

人人都会Vibe Coding

面向普通人/ 开发者的科普与实用指南

目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

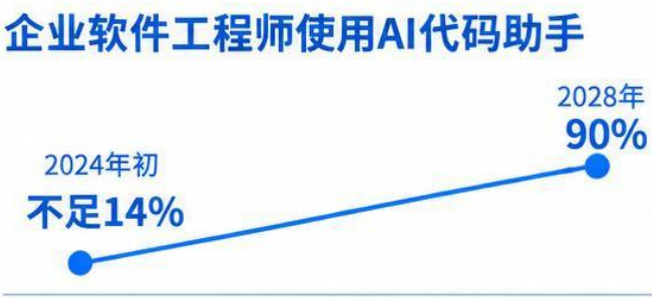
04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

AI编程进入快速增长期

编程曾长期是专业程序员的专属，Vibe Coding以自然语言描述需求，把创造重心从代码语法转向创意与需求的表达：非开发者无需编码基础即可做出解决具体问题的小工具，开发者则将精力从重复编码转向系统设计、需求判断与质量控制。市场价值同步显现：Gartner预计到2028年90%的企业软件工程师将使用AI代码助手（2024年初不足14%）；Mordor Intelligence预计全球AI代码工具市场将由2025年的73.7亿美元增至2031年的299.6亿美元，2026—2031年复合增长率26.23%；IDC数据显示，中国市场将从2025年的3.99亿元增至2026年的11.73亿元。



来源：Gartner, 2025



来源：Mordor Intelligence, 2026



来源：IDC《2025年中国AI编程市场份额报告》

Vibe Coding 让自然语言成为软件创造的新入口

2025年2月，Andrej Karpathy用“Vibe Coding”描述一种基于自然语言和AI的全新编程方式：开发者只需提出需求、运行程序、将报错信息反馈给AI，在生成可用结果后继续推进，全程无需纠结于代码的具体写法。推文发布后48小时内获得超过450万次浏览，迅速在开发者社区中引发广泛讨论。2025年底，柯林斯词典将其评为年度词汇，标志着这一概念已从开发者圈层扩展为大众话题。



非开发者·个人日常生活场景

Vibe Coding让没有编程经验的用户也能尝试制作个人网站、家庭记账、学习打卡、活动报名和简单数据看板等工具。使用者先用自然语言描述需求——说明谁在用、要完成什么操作、希望得到什么结果，AI据此生成一个可以直接用的版本。用户通过实际试用，指出问题并提出修改意见，AI持续调整，直到满足需求。该方法更适合流程清晰、数据规模小、影响范围有限的工具型应用。



非开发者·职员办公提效场景

非开发人员的日常工作涉及大量信息收集、整理与分发，这些工作完全可以通过软件工具提效。Vibe Coding让人力资源、财务、行政、教育、科研和销售等岗位能够自行制作表单、内部看板、数据整理工具和流程原型。其直接价值在于让业务人员在需求产生时先做出可操作样品，用真实流程发现遗漏，再与IT或开发团队讨论。低风险工具可由业务人员持续维护，但接入企业账号、核心数据和正式流程后，仍需遵循权限、安全、审计与运维要求。



开发者·人机协作的分工场景

Vibe Coding让专业开发者可以借助AI处理代码检索、功能实现、测试补充和缺陷修复等可明确描述和验证的任务。具备Agent能力的工具还能读取代码库、调用终端、运行测试，并根据结果持续修改。随着AI承担更多编码和操作工作，开发者的注意力转向任务拆解、技术方案、关键设计和结果审查，并在关键节点决定继续、调整或回退。开发者仍需理解关键代码，通过Diff、测试、CI和运行结果判断修改是否可靠，再决定是否合并到项目。



使用者差异对比：普通人追求可用，专业开发者要保证可交付

普通人通常从生活或业务问题出发，通过实际试用确认是否满足需求，目标是尽快得到可以使用和分享的工具；开发者通常从需求单、缺陷或现有代码库出发，还要检查架构、依赖、测试、安全和上线影响。两者都遵循“说明目标-生成结果-试用反馈-验证修改”的循环，但交付依据不同：个人工具以实际体验为主，团队和生产软件还需要代码审查、版本管理、自动测试、权限控制与运行监控等。



目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

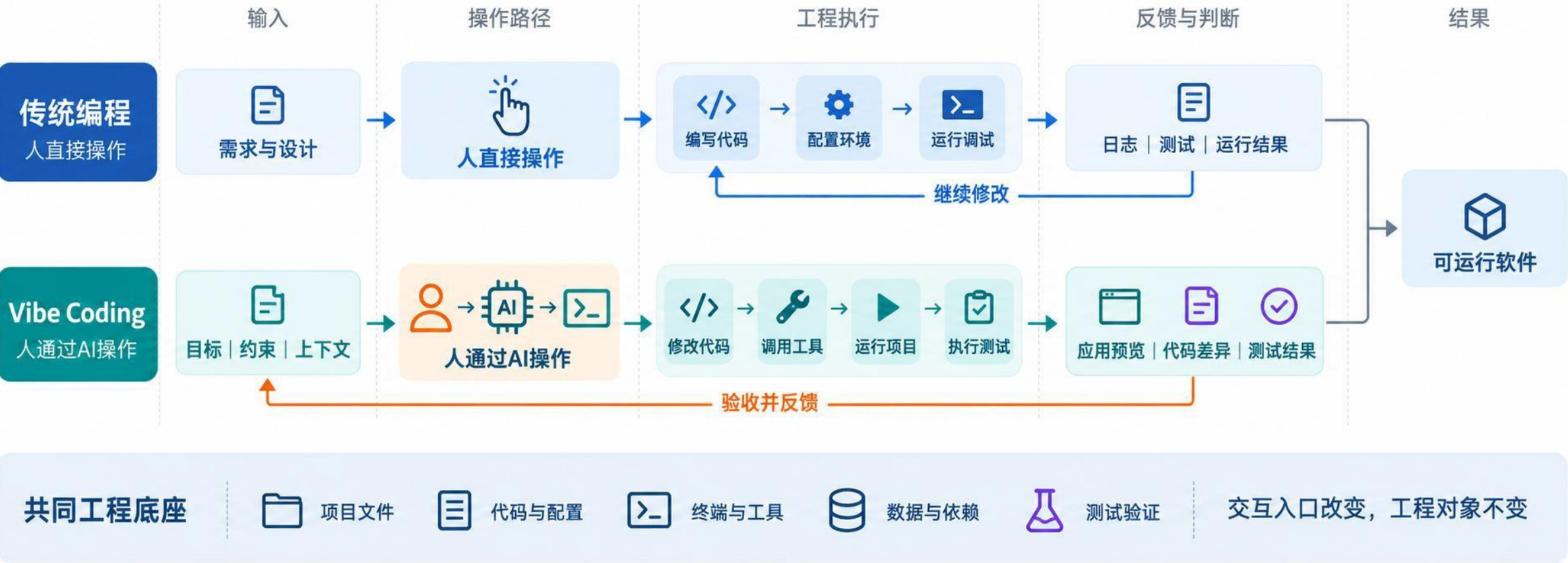
04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

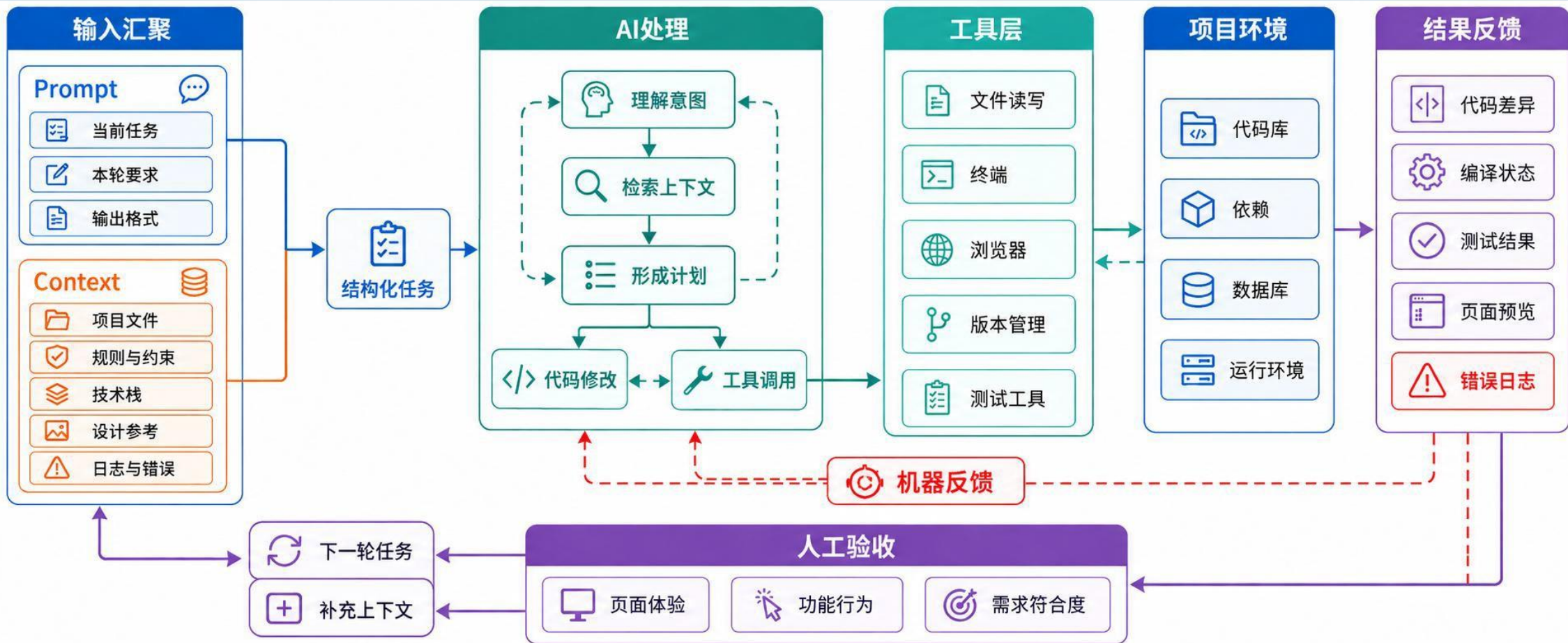
Vibe Coding 的核心理念

Vibe Coding改变了人与软件开发工具的交互方式。在传统编程中，人必须把想法翻译成机器能理解的精确指令——语法、算法、数据结构、接口规范，每一步都不能出错。Vibe Coding借助大模型解析自然语言意图，并进一步生成计划、代码和工具操作。使用者工作重心从直接实现转向清楚表达目标、提供必要上下文、根据运行结果持续校正，并判断最终结果是否可以使用或交付。



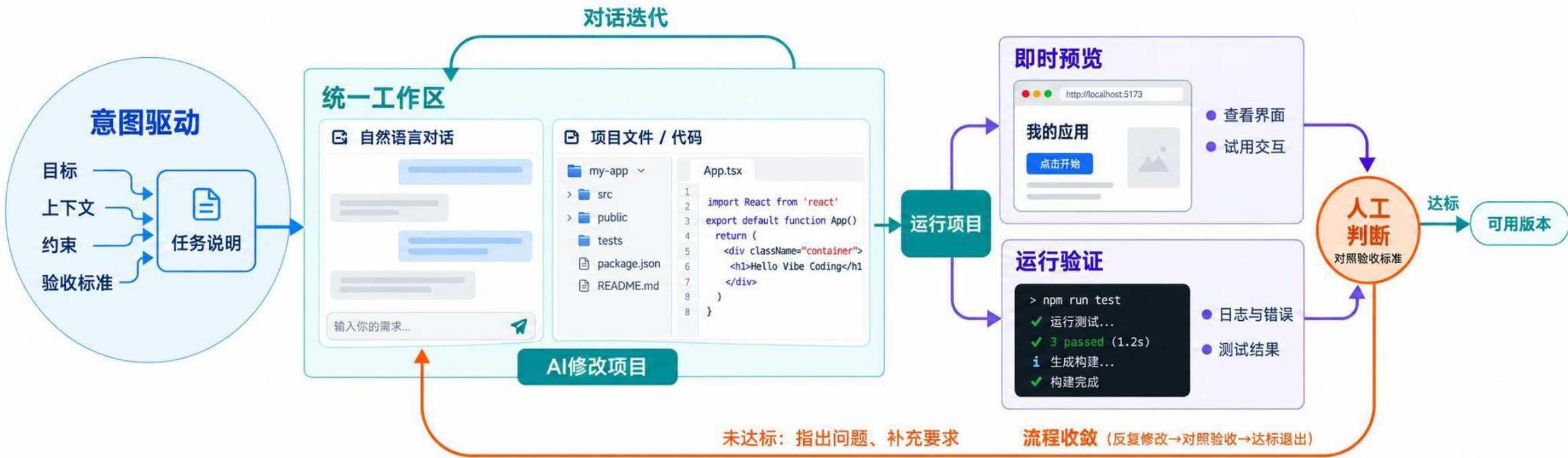
Vibe Coding 的工作原理

Vibe Coding 以用户意图作为开发起点。用户通过自然语言、截图或已有代码描述目标，AI 结合代码库、项目规则和运行状态理解当前任务，再将目标拆解为具体操作，通过编辑文件、执行命令、调用工具和运行应用完成实现。测试结果、报错信息和实际页面效果会继续反馈给 AI，推动下一轮分析和修改。整个过程形成“表达意图-理解上下文-执行修改-运行验证-反馈迭代”的循环，全程由人把控项目方向，持续循环迭代直至应用满足交付标准。



Vibe Coding 的核心特征

Vibe Coding的核心特征可以归纳为意图驱动、对话迭代、即时预览与流程收敛。**意图驱动**指开发者用自然语言描述功能目标和预期体验，AI负责技术实现。**对话迭代**指通过多轮自然语言对话逐步接近最终目标。**即时预览**指AI生成的应用可在浏览器或模拟环境中立即运行，开发者能实时看到效果、感知产品形态。**流程收敛**则实现所有开发环节在同一个工作区内形成完整闭环，需求澄清、界面调整、代码修改和运行验证无需切换工具，可随时来回调整，快速产出可用原型。



Vibe Coding 与传统编程的全面对比

对比维度		传统软件开发	Vibe Coding
交互过程	 主要入口	代码、配置、设计文档与 开发工具	自然语言 、参考材料、运行反馈与代码
	 迭代节奏	需求、设计、开发和测试按 工程流程 推进	描述、生成、运行和反馈形成 高频短循环
控制与能力	 实现控制	开发者直接 设计、编写、调试和维护代码	AI生成 或修改代码，使用者持续指导和检查
	 上下文来源	团队通过 文档、代码和协作 积累系统理解	AI依赖 当前对话 、项目文件、规则与工具反馈
	 使用者职责	团队分工 完成需求、架构、实现、测试与运维	定义目标、提供上下文、判断结果并承担 交付责任
	 技能要求	专业开发能力 贯穿设计、实现和维护	起步门槛降低，风险越高越需要 工程能力
交付与风险	 首版形成	时间取决于团队、系统复杂度与 工程流程	简单原型通常能够 更早形成
	 质量保障	依靠 测试、代码审查、CI 、监控与运维	沿用工程保障，并增加AI生成 结果核验
	 适用范围	各类软件，适合 复杂、长期维护 的生产系统	原型、小工具 和边界清晰的任务更具优势
	 主要限制	开发成本 和专业门槛较高	上下文遗漏、错误累积和维护难度 随规模上升

目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

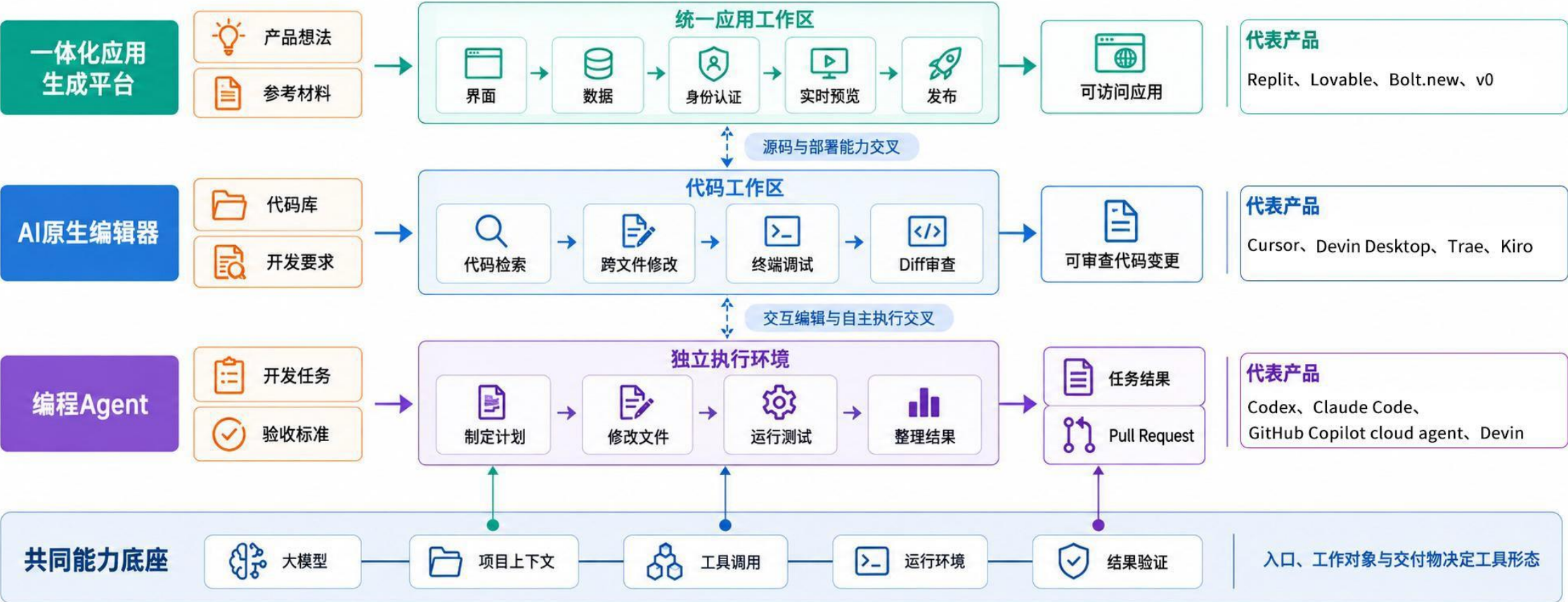
04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

