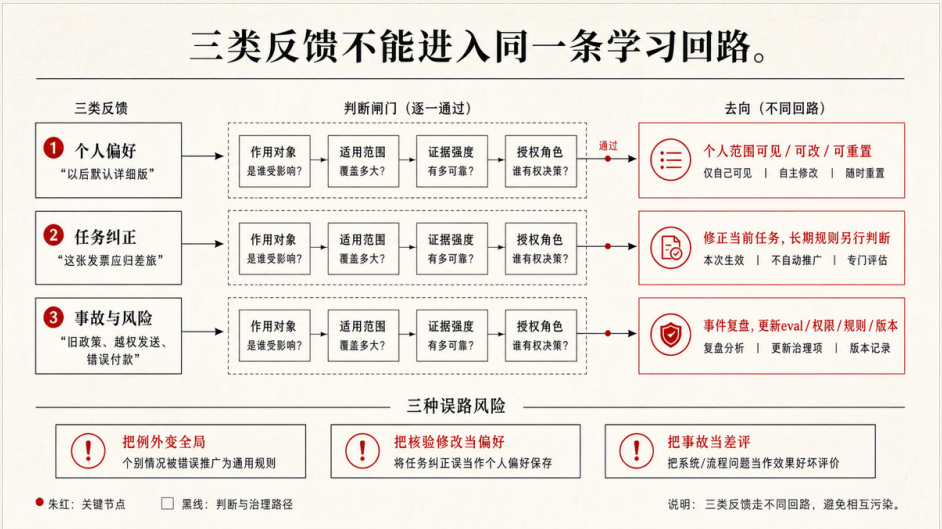

图 7-5 三类反馈需要分流、授权与回退

7.5.3 团队需要维护失败、版本和质量主张

持续维护并不是每天人工修正模型输出，而是由团队建立让系统能够被观察、校准和约束的机制。

设计师需要参与组织、并与相应责任角色共同维护的内容包括：

1. 真实失败与近失事件；
1. 用户修改、拒绝和接管的原因；
1. 代表性任务与边界样本；
1. 组件、规则、提示词、模型和工具版本；
1. 个体化假设和记忆的来源；
1. 自动化等级与权限变化；
1. 质量主张、验收门槛、eval 和人工评审结果；
1. 事故后的修复、降权、回滚和通知。

这些工作会改变“设计完成”的定义。一张界面通过评审，并不能证明它在新的数据、模型和任务中继续成立；一条规则曾经有效，也不能保证用户目标变化后仍然合理。设计师需要把规则当成可以被证据推翻的假设，而不是永远正确的规范。

7.5.4 受控适应需要变化说明和回退

未来产品可能拥有更强的适应能力：根据用户明确设置改变输出和界面，或在获准范围内根据反复出现的需求提出新的工作方式。适应不等于自主生效；影响业务规则、权限和真实后果的变化，仍然需要由有权角色评审、验证和发布。

“系统会自己进化”不应成为回避责任的说法。越是能够变化，越需要让人知道：什么发生了变化，为什么变化，依据了哪些反馈，影响哪些任务，是否经过验证，以及怎样恢复上一版本。

对个人用户来说，重要变化应该进入可理解的记录，例如“根据你最近三次修改，摘要默认从三段调整为五点；你可以撤销”。对团队来说，模型、提示词、工具、权限和设计规则要能够分别版本化，出现问题时才能定位变化来源。

系统适应还需要速度分层。输出格式和临时布局可以快速变化，业务政策、权限和高风险行为应该经过更慢的评审与验证。并不是所有部分都应该用同一种“实时学习”速度。

从一次性交付到持续维护，设计工作的范围会从塑造一个完成品，扩展为参与维护一组长期关系：系统与用户目标的关系，行为与规则的关系，反馈与变化的关系，以及能力增长与责任边界的关系。谁能修改规则、批准版本、扩大权限和处理事故，仍要由组织明确指定，不能笼统归给“设计师”。

