InfoQ 极客传媒

面向 Skills 的上下文工程

CodeBuddy Spec-Coding 的结合实战

演讲人：汪晟杰

腾讯 / 资深技术产品专家

AiCon

全球人工智能开发与应用大会

AiCon
全球人工智能开发与应用大会

目录

极客邦科技 2026 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

4月 · 北京 · 1200人

QCon 全球软件开发大会

会议时间：4月16-18日

- Agentic Engineering
- AgentOps
- 下一代模型架构与推理优化
- AI 原生基础设施
- 知识工程实践
- AI 安全

6月 · 上海 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：6月26-27日

- AI Infra 系统工程
- 多 Agent 协作与实践
- 多模态融合
- 模型训练与推理创新
- 数据平台与特征服务

8月 · 深圳 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：8月21-22日

- Agentic AI
- 轻量化与高效推理
- 多模态应用
- AI + IoT 场景实践
- AI 工业化落地

10月 · 上海 · 1200人

QCon 全球软件开发大会

会议时间：10月22-24日

- AI Agent
- Vibe Coding
- 智能可观测
- 推理基建
- 模型攻防
- AI x 创造力

12月 · 北京 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：12月18-19日

- 大模型架构创新
- 多模态 AI 产业融合
- 具身智能
- AI for Science
- 大模型安全

InfoQ 极客传媒

01 Spec Coding

提示词工程 — FEIT


F
E
I
T
目标：
为单个任务找到
“神奇的提示
词”。
反复试错和评测
测试指令跟随能力
增加示例和规则
按我的示例规则来
调整措辞内容（中/英）
不同模型的措辞理解、上下语意不一致
特殊强调
定义不同场景下有不同的思考策略

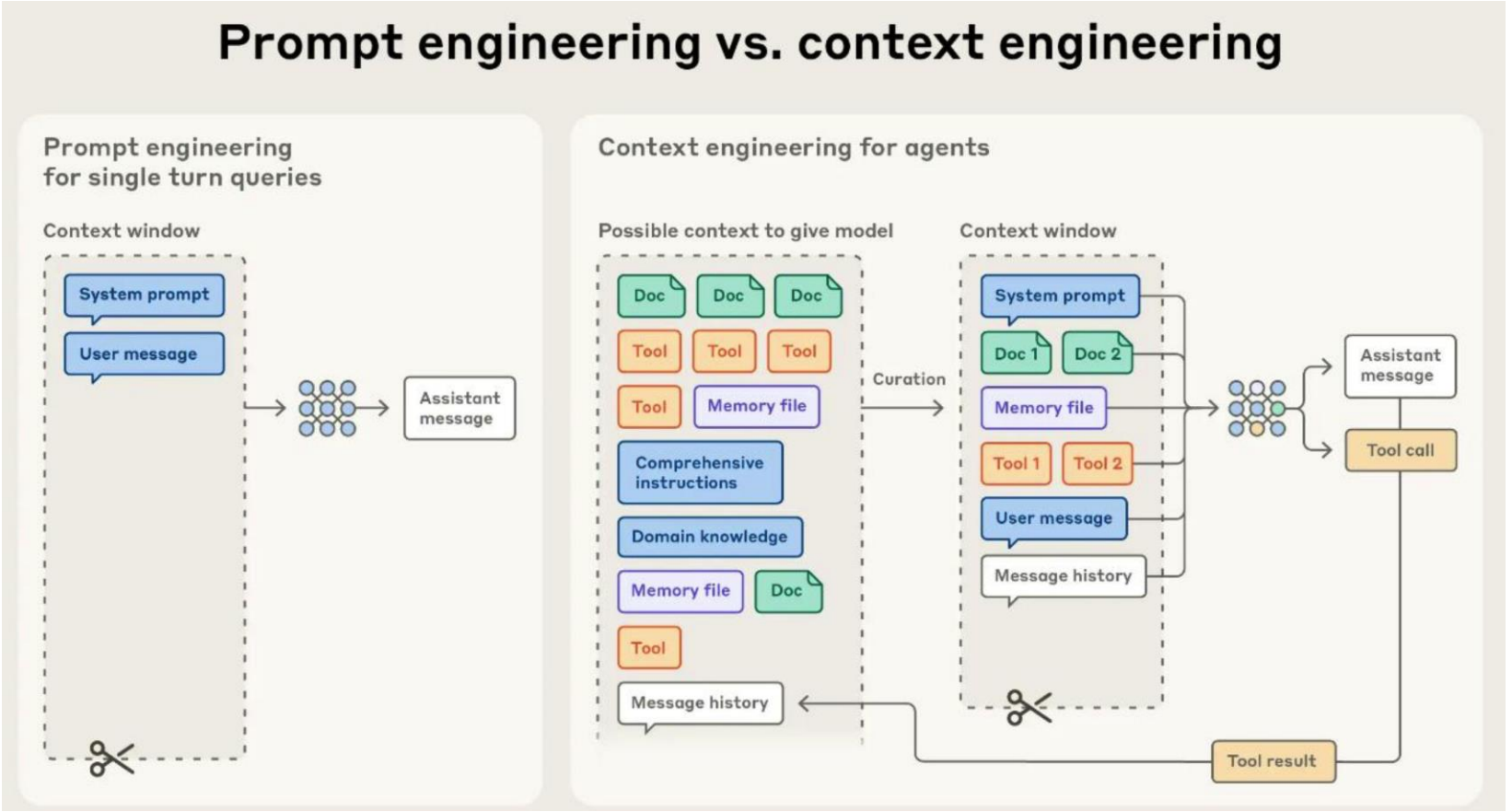
提示词工程带来的问题


一句话需求?
FAIL
一句话需求?
产品需求
产品验证交付
设计原型
草图?
测试验证
没单测没评审?
需求理解
使用 LLM 进行提效
持续运营
要用什么模型?
方向错误?
技术调研
技术可行性
计算加速
实现不了? BUG?
提交任务
任务
测试计划
任务过大过杂?
哪个是关键路径?