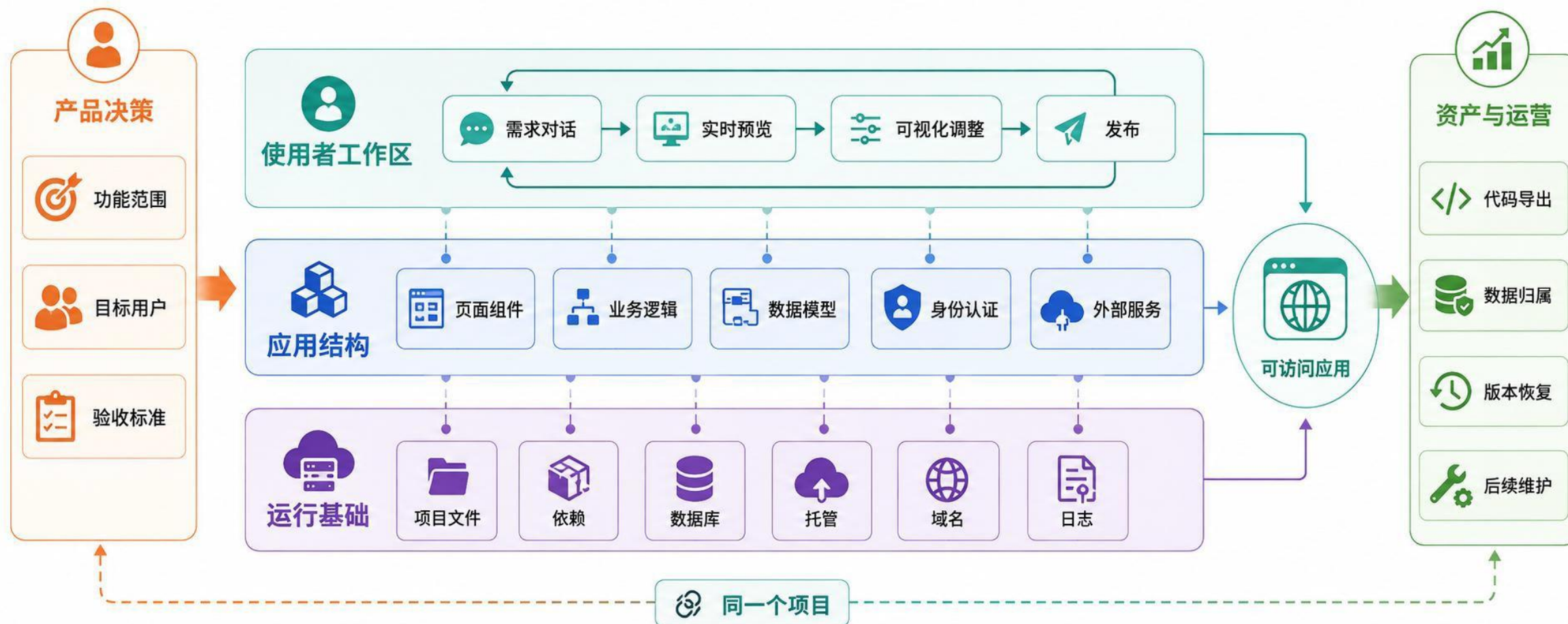
Vibe Coding 工具生态

当前Vibe Coding工具大致形成三种产品形态。**一体化应用生成平台**把界面、运行环境、托管和发布封装在统一云端环境中，适合从零开始快速制作完整小应用和 Demo；**AI原生编辑器**保留传统代码工作台形态，AI 能够理解整个项目的上下文，支持跨文件的代码修改、重构和生成，用户可以随时查看和编辑 AI 生成的全部源码；**编程Agent**以开发任务为执行单元，可以调用终端、运行测试、排查报错、处理 Issue 或提交 PR，用户设定目标和验收标准，Agent 自主拆分执行步骤、自动迭代调试，独立完成闭环开发任务。



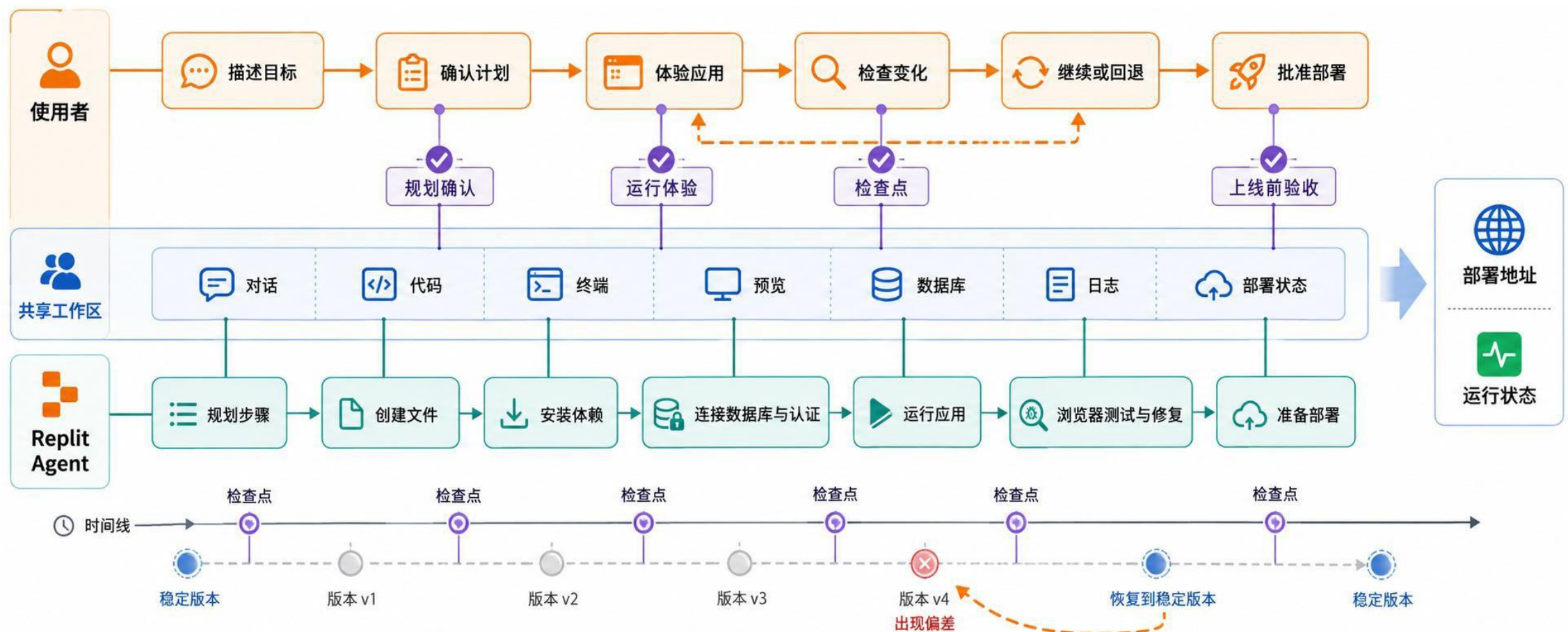
一体化应用生成平台

一体化应用生成平台将需求描述、界面生成、项目运行、数据存储、身份认证、实时预览和部署发布组合在同一工作区。使用者通过对话描述产品，通过实时预览检查结果，再借助可视化编辑或继续对话完成迭代。平台同时承担技术选型、依赖配置和云端资源连接，显著缩短从想法到可运行应用的距离。这类工具尤其适合从零创建网站、个人工具、数据应用和轻量业务系统，代码导出、数据归属和脱离平台后的维护成本，直接影响项目的长期发展空间。



Replit: AI应用和网站构建器

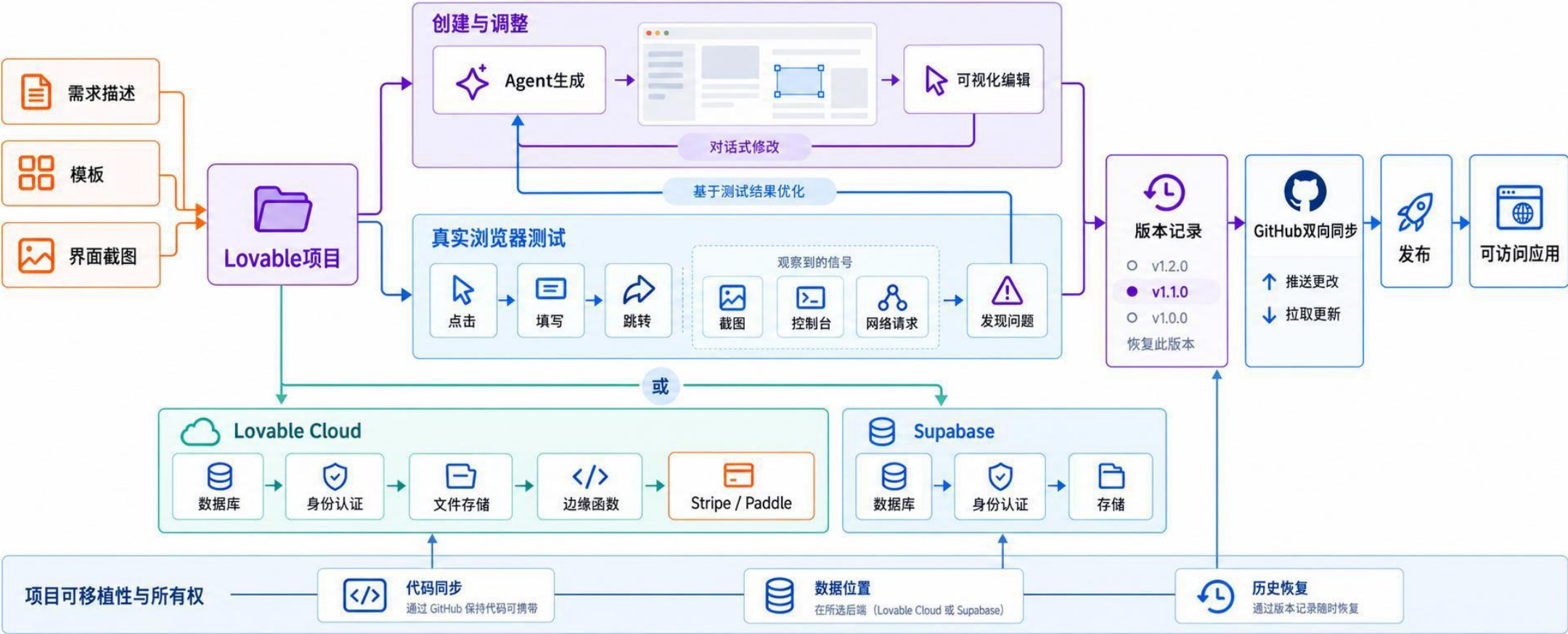
Replit 支持先在规划模式中与 Agent 讨论目标、功能范围和实施步骤，Agent 在此阶段只做分析不修改代码，确认后再切换到构建模式执行。Agent 能够创建项目文件、安装依赖，并配置数据库和身份认证。项目运行后，Agent 会使用真实浏览器测试应用，点击按钮、填写表单、检查链接和API，发现问题后自动修复，可连续自主运行长达200分钟。检查点记录每个阶段性版本，出现偏差时可一键恢复。开发完成后可在同一平台完成部署和自定义域名配置。代码、终端、运行日志和版本状态始终向使用者开放。



Lovable：把描述变成可用的应用

Lovable可从需求描述、模板或界面截图启动项目，生成后可继续对话或使用可视化编辑直接调整页面元素。内置AI Agent可在真实浏览器中测试应用，模拟点击和填写表单，发现报错后分析并尝试修复。项目可使用Lovable Cloud提供的数据库、身份认证、文件存储和边缘函数，也可连接Supabase。Lovable Cloud项目可启用内置Stripe或Paddle支付，其他后端路径需按对应集成方式配置。版本记录支持查看和恢复历史状态，GitHub双向同步为代码备份和团队协作提供接口。

。



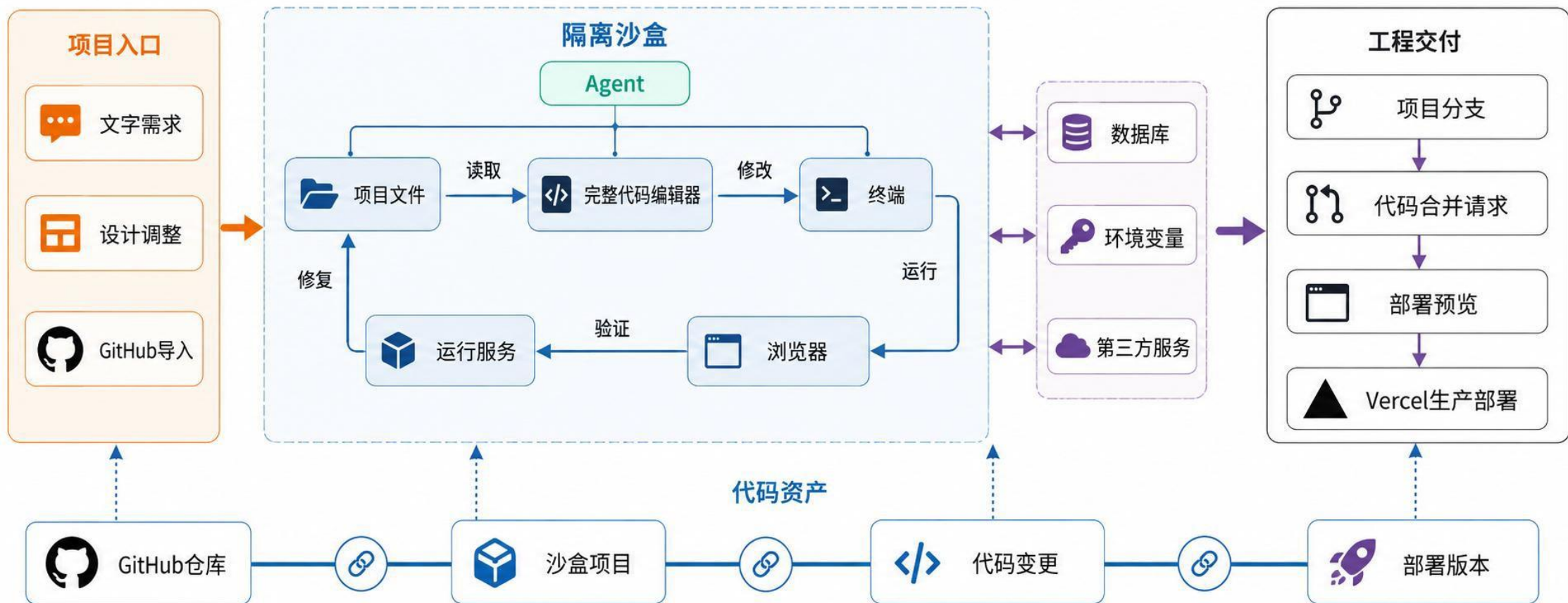
Bolt.new：基于浏览器的AI驱动全栈Web 开发

Bolt.new 支持从自然语言、Figma 设计稿或 GitHub 仓库启动项目，依托 WebContainers 在浏览器中提供完整的文件系统、依赖安装、终端和实时预览。用户可直接查看和修改代码，AI 持续自主调试并迭代。Bolt Cloud 内置 PostgreSQL 数据库、用户认证、文件存储、边缘函数、托管服务和 Stripe 支付。平台也可生成Expo移动应用项目，通过Expo Go预览，再使用Expo工具链打包发布。项目支持版本恢复和GitHub协作，发布前可运行安全审计，检查访问控制、密钥、依赖和常见漏洞。



v0：使用AI构建全栈Web应用

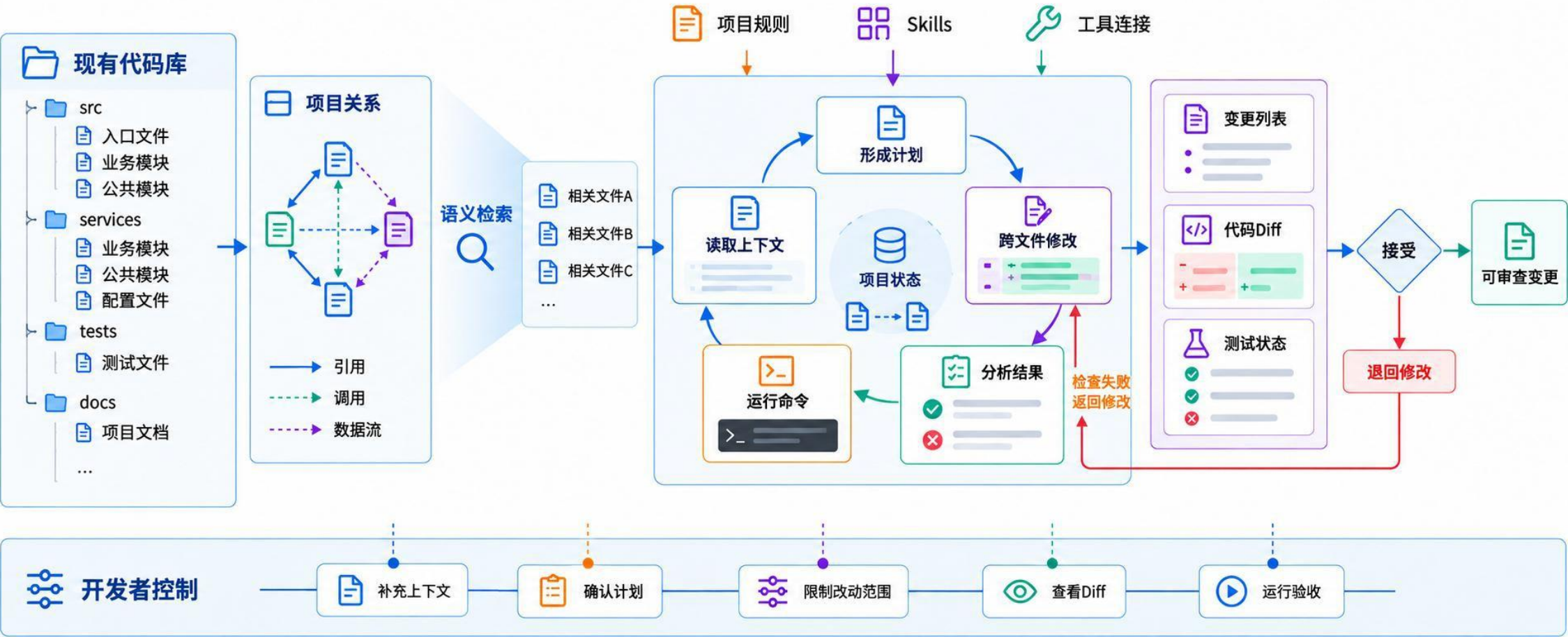
v0 已从早期的 UI 生成工具扩展为面向完整应用开发的 AI 开发工作区。用户可通过自然语言、Figma 设计稿或 GitHub 仓库启动项目，Agent 在隔离云端环境中读取和修改代码、执行终端命令、联网搜索，并通过浏览器实际操作应用、调试和修复错误。平台提供源码编辑、Design Mode、实时预览、版本快照及 Git 分支能力，并深度打通 GitHub 与 Vercel，覆盖 Preview、PR、CI、合并和生产部署流程。同时支持 Next.js、Nuxt 及 Python 服务，并可连接 Stripe、Supabase、Neon 等外部服务。v0 强调从产品想法和设计直接走向可运行、可部署的应用。



