Baidu Comate

人机协同的AI原生开发范式革命及落地

百度工程效能部　　李杨

个人简介

百度工程效能部
李杨

百度资深研发工程师， 文心快码（Baidu Comate）商业化负责人，云端开发平台（iCoding）的技术负责人。

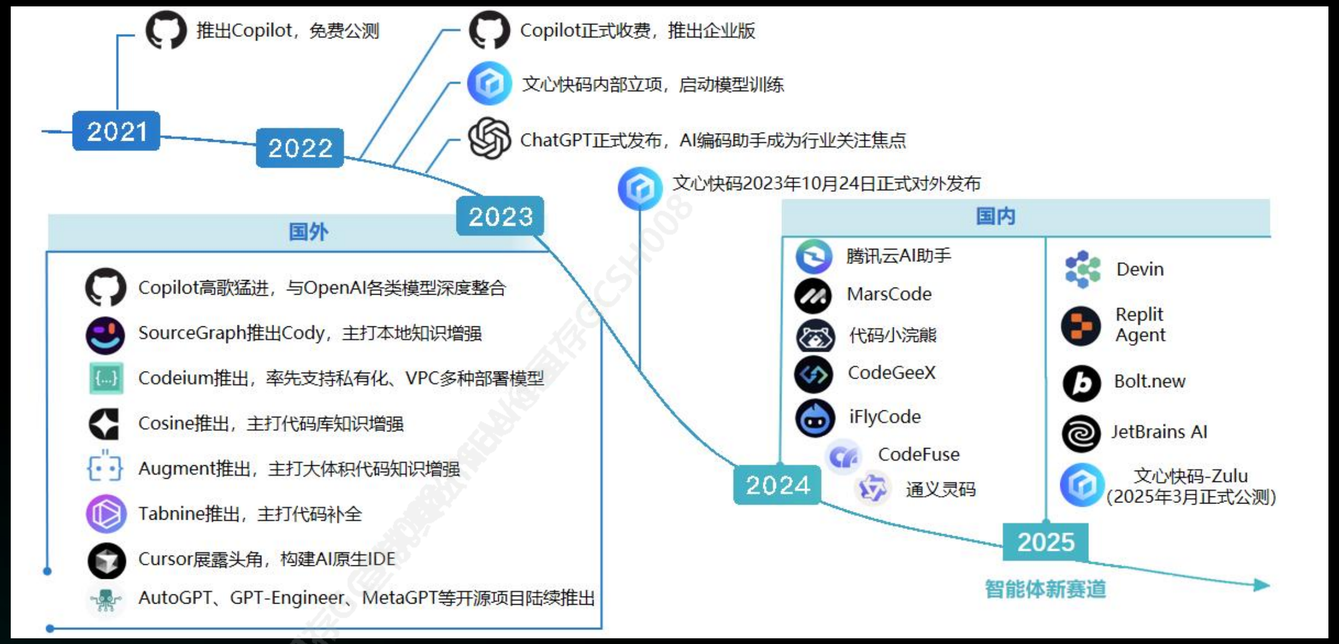
百度一级专利发明人，名下国内外发明专利10余个，已获授权

国家重点研发计划『基于人机结对编程与协同进化的智能敏捷开发云平台』技术骨干

目录

CONTENTS

业内普遍看好大模型在研发领域的应用场景



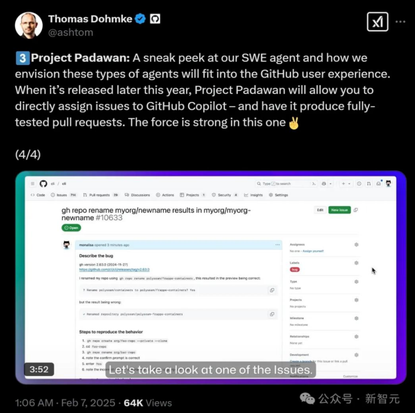
大模型在ToB领域最成功的领域， 50+AI智能编码助手

CURSOR	GitHub Copilot	replit	lovable
5亿美元	4亿美元	1亿美元	7500万美元
打破历史记录	年增长4倍	6个月10倍增长	7个月时间

用户愿意为AI Coding付费

Devin Ai

The World's First Fully Autonomous AI Software Engineer

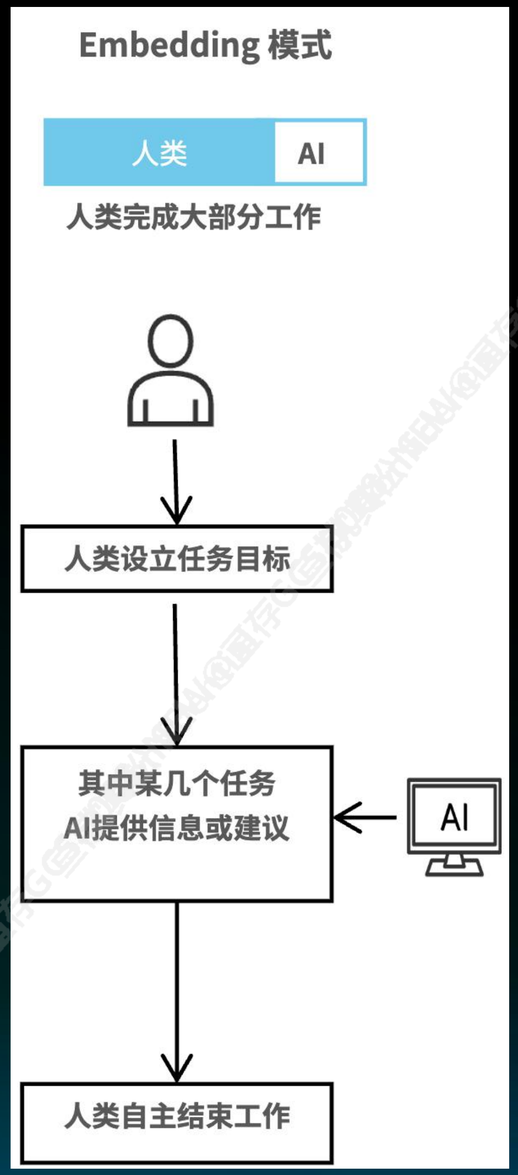


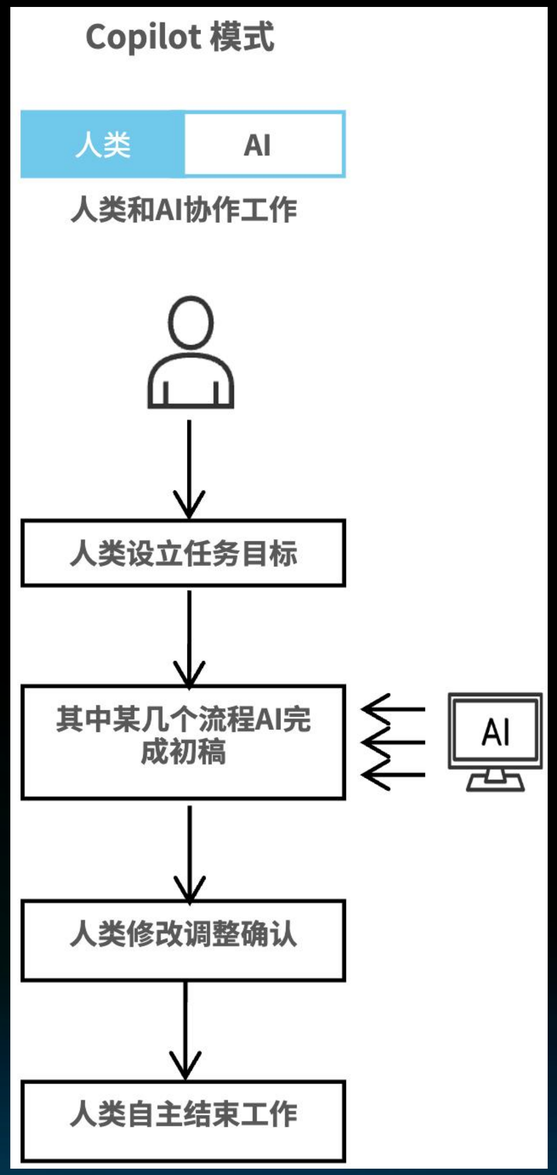
SWE Agent修复issue

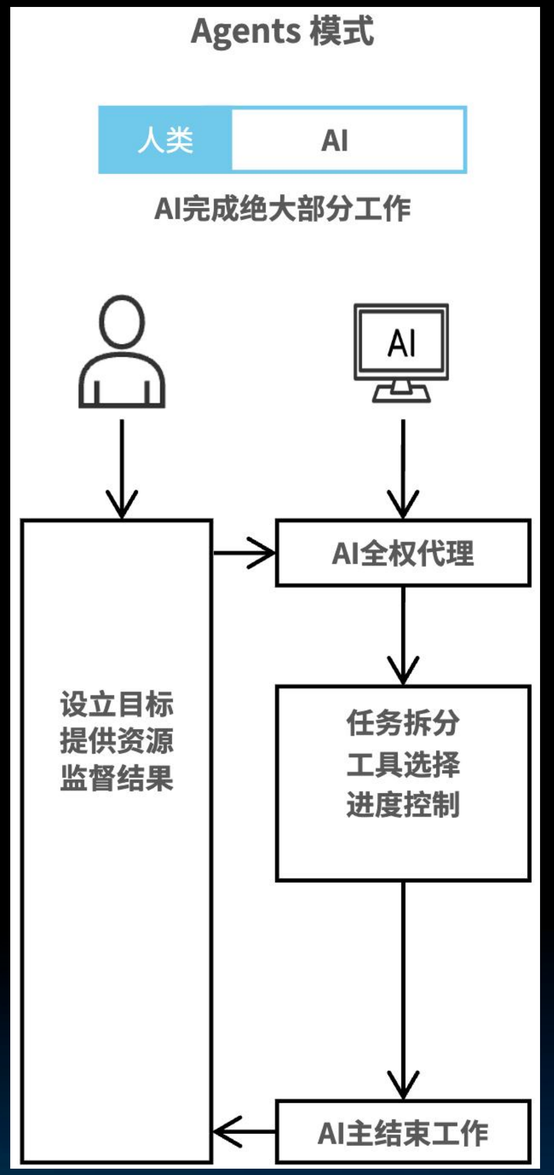
命令行形态集成

研发智能体产品形态多样

人类与AI协同模式 - 从工具到员工的范式迁移







大模型为软件开发者带来全方位的改变

AI是生产力工具，赋能成为更好的个人(Tool → Capability)