7.6 设计最终仍然是在为人保留选择

个体化、生成式与环境化，都在试图降低人与系统之间的摩擦。

个人系统减少重复设置，运行时界面减少寻找功能，环境化交互减少打开应用和翻译意图，Agent
减少手工执行，代码生成减少从想法到实现的距离。

这些变化可以增强人的能力，也可能悄悄缩小人的选择。

7.6.1 越顺滑的系统，越可能隐藏决定

个人系统会根据过去预测用户需要什么，因此可能不再展示其他可能；运行时界面会为当前任务选择“最合适”的控件，因此可能隐藏系统没有选中的功能；环境化助手在用户开口之前主动出现，因此可能把推测变成默认意图；持续学习会吸收用户反馈，因此产品规则可能在用户没有意识到时改变。

当所有决定都被包装成便利，用户可能只看见最后一条路径。

问题不在于系统做了选择。任何界面都在排序信息、设置默认和限制行为。新的风险在于，这些选择开始实时发生、因人而异，而且不一定留下稳定页面让用户比较。设计如果只优化步骤和点击，就可能在提高效率的同时降低可见性。

7.6.2 选择权需要变成具体的产品能力

“以人为本”不能只停在价值口号。它至少应该被转化为六种可以检查的产品能力。

表 7-5 把选择权转化为可检查的产品能力

能力	用户应该能够做什么	设计需要提供什么
知道	理解系统使用了什么上下文、为什么给出当前结果	来源、个体化标记、关键依据和变化说明
选择	看到真实替代方案，比较不同目标和取舍	分支、对比、默认项说明和影响预览
调整	修改目标、记忆、界面和自动化程度	可编辑偏好、权限刻度和局部设置
拒绝	不使用某项个体化或自动化，转向其他路径	暂停、关闭、跳过与非 AI 方案
恢复	撤销动作、纠正结果、申诉并重新接管	版本、回滚、补偿、人工升级和证据链
离开	清除记忆、导出必要数据并退出系统	删除、可携带性、退订和清楚的后续影响

OECD 的人本 AI

原则把个人自主、人类能动性和监督列为重要保障。UNESCO 的 AI 伦理建议进一步提出，人可以在有限情境中选择让渡控制，但最终责任与问责仍要归属于相应的人或法律实体；在数据治理框架中，个人还应能访问和删除自己的个人数据，并获得纠正决定与损害补救的机制。[1314] 这些原则的适用方式仍受具体法律、角色和情境约束，不能被压缩成一个让末端用户独自承担后果的“同意”按钮。

这些原则最终都需要通过设计落地。一个关闭按钮如果藏在多层设置里，拒绝就不是低成本选择；一个批准按钮如果没有提供证据，选择只是责任转移；一个记忆删除入口如果无法说明影响范围和生效时间，退出也不完整。