Context Rot

提示词工程是为了获得最佳推理结果而编写和组织 LLM 指令的方法，

上下文工程则是指在 LLM 推理过程中，动态规划和维护最优的输入 token 集合（集合包括任何可能进入上下文的信息）



魔法的真相：缺失的上下文

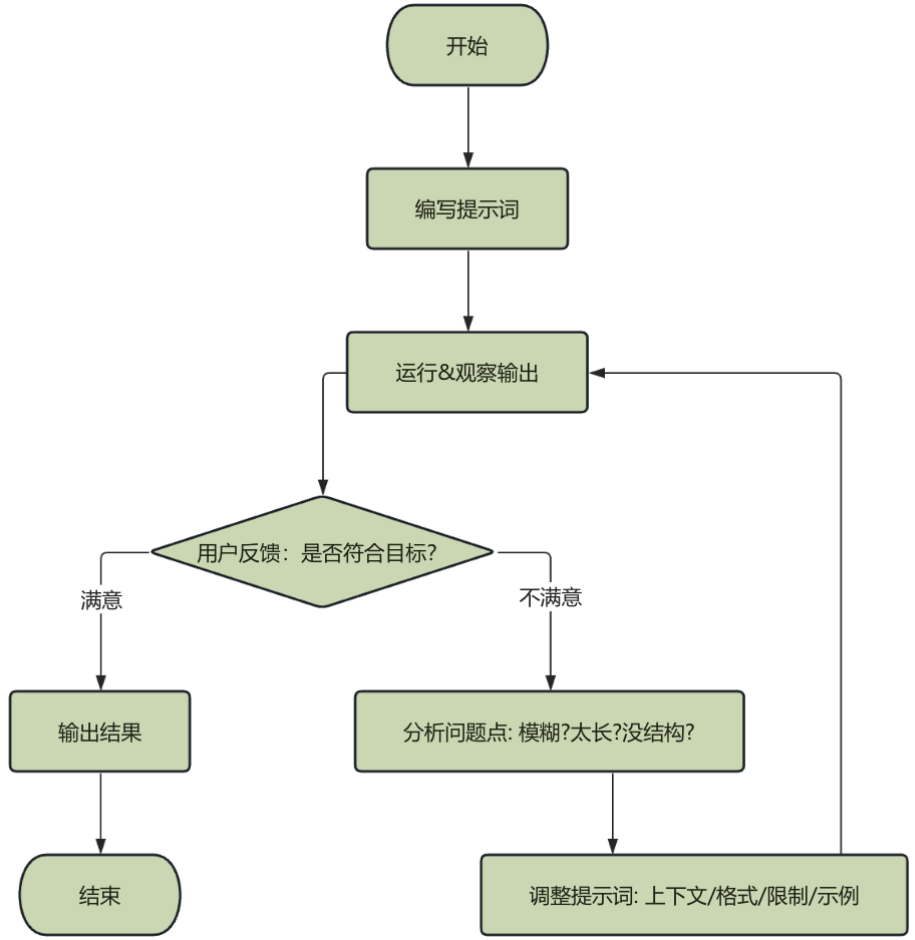
1 代码读取 / 上下文工具
- Read File（读取单文件）
- Read Directory / Tree
- Search（全文 / 语义搜索）
- Symbol / Definition Lookup
- Cross-file Reference

2 代码修改工具
- Edit File（Diff Patch）
- Insert / Replace Code
- Multi-file Edit
- Refactor（rename / extract）

3 终端 / 命令执行
- Run Command
- Run Tests
- Build / Lint

真正的差距不在于模型，而在于我们提供给它的信息生态系统。

Context Rot

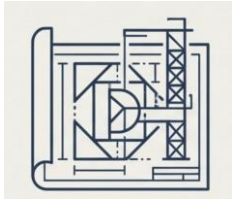


上下文工程是一门设计和构建动态系统的学科

我们的角色正在演进：从文字工匠到 AI 架构师

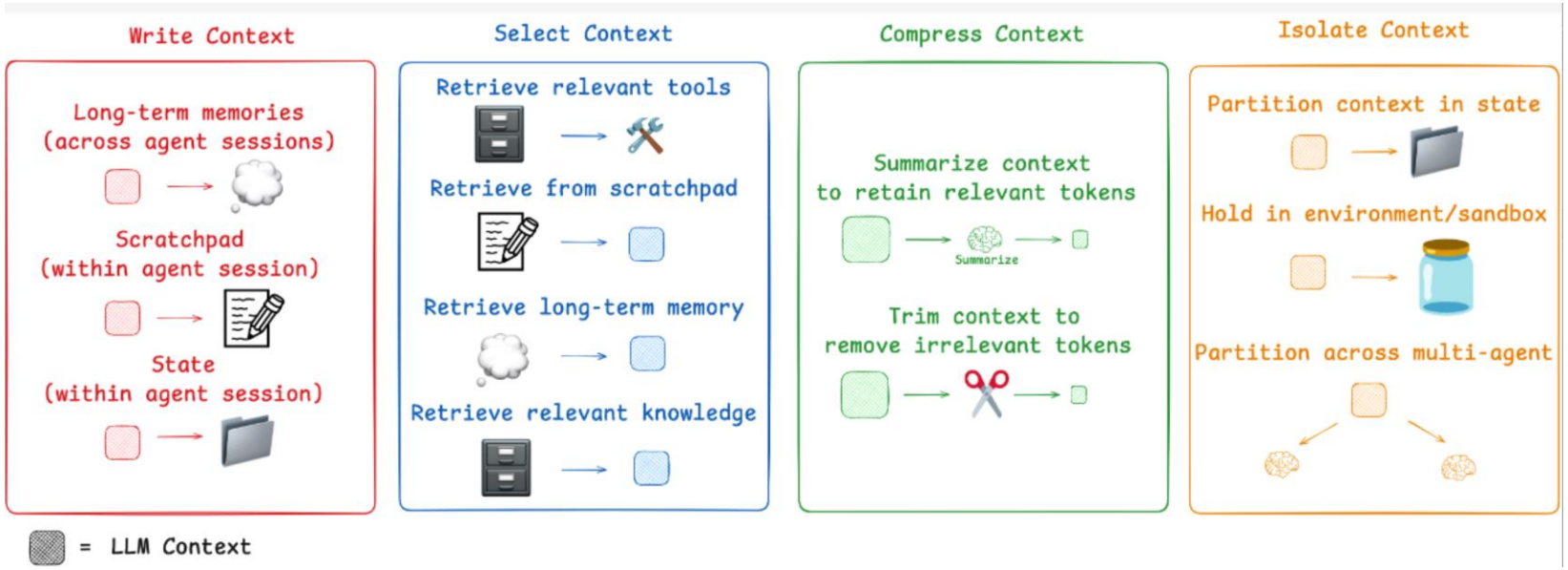
文字工匠 (Word-Smith)