应用生成平台对比

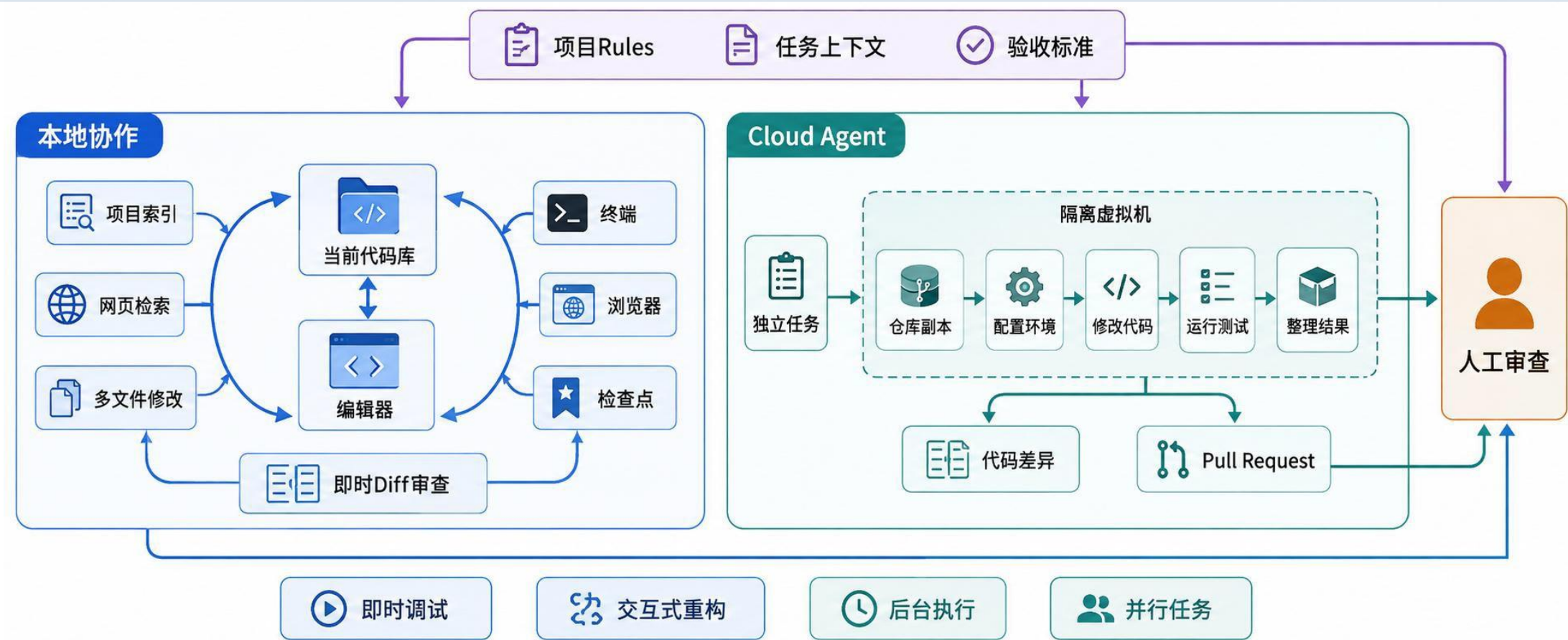
对比维度		Lovable	Replit	Bolt.new	v0
产品入口	产品模式	对话式全栈应用生成	云端开发平台 + Agent	浏览器全栈开发环境	Vercel生态AI开发工作区
	启动材料	自然语言、模板、截图	自然语言、模板、现有项目	自然语言、Figma、GitHub	自然语言、设计稿、GitHub
开发控制	技术范围	React与TypeScript Web应用	多语言、Web、后端与移动	JavaScript前端 + Node.js后端	Next.js与React全栈应用
	代码控制	可视化编辑、GitHub双向同步	完整云端 IDE、终端与Git	浏览器代码视图、终端与GitHub	完整编辑器、分支与PR
	后端与集成	Lovable Cloud 或 Supabase； Stripe、Paddle	内置后端、数据库、 存储与认证	Bolt Cloud 或 Supabase； Stripe、MCP	Marketplace 数据库与存储； Stripe、MCP
	执行与验证	多轮生成、浏览器测试与修复	长任务、实时监控与检查点	预览、运行调试与版本历史	沙盒命令、测试与权限模式
交付选择	原生移动	Web应用为主	原生iOS与Android项目	Expo项目；EAS打包发布	Web与响应式界面为主
	部署与迁移	Lovable Cloud、 GitHub、自托管	Replit发布、自定义域名	Bolt Cloud、Netlify、GitHub	Vercel部署、GitHub协作
	适用项目	非开发者、产品经理、Web MVP	多语言、后端或移动项目	JavaScript全栈与Expo原型	Next.js项目与高保真界面

AI原生编辑器在保留文件树、代码编辑、终端、调试和版本控制等专业开发界面的基础上，让Agent具备读取项目结构、检索相关代码、修改多个文件、执行命令并分析运行结果的能力。开发者可以在每轮操作中查看计划、文件变化、代码差异和测试状态，随时补充上下文或调整方向。项目规则、Skills和工具连接还能持续提供团队规范与专业流程。这类工具既适合已有代码库的迭代，也适合采用复杂技术栈、强调精细控制和长期维护的新项目。



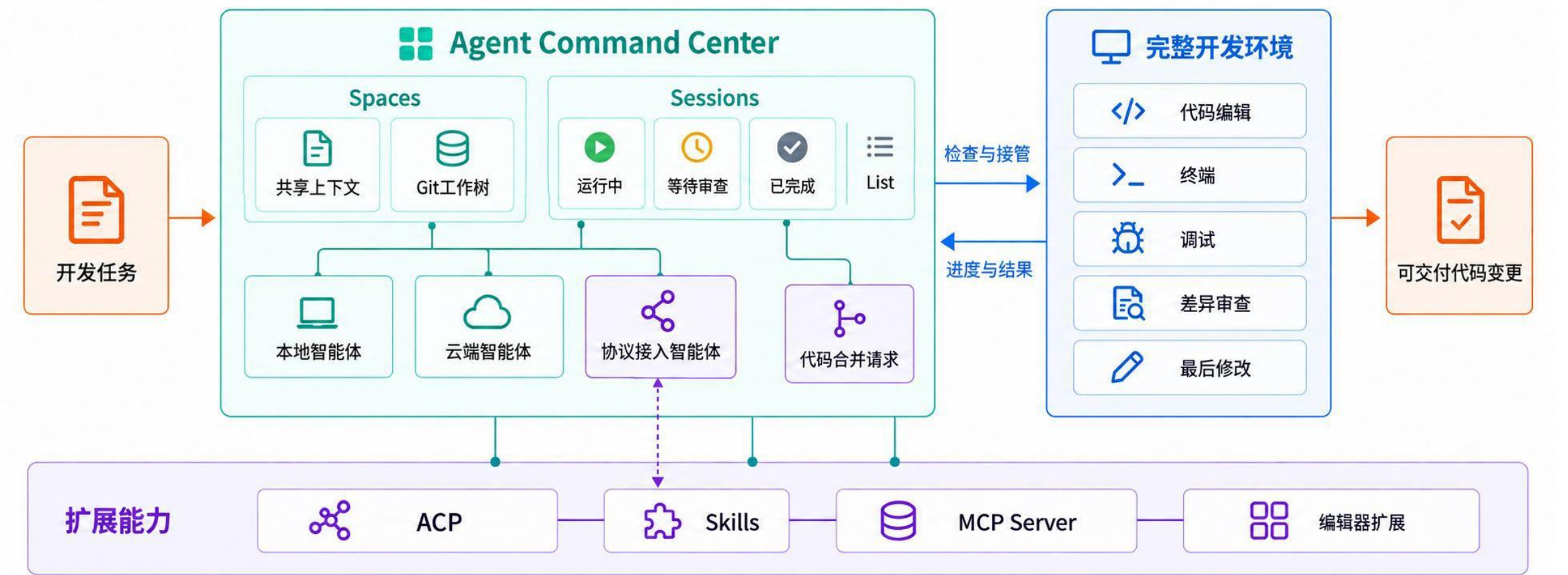
Cursor：面向大型、复杂项目的AI 编程智能体

Cursor 已从 AI 代码编辑器扩展为以 Agent 为核心的开发环境。Agent 可搜索代码库与网页、跨文件修改代码、执行终端命令，并操作浏览器测试应用、读取日志和网络请求，根据结果持续修复；检查点自动保存关键修改状态，项目规则则通过 Rules 体系持续约束开发。除本地协作外，Cloud Agents 可在独立虚拟机中并行完成开发、测试和验证，生成可合并 PR，并持续跟进 CI 与评审反馈。Cursor 还支持长程目标、事件触发、云端 Subagents 及 Self-hosted Machines，使 Agent 从辅助编码进一步走向持续运行的软件工程执行者。



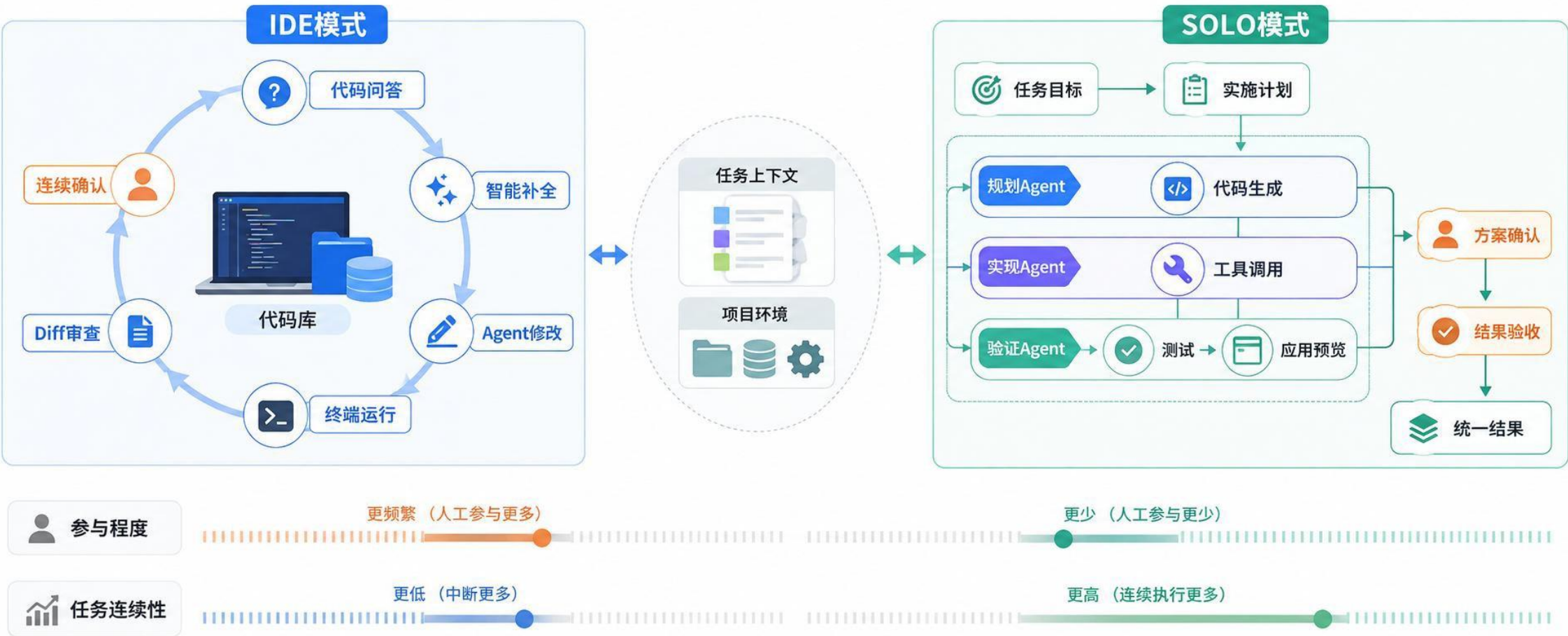
Devin Desktop(原Windsurf): Agent 指挥中心

2026 年, Windsurf 更名为 Devin Desktop, 并统一到 Devin 品牌体系下。产品保留完整 IDE 能力, 但定位已从 Agentic IDE 转向多 Agent 开发工作区: Agent Command Center 成为默认界面, 通过 Spaces、Kanban 等界面统一调度和管理本地与云端 Agent, 开发者可在 IDE 中直接审查、调试和接管 Agent 生成的代码。平台支持 ACP 接入不同 Agent, 并通过 Skills、MCP 等机制扩展工具与可复用开发流程, 使本地编辑、Agent 委派、任务并行和代码审查逐步汇聚到同一开发环境。



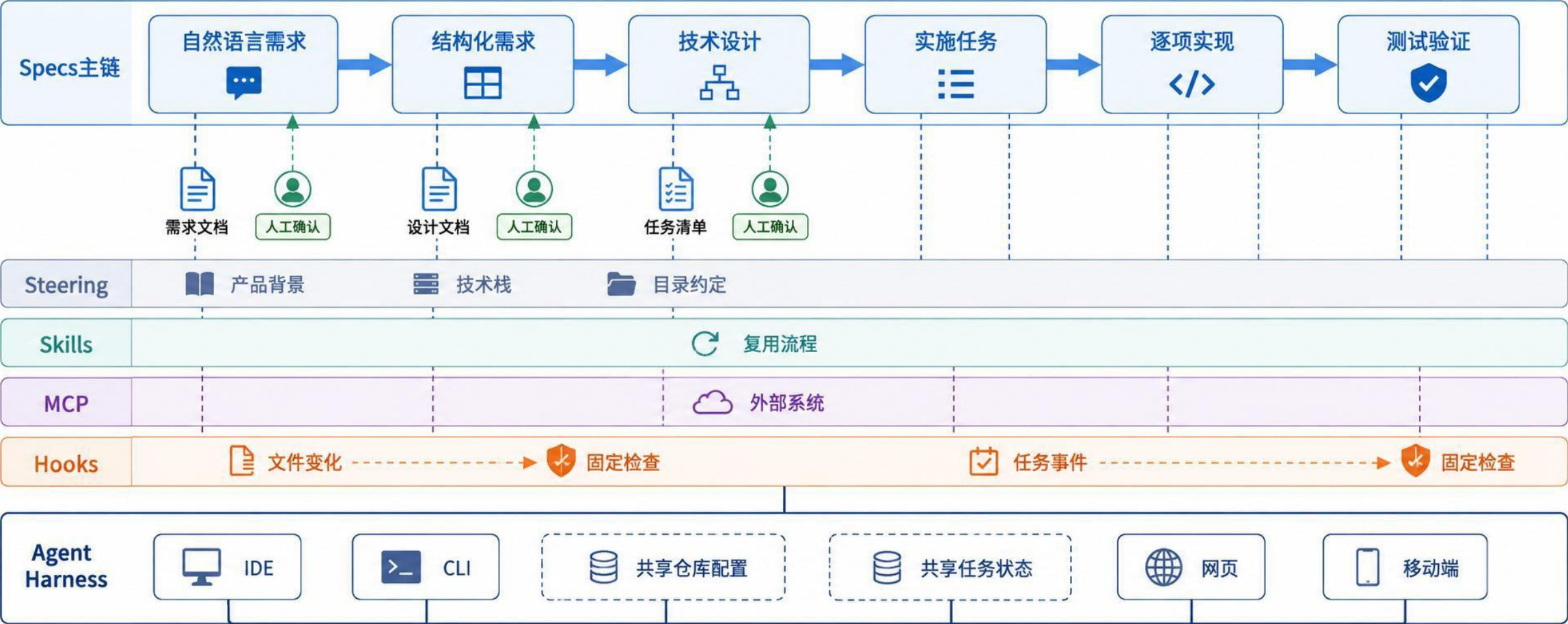
Trae Code： 一站式AI开发

TraeCode以 IDE Mode 与 SOLO Mode 提供两种 AI 编程方式。IDE Mode 将代码问答、智能补全和 Agent 能力融入传统开发流程，开发者可在编辑、终端、调试和代码审查中持续与 AI 协作；SOLO Mode 则将需求理解、规划、代码生成、工具调用、测试、调试和部署串联起来，由 Agent 自主推进端到端开发任务。平台还支持内置与自定义 Agent，不同 Agent 可独立执行或作为 Sub-agent 协同处理复杂任务。开发者可在 IDE 与 SOLO 间灵活切换，在高频结对开发与自主任务委派之间调整参与程度。



Kiro：智能体 AI 开发平台

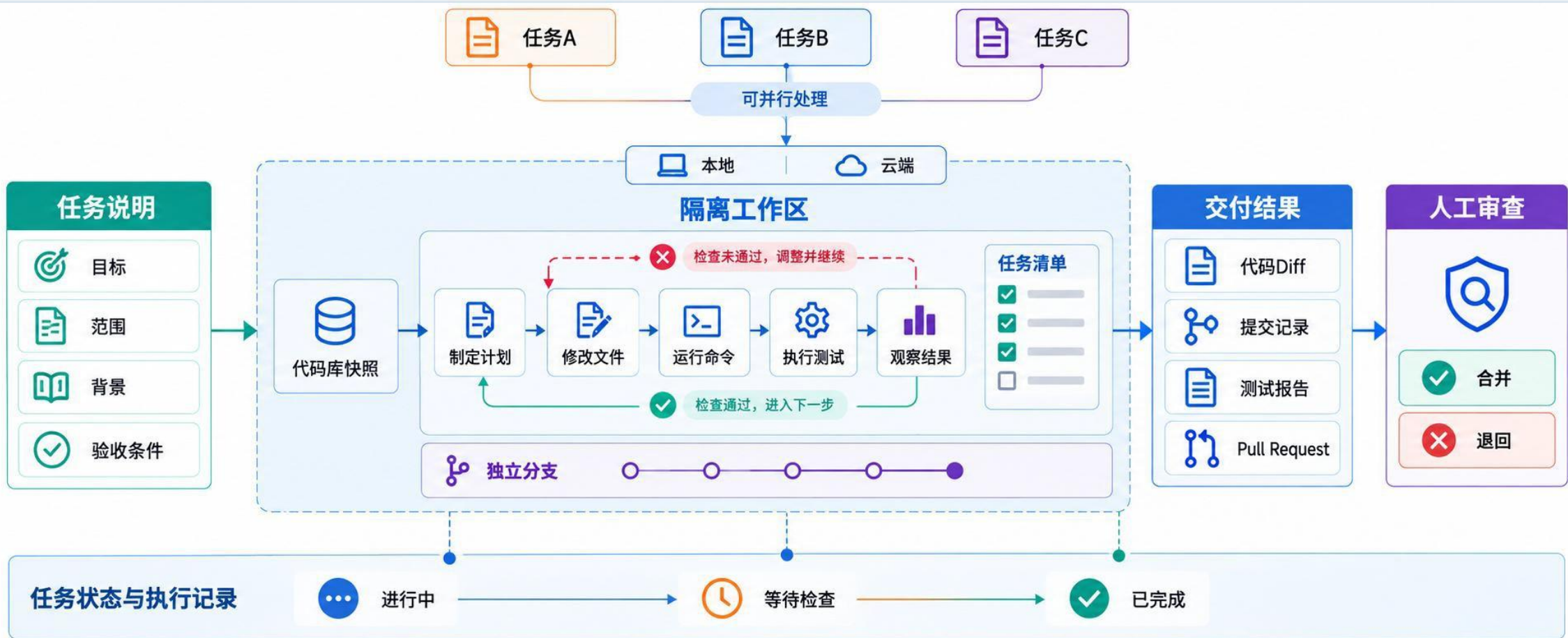
Kiro 以 Specs 为核心组织开发任务，将自然语言需求转化为需求文档、技术设计和实施任务，允许分阶段审查后再执行。Steering 持续提供产品背景、技术栈和项目规范，Skills 沉淀可复用流程，Hooks 基于文件、工具及 Agent 生命周期事件自动执行操作，MCP 连接外部工具与数据。IDE、CLI、Web 和 Mobile 由统一 Agent Harness 驱动，共享 .kiro/ 配置，支持 Custom Agents、Sub-agents 与云端会话，使同一开发任务可跨本地、终端、浏览器和移动端持续推进。