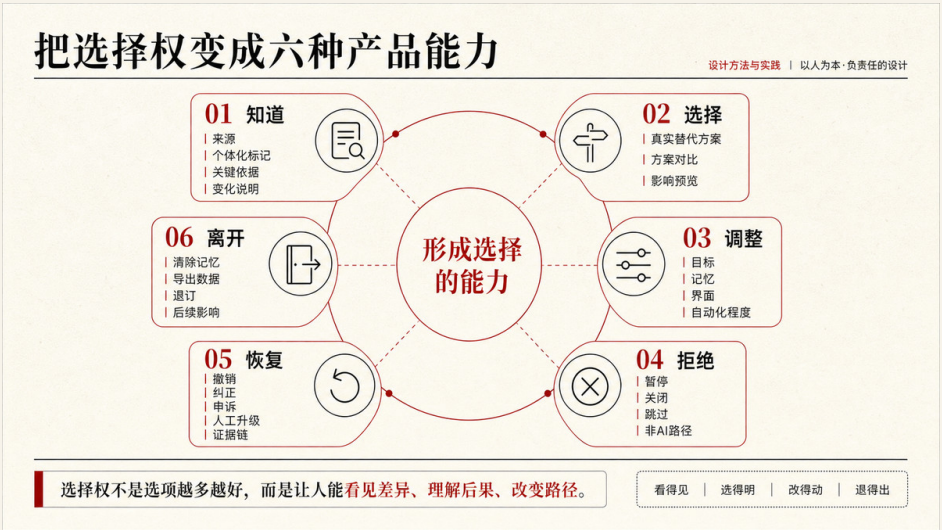

图 7-6 把抽象的选择权落实为六种产品能力

7.6.3 设计不是保留最多选项，而是保留形成选择的能力

替用户保留选择，并不意味着所有任务都要展示十种方案，也不意味着系统不能自动完成重复工作。选项过多同样会增加负担。

真正需要保留的是用户形成选择的能力：他知道目标是什么，能够看见关键差异，理解后果，在需要时改变路径，并拥有承担责任所需的信息与权力。

因此，AI 可以帮助用户搜索更多方案、模拟后果、发现冲突和完成执行，但不应该悄悄替用户决定什么值得追求。系统可以根据过去偏好安排一次旅行，却不能把过去的生活方式当成用户永远不变的价值；可以生成一个高效工作流，却不能因为效率更高就取消用户保留的学习、协商和判断过程。

有些摩擦应该被消除，例如重复录入、格式转换、机械查找和低价值等待；有些摩擦则应该被保留，例如理解目标、比较价值、确认不可逆后果和处理利益冲突。设计师需要分辨两者，而不是把“步骤更少”当成唯一进步。

7.6.4 未来设计师仍然在设计人与世界的关系

本书从一个变化开始：人与机器之间的关系，正在从人操作工具，转向人表达意图并委托机器行动。

第一章解释了确定性软件的基础假设为什么发生变化。第二章把设计对象从界面扩展到意图、上下文、能力、行为、表达和治理。第三章讨论用户怎样从操作者变成委托者、监督者和否决者。第四章把设计思维从规定单一路径扩展为管理可能性空间。第五章重新定义好设计：不仅要有任务与使用质量，还要检查行为边界、校准依赖、回溯、接管恢复、长期影响，以及特定情境中的判断能力保护。第六章再讨论设计师怎样把代码、设计系统和结构化交付材料带进 Agent 工作流。

来到最后一章，这些变化汇合成一组有条件的方向：产品可能更个体化，部分界面可能在运行时形成，交互可能进入更多环境，设计会扩展出新的问题域，系统也需要在上线后持续维护。代码、模型和规则如何成为设计材料，已经由第六章具体展开。

但技术能力不会自动给出设计方向。系统能够理解更多，不代表应该知道一切；能够生成更多，不代表每次都应该变化；能够行动更远，不代表可以替人承担责任；能够无处不在，也不代表应该占据人的全部注意力。

设计的价值，仍然在于为能力建立秩序，为变化规定边界，为不确定性提供理解，为行动保留责任，也为人保留重新选择的机会。

AI 产品可以让用户更少学习软件结构，更少重复执行步骤，也更少在不同工具之间搬运信息。判断这种产品是否优秀，不能只看它省掉了多少步骤，还要看它是否要求用户用便利交换控制：人能否知道系统正在怎样理解自己，能否修改这种理解；机器承担规模、速度和重复之后，目标、价值、例外和后果是否仍交给有信息、有权力也有职责的人判断。

当软件开始越来越像一个行动者，设计师要做的，不是让机器看起来更像人。

而是让人在与机器共同生活和工作时，仍然能够理解、选择、改变，并成为自己目标的定义者。

本章小结

本章提出六个相互关联的未来方向。它们是供团队检验的设计命题，不是行业发展的必经阶段。

第一，部分产品可能从统一软件走向个人系统。个性化可以从推荐内容扩展到信息结构、工作流、Agent 角色、自动化程度、记忆和确认方式。设计师需要让差异可见、可改、可暂停和可重置，防止系统把过去固化成未来。