雕琢单一、静态的提示词

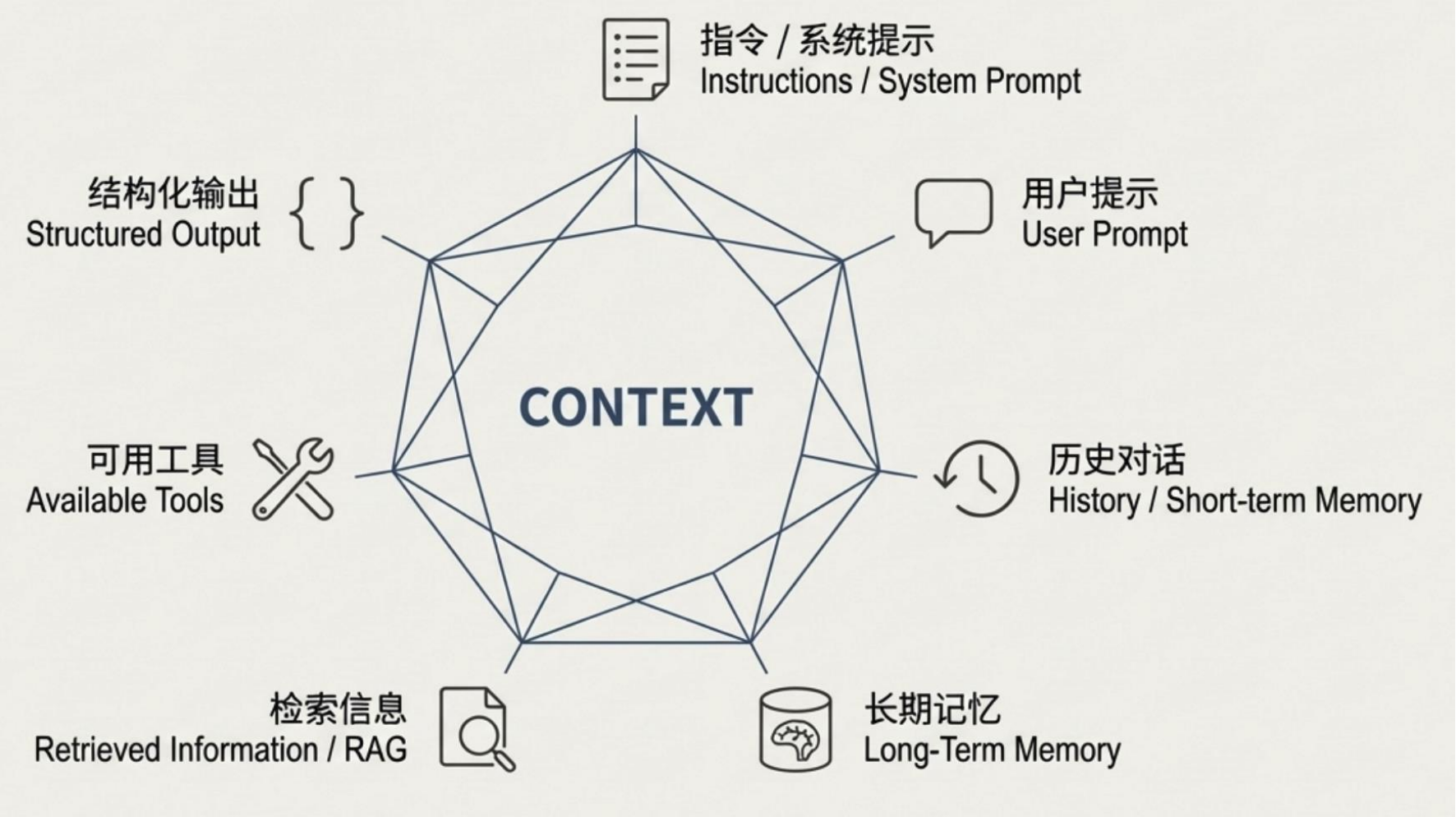


AI 架构师 (AI Architect)

设计动态、完整的信息生态系统



上下文的完整解构


指令 / 系统提示
Instructions / System Prompt
结构化输出
Structured Output
用户提示
User Prompt
CONTEXT
可用工具
Available Tools
历史对话
History / Short-term Memory
检索信息
Retrieved Information / RAG
长期记忆
Long-Term Memory

上下文工程2.0 的三大支柱

规范驱动的开发 (Specification-Driven Development)

设计蓝图 (The Blueprint)

智能体的执行循环 (The Agentic Loop)

动态引擎 (The Engine)

上下文高效的系统 (Context-Efficient Systems)

智能流水线 (The Assembly Line)

传统规约 到 AI 驱动型规约的变革

1. 函数规约（Preconditions, Postconditions, Invariants）

假设你在开发一个计算银行账户余额的系统，使用规约编程来确保“存款”和“取款”操作的正确性。

```python
class BankAccount:
    def __init__(self, initial_balance: float):
        self.balance = initial_balance

    # 预条件：存款金额必须大于0
    # 后条件：账户余额会增加相应的金额
    def deposit(self, amount: float):
        assert amount > 0, "存款金额必须大于0"
        self.balance += amount

    # 预条件：取款金额必须大于0且账户余额充足
    # 后条件：账户余额会减少相应的金额
    def withdraw(self, amount: float):
        assert amount > 0, "取款金额必须大于0"
        assert self.balance >= amount, "账户余额不足"
        self.balance -= amount
```

2.模块间规约（Interface Contracts）

在微服务架构中，两个服务之间可能需要遵循一个共享的接口协议。这个协议定义了双方期望的输入输出格式和行为，从而保证它们能够正确协作。

```python
class WeatherService:
    # 输入：城市名称
    # 输出：天气数据字典，包含气温、湿度、天气状况等信息
    def get_weather(self, city: str) -> dict:
        pass  # 模拟天气查询

class NotificationService:
    # 输入：天气数据（字典），包含温度、湿度等信息
    # 输出：发送通知，返回发送状态
    def send_weather_alert(self, weather_data: dict) -
        pass  # 模拟发送通知
```