AI 是放大器，提升需求、设计、开发、测试、上线等环节的有效产出

- AI prototype
- Spec研发
- TDD

AI能改变生产关系，组织能力（Role → Boundary Fusion）

AI 是倍增器，角色边界逐渐融合，跨角色沟通成本显著下降

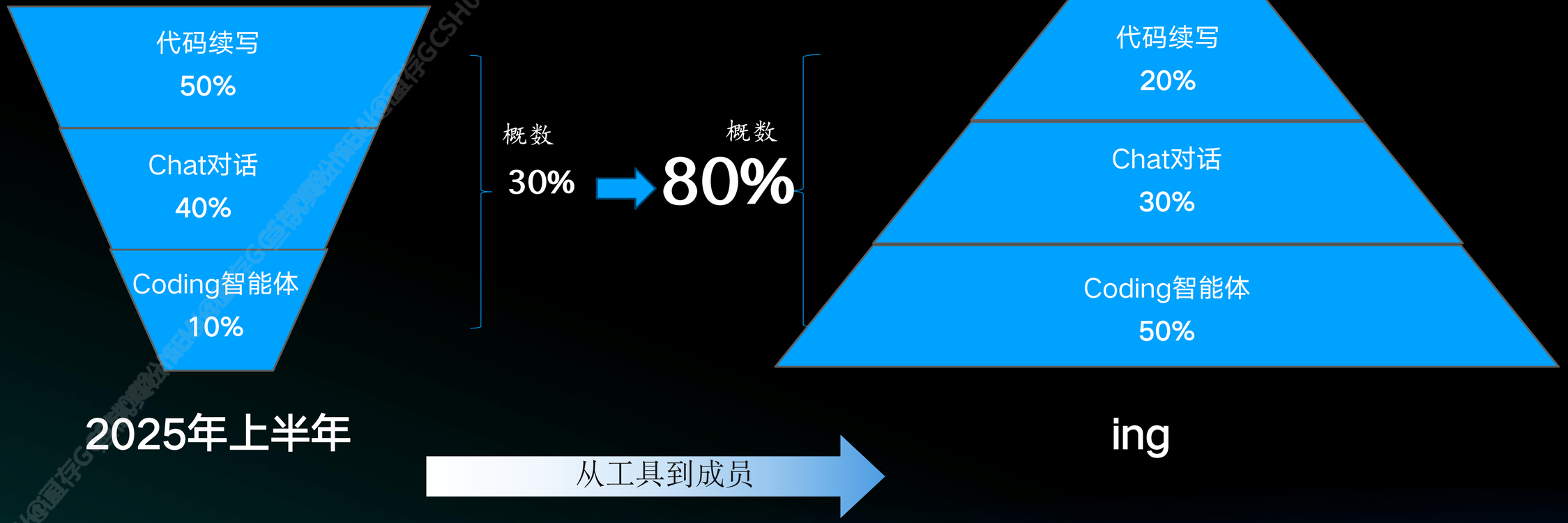
- 流程重构
- 角色重定义
- 能力模型

AI Native研发团队

- AI是主要生产力
- 人:执行者 -> 设计者
- (半)全栈角色

变化1：AI代码生成占比继续提升，构成正悄然发生变化

AI代码生成占比数值还将继续提升，增速还将加快，达到临界值后团队将产生质变



变化2：智能化进程多阶段并存，行业变化快，拥抱变化

代码续写 ⟷

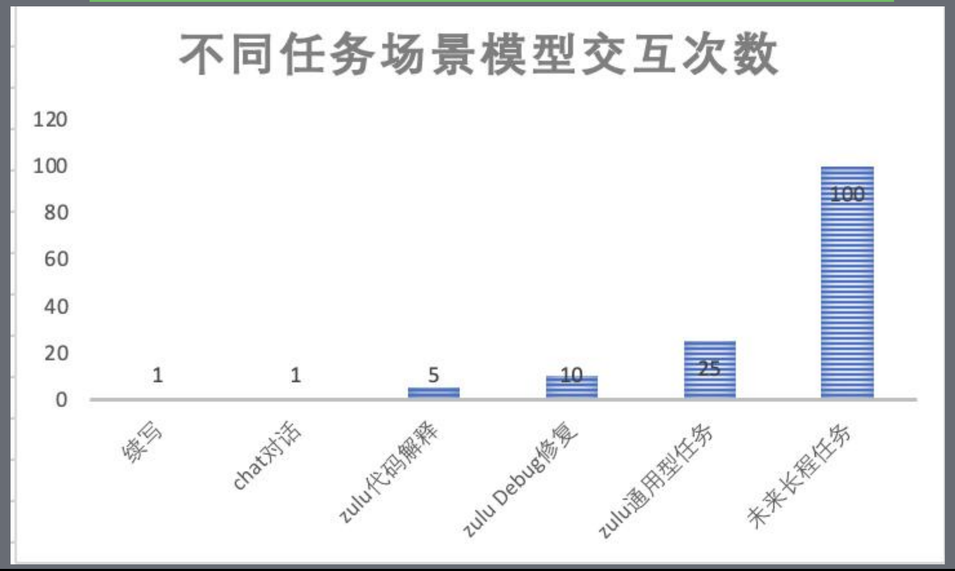
Chat对话 ⟷

智能体Agent ⟷

Vibe Coding ⟷

拥抱『变化』常用常新

AI参与程度变化变大



生成粒度

词 | 行 | 多行 | 结构段 | 函数、类 | 需求 | 产品

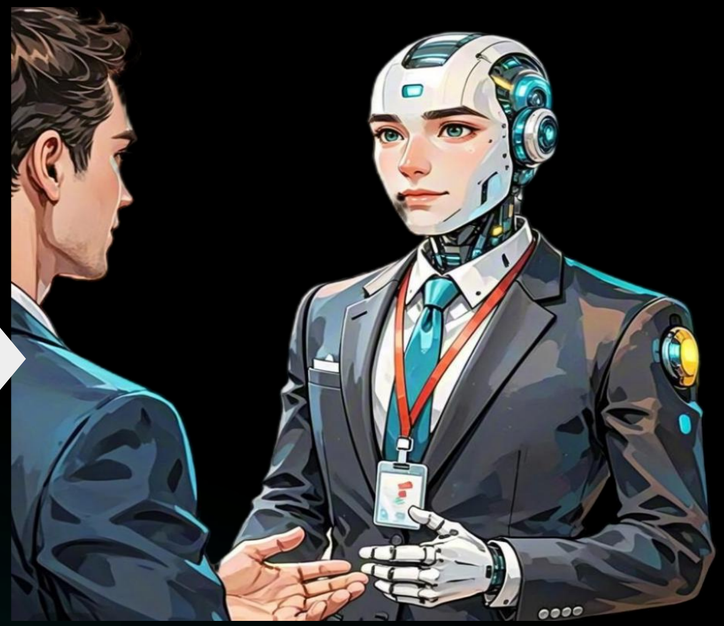
变化3：新的产品形态出现，更好实现软件研发场景的协同

最近爆火的Clawdbot，AI正从简单的聊天响应进化为能自主执行任务的智能体

工程师日常协同研发

工程师 委托 同事 反馈

人机协同研发新范式



工程师 委托 研发数字员工 反馈

类人协作，自主决策并执行闭环

研发任务感知
事件触发 | 主动感知

研发任务执行
多环境联动 | 异步委托与授权

研发任务追溯
研发数字员工可视化工作台

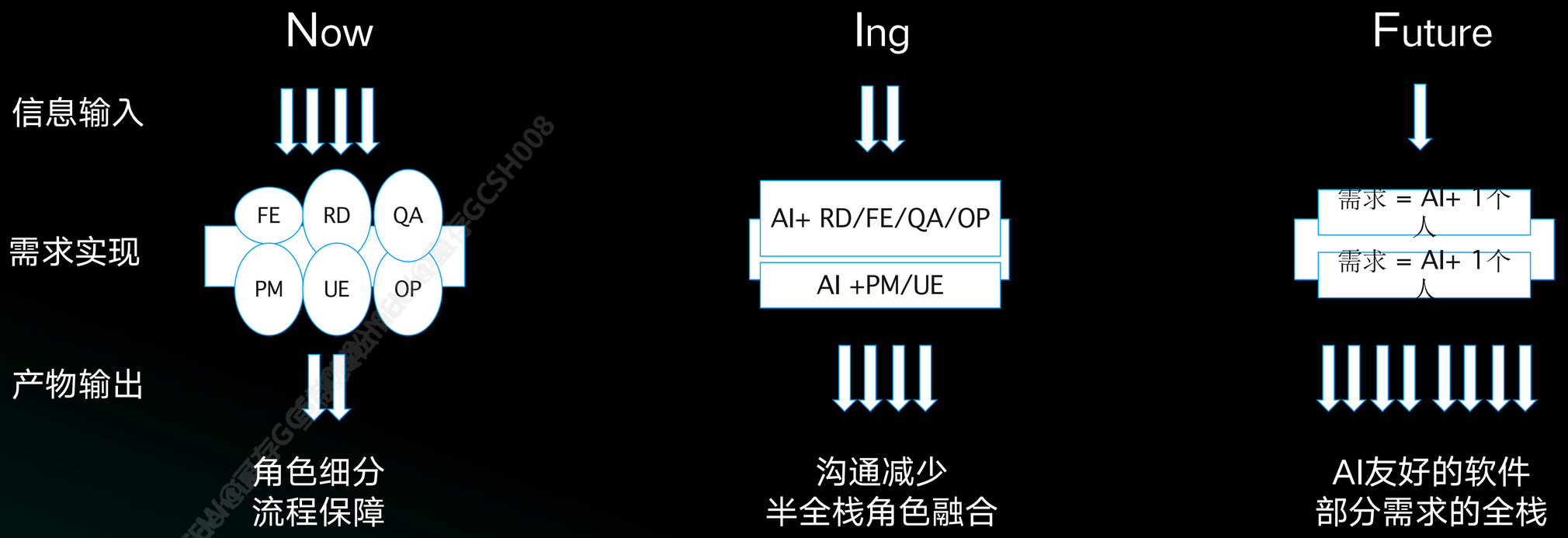
人机同权，研发全场景触达

本地场景
IDE | PC | Browser
.....