AI原生编辑器对比

对比维度		Cursor	Devin Desktop	TraeCode	Kiro
产品入口	产品形态	Agent驱动的AI开发环境	IDE与多Agent指挥中心	IDE与SOLO双模式开发工具	Specs驱动的多端Agent开发环境
	核心交互	编辑器协作、Agents窗口、云端任务	Agent Command Center、Kanban、Spaces	IDE模式持续协作；SOLO自主执行	Specs工作流与对话式开发
开发控制	任务组织	计划模式、检查点、并行Cloud Agents	统一管理本地、云端及第三方Agent	实施计划、多任务管理、子Agent分工	需求、设计、任务三阶段Specs
	上下文与扩展	Rules、Skills、Subagents、Hooks、MCP	Spaces共享上下文；Skills、ACP、MCP	项目上下文、自定义Agent、MCP	Steering、Skills、Hooks、MCP
	执行与并行	本地编辑；隔离云端虚拟机并行	本地Agent与Devin Cloud协同	本地IDE；SOLO多任务与多Agent	统一Agent Harness；本地与云端会话
	审查与恢复	Diff、检查点、PR与验证产物	集中查看Agent Diff、PR与测试结果	Diff视图、测试结果与应用预览	权限规则、检查点与Rewind
交付选择	适用场景	高频编写、重构与云端并行任务	多Agent并行与团队任务调度	在人工主导与AI自主之间灵活切换	需求复杂、强调规范与追溯的项目
	核心优势	本地协作与异步任务衔接紧密	多Agent状态、上下文和审查集中管理	双模式覆盖结对开发与任务委派	结构化规格与跨端上下文一致

编程Agent以目标为驱动，在接收明确的任务后，自主完成从规划到交付的完整流程。它被赋予目标、范围、验收条件和背景后，在本地或云端沙盒中自主读取代码、分析依赖、制定计划、修改文件、运行测试，并根据执行结果自主调整。过程中调用终端、浏览器、Git等工具，完成任务后以代码差异、提交记录、拉取请求或测试报告交付成果，供开发者审查。适合Bug修复、测试补充、接口迁移等边界清晰的任务，部分支持并行处理团队积压任务。



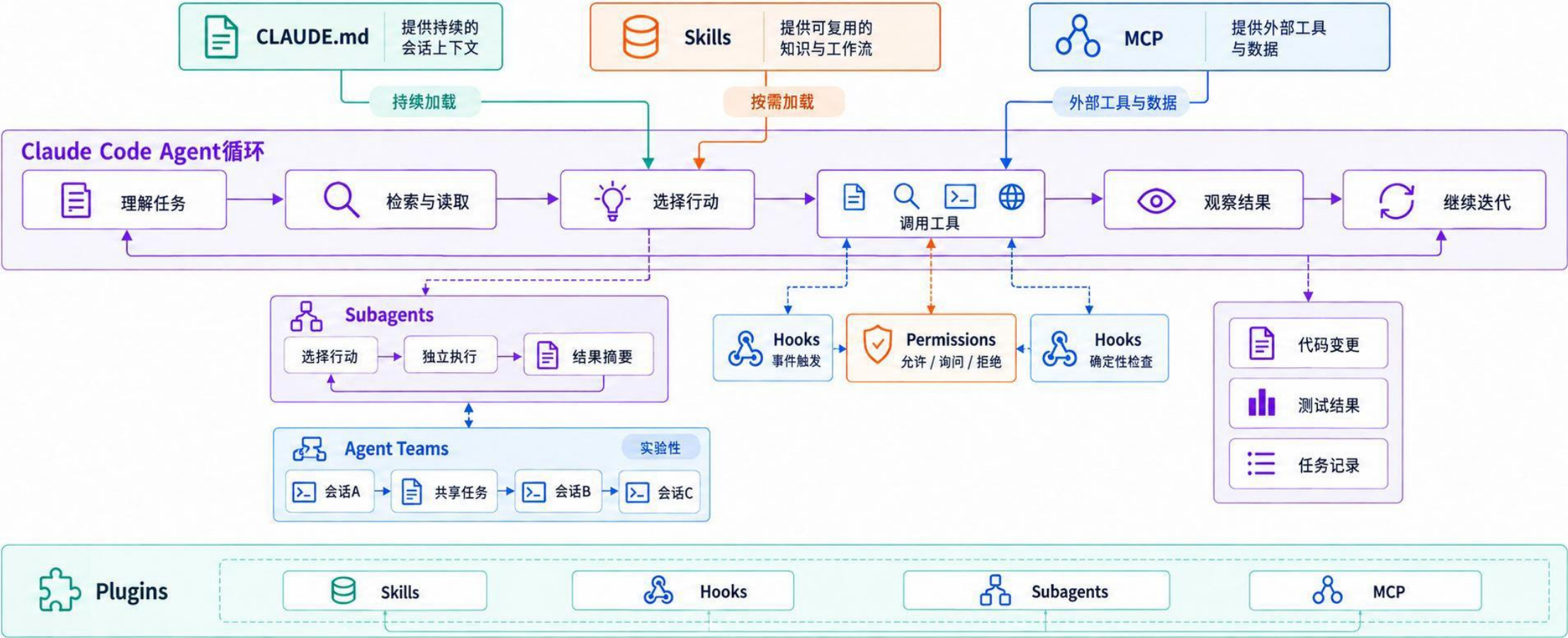
Codex：AI编程智能体和助手

Codex 的核心正在从单次代码生成转向可委派、可并行的软件工程执行。开发者既可让 Agent 直接操作本地项目，也可利用独立 Gitworktree 隔离多个任务，或将工作交给云端环境后台执行，使多个 Agent 在不干扰主工作区的情况下并行开发、测试和验证。AGENTS.md 与 Skills 为这些执行环境提供一致的项目约定和可复用能力，最终通过代码 Diff、测试结果和 提交内容 交回开发者审查。相比始终围绕 IDE 交互的产品，Codex 更强调把完整工程任务交给 Agent 持续执行。



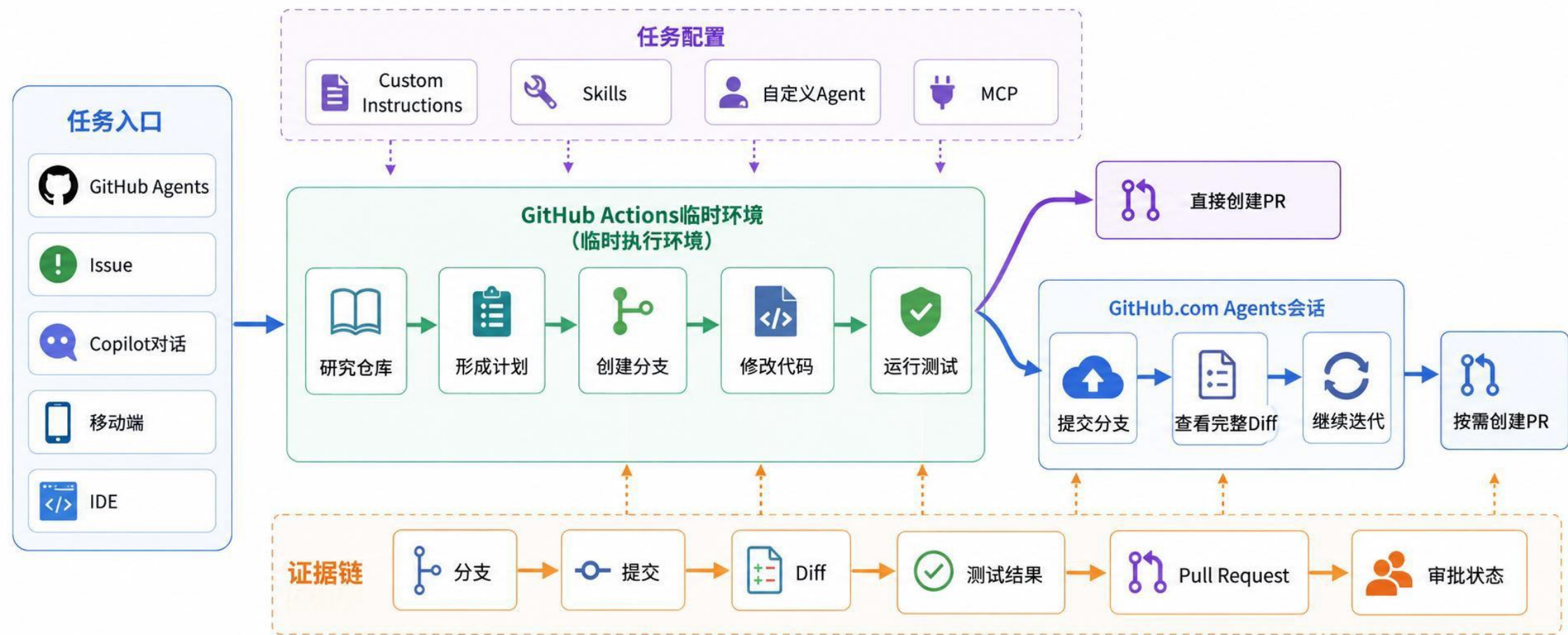
Claude Code：AI编程智能体、终端、IDE

Claude Code 提供了一套高度可组合的 Agent 工程体系。CLAUDE.md 将项目约定持续注入上下文，Skills 把知识与工作流封装为可复用能力，Hooks 在关键生命周期事件中执行确定性检查，MCP 扩展外部工具，Permissions 则控制 Agent 的行动边界。在此基础上，Subagents 可将任务拆入独立上下文，Agent Teams 进一步让多个 Claude Code 实例共享任务并协同工作。团队可以围绕自己的代码库和工程流程定制 Claude Code 的 Agent Harness。



GitHub Copilot cloud agent: GitHub 原生编程智能体

GitHub Copilot cloud agent 于 2026 年 4 月由 Copilot coding agent 更名，进一步融入 GitHub 原生开发流程。任务可从 Agents、Issue、IDE、Mobile 等入口发起，在 GitHub Actions 临时环境中研究代码库、制定计划、修改代码并运行测试。GitHub.com 上 Agents 会话可先在独立分支中持续迭代、查看 Diff，再按需创建 PR，部分入口也可直接创建 PR。Copilot Automations 可按计划或仓库事件自动启动 Agent，将 Issue、代码修改、测试与 PR 串联成持续自动化流程。



Devin : 云端自主AI软件工程师

Devin 以独立 Session 承接完整软件工程任务，可研究代码库、制定计划、编写代码、运行命令、操作浏览器、测试应用并提交 PR。Ask Devin 用于探索代码和规划任务，再转交 Agent 模式持续执行。每个 Session 保留完整的操作、Diff、测试与上下文轨迹，并可利用历史会话经验推进后续工作。多个 Devin 可并行处理跨仓库、长周期项目，分别完成开发和验证。端到端测试可直接操作真实应用并生成视频证据，最终结果不仅有代码 Diff，也有可视化的功能验证。

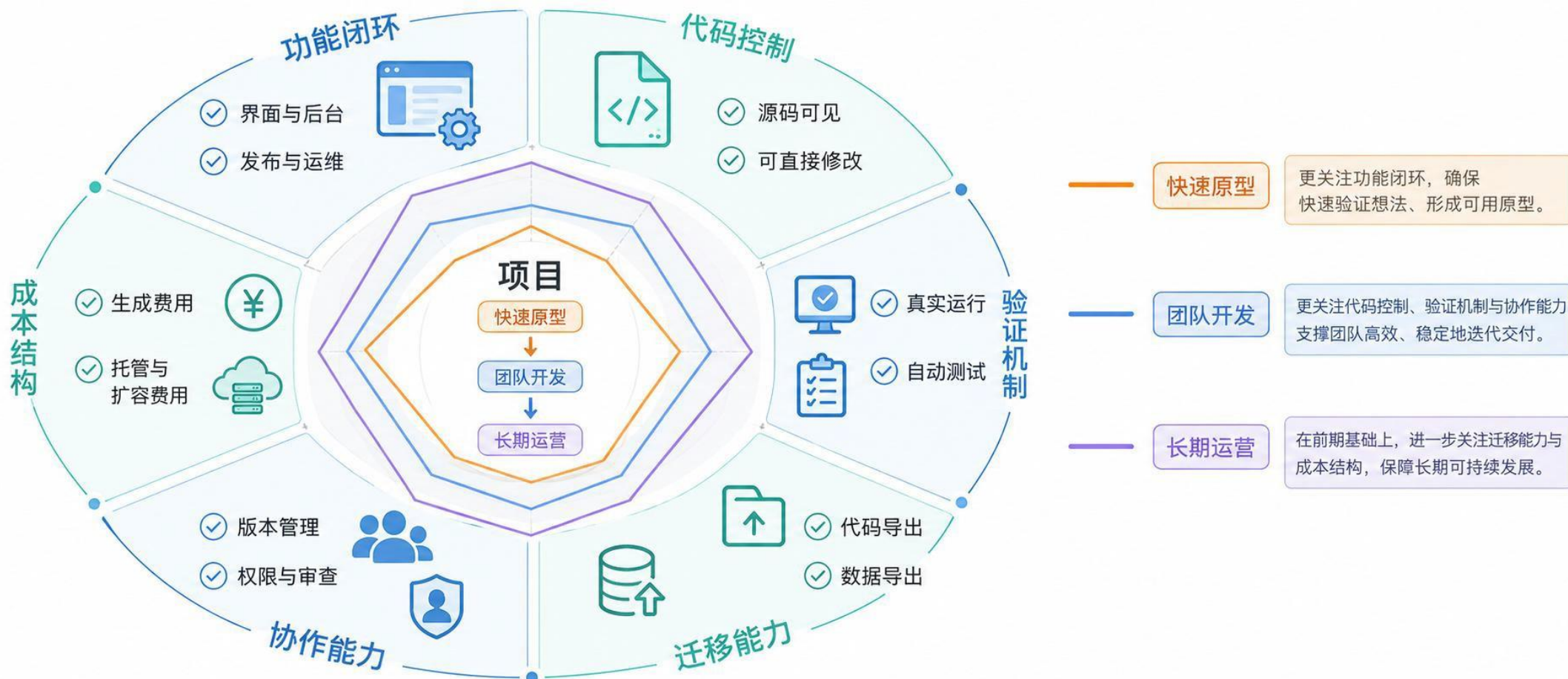


编程Agent 对比

对比维度		Codex	Claude Code	GitHub Copilot cloud agent	Devin
产品入口	产品形态	本地与云端协作的编程Agent	终端优先并延伸至桌面和IDE的编程Agent	融入GitHub流程的云端异步Agent	运行于独立云端环境的自主软件工程Agent
	任务入口	桌面应用、CLI、IDE与云端任务	CLI、Desktop、VS Code、JetBrains与网页	Agents页面、Issue、Copilot对话、IDE与集成	Web、Devin Desktop、Slack、Teams与任务集成
执行控制	执行环境	本地目录、Git worktree或云端容器	本地、Git worktree、云端或SSH远程环境	GitHub Actions临时云端环境	每个Session使用独立云端环境
	工作流与并行	连续协作、后台任务与子Agent并行	动态任务循环、并行Session与Agent teams	研究、计划、分支迭代与自动化触发	自主规划、执行和测试；多Session并行
	规则与扩展	AGENTS.md、Skills、MCP与自定义Agent	CLAUDE.md、Skills、Subagents、Hooks、MCP	Custom instructions、Skills、Hooks、MCP与Custom agents	Knowledge、Playbooks、Skills与MCP
	验证与审查	Diff、命令与测试结果、提交或PR	Diff、测试、可视化预览与PR监控	Session日志、Diff、自动检查与代码审查	测试报告、操作录像、PR与Devin Review
交付选择	主要交付	本地变更、提交、分支或PR	本地变更、提交或PR	分支变更，可按需创建PR	PR、测试证据与任务结果
	适用场景	跨本地、云端与多工具协作的开发任务	终端深度协作和复杂代码库任务	GitHub内的Bug修复、增量功能与积压任务	长时间自主执行、端到端验证和团队任务队列

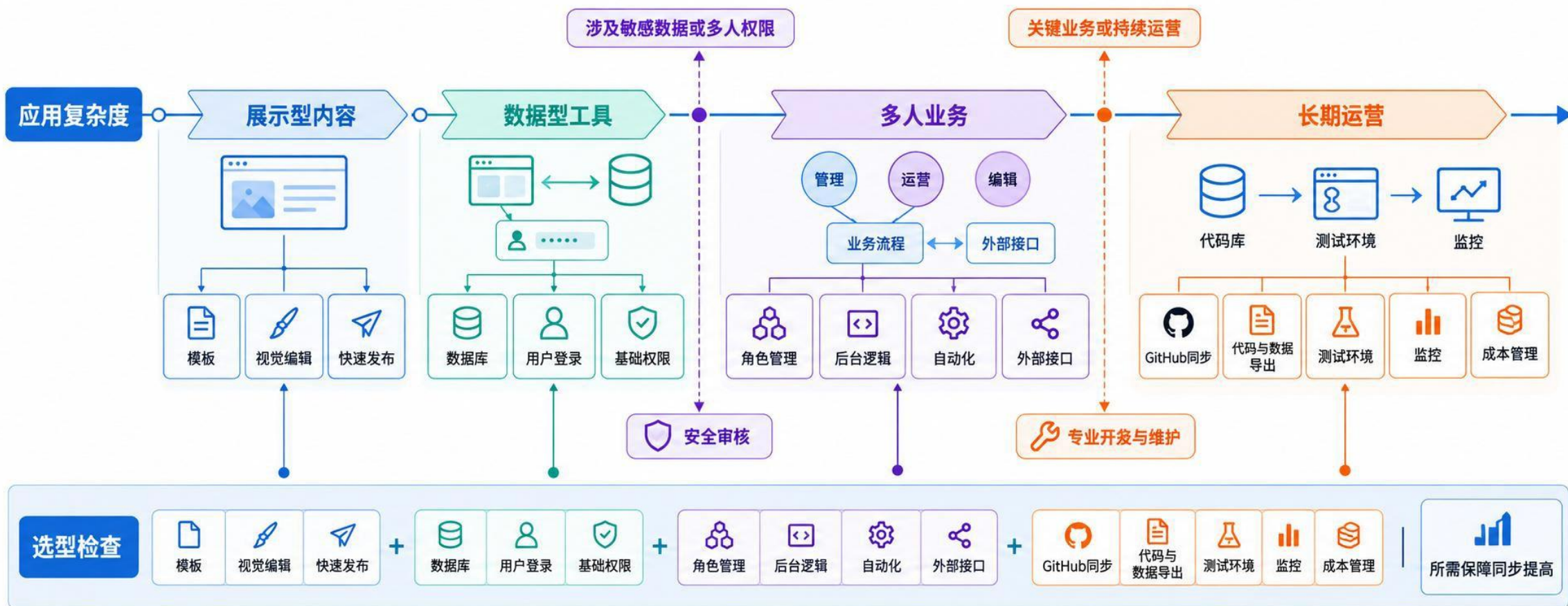
工具选型的六个维度： 闭环、控制、验证、迁移、协作与成本

工具选型需要同时考察六项能力。**功能闭环**决定项目能否覆盖界面、数据、认证和发布；**代码控制**决定使用者能够深入修改到什么程度；**验证机制**决定生成结果是否经过运行和测试；**迁移能力**关系到代码与数据的长期归属；**协作能力**影响团队开发和版本管理；**成本结构**决定项目扩大后的可持续性。真实项目还需要评估上线、维护、故障处理和持续迭代能力，并据此形成项目自己的优先级排序。