3. 异常处理规约

假设我们正在开发一个在线购物系统，用户需要提供有效的信用卡信息进行支付。如果支付失败，我们需要提供一个清晰的错误信息，并要求在支付流程中进行适当的异常处理。

```python
class PaymentSystem:
    def __init__(self):
        self.balance = 1000  # 用户余额

    # 预条件：信用卡信息必须有效，支付金额必须大于0
    # 后条件：支付成功后余额减少
    def process_payment(self, credit_card_info: dict, amount: float):
        assert self._is_valid_credit_card(credit_card_info), "信用卡无效"
        assert amount > 0, "支付金额必须大于0"
        if self.balance < amount:
            raise ValueError("余额不足")
        self.balance -= amount
        return True

    def _is_valid_credit_card(self, credit_card_info: dict) -> bool:
        # 假设验证信用卡逻辑
        return True
```

AI 编程从 Vibe Coding 到氛围编程

传统编程 (Traditional Coding)

特征

- 传统开发者主导的完全纯手动编码
- 手动调试过程
- 依赖浏览器搜索引擎/开发者社区等辅助开发

效果

- 逐行编码，比较低效

学习曲线及要求

- 陡峭的学习曲线，需了解编程底层技术，需自主学习能力和积累项目中实战经验

氛围编程 (Vibe Coding)

特征

- 基于AI 智能体和AI对话至上，采用自然语言描述需求，实现多文件代码生成，生成执行的应用
- 精准描述需求和任务表达有助于 AI 生成代码质量

效果

- 实现工程级别开发，中等效率

学习曲线及要求

- 学习曲线中等，非程序员(如 产品、设计等小白用户）也可编码，侧重和锻炼表达功能能力

规约编程 (Specification-Oriented Coding)

特征

- 在 Agent 至上，基于规范和设计共识驱动 AI 全栈开发，批量生成业务代码
- 结构化沟通和系统设计，生成代码包含所需的前提和意图
- 多智能体协作，先共识，帮你理清思路，集成系统思维澄清

效果

- 规范文档和共识协作，实现系统级别完整意图的规范化代码，效率高

学习曲线及要求

- 学习曲线高，弥补 Vibe Coding 中的痛点，对规范化、系统化有更高全局要求和把控能力，锻炼编写能充分捕捉意图和价值观的规范能力

未来，基于AI的个人氛围编程，以及过渡和适应专业团队协作的规约编程，两种开发范式并存

Speckit 成为规约编程的标配

规范驱动开发(Specification-Driven Development - SDD)

“Every hour spent on planning saves 10 hours of rework.”

02

CodeBuddy 如何落地上下文工程

用工程纪律约束AI:从规则引擎得到的启示

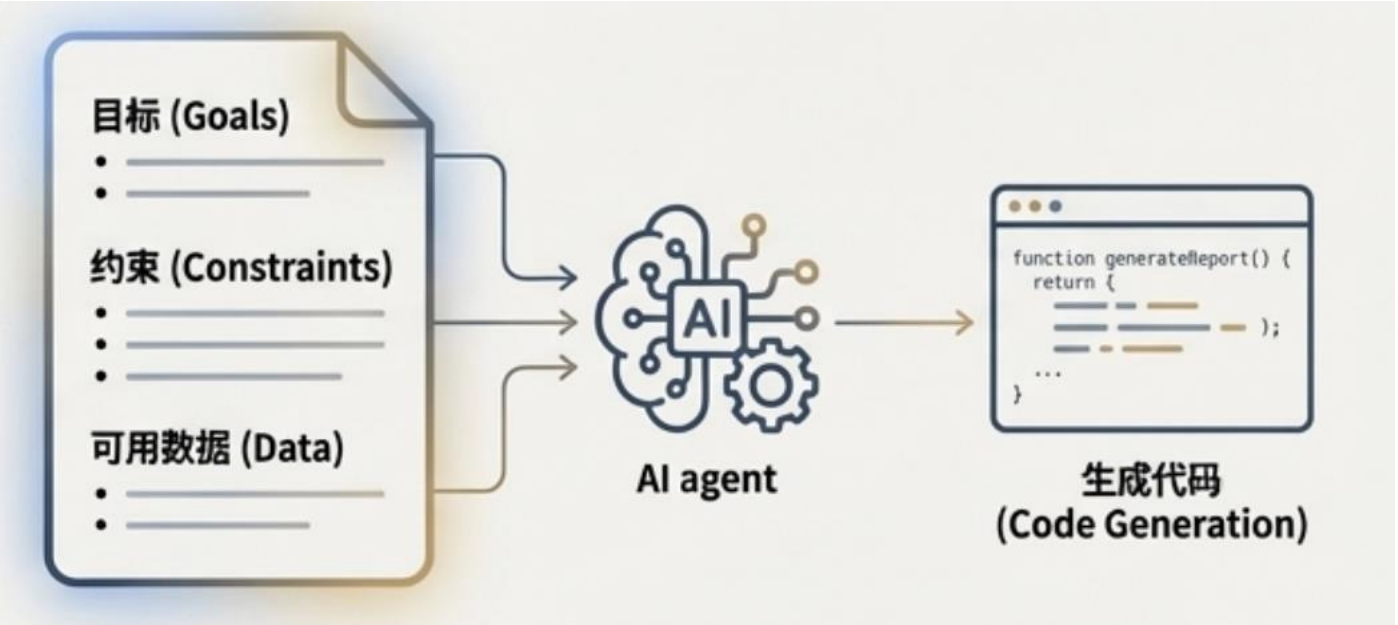
文件名	仓库数量/采用情况	文件位置	主要支持工具	文件类型	标准化状态
AGENTS.md	超过20,000个开源项目	仓库根目录或子目录	OpenAI Codex, GitHub Copilot, Google Jules, Cursor, Aider, RooCode, Zed, Factory等20+工具	Markdown指令文件	开放标准，正在成为主流
Claude Code Skills	无具体统计数据	作为技能包存在于.claude目录	Claude Code, Claude.ai Web, Claude Desktop, Claude API	技能包（包含SKILL.md + 脚本 + 资源）	Anthropic专有格式
GEMINI.md	无具体统计数据	仓库根目录或.gemini/GEMINI.md	Google Gemini CLI	Markdown指令文件	Google专有格式
Github Copilot Instructions.md	无具体统计数据	.github/copilot-instructions.md 或 .github/instructions/*.instructions.md	GitHub Copilot (包括Agent模式)	Markdown指令文件	GitHub专有格式
CodeBuddy.md	**腾讯内部、COdeBuddy 开发者**	**仓库根目录（基于CodeBuddy.ai规范）**	**CodeBuddy AI Assistant (VS Code扩展)**	**Markdown指令文件**	**CodeBuddy 的 MD 格式**