办公场景
IM | 知识库 | 邮箱
.....

DevOps平台
需求管理| 代码托管| 应用管理
.....

变化4：角色越来越「独」，信息交换「管线」变少，组织变化团队变小



小团队能干大事情，创业公司开始，大公司逐步转变中

YC 2025年数据显示，孵化项目中24%初创企业， 95%代码由AI生成，团队人均规模3.2人

Cursor、GenSpark、Devin等公司人比较少，产出非常惊人

目录

CONTENTS

百度Comate平台产品全景图：定义AI研发应用新范式

Comate平台端

Cli	IDE	Plugin for IDE		
Comate Cli	Comate AI IDE	Vscode	jetbrains	visualstudio

能力层

基础能力	基础组件	通用Agent	垂类Agent	开放能力	企业应用
单行补全	文件编辑	Ask模式：只做问答	代码评审 Agent	MCP打通研发工具链 NEW	权限管理
多行补全	代码检索	Code模式：代码编辑	安全扫描 Agent	Skills NEW	AI采纳数据分析
光标预测 NEW	Todo Write	Plan：计划任务	FigmaToCode Agent	自定义SubAgent NEW	知识管理
代码编辑	Create Plan	Spec：SDD研发模式 NEW	Page Builder Agent		开发中心
基于RAG补全	Other System Tools		Debug Agent		模型管理

架构层

SubAgent架构		记忆存储&检索系统			私域知识增强
隔离的上下文窗口	Tools	上下文感知处理	智能去重机制	冲突解决能力	私域代码/文档

模型基础设施

文心系列大模型	开源大模型（GLM，Qwen，MiniMax）	自定义模型

Copilot编码实践| AI加速写代码的速度

0	1	2	3	4	5
无助手	**辅助写代码**	代码私域问答	chat模式	代码智能体	自适应全自主

Human引领 单指令交互 | 基于通用知识 | 快思考

AI驱动 多轮交互 | 基于私域+领域知识 | 慢思考

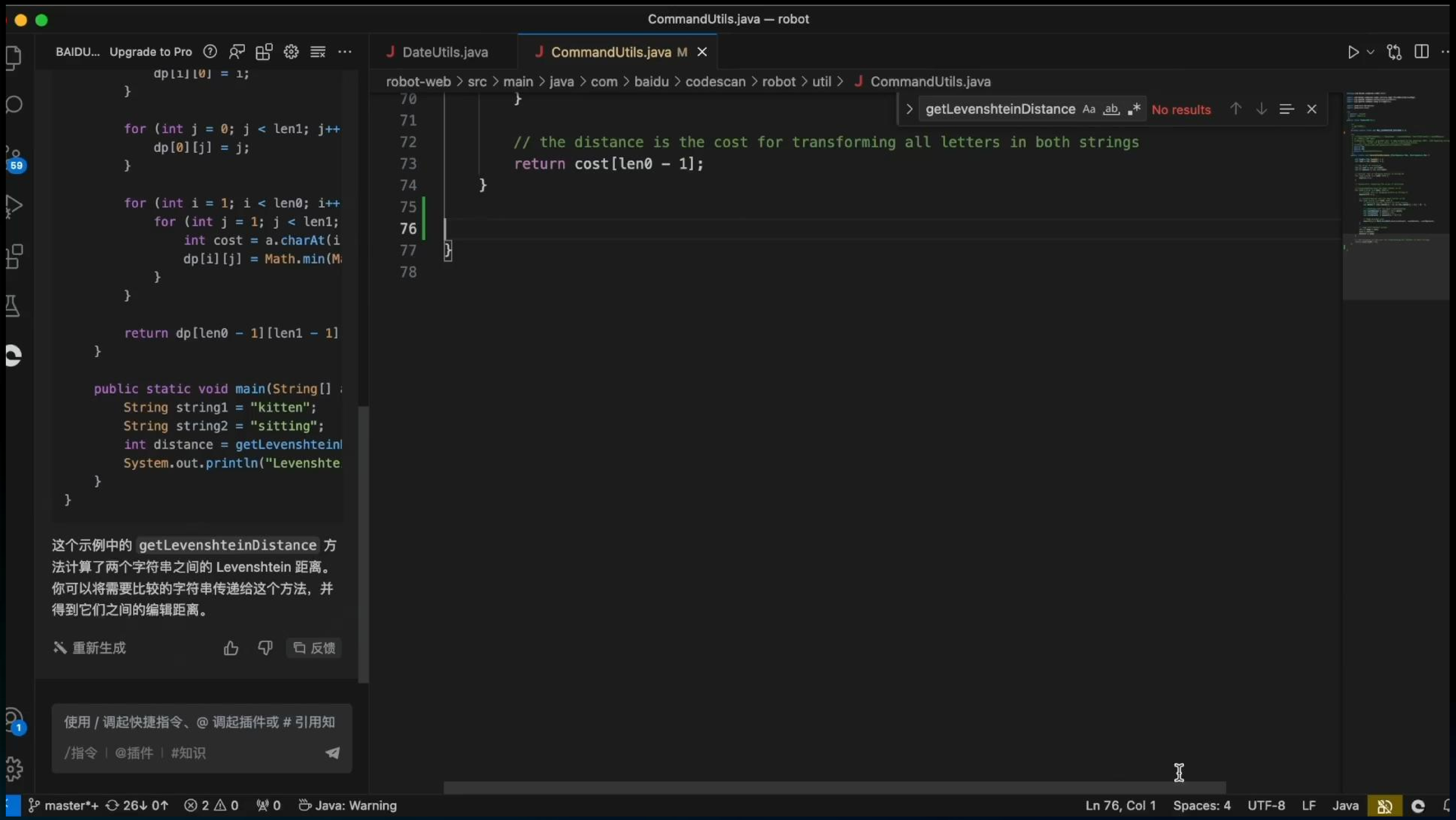
AI 引领

从"工具"走向"Pair Engineer"
习惯的主动改变

代码续写

自然语言转代码

落地实践：写代码之前先写注释



Copilot编码实践 | 大模型结合私域知识，打造更懂你的智能编码助手

0	1	2	3	4	5
无助手	辅助写代码	**代码私域问答**	chat模式	代码智能体	自适应全自主
Human引领 单指令交互 | 基于通用知识 | 快思考			AI驱动 多轮交互 | 基于私域+领域知识 | 慢思考		AI 引领

把你的私域知识告诉AI
个性化、组织知识沉淀

提供上下文，让AI举一反三

参考本地代码生成新代码

基于文档的问答助手

接口文档生成业务代码

基于链接生成专属代码

1、文字描述为主，多模态不够精确

2、段落标题、目录清晰

```
# Linux上GPU预测部署示例

## 1 C++预测部署示例

C++示例代码在[链接](https://github.com/PaddlePaddle/Paddle-Inference-Demo/tree/master/c%2B%2B/cuda_linux_demo)，下面从`流程解析`

### 1.1 流程解析

#### 1.1.1 准备预测库

请参考[推理库下载文档](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/guides/05_inference_deployment/inference/

....省略部分内容

#### 1.1.4 设置Config

根据预测部署的实际情况，设置Config，用于后续创建Predictor。

Config默认是使用CPU预测，若要使用GPU预测，需要手动开启，设置运行的GPU卡号和分配的初始显存。可以设置开启TensorRT加速、开启IR优化、开启内存优化。

```cpp
paddle_infer::Config config;
if (FLAGS_model_dir == "") {
config.SetModel(FLAGS_model_file, FLAGS_params_file); // Load combined model
} else {
config.SetModel(FLAGS_model_dir); // Load no-combined model
```
```