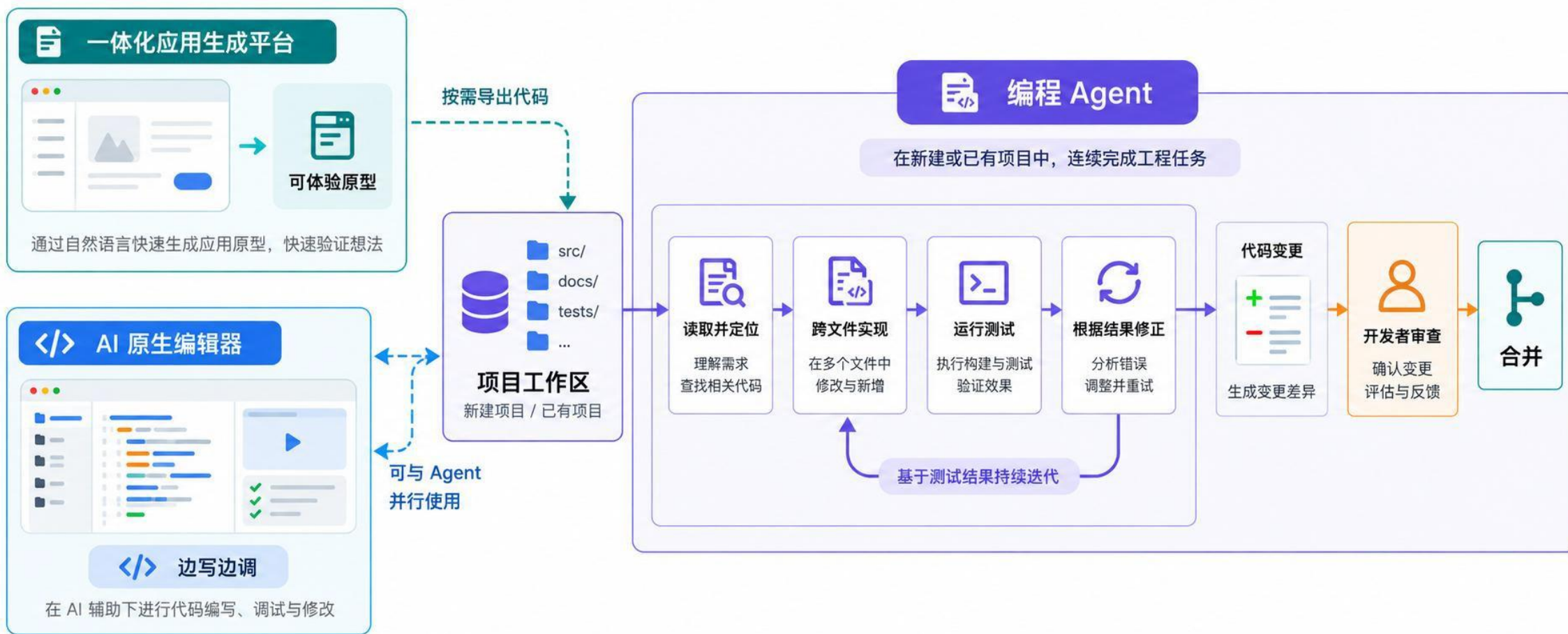
非开发者工具选择：应用复杂度决定平台能力组合

非开发者通常可以从一体化应用生成平台起步。展示型网站重点考察模板、视觉编辑和发布；带数据的小工具需要数据库、用户登录和基础权限；多人业务应用还需要角色管理、后台逻辑、自动化和外部接口；长期运营产品则需要GitHub同步、代码与数据导出、测试环境、监控和成本管理。不同功能层级对应不同的平台能力需求，涉及敏感数据或关键业务时，仍需引入专业开发、安全审核和持续维护机制。



开发者工具选择：任务特征决定工具匹配策略

开发者可根据任务特征选择不同工具。一体化应用生成平台适合快速搭建可体验的产品原型；AI 原生编辑器适合开发者边写边看、即时调试和调整代码。编程 Agent 则能连续完成读取现有代码库、定位相关模块、跨文件开发或修复、运行测试，再根据结果迭代。在实际团队中，三类工具可组合使用，目前编程Agent正越来越多地进入真实项目的日常开发，但任务边界、代码审查和最终合并仍由开发者把关。



目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

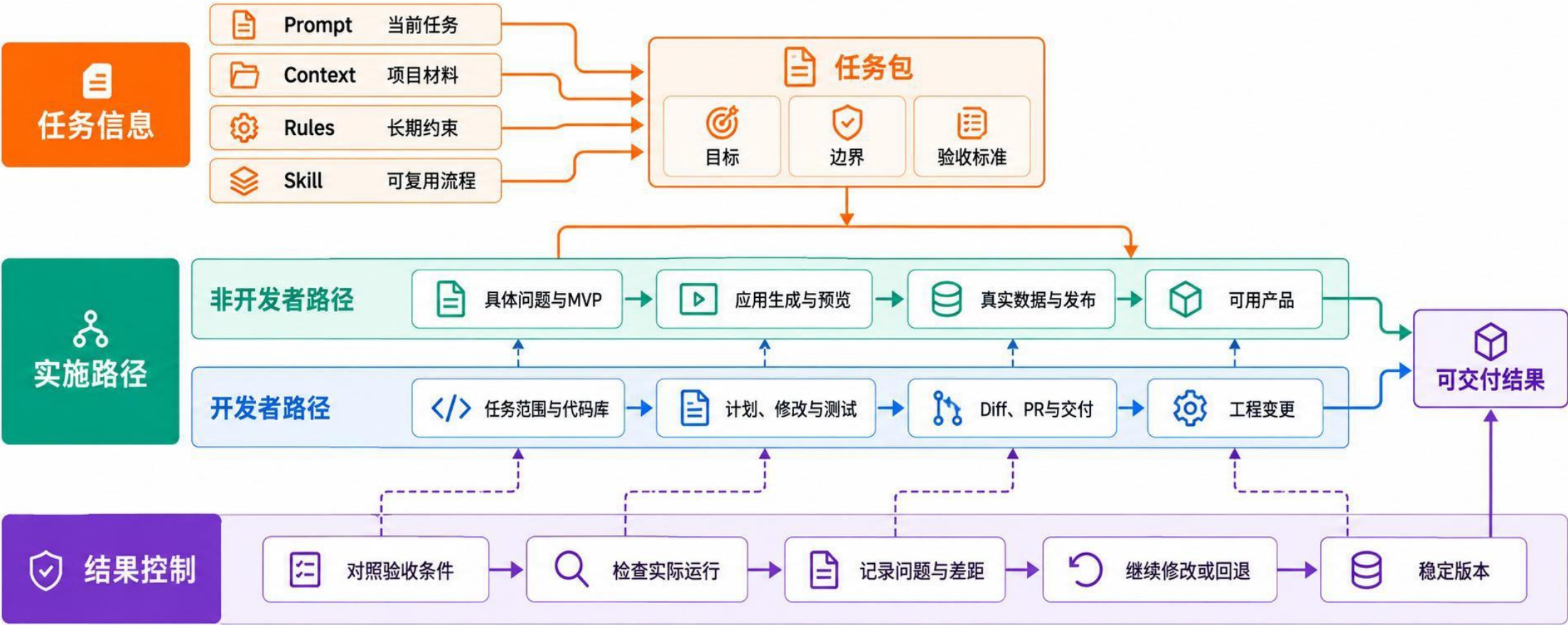
04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

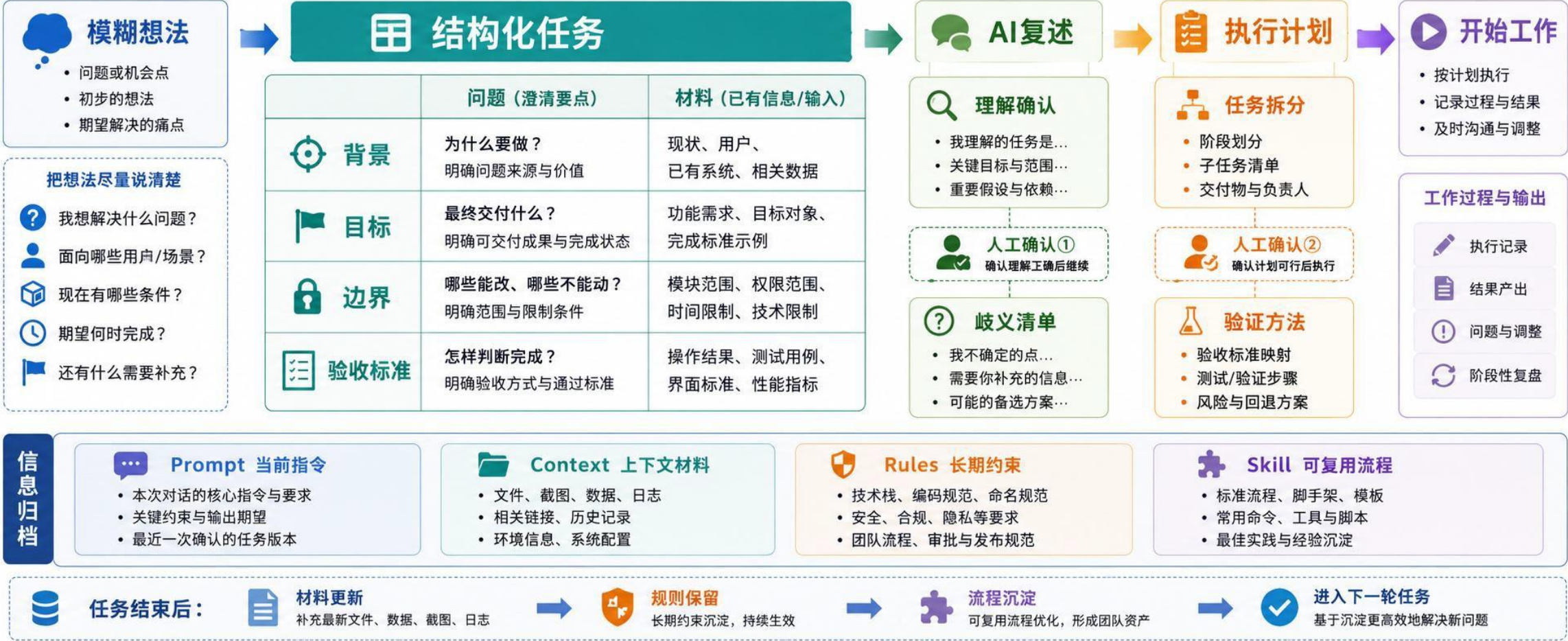
Vibe Coding 的实践框架

一个可靠的Vibe Coding流程包含三个要素：任务信息、实施路径和结果控制。**任务信息**用于说明当前目标、项目上下文、长期规则和可复用方法；**实施路径**根据使用者与项目类型展开，非开发者从产品问题和最小版本开始，开发者从任务范围和现有代码开始；**结果控制**贯穿每轮修改，使用者依据验收标准检查结果，保存稳定版本，并在偏离目标时回退。三部分共同决定生成结果能否进入真实使用。



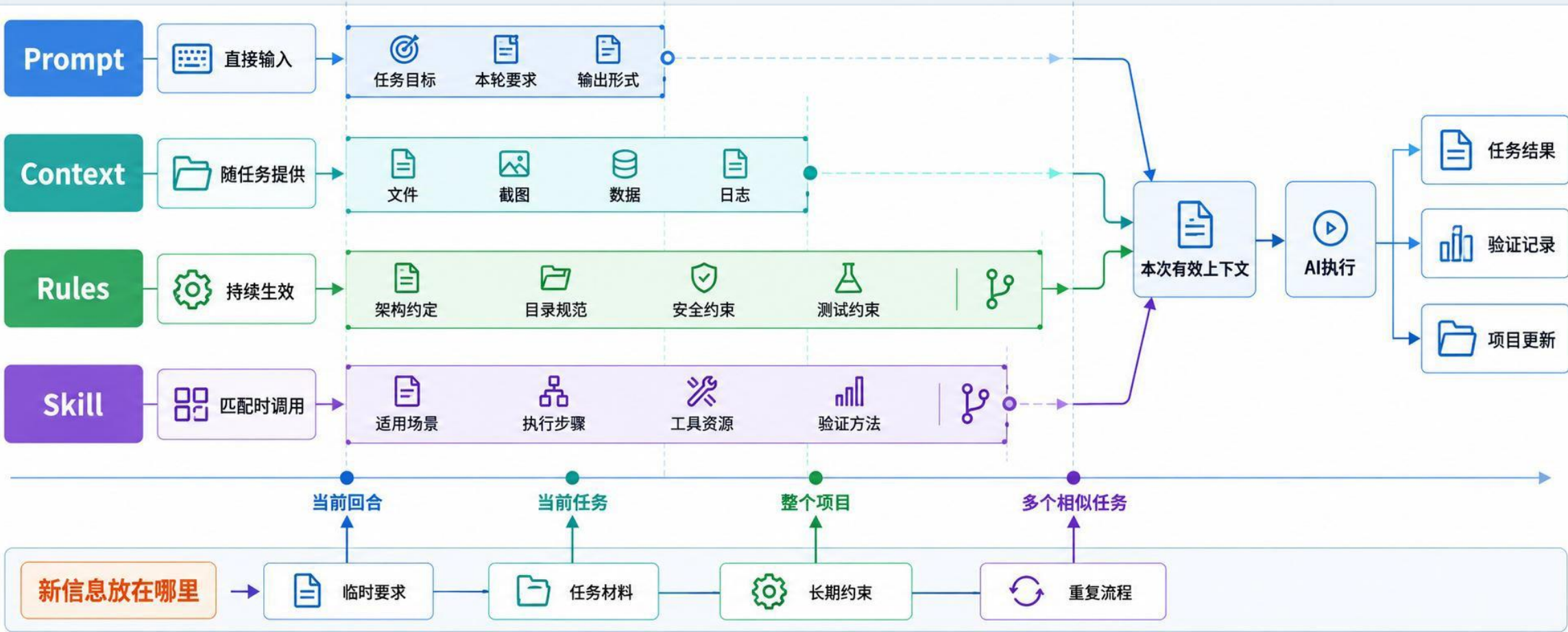
任务信息： 背景、目标、边界与验收标准

Vibe Coding不需要追求复杂的提示词格式，但必须向AI提供完成任务所需的信息。**背景**说明当前项目、目标用户和问题场景，**目标**描述本轮要产生的可观察结果，**边界**规定不能修改的功能和必须遵守的限制，**验收标准**说明怎样证明任务完成。任务较复杂时，应先让AI复述理解并给出计划。清楚的任务说明可以减少模型自行补全需求，也方便使用者在完成后逐项核对。



任务信息的四种载体： Prompt、Context、Rules 与 Skill

Prompt表达当前要完成的任务，通常在当前任务或会话中使用。**Context**提供与任务有关的文件、截图、数据结构、日志和参考资料。**Rules**保存项目长期遵守的架构、目录、安全和测试约束。**Skill**封装某类重复任务的执行步骤、工具资源和验证方法。四类信息共同进入AI的工作过程，但生命周期和用途不同。把稳定内容留在项目中，可以减少反复解释，也能避免长对话中出现版本冲突。



Skill工作方式：把重复经验变成可复用流程

Skill可以理解为供AI调用的任务操作手册。一个完整Skill会说明适用场景、需要的输入、执行步骤、可使用的脚本或模板、失败后的处理方式以及完成后的验证标准。使用者可以先采用来源可信的内置或官方Skill。当某个流程反复出现，或者现成能力不符合项目规范时，再把自己的步骤与检查方法沉淀为专属Skill。Skill需要测试、审查和更新，才能长期保持可靠。



非开发者Vibe Coding流程

非开发者的 Vibe Coding 可以从一个具体问题和最小可用版本开始，再通过 AI 补齐需求，形成产品说明和初步构建计划。首次生成前提供必要的视觉参考和真实内容，用示例数据快速完成可点击原型，并通过小步迭代逐渐跑通核心流程。主流程稳定后，再逐步接入真实数据、用户权限和外部服务，通过实际操作测试、记录问题并在必要时恢复稳定版本。完成上线前验证后正式发布，再根据运行数据和用户反馈持续迭代。

用户反馈 · 问题回流 · 下一轮迭代



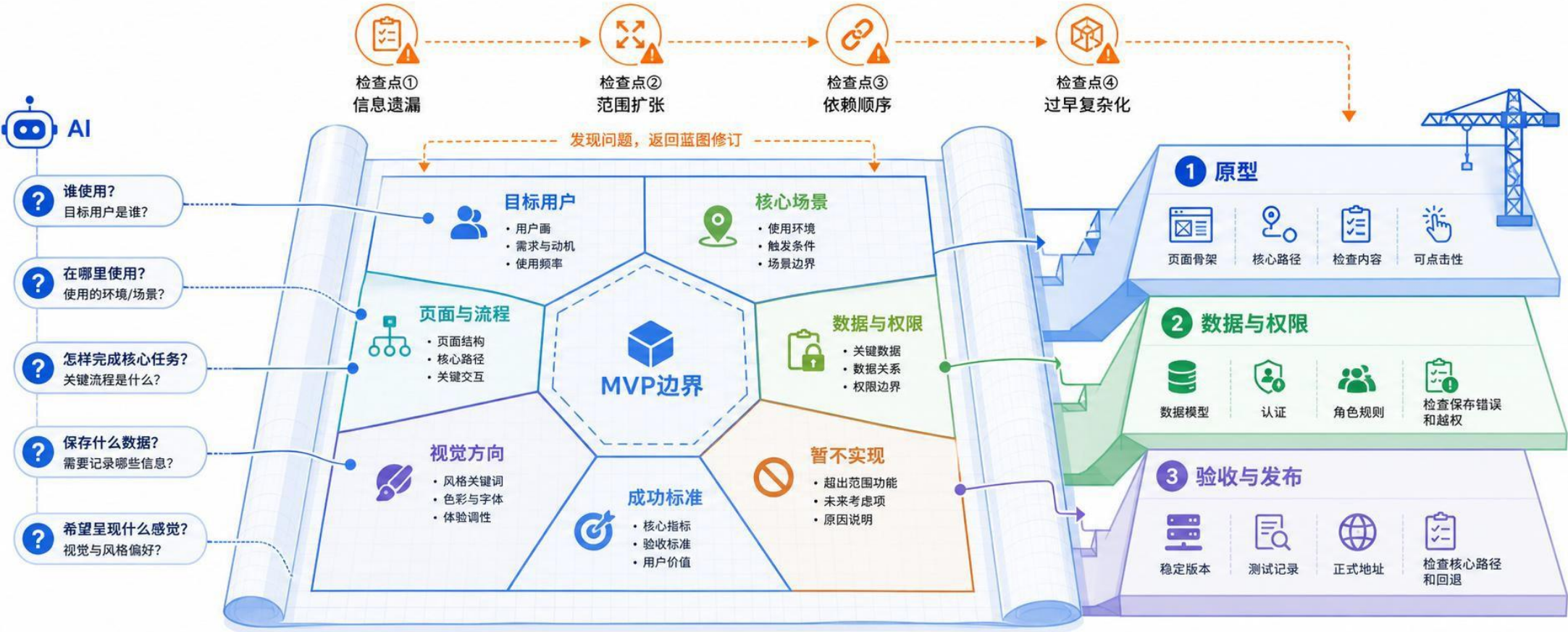
第一步： 从一个具体问题开始， 先做最小版本

Vibe Coding 项目应从明确的问题和使用场景出发，而不是从功能清单开始。首先需要确定目标用户、实际问题和预期改善的环节，再据此划定最小可用产品（MVP）的范围。首个版本不追求功能完整，而是用尽可能少的功能跑通一条核心流程，验证产品是否真正解决问题。例如开发活动管理工具，可以先实现“创建活动 → 用户报名 → 查看名单”的流程。清晰的范围既能减少 AI 在需求理解和实现过程中的偏差，也便于根据实际结果判断产品价值，再逐步扩展后续功能。