用工程纪律约束AI:从规则引擎得到的启示

规范即合约 (Specification as Contract)

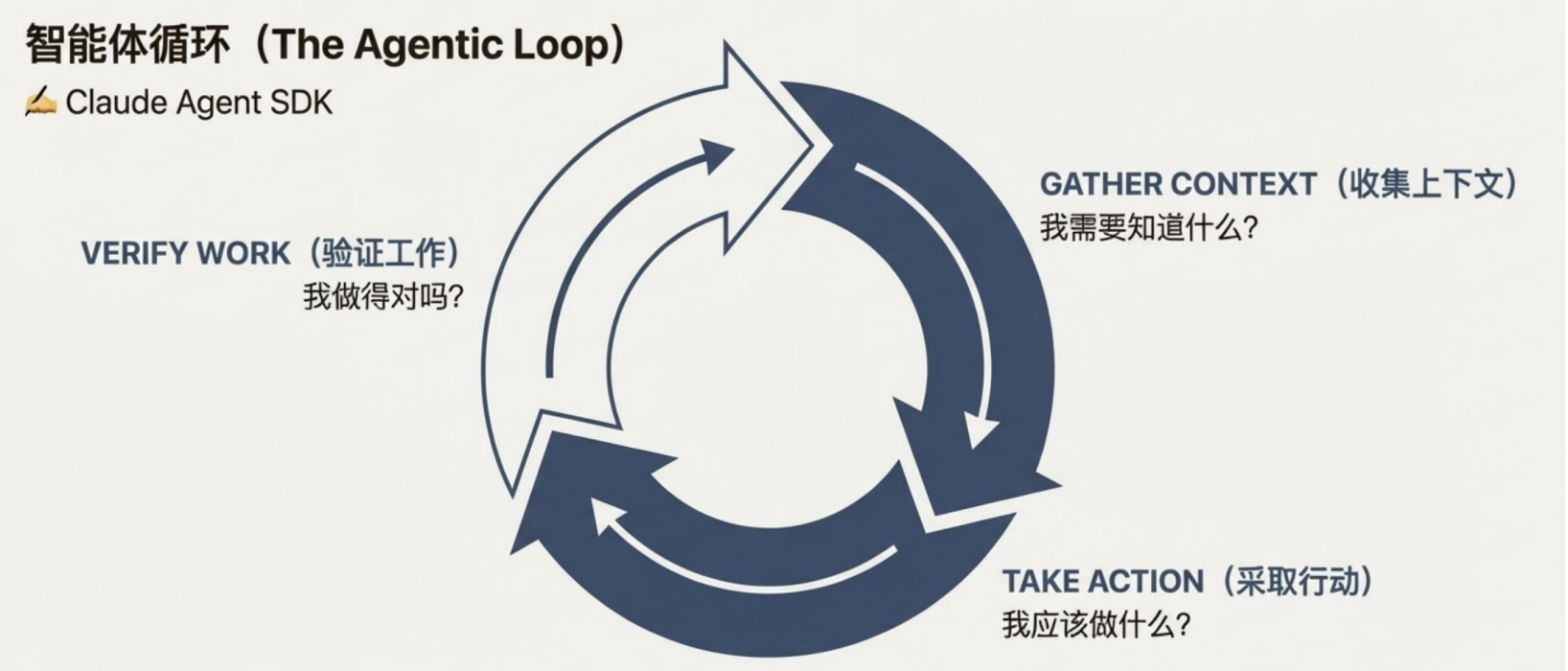
Rules 和 SpecKit

为 AI Agent 提供对任务目标、约束和可用数据的明确定义。
Agent 基于这份“合约”生成代码。



动态引擎 – Agent 的思考与行动循环

- Gather Context：理解工程与目标
- Take Action：真实修改代码与系统
- Verify Work：自动验证结果并驱动下一轮



“给 Claude一台计算机”——让 Agent 像人类一样工作,通过终端访问文件、运行代码、迭代修正。

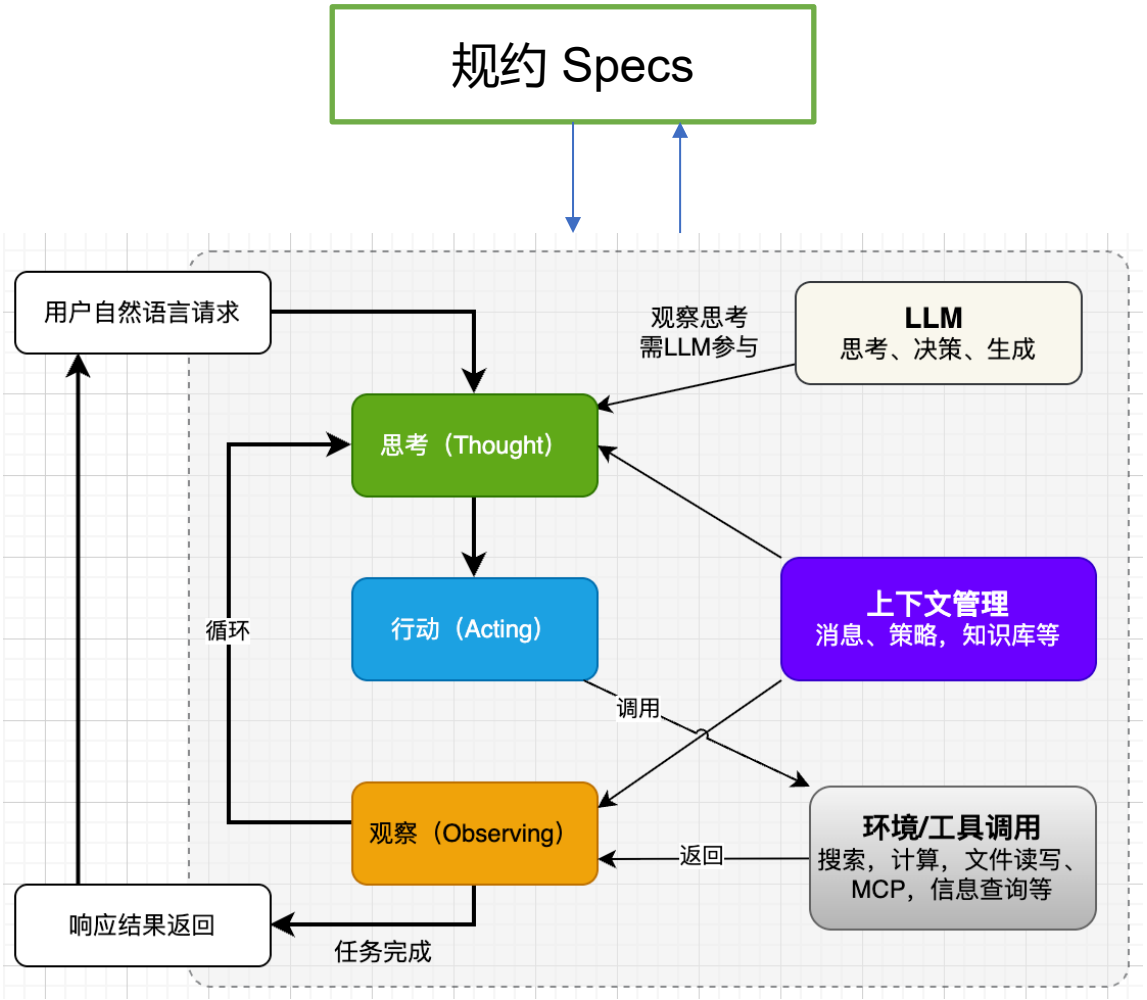
构建一个可靠的执行循环