Copilot编码实践 | 写代码之前问一问，当搜索引擎来用

0	1	2	3	4	5
无助手	辅助写代码	**代码私域问答**	chat模式	代码智能体	自适应全自主

Human引领 单指令交互 | 基于通用知识 | 快思考

AI驱动 多轮交互 | 基于私域+领域知识 | 慢思考

AI 引领

在动手实现之前，先去问一下AI的实现

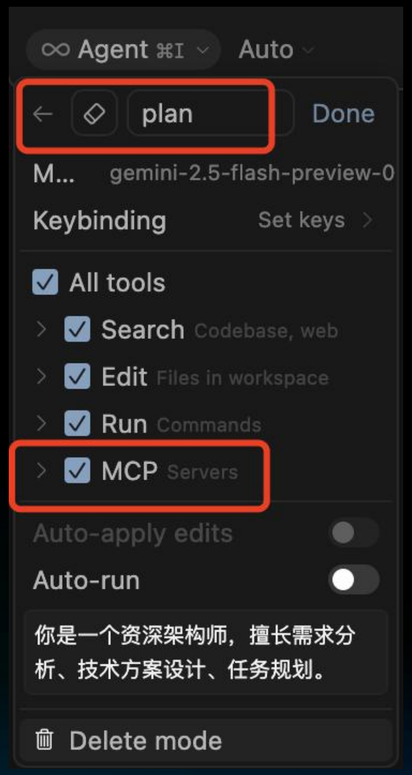
利用ASK/Plan模式做需求澄清
厂内MCP工具让模型知道私域知识

实现逻辑不了解，问一问

没有思路的时候，问一问

忘记知识的时候，问一问

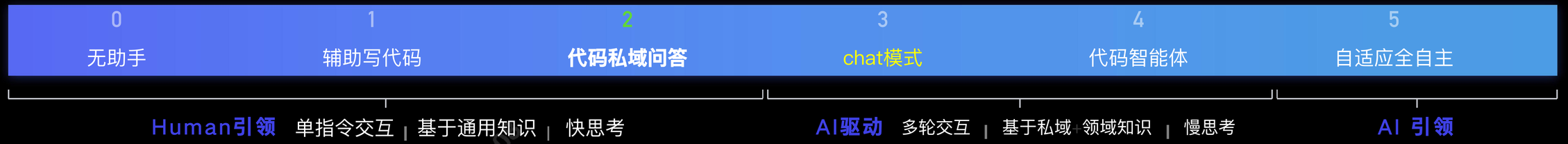
代码出错的时候，问一问



Plan模式

MCP工具

Agent编码实践 | Coding Agent推进Rules编写，AI在约束下发挥



Coding Agent很强大，但很健忘，在实际代码库落地需要Rules的约束和上下文提供

无法掌控的感觉是无法接受
Coding Agent越强大
越应该约束

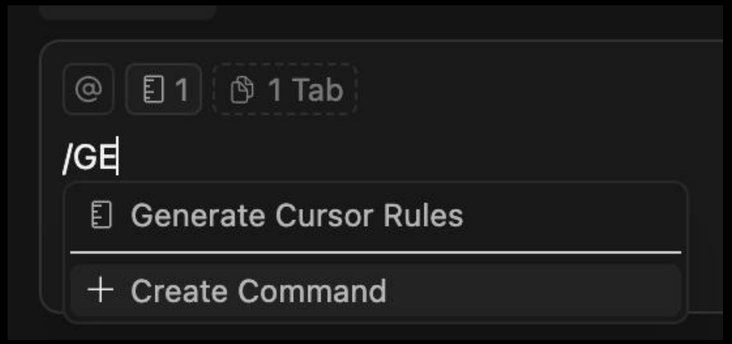
不能要求人人都成为专家
驾驭AI是有一定门槛
经验流程要固化

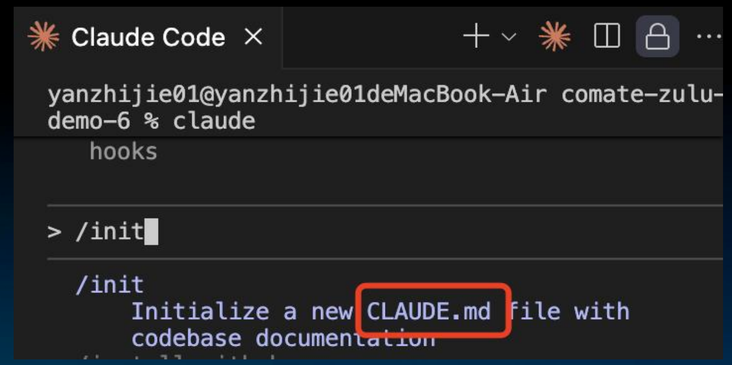
Rules可以分层

- 技术方案编写规范
- 项目架构（目录、技术栈、业务）
- 代码开发规范（命名、注释、格式、日志、错误码）
- 常见功能开发流程模板和示例
- 代码提交与评审规范

```
├── plan
│   └── p-design-tech-solution.mdc          // 技术方案编写规范、模板
├── develop
│   ├── d-gen-data-center-crud-from-sql.mdc  // d-gen 开头的是总结的某类功能的代码生成模版、开发流程
│   ├── d-gen-cronjob.mdc                    // cronjob 开发流程、代码模板;
│   ├── d-gen-api.mdc                        // http api 开发流程、代码模板;
│   └── d-test-unit-test.mdc                 // 单元测试规范、模板
└── review
├── r-code-review.mdc                    // 代码评审规范
├── r-git-commit.mdc                     // commit message 规范
├── r-merge-request.mdc                  // 创建远程合并请求流程
└── r-summary-sop.mdc                    // 总结本次代码修改的标准流程生成 p-gen-xx 文件，供下次开发使用
├── g-code-spec.mdc                      // 项目代码规范
├── g-project-hotkey.mdc                 // 项目中的 hotkey 比如`提示语中出现 cr 则参考 xxx.rules 执行`
├── g-project-structure.mdc              // 项目结构、技术架构、业务架构...
├── g-system.mdc                         // 项目 AI 聊天规范，比如强调多轮对话、引导补全...
```

善用工具辅助生产





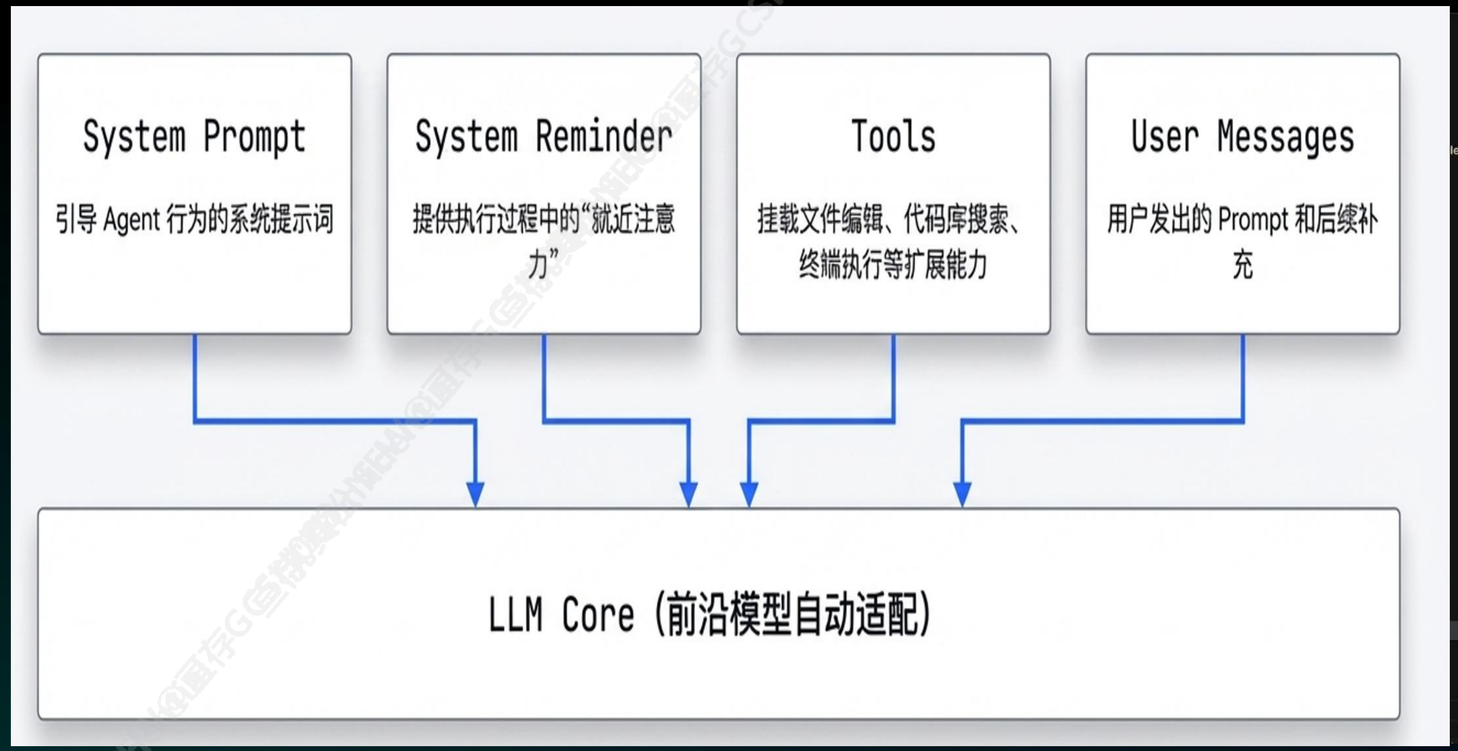
Coding Agent编码任务端到端生成

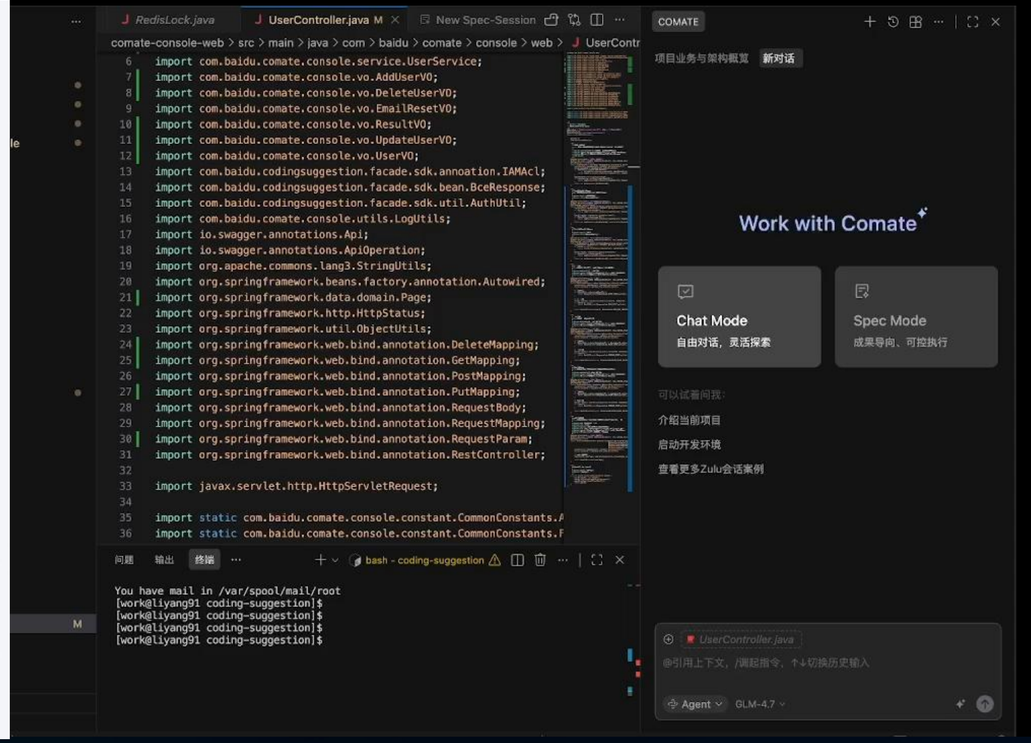
0	1	2	3	4	5
无助手	辅助写代码	**代码私域问答**	chat模式	代码智能体	自适应全自主