第二，部分界面可以在运行时形成。生成内容、编排界面和生成能力具有不同风险。在采用这种方式的项目中，设计师不仅画页面，还要定义稳定底座、组件语义、生成规则、权限、回退和验收标准。固定界面仍然适合高频、稳定、高风险和依赖熟练度的任务。

第三，交互可以从单一应用扩展到环境。多模态的关键不是输入数量，而是共享上下文、协调状态、跨设备接力和根据注意力选择合适的表达。位置、距离、身体、在场他人、隐私与社会礼仪也会随之成为设计变量。

第四，设计需要增加意图、上下文、委托关系、不确定性、Agent 行为、生成式 UI、eval 和运行时治理等问题视角。它们未必成为独立岗位，也不要求单个设计师全部负责，但能帮助团队看见页面之外的系统问题。

第五，具有运行时变化的产品需要从一次性交付走向持续维护与再验。真实使用会产生新证据。个人偏好、任务纠正和事故风险需要进入不同反馈回路，模型、工具、权限、规则和质量主张都需要被持续观察、版本化、验证和回退。

第六，所有未来方向最终都要回到人的选择权。好的系统应该让用户能够知道、选择、调整、拒绝、恢复和离开。设计不是让机器替人决定一切，而是在机器拥有更多能力之后，重新安排人如何理解、参与和负责。

AI 改变设计，最终改变的不是某个工具、某个界面或某个岗位，而是人怎样把意图交给机器，机器怎样把行动带回世界，以及设计怎样守住两者之间的边界。

注释

1 Google, “Get personalization with memory of your past Gemini chats” 与 “Connect your Google apps to personalize your Gemini experience,” Gemini Apps Help，https://support.google.com/gemini/answer/16598469 ，https://support.google.com/gemini/answer/16598406 。

2 Apple Developer, “App Intents” 与 “Providing contextual cues to Apple Intelligence and Siri,” https://developer.apple.com/documentation/appintents ，https://developer.apple.com/documentation/appintents/providing-contextual-cues-to-apple-intelligence-and-siri 。

3 Yi-Hao Peng, Samarth Das, Jeffrey P. Bigham, Jason Wu, “Efficient Personalization of Generative User Interfaces,” arXiv:2604.09876, 2026，https://arxiv.org/abs/2604.09876 。

4 Google PAIR, “Feedback + Control,” People + AI Guidebook，https://pair.withgoogle.com/guidebook-v2/chapter/feedback-controls/ 。

5 Jiaqi Chen et al., “Generative Interfaces for Language Models,” arXiv:2508.19227, 2025，https://arxiv.org/abs/2508.19227 。

6 A2UI, “A Protocol for Agent-Driven Interfaces” 与 “A2UI Specification,” 当前稳定版 v0.9.1、v1.0 候选版，访问于 2026 年 8 月，https://a2ui.org/ ，https://a2ui.org/specification/ 。

7 Piyush Arora et al., “Self-Evolving Systems: Moving Beyond Deterministic Interfaces to Adaptive Generative Interfaces,” Google Research, 2026，https://research.google/pubs/self-evolving-systems-moving-beyond-deterministic-interfaces-to-adaptive-generative-interfaces/ 。

8 Microsoft Research, “Tools for Thought — Dynamic UI for AI lowers the barrier for steering AI outputs,” https://www.microsoft.com/en-us/research/project/tools-for-thought/ 。

9 W3C, Multimodal Interaction Framework, 2003，https://www.w3.org/TR/mmi-framework/ 。

10 Google DeepMind, “Project Astra,” https://deepmind.google/models/project-astra/ 。

11 Apple Developer, “Designing for visionOS,” Human Interface Guidelines，https://developer.apple.com/design/human-interface-guidelines/designing-for-visionos 。

12 NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023，https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf 。

13 OECD.AI, “Human-centred values and fairness (Principle 1.2),” https://oecd.ai/en/dashboards/ai-principles/P6 。

14 UNESCO, Recommendation on the Ethics of Artificial Intelligence, 2021，尤其第35—43、54—55、73段，https://www.unesco.org/en/legal-affairs/recommendation-ethics-artificial-intelligence 。