文件系统 Agentic Search	子智能体(Subagen)	定义规则	反馈
将文件结构作为上下文工程的一部分。	并行处理任务，隔离并压缩上下文。	如代码linting 或校验邮件地址格式。	视觉反馈(Visual Feedback):提供HTML渲染截图，让模型进行 视觉验证。

获取上下文的几个大模块

Code Agent 整体架构

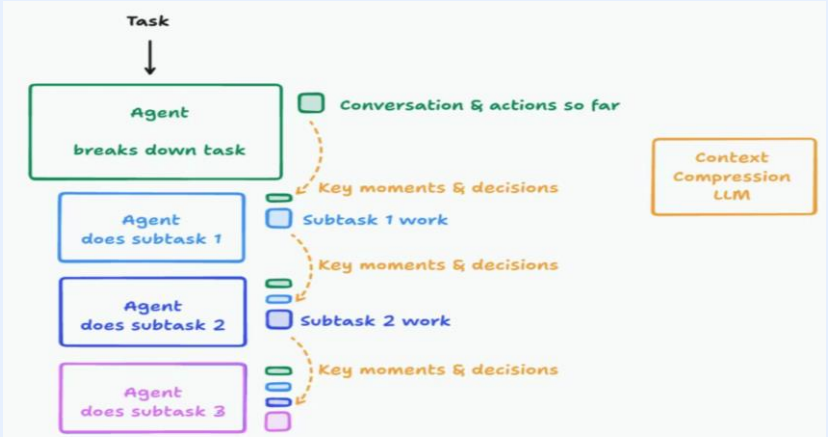
- 基于 ReAct 范式，结合多种扩展组件，系统性提升智能体的自主编码能力
  - 环境/工具集定义：项目探索，代码编辑，命令执行，MCP扩展等
  - 思考模式：调用工具前，进行思考过程，包括：任务拆分，错误分析，需求理解，反思等
  - 上下文管理：阶段式消息总结、策略调整和记忆管理等
  - 工程加速：Prompt Caching, 减少重复计算，提升响应效率