Human引领　单指令交互 | 基于通用知识 | 快思考

AI驱动　多轮交互 | 基于私域+领域知识 | 慢思考

AI 引领

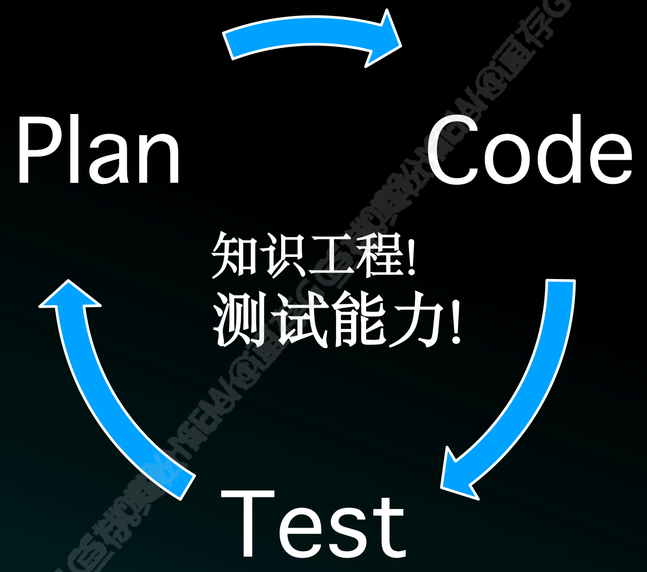




Coding Agent 0Day 接入前沿模型，开发者只需要关注业务逻辑

工程能力越发重要，软件研发流程的适配改变

0	1	2	3	4	5
无助手	辅助写代码	**代码私域问答**	chat模式	代码智能体	自适应全自主



知识工程越来越重要

- 1.Prompt工程变成了必要技能
- 2.需求的Spec也是知识，充分利用AI完善文档，形成闭环
- 3.充分利用文件系统便于Agent获得知识

项目结构

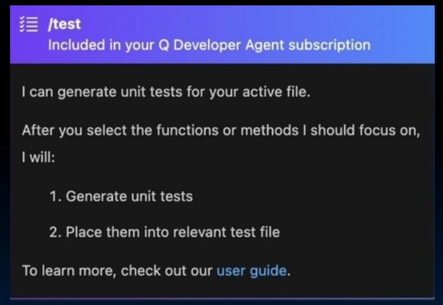
```
packages/
├── core-service/
├── image-editor/
└── example/
```

文档结构

```
design/
├── core-service/
├── image-editor/
└── example/
```

测试能力是关键因素

- 1.大模型是概率模型，@1pass不可能
- 2.确定性的测试能力对生成代码的验证是效果提升的关键因素，AI可以多试几次
- 3.TDD有点太理想，但测试能力是跨过80分危机的良药