第二步：把需求整理成产品说明，再制定构建计划

确定 MVP 的范围后，应先让 AI 通过提问补齐需求中的关键信息，并将确认后的结论整理成产品说明。内容通常包括目标用户、使用场景、核心流程、首版页面、数据需求、视觉方向、暂不实现的功能和成功标准，用于明确后续开发的范围与目标。在此基础上，再由 AI 拆解分阶段构建计划，明确每一步的任务、产物和检查方式。使用者需要重点确认业务流程是否完整、是否偏离 MVP 范围，以及是否过早引入不必要的复杂功能。



第三步： 提前确定视觉规范， 并尽量使用真实内容

产品的视觉规范和实际内容直接影响页面结构，不宜全部留到开发末期再调整。生成前应向 AI 提供基本视觉要求，包括品牌色、字体风格、信息层级和移动端适配，并尽量使用真实的标题、按钮文案、图片比例和示例数据；涉及业务数据时可先脱敏。修改已有页面时，可提供截图或设计稿，明确调整区域、预期变化及必须保留的内容。相比“更高级、更好看”这类主观描述，视觉参照和真实内容能减少 AI 对设计意图的猜测，也让首轮结果更接近最终产品。



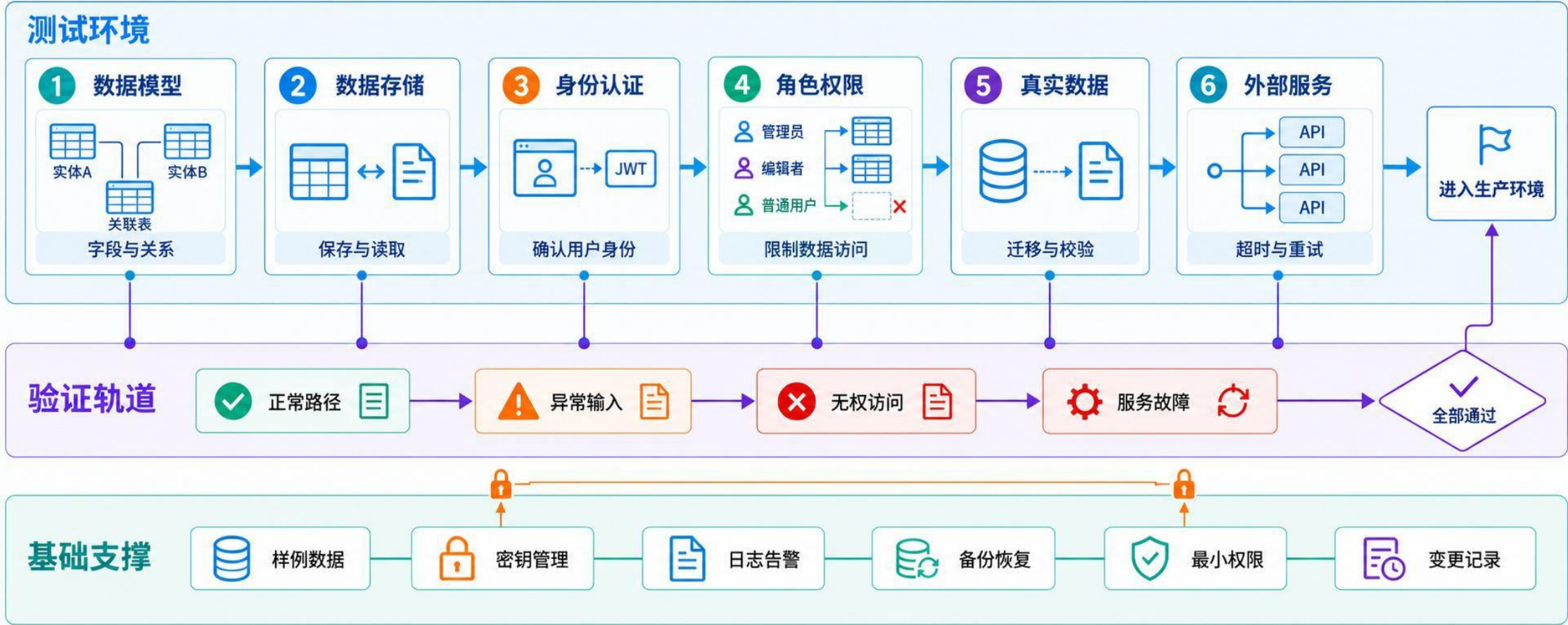
第四步：先做出可点击原型，再逐步增加功能

面向用户的小型应用可以先使用示例数据生成可点击原型，重点验证页面顺序、内容表达和主要操作是否符合预期，再逐步接入真实数据和后台能力。进入正式构建后，可将应用拆成若干可以独立检查的小阶段，每一轮只增加一个主要功能，并在继续下一轮之前确认页面体验和运行结果。小步推进不仅便于及时发现和定位问题，也能减少一次修改对整个项目的影响；如果生成结果偏离预期，还可以恢复到最近的可用版本后重新调整。



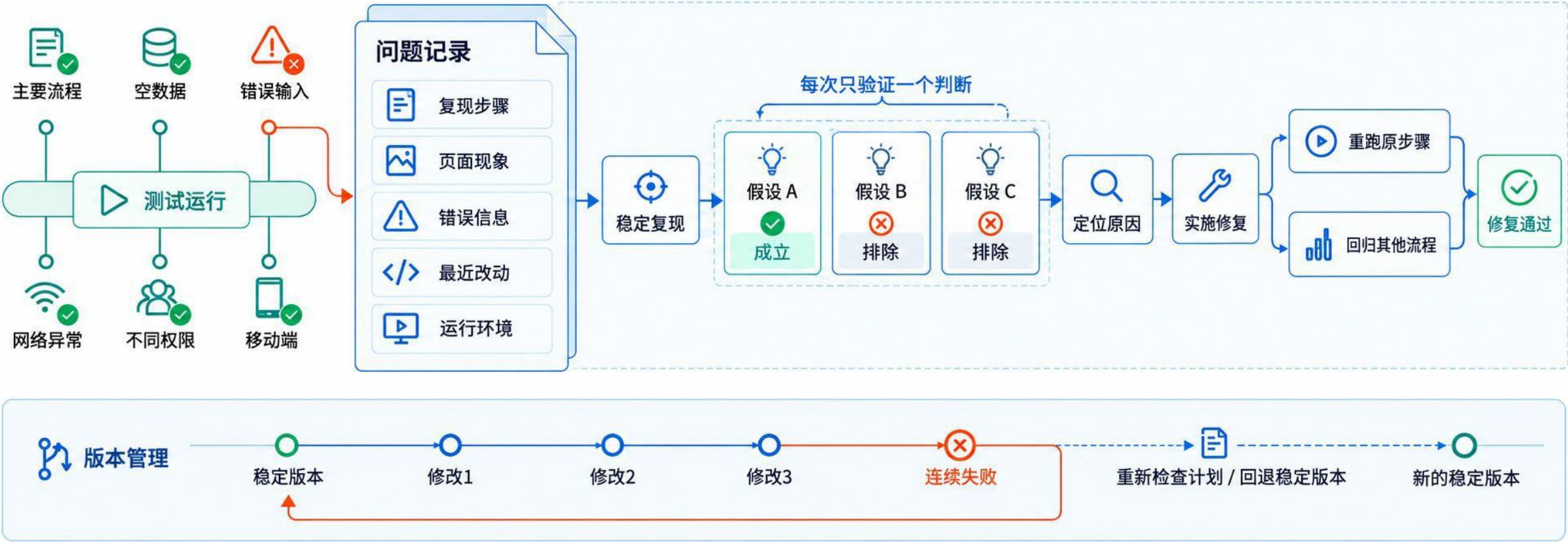
第五步：逐步接入真实数据、用户权限和外部服务

核心流程稳定后，再逐步接入数据库、用户身份和外部服务。明确需要保存的数据、数据关系及访问规则，在测试环境中使用示例数据验证。涉及多用户时，加入登录和权限，确认每类用户的操作范围。完成后端基础能力后，再接入真实数据、配置邮件、地图、支付等外部服务，并处理调用失败、重试等异常。每增加一层真实能力，都应同步检查正常、错误和无权限等状态。即使不负责具体开发，至少也需要了解数据存储位置、访问权限及外部服务故障对产品的影响。



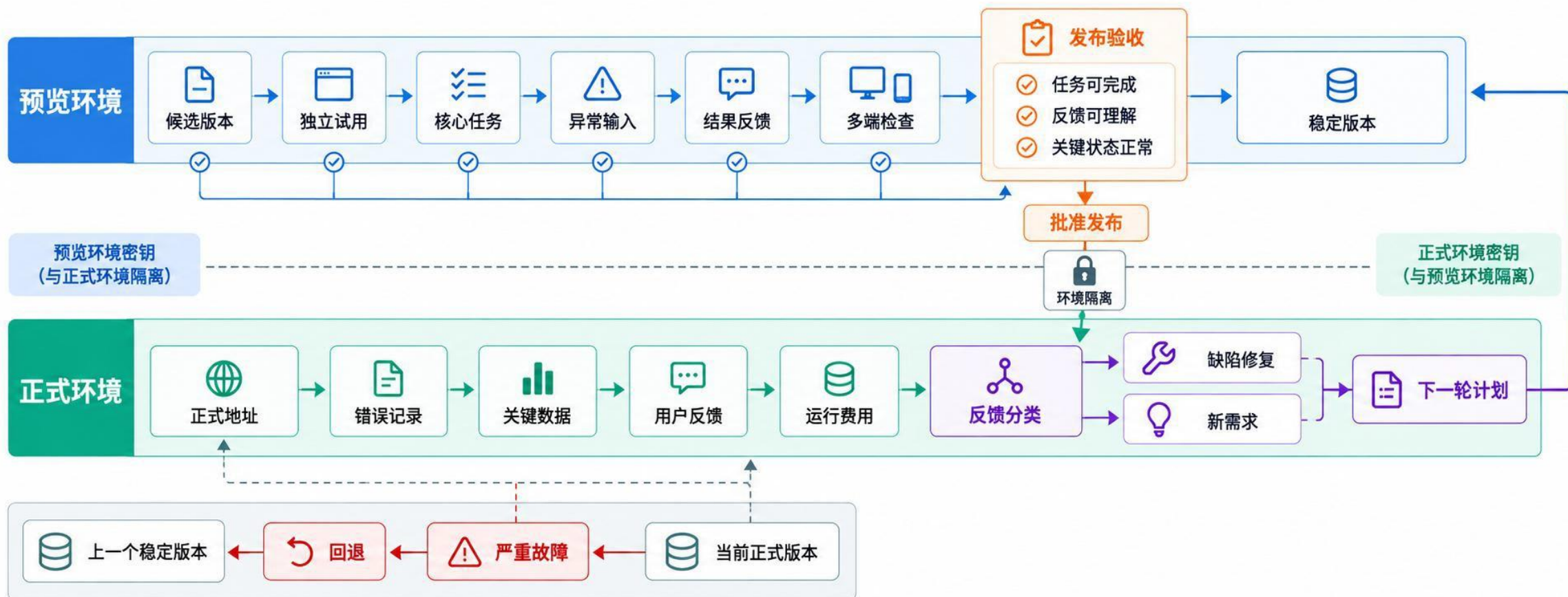
第六步：用真实操作测试，出现问题时定向排查

非开发者无需读懂全部代码，也可以通过可重复的操作步骤验证软件是否正常。测试不仅要跑通主要流程，还应覆盖空数据、错误输入、网络异常、不同用户权限和移动端显示等情况。出现问题时，先记录触发步骤、页面现象、错误信息和最近改动，再让AI分析可能原因，每次只验证一个判断。多轮修改仍未改善时，应停止继续叠加补丁，重新检查计划或恢复到最近稳定版本。修复完成后重新执行原来的操作，确认问题消失且已有功能仍然正常。

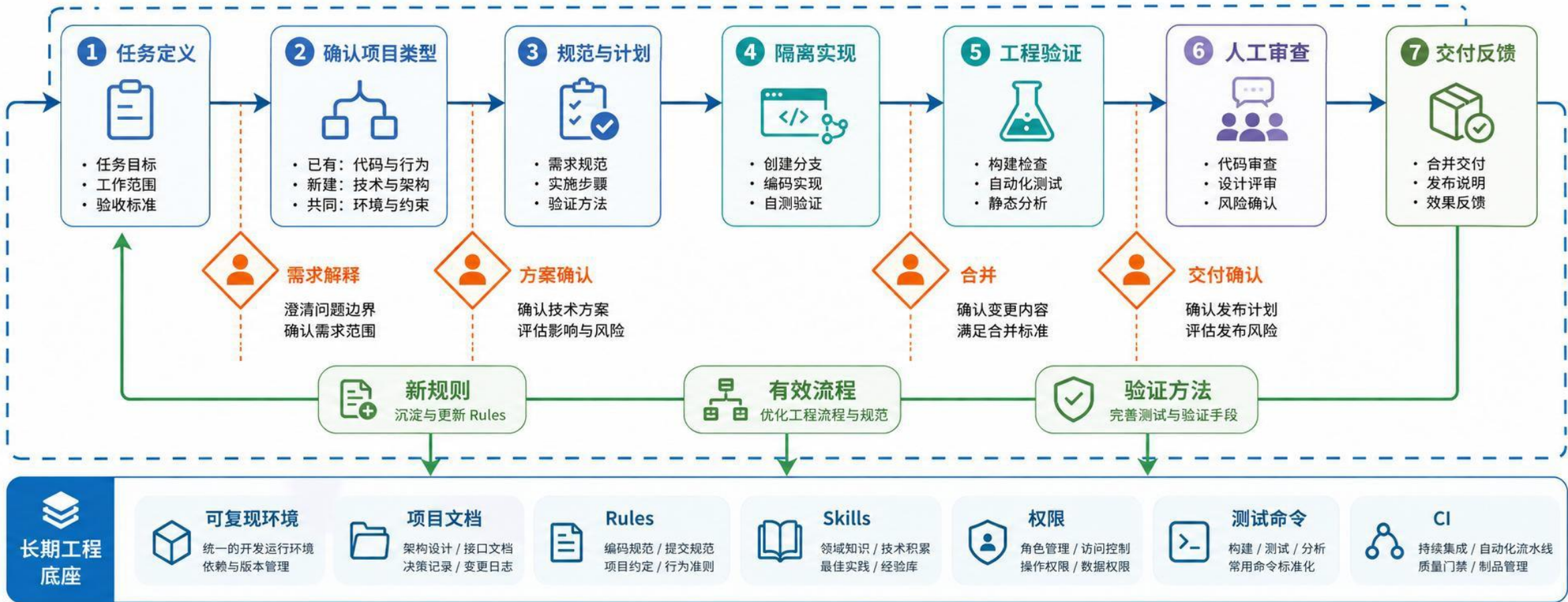


第七步： 正式发布， 并根据真实反馈持续迭代

正式发布前，可以邀请未参与构建的人测试使用，观察用户在哪一步遇到困难、是否出现预期之外的输入，以及能否理解操作成功或失败后的反馈。通过实际使用验证后，再将预览版本发布到正式环境，并保留对应的稳定版本。上线后仍需持续关注错误记录、关键数据、用户反馈和运行费用。后续修改应先在预览环境完成测试，再决定是否更新正式版本。真实用户发现的问题和新的需求则进入下一轮产品调整与构建计划，使“构建-测试-发布-反馈-迭代”形成持续循环。



开发者需要先明确任务目标、工作范围和验收标准。随后确认项目类型，已有项目需要了解相关代码和现有功能，新项目则需要确定技术选型、基础架构和开发环境。在此基础上，用 SDD 整理需求规范、实施步骤与验证方法。方案确认无误后，Agent 在隔离环境中分批实现，每完成一部分就执行相关验证，出现失败时，根据日志和测试结果定位原因。验证通过后，开发者审查代码变更及潜在风险，完成合并、交付，并将沉淀的经验补充至项目文档与规则库。



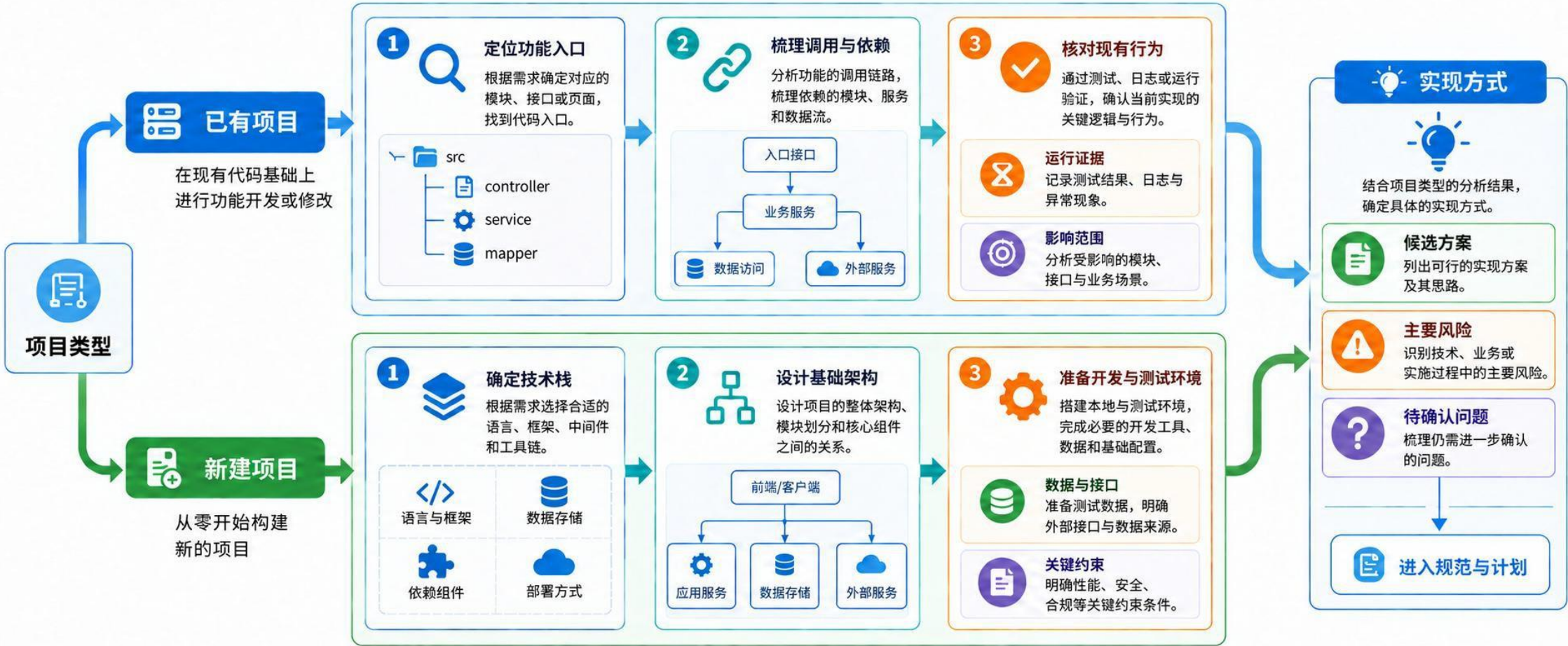
第一步：明确任务范围和验收标准

交给编程 Agent 的任务应具有明确目标、合理范围和可验证的完成条件。执行前需要说明当前行为、期望结果、相关模块、限制条件和验收标准，并明确本次任务不应修改的内容。功能开发可通过用户行为、接口结果或测试用例定义验收。缺陷修复还应提供复现步骤和回归要求。任务范围过大时，应拆分成能够独立实现和验证的小任务。涉及生产数据、关键权限或不可逆操作时，则需要提高人工确认和审批要求，限制 Agent 的自主操作范围。