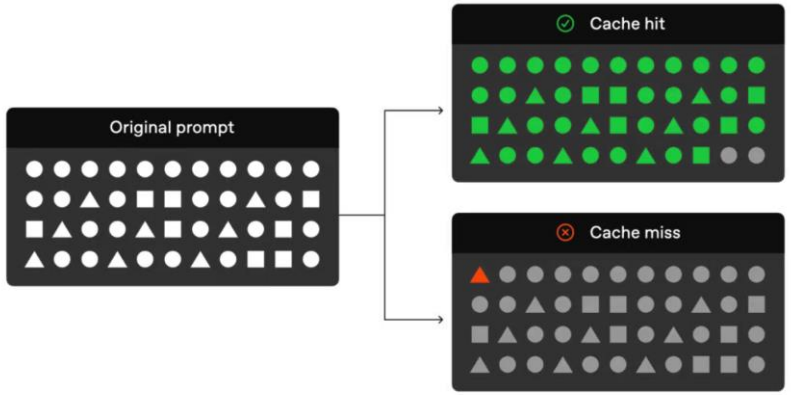


基于特定规则的上下文压缩

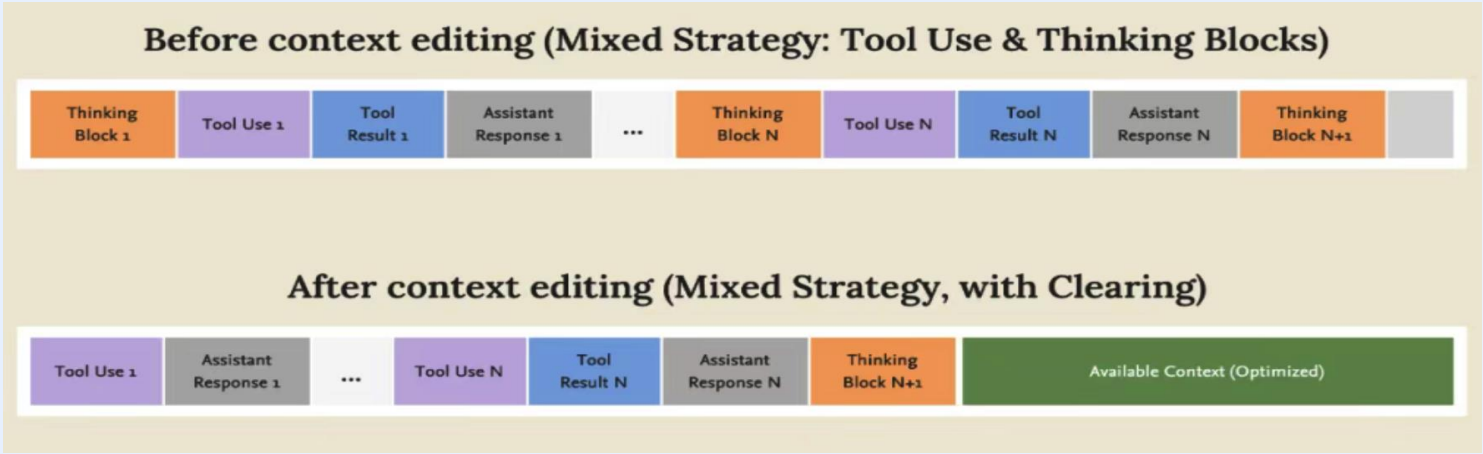
问题 — Prompt Caching Failure

基于模型的语义压缩



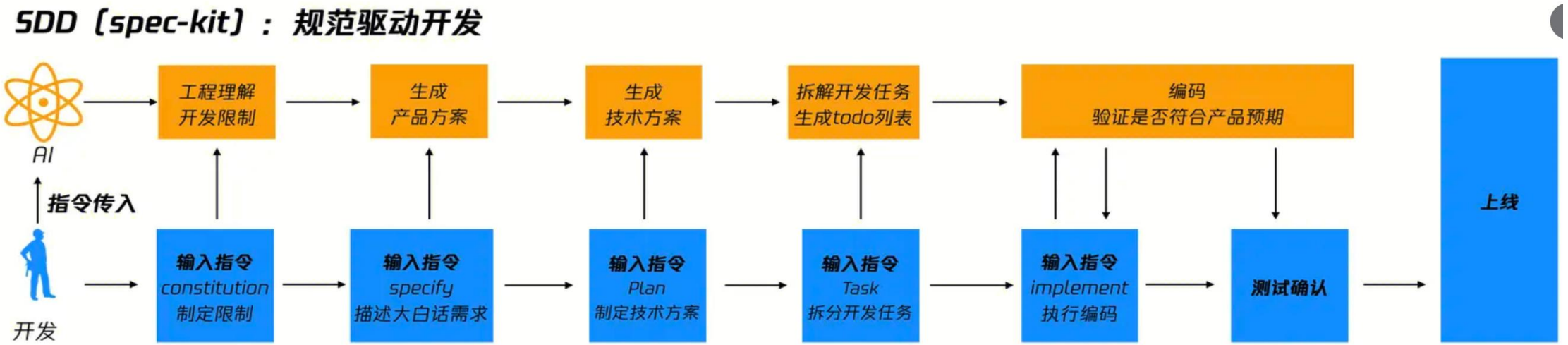


基于规则的历史结构化剪枝

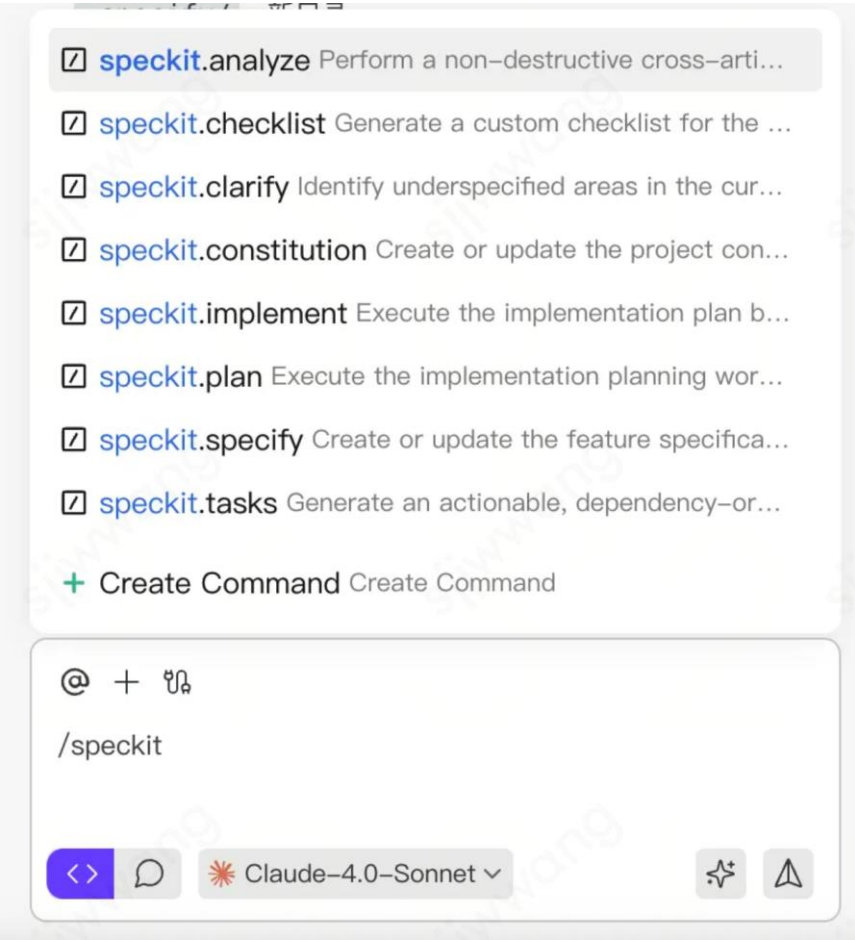


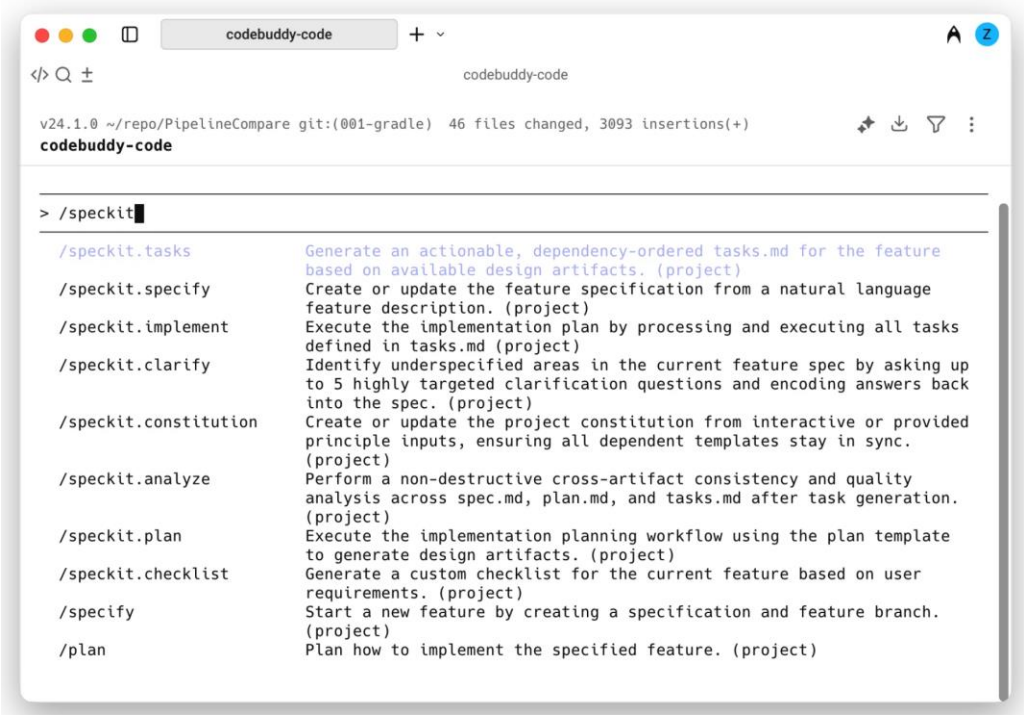
SDD 驱动

- 明确表达目标与动机（What/Why），规避过早技术选型（不要在规格阶段锁死 How）。
- 反复迭代与澄清，先完善规格，再进入实现。
- 先验证计划（可行性、契约、测试口径）再编码。
- 让代理/自动化负责细节实现，人侧把控规格质量与决策。