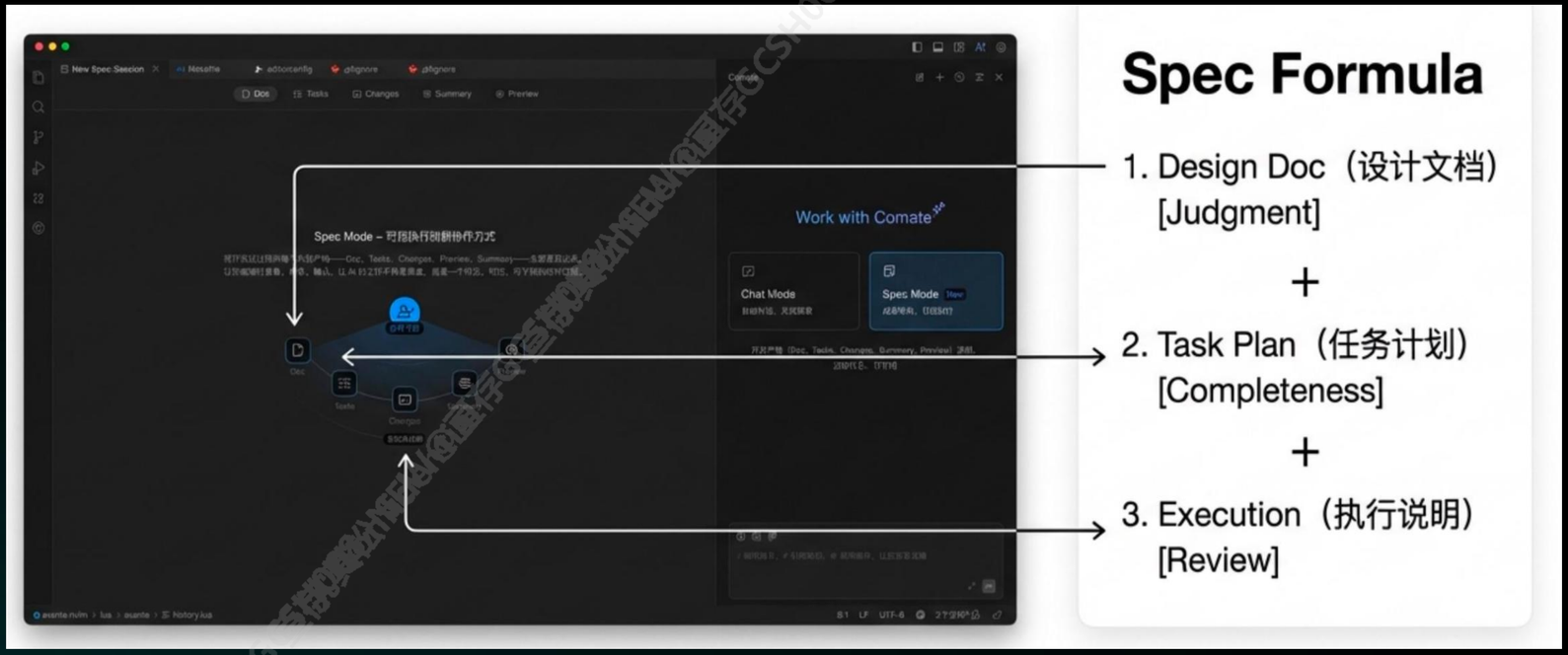
Spec driven很有效

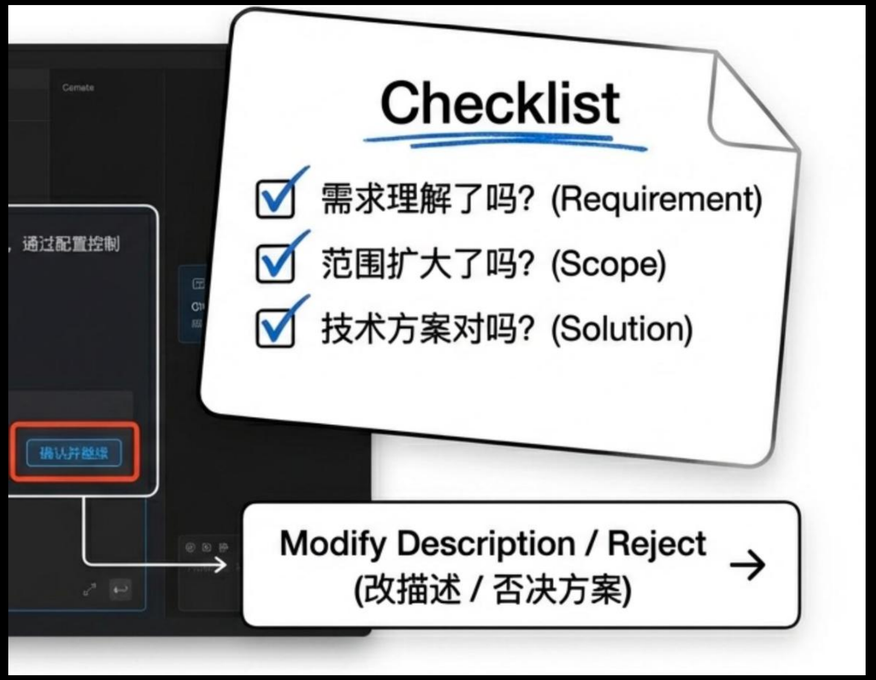
从写代码=>写文档

工程能力是保障

Agent编码实践I工程能力越发重要，软件研发流程的适配改变

0	1	2	3	4	5
无助手	辅助写代码	**代码私域问答**	chat模式	代码智能体	自适应全自主



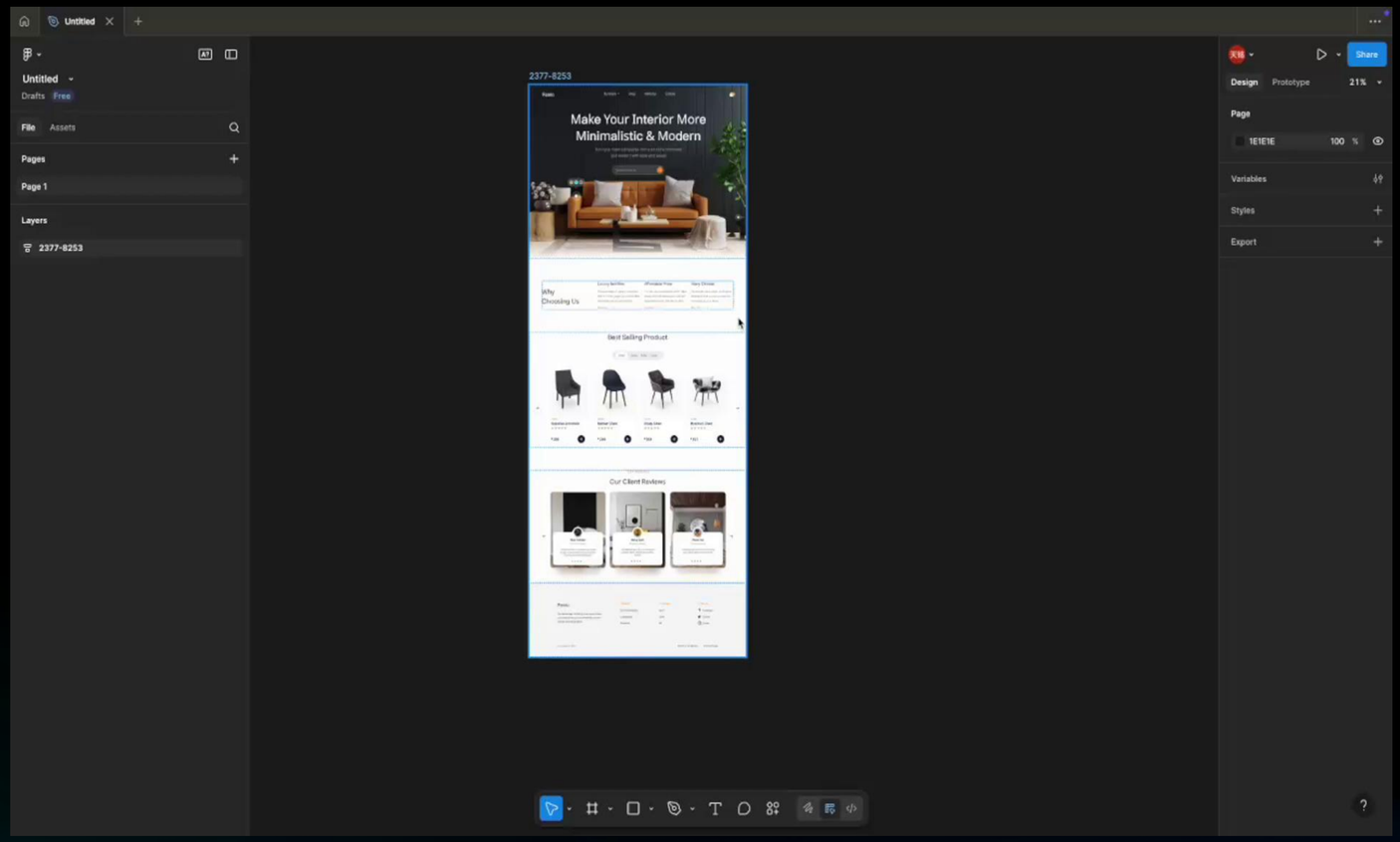


Spec：让AI写之前先把打算怎么做写出来

Agent设计实践|F2C应用落地广泛，先标准化再智能化

充分利用
大模型
现有能力

先标准化
再智能化



Agent实践同DevOps平台的结合|编译修复从IDE、云端到CR，多形态

构建“工具+知识+Skills+Agents”的开放研发新生态

Comate kenel开放平台

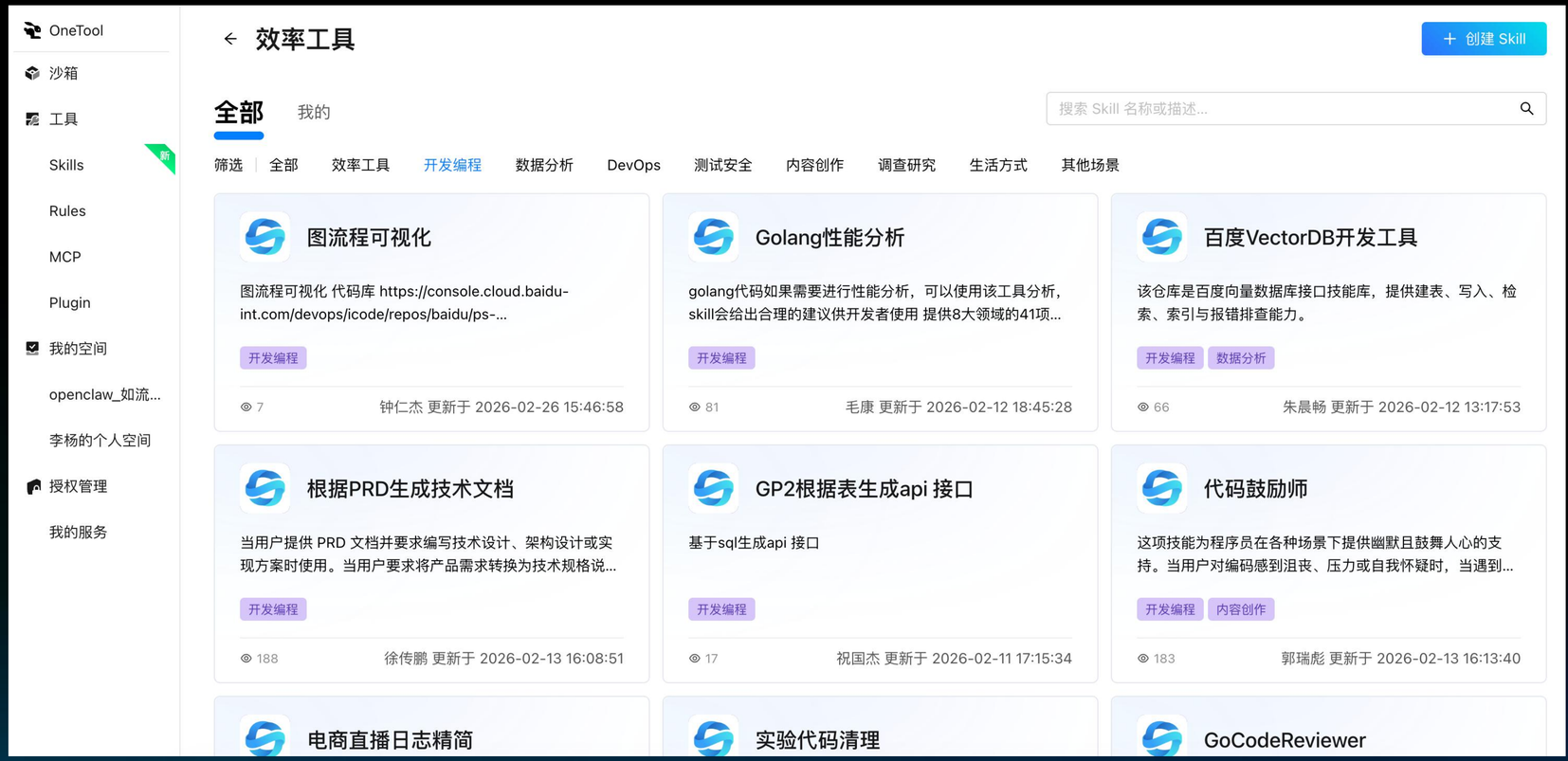
需求拆解	架构规范	研发编码
01	02	03
需求问答 需求文档创建 项目计划建议	前端规范 后端规范 业务流程梳理 安全规范	基于技术约束、 规范、项目约束 自动编码
调试	测试	部署
04	05	06
Debug 智能体调试 自动修复	测试用例生成 测试问题反馈分析	部署至沙箱 部署至线上e

- 企业知识融合能力
- Skills扩展能力
- MCP全流程工具接入能力
- 行业智能体构建能力

MCP+Skills 扩展研发全流程

MCP市场

Skills市场



SubAgent：适配研发规范和业务逻辑的Workflow

创建Agent

编辑Agent

什么是 Subagent?