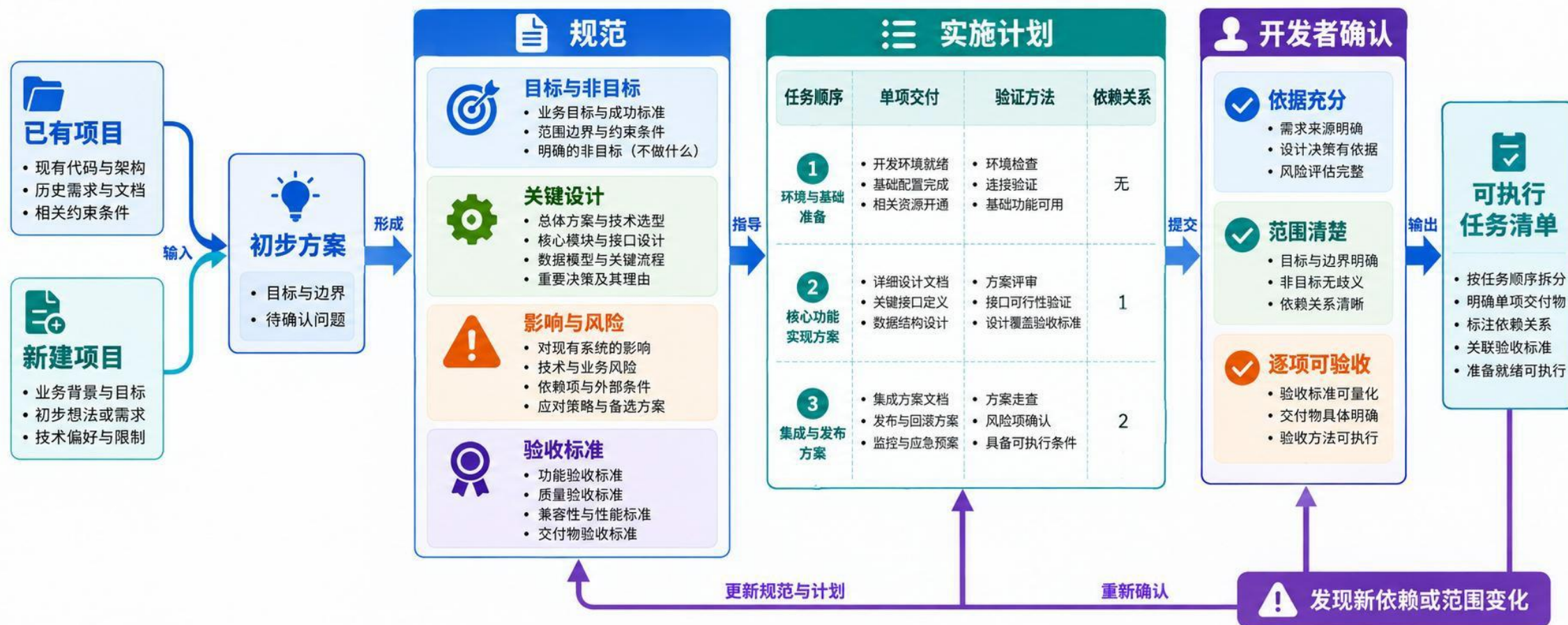
第二步： 确认项目类型， 形成初步方案

开发者需要先确认项目类型，再梳理实现任务所需的条件。已有项目应让 Agent 沿功能入口查找相关代码、调用关系和测试，核对当前行为。修复缺陷时，还需复现问题并保存日志与运行结果。新建项目则要根据需求确定技术栈、基础架构、数据与接口边界，并准备开发和测试环境。两类任务最终都应形成候选实现方案，列明主要风险和待确认问题。



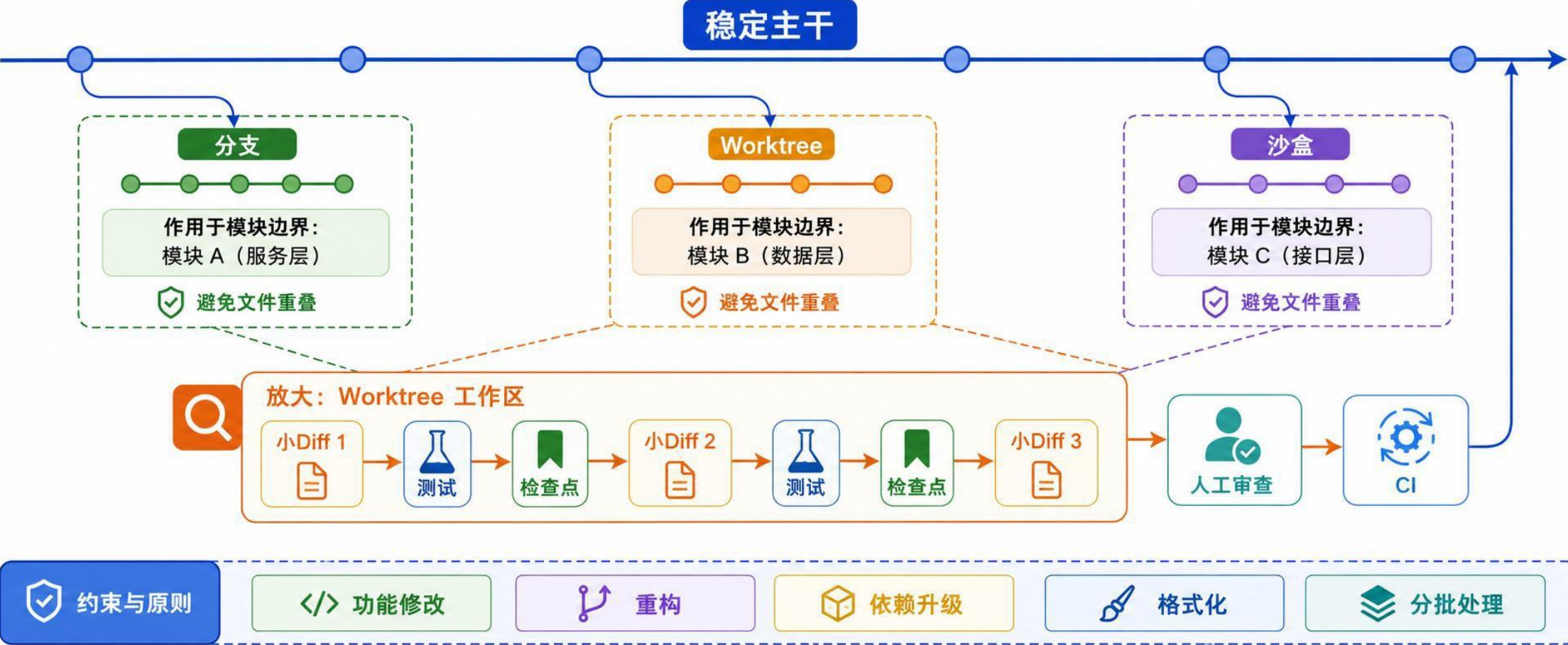
第三步：用SDD 形成可审查的实现计划

确认项目类型和初步方案后，开发者用 SDD（规范驱动开发）整理可审查的规范与实施计划。规范写清目标、非目标、关键设计和验收标准；计划列出任务顺序、每一步的交付结果与验证方法。已有项目还需说明准备复用或修改的代码，以及接口、数据和兼容性可能受到的影响；新建项目则将架构选择和基础约束写入规范。开发者检查这些判断是否有依据、各项任务能否单独验收。执行中若发现新依赖或范围变化，应先更新规范并重新确认，再继续修改。



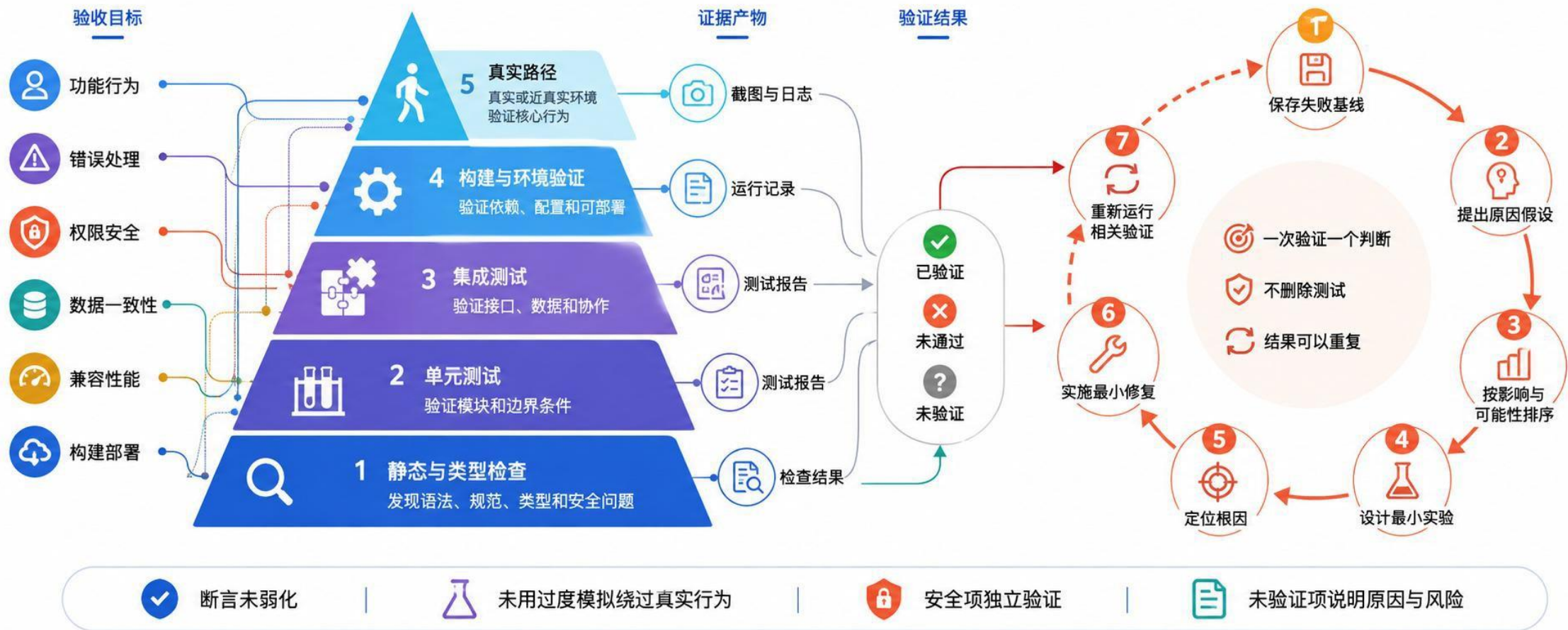
第四步： 隔离环境执行， 控制每次修改的范围

AI 可以在短时间内修改大量文件，因此每项任务最好在独立分支、Worktree 或沙箱中执行，并保留开始时的稳定基线。实现过程应小步推进，每次只完成一个明确目标，避免同时加入新功能、重构代码、升级依赖或大范围格式化。每个增量都应配套验证，并形成便于审查的 Diff。多个 Agent 并行工作时，还应尽量减少对同一模块的重叠修改。隔离环境、小范围 Diff 和稳定检查点能够限制错误影响，也为比较和回退提供依据。



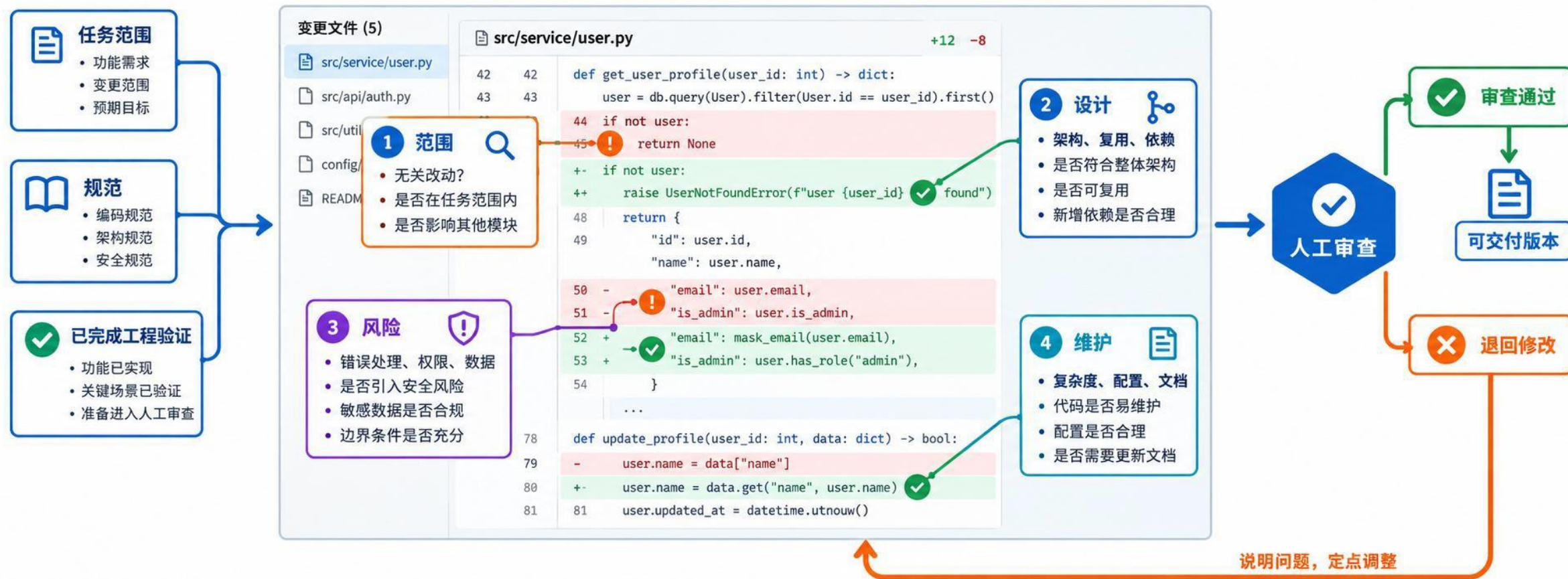
第五步： 执行工程验证， 用测试结果确认符合预期

Agent 声称任务已经完成，并不代表代码真正可用。开发者应根据任务类型运行静态检查、类型检查、单元或集成测试、构建和实际功能测试，并将结果逐项对应验收标准，同时记录尚未验证的部分。如果测试失败，应先稳定复现问题，再通过错误信息、日志和最小实验定位根因，修复后重新执行相关测试。开发者还要检查 Agent 是否为了让测试通过而删除测试、降低原有测试要求或绕过真实场景。涉及权限、安全等关键行为时，还应进行独立验证。



第六步：人工审查代码，确认实现质量

工程验证完成后，开发者对 Agent 产出的代码进行人工审查，判断它能否纳入项目长期维护。对照任务范围查看 Diff，确认没有无关改动，检查关键设计是否符合项目架构，是否重复实现已有能力，以及新增依赖和复杂度是否必要。还需审视错误处理、权限与数据操作中的潜在风险，并核对配置、文档是否随代码同步更新。遇到依据不清或难以维护的实现，应要求 Agent 解释并定点修改。确认实现质量后，再按项目流程接受结果、提交评审或准备合并。



第七步：交付开发成果，跟踪实际反馈

审查通过后，开发者按项目要求让成果进入实际使用。个人项目可以提供可运行版本或部署到可访问环境。团队项目则按既定流程合并代码、核对配置，并在需要时发布。交付时应附上变更说明、验证结果和必要的使用说明。投入使用后，关注真实操作、日志与错误反馈，确认功能在实际环境中符合预期。出现异常时保留证据，及时修复或回退到稳定版本。后续将新的项目知识更新到文档，反复验证有效的做法再沉淀为 Rules 或 Skills。



主流 SDD 与配套工具

SDD是约束 AI 开发行为、保障代码可维护的核心。OpenSpec、Spec Kit 和Kiro Specs 把需求转成可审查的规格，GSD与BMad Method 继续管理长任务的执行过程，Task Master 则从已有 PRD 生成任务。选型时应先确认项目缺少的是规格、上下文管理，还是任务拆分。

类别	工具	定位	关键产物或流程	适合场景	使用边界
SDD框架	OpenSpec	轻量级规范管理框架	提案规范、任务清单	已有代码库增量修改	需要主动维护并审查规范
	Github Spec Kit	GitHub 开源结构化 SDD 工具包	项目原则、需求规范、技术方案、任务清单	团队统一方法跨 Agent 使用	步骤完整学习成本更高
	Kiro Specs	Kiro 开发环境内置	需求文档、设计文档、任务清单	已经使用 Kiro，在环境内形成闭环	与 Kiro 环境结合较深
上下文管理框架	GSD	面向长周期任务和上下文管理	规范、计划、研究报告	长周期任务上下文容易衰减	比轻量 SDD 覆盖更多流程
全流程框架	BMad Method	按角色组织的软件开发方法体系	PM、架构、开发测试等角色产物	新项目角色与流程复杂	配置和学习成本较高
配套工具	Task Master	把现有 PRD 转成可跟踪任务	任务、子任务、状态	已有 PRD 需要执行拆分	不能替代完整的Spec流程

值得优先配置的通用工程 Skills

Skills 把开发经验和团队规范变成 Agent 可以重复调用的工作流程。实用的 Skill 会同时规定任务目标、执行顺序、检查方法和交付结果。优先补齐代码库探索、计划、测试、调试、审查和验证等高频能力，可以减少盲目修改，也便于开发者判断任务是否真正完成。

1 代码库探索

acquire-codebase-knowledge

描述：沿入口、业务逻辑和依赖关系理解陌生代码库

产出：项目结构、请求链路说明

GitHub Awesome Copilot

2 需求与计划

brainstorming writing-plans

描述：澄清需求、对比技术方案并拆分实施步骤

产出：经过确认的实现计划

Superpowers

3 隔离环境

using-git-worktrees

描述：基于 git worktree 建立独立工作区并检查初始状态

产出：干净、可回退的任务环境

Superpowers

4 测试驱动开发

test-driven-development

描述：遵循 TDD 范式，先编写失败测试用例，再完成最小可行实现

产出：可重复运行的回归测试

Superpowers

5 系统化调试

systematic-debugging

描述：复现问题、收集证据并验证假设。

产出：根因说明与最小修复方案

Superpowers

6 代码审查

requesting-code-review

描述：对照需求和 Diff 查找实现问题

产出：分级问题清单与修改建议

Superpowers

7 安全审查

security-review

描述：检查输入、权限、依赖和敏感数据路径

产出：安全风险与修复建议

GitHub Awesome Copilot

8 完成前验证

verification-before completion

描述：运行测试、构建和检查命令。

产出：证明任务完成的执行结果

Superpowers

同类 Skill 无需重复安装，优先保留一套主流程

按技术栈添加的专项 Skills

专项 Skill 只处理特定技术对象，适合在确认项目技术栈后按需加载。界面生成和界面复核属于不同任务，浏览器验证负责检查真实用户流程，数据库和部署 Skills 则进入后端开发与交付环节。只加载当前任务需要的能力，可以减少无关指令和规则冲突。