Subagent是 Comate 主Agent可以灵活委派任务的专业化Agent助手。它通过在主对话外拆解复杂任务，助力主Agent实现多线协同。

每个Subagent在独立的上下文窗口中运行特定类型的工作，并将结果精准返回给主Agent，确保主对话上下文的绝对纯净。

上下文隔离

每个子代理拥有自己的上下文窗口。耗时较长的调研或探索任务不会占用主对话的上下文空间。

并行执行

可同时启动多个子代理。能够在代码库的不同部分并行工作，无需等待低效的顺序执行完成。

专门领域能力

通过自定义提示词、工具访问权限和特定模型，为子代理精准配置专属特定领域的任务能力。

灵活可复用

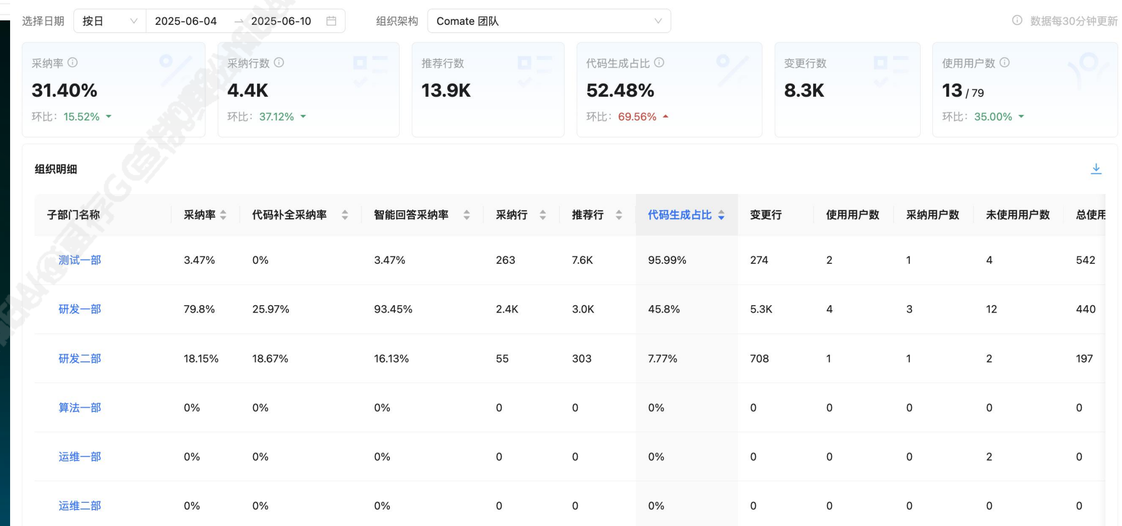
支持开发者自定义配置子代理，实现一次构建，即可在多个项目之间无缝复用。

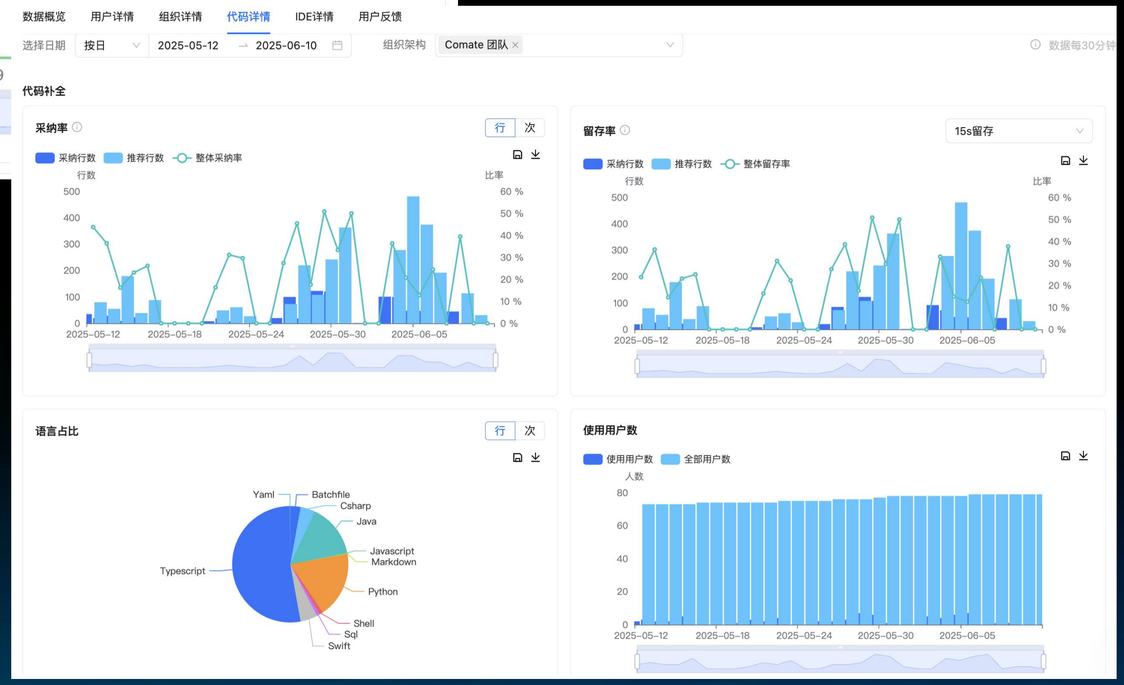
Comate支持项目及个人Subagent自定义

如何衡量AI Coding对组织效率的提高

衡量组织效率，指引组织落地

- 采纳率、AI代码生成占比的实时跟踪
- 按照组织、个人查看详细内容
- 查看各个语言、功能使用情况





目录

CONTENTS

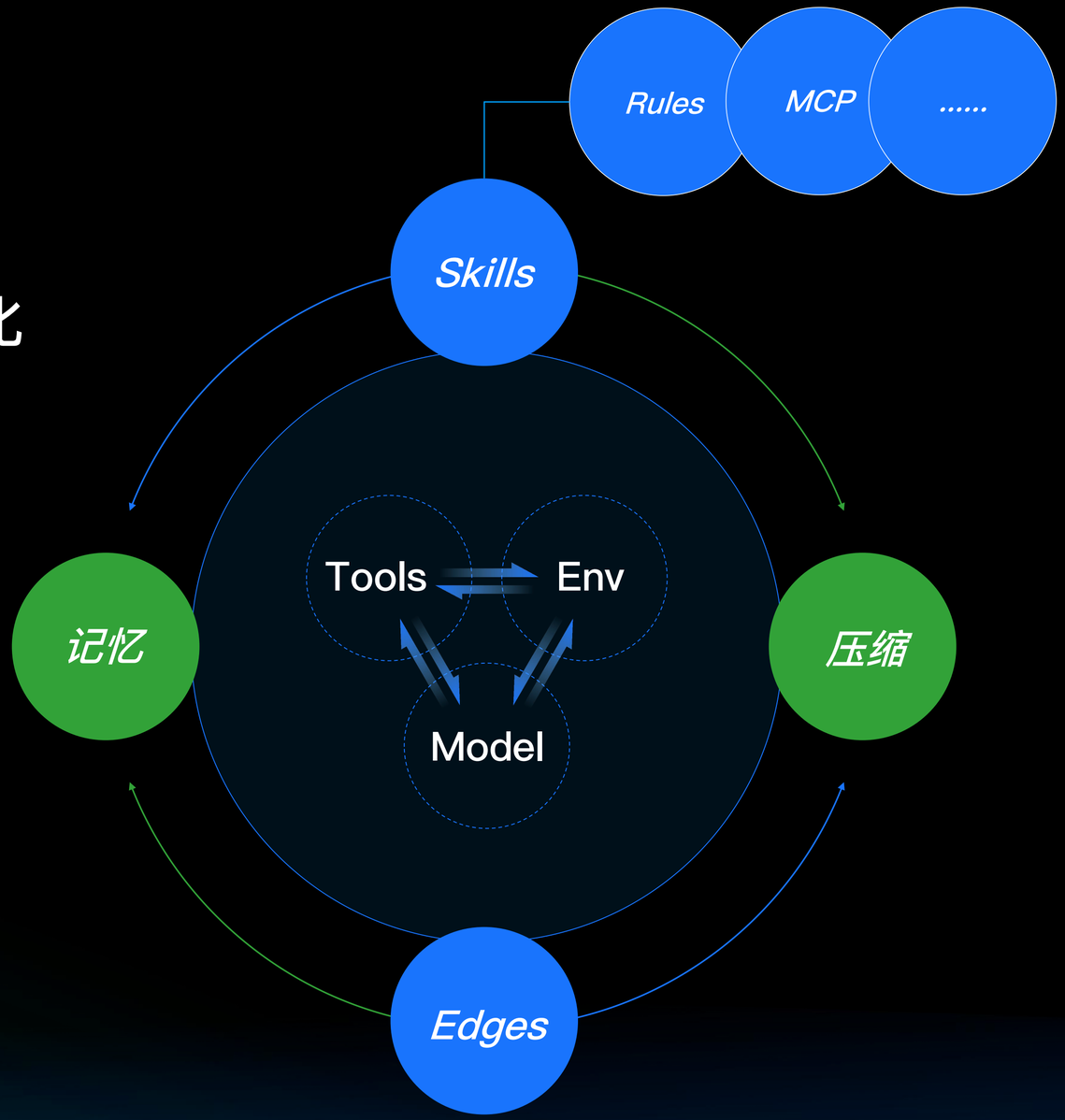
Agent Loop的基本框架

核心Loop：驱动Agent运转，感知外部环境变化

包含模型、工具以及Agent所运行的环境。很多Agent框架忽略模型本身「特性」的变化，尝试打造统一的框架适配所有模型，Comate的实践让我们清醒认识到这是错误的。因此，Agent框架必须能够发现模型的「变化」。

外层Loop：为Agent提供扩展性、鲁棒性

通过记忆、Skills等Context来为Agent提供更多接触环境、感知环境的手段，从而大幅延长Agent的「触角」。通过「压缩」和「Edges（边界情况处理）」来让Agent具备自愈能力。



打造一个通用Agent产品时，构建Agent Loop面对的核心问题

对模型解题能力的持续观测，动态调整Agent框架。

1. 随着模型的「解题」能力越来越强，构建Agent核心原则从「流程定义」变为「状态感知」，不再需要教模型怎么做，只需要告诉模型现在发生了什么
2. 针对Claude 4.6系列、GPT-5.3/5.4系列模型对与编码的细节处理已经足够优秀，完全不需要Spec Driven Development、Superpowers等开发模式/脚手架，反而拖累模型效果，浪费上下文

构建有效的评测级，重点不再是解题分数而是发现异常值。

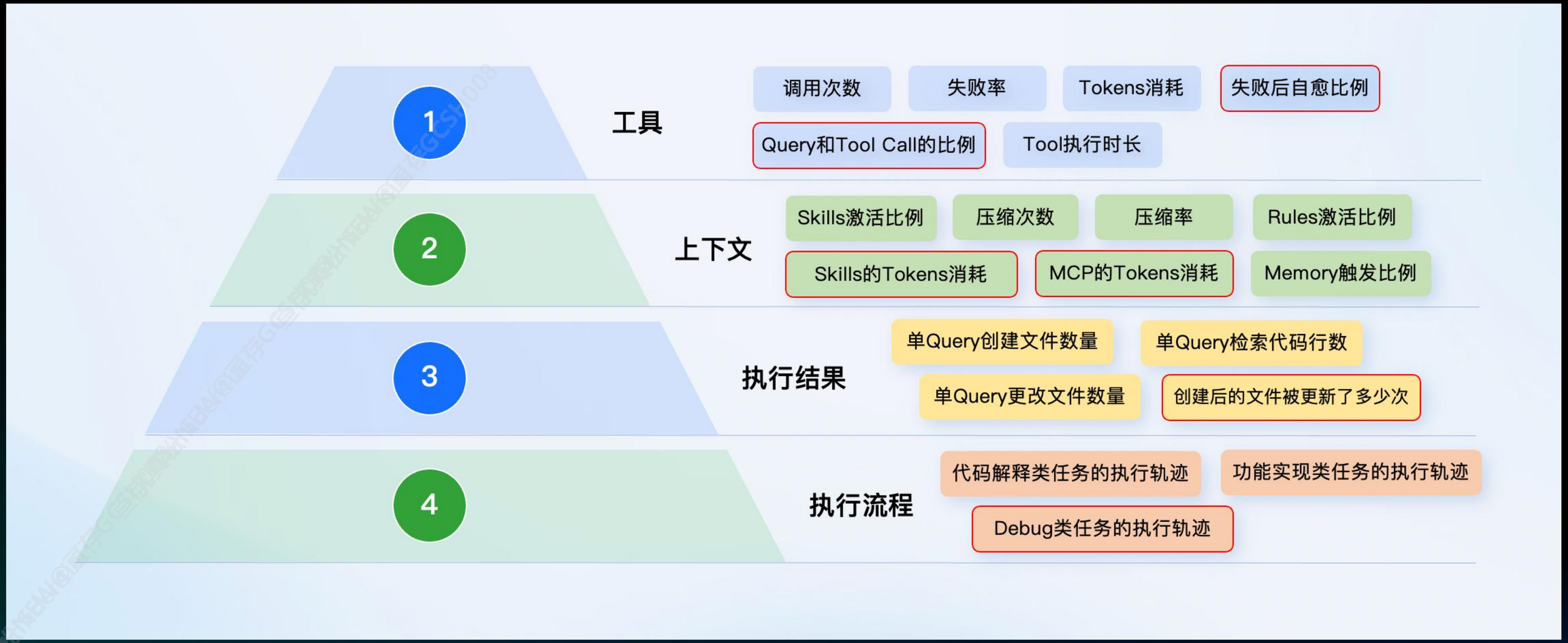
1. 单纯做面向SWE-Bench等通用方法的评测无法真正体现Agent的效果，评测集和对应的评判标准体现了Agent的「调性」
2. 评测单纯看分数是不够的，需要发现异常值，通过异常值发现模型的偏好

「人」也是Agent的Tool、Context，需要打破角色分工。

1. 传统软件研发协作流程将开发者分成前端、后端、测试等各个角色，这种协作适用于「稳定态」的软件，而基于模型构建的Agent应用是「混沌态」，无法有效适用

构建线上Agent执行数据的观测体系

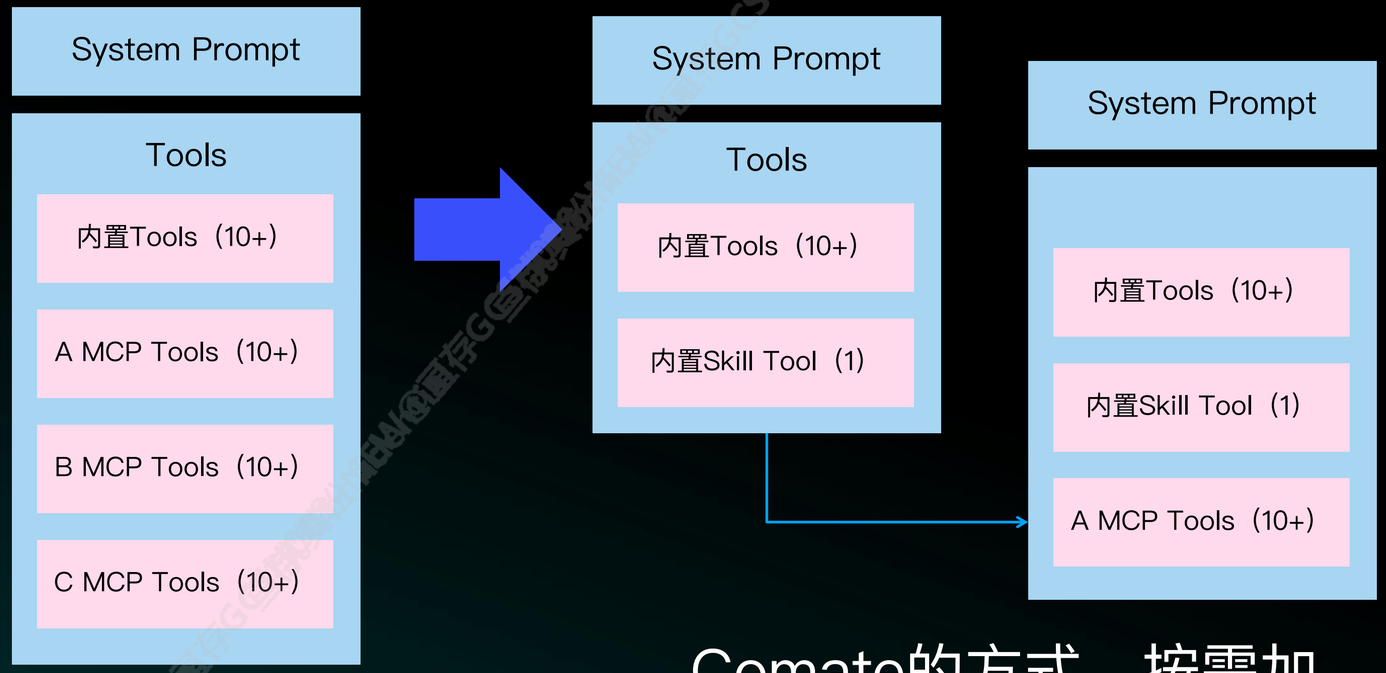
Feedback Loop：让 Agent 的行为可观测



实践：MCP和Skills的Tokens消耗对比强烈，推动我们快速优化

■使用Skills的方式实现MCP动态加载，节省98%的Tokens消耗

- ✓ Comate自动为每个MCP Server生成Skill风格的描述，并作为一个虚拟的Skill加载
- ✓ 模型根据Query自动判断是否需要加载整个MCP Server的Tools



业内主流的方式

Comate的方式，按需加载

```
用户在 Comate MCP 配置界面安装 MCP 服务器
            |
            ▼
Comate 自动为每个 MCP 服务器生成对应的 Skill 配置
（名称 + 高级描述，记录服务器能力概要）
            |
            ▼
每次请求时，skill 工具 description 中嵌入所有 MCP Skill 的轻量列表
（模型可见：服务器名称 + 能力概要）
            |
            ▼
模型根据任务判断：是否需要使用某个 MCP 服务器?
    |                              |
   否                             是
    |                              |
（不加载）          调用 skill(name="<server_name>")
                                   |
                                   ▼
                    Comate 实时拉取该 MCP 服务器最新工具列表
                    返回所有工具的完整 Schema 定义
                                   |
                                   ▼
                    模型阅读 Schema，选择合适工具
                    调用 use_mcp_tool(server_name, tool_name, arguments)
                                   |
                                   ▼
                          Comate 执行 MCP 工具调用
                          返回结果给模型
```

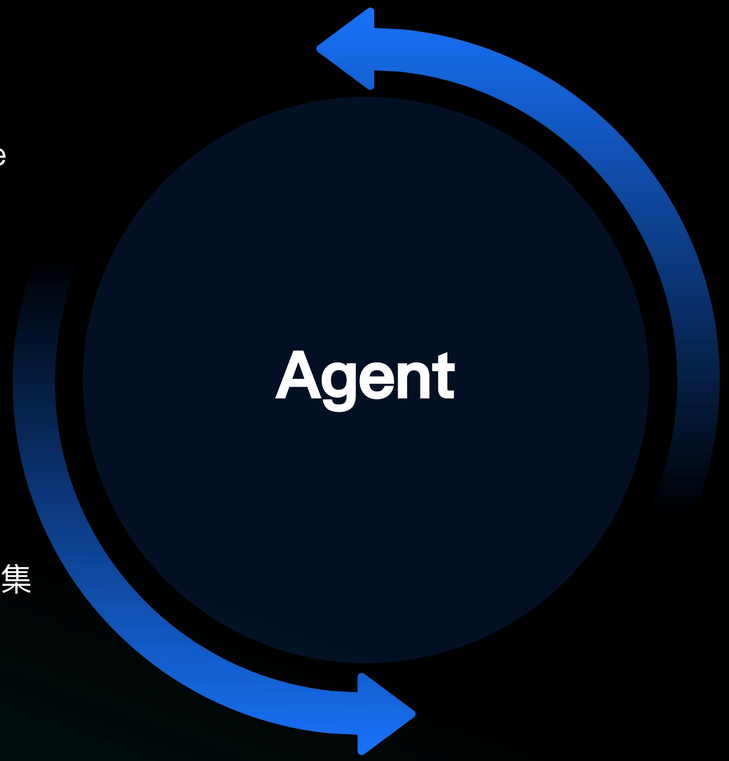
从业务本身挖掘评测集，善用Git Blame

为Agent提供作为评测集的代码库

- 必须是一个Git仓库，且有良好的Commit Message
  - 告诉Agent希望提取什么类型的Case作为评测

Agent通过分析Git Logs提取信息

- Agent开始逐个分析Commit判断是否适合作为评测集
  - 如果适合则抽象给出评测的Query、验证方法



启动另一个Agent进行验证

- 检查无误则启动另一个独立Agent进行验证
  - 验证结果经过Agent和人工检验

人工检查评测集的合理性并修正

- 检验Agent提取Case的合理性
- 请注意，一条评测Case一般要跨多个Commit，需告知Agent这样的情况存在

评测结果的度量，分析流程和异常值 —— 四象限分析法

Overall Score = 0.6 × Outcome Score + 0.4 × Execution Score

Outcome Score（结果质量 · 权重 60%）

- *Test Pass（实际测试通过，0.5）*
- *Functional Correctness（功能正确性，0.2）*
- *Completeness（完整性，0.1）*
- *Patch Quality（代码质量，0.1）*
- *Regression Risk（回归风险，0.1）*

Execution Score（执行效率 · 权重 40%）

- *Problem Understanding（理解能力，0.25）*
- *Agent Execution Quality（执行质量，0.75）*
  - *Investigation*
  - *Tool Usage*
  - *Control Flow*
  - *Task Management*
  - *Validation*
  - *Craftsmanship*

📐 OUTCOME 轴定义

Test 通过 outcome = 0.5 + llm_score/2，范围 [0.5, 1.0]

Test 不通过 outcome = 0 + llm_score/2，范围 [0, 0.5)

```
llm_score = (
  functional_correctness +
  completeness +
  patch_quality +
  regression_risk
) / 4
```

⚙️ EXECUTION 轴定义

核心指标 execution = agent_execution_quality（LLM评分）

高 Execution execution_quality ⩾ 0.7（执行规范、高效）

低 Execution execution_quality < 0.7（存在循环/混乱）

```
agent_execution_quality 含:
· Investigation & Localization
· Tool Usage / Control Flow
· Task Management / Validation
· Execution Craftsmanship
```

把人放到Loop里的两层含义

1

全员转型 —— 每个人都成为Agent Engineer

- 不只是开发者在用Agent，PM、UE、测试、售前都在用IDE写代码 —— 角色边界正在消失
- 转型的核心不是"学会写Prompt"，而是从"执行者"转变为"问题提出者"，能力瓶颈从编码速度变成了提出好问题的能力
- 协作模式从"大Feature拆小Task分给多人"变为"一人 + Agent一竿子捅到底"

2

把人看作Agent的工具和Context

- 理解中人是Agent的使用者、监督者。但在实践中，人本身就是Agent Loop的一部分 —— 人提供的判断、审美、业务知识是Agent无法自己生成的Context
- 当Agent遇到需要产品决策、视觉判断、业务逻辑确认的节点时，"调用人"和"调用Tool"没有本质区别

Baidu Comate
拥抱AI原生研发新范式

感谢聆听

扫码加好友(李杨)

AI原生研发新范式更深入交流

欢迎扫码了解

用科技让复杂的世界更简单