项目场景	推荐 Skill	具体用途	什么时候加入
UI 页面起稿	frontend-design	根据视觉规范/参考稿生成页面与组件，降低模板化同质化问题	需要从零设计界面时
UI 质量复核	web-design-guidelines	校验页面可访问性、前端性能、交互反馈、样式细节与一致性	页面基本完成后
React / Next.js	vercel-react-best-practices	检查渲染逻辑、数据加载策略、组件结构和前端性能问题	项目使用对应技术栈时
浏览器流程验证	agent-browser	驱动真实浏览器，复现用户操作流程，录制/截图留存验证结果	需要验证完整操作路径时
Supabase 项目	supabase	管理数据库、身份认证、文件存储，执行 Supabase 平台相关操作	项目使用 Supabase 时
Postgres 工程	supabase-postgres-best-practices	检查表结构、索引、数据库迁移脚本、SQL 性能与 RLS 行级安全策略	涉及数据库设计时
预览环境	vercel-deploy	构建可访问、可对外分享的预览部署环境	需要交付或评审页面时

通用方法长期启用，专项 Skill 随项目加载

共同原则： 控制修改范围， 用验证结果判断完成

非开发者主要通过页面体验、数据结果和真实用户操作判断软件，开发者还会检查代码差异、测试、CI和运行状态。两类使用者掌握的技术深度不同，但无论技术深度如何，都需要明确要解决的问题和本次修改的范围，通过可重复的验证确认结果，并保留稳定版本和回退能力。AI扩大了个人的软件创造能力，也扩大了错误传播速度。最终是否合并、发布和投入真实使用，仍然需要由人做出判断。



目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

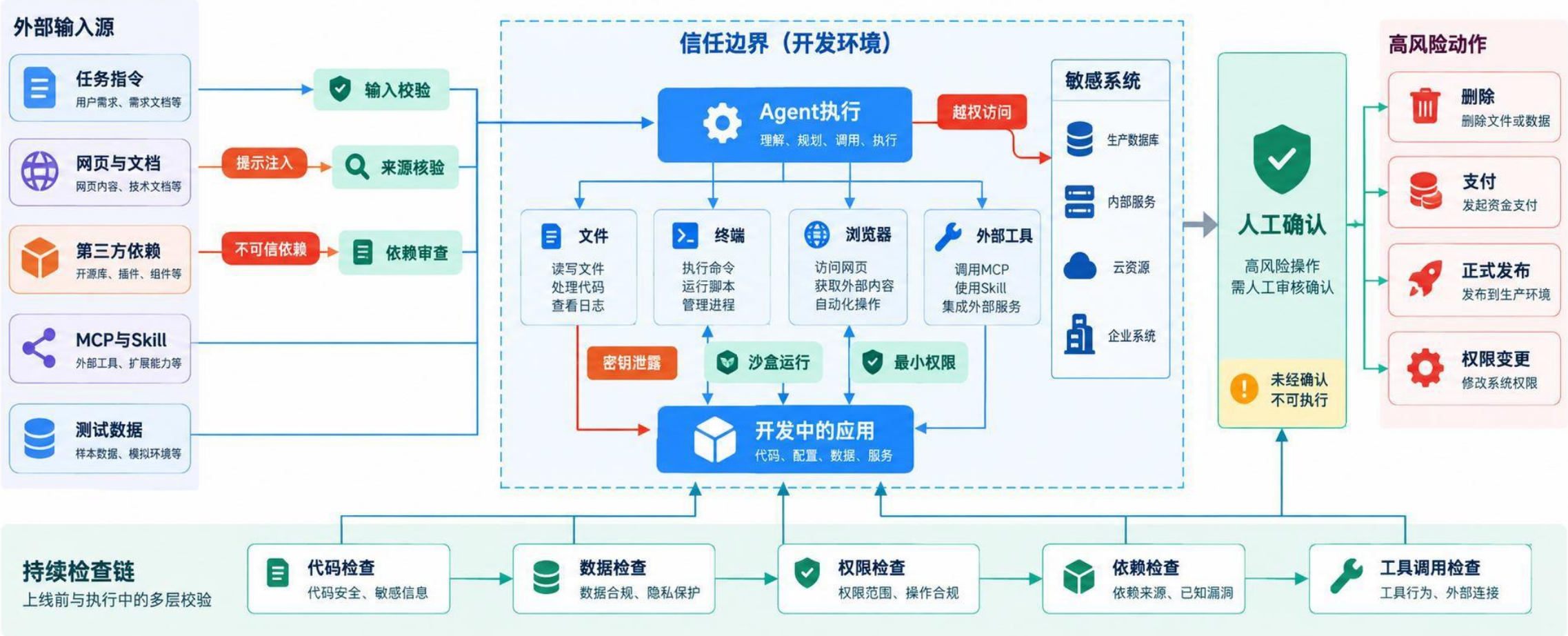
技术债务与代码质量

Vibe Coding 的对话式迭代倾向于用局部补丁快速解决问题，而非同步调整整体结构。重复代码、规则冲突、废弃依赖和测试缺口随之累积，应用仍能运行，但理解和修改的难度持续上升。经过多轮迭代的工具可能出现多套并存的权限判断逻辑，即使是小改动也可能影响已有功能。若无法清晰说明关键逻辑、数据流向和改动原因，还会形成理解缺口，进一步加剧维护成本。定期整理代码结构、清理依赖、补充回归测试并记录关键设计决策，才能保持软件的可维护性。



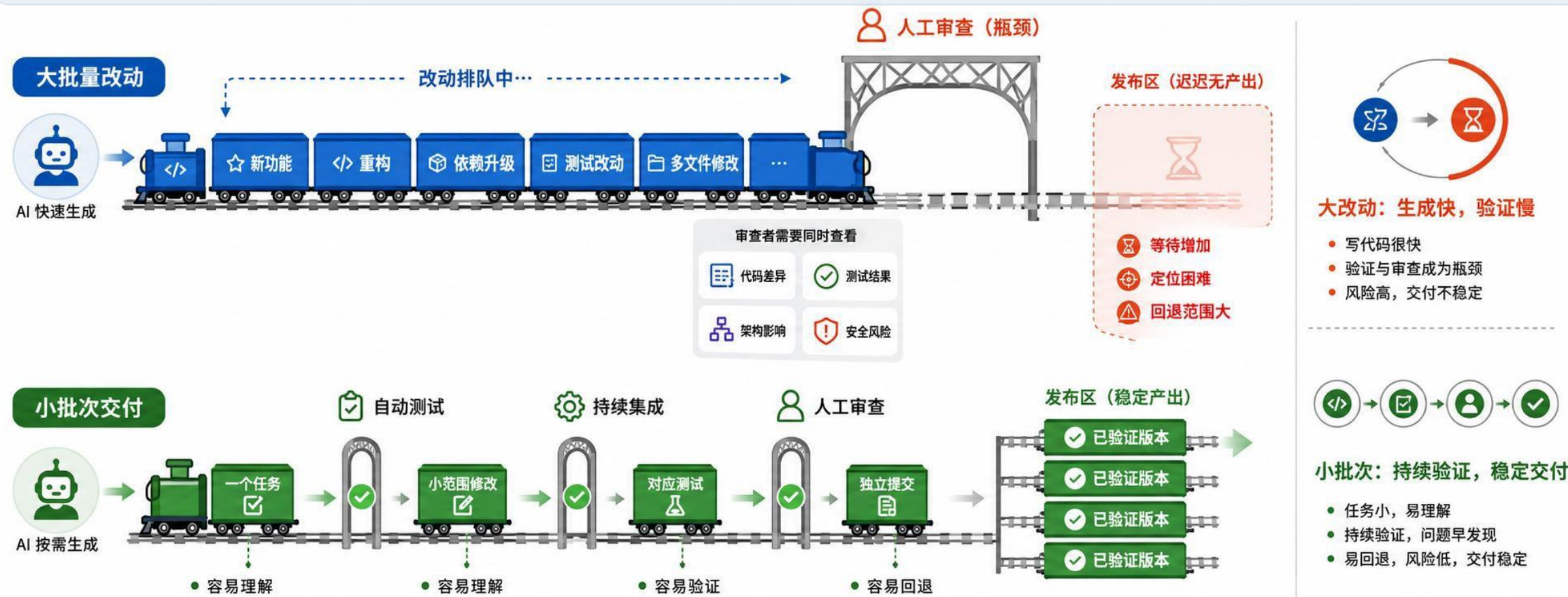
安全漏洞与工具风险

AI可能生成缺少输入校验、身份认证或权限边界的代码，也可能把密钥写入文件、误用真实数据，或安装名称相近却不可信的依赖。页面即使能够正常使用，后台仍可能允许普通用户读取他人记录、绕过登录或调用管理员功能。Agent接入浏览器、终端、MCP和Skill后，外部内容还可能诱导其执行越权操作。安全检查需要覆盖代码、数据、权限、依赖来源和工具调用，高风险动作应在沙盒中运行，并对删除、支付和正式发布设置人工确认。



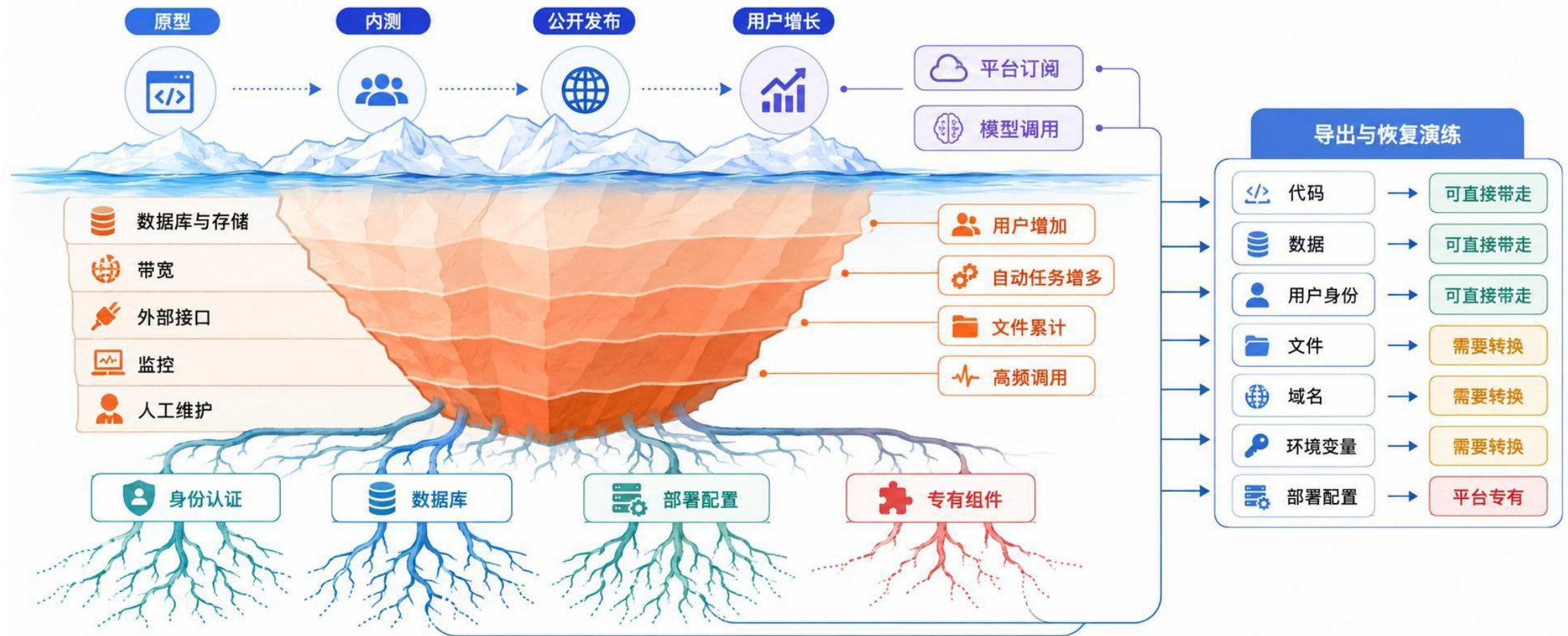
审查瓶颈与验证体系

AI短时间内大量修改文件，而开发者审阅Diff、运行测试、判断设计和处理异常仍需时间。变更范围过大、多个任务并行、或测试与实现由同一个Agent同时修改，都会增加结果核验难度，审查积压可能抵消编码阶段节省的时间。开发流程的主要压力正从代码生产逐步转向结果验证。团队需控制任务粒度、保持小批量变更，使验收标准对应独立测试，再通过持续集成、安全扫描和人工审查建立与生成速度相匹配的验证能力。



使用成本与平台依赖

原型阶段的显性成本主要来自平台订阅、生成积分以及模型或Agent额度。应用上线后，还会持续产生数据库、存储、带宽、域名、第三方接口和监控等运行费用。免费额度下成本可能不明显，用户增长、自动任务和模型调用增加后，完整费用结构才会显现。平台对认证、数据库、部署和专有组件封装越深，迁移时需要重建的能力越多。选型时应确认计费方式、费用上限、代码与数据导出能力、核心服务替换路径，以及停止订阅后应用能否继续运行和维护。



责任归属与治理缺失

Vibe Coding将更多实现工作交由 AI 完成，但需求合理性、数据可用性与代码上线决策仍须由人判断。个人项目由创建者承担结果，团队项目需要明确任务提出、技术审查、发布批准、运行维护和事故处理的职责，并保存代码变更、测试结果与操作记录。长期依赖 AI 输出而不加审查，会削弱使用者的异常发现与系统接管能力，因此团队仍需保留需求判断、架构理解和故障处理等核心能力。涉及资金、隐私或关键业务时，还应提升权限管理、审批流程与专业审查等级。



目录

01  Vibe Coding的兴起与应用场景

02  Vibe Coding的特征与工作方式

03  Vibe Coding的工具与选择方法

04  Vibe Coding的流程与实用技巧

05  Vibe Coding的挑战与风险边界

06  总结与未来展望

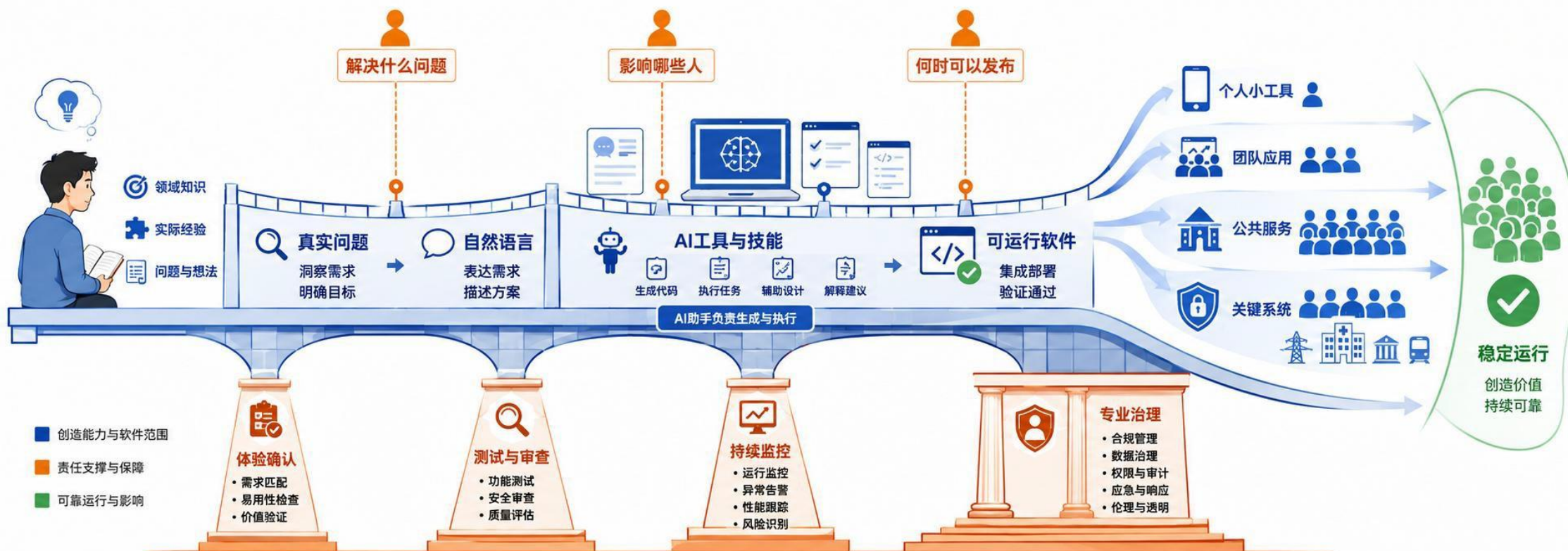
Vibe Coding 的发展方向：从一次性Prompt 走向可复用能力

Vibe Coding 早期主要关注如何用自然语言获得最终结果，下一阶段的竞争将更多转向如何让AI稳定理解项目、复用经验并完成可验证流程。项目知识、Rules、Skills、模板、测试和运行环境共同构成AI工作的长期上下文。Prompt 不会消失，它仍负责表达本次目标，但重复任务将越来越多地由Skill承载，团队方法也会变成可以版本化、共享和评测的资产。未来真正拉开差距的，不只是使用了哪个模型，而是谁能把领域知识和工作流程组织成AI可执行的系统。



软件创造的门槛正在重构，人的责任仍是最终边界

Vibe Coding的真正意义，不是让所有人都变成传统意义上的程序员，而是让更多人能够把场景知识转化为可运行的软件，也让专业开发者可以用新的方式实现工作。自然语言降低了表达意图的门槛，工具和Skill扩展了AI的执行能力，反馈与验证则决定结果是否可靠。未来会出现更多由个人和小团队创造的长尾软件，也会出现更严格的工程治理。但无论工具怎样进步，选择解决什么问题、判断何时可以发布、决定系统使用范围和受众，这些关乎判断和责任的事，始终需要人来完成。



门槛降低，让更多人能够创造；影响扩大，需要更强的验证与治理。

版权声明

一、内容版权说明

文字内容：本 PPT 所有文字内容均基于公开渠道信息整理汇编，包括但不限于行业报告、学术论文、厂商白皮书、开源社区文档及公开新闻资讯，版权归原作者或发布机构所有。

图片与素材：所有示意图、表格等素材均来源于公开网络资源（如各类无版权图库，或企业官网、行业分析报告的公开配图）或由AI辅助生成并经过人工设计与校对。产品名称、商标、界面元素及公开资料中的素材归其权利人所有，仅用于技术科普和说明，无商业用途。

二、知识产权声明

本 PPT 内容为知识整理与二次创作成果，不涉及对任何企业、机构或个人的商业信息、技术专利的侵权。若部分内容无意中侵犯了您的知识产权，请通过以下联系方式告知，我们将立即删除或补充版权说明。

三、免责条款

本 PPT 仅用于技术交流、学习分享，禁止用于商业用途或盈利性传播。内容力求准确，但不保证完全无误，对因使用本 PPT 内容引发的任何直接或间接损失，不承担法律责任。图片与数据仅供参考，具体以官方最新发布信息为准。

四、联系方式

如有任何问题，请联系我们。