Speckit 成为规约编程的标配





Fast Path Workflow

小缺陷开发模式的推荐的 SpecKit 指令工作流

场景类型	推荐流程	是否生成 spec.md
复杂 bug（跨模块）	Clarify → Plan → Tasks → Implement	否（可引用旧 spec）
小 bug（微逻辑、UI、配置）	Plan → Tasks → Implement	否，直通快修流程
配置错误 / 更新脚本	Tasks → Implement	否，仅单步任务

缺陷修改类的最佳流程

独立的产设研上下文

规划 → 设计 → 开发

产品经理　需求智能体

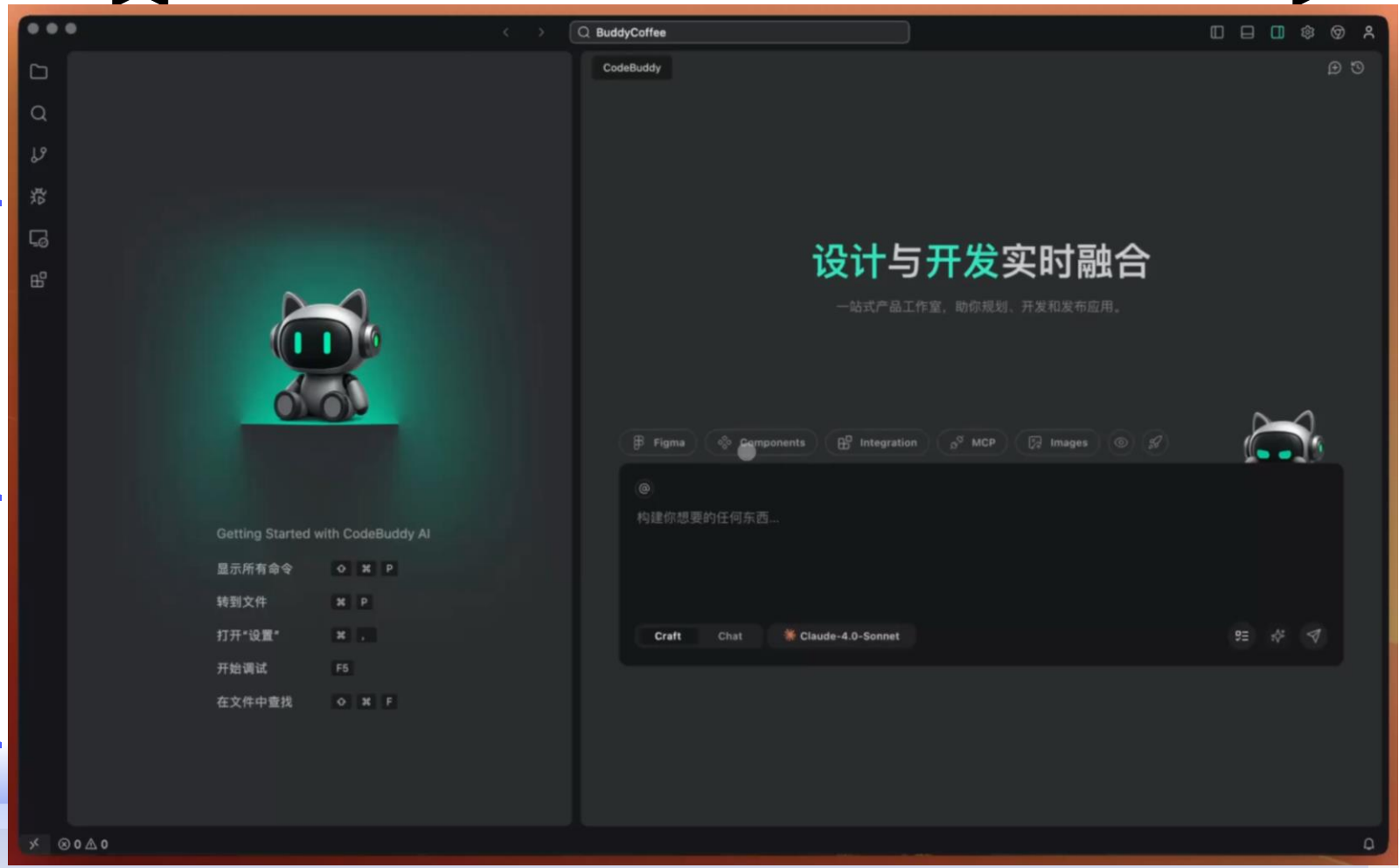
为产品需求创建详细的问题，
为AI提供清晰的蓝图

设计师　设计智能体

精准理解需求将创意和现有的产品设计变成功能性原型、网页应用程序和交互式用户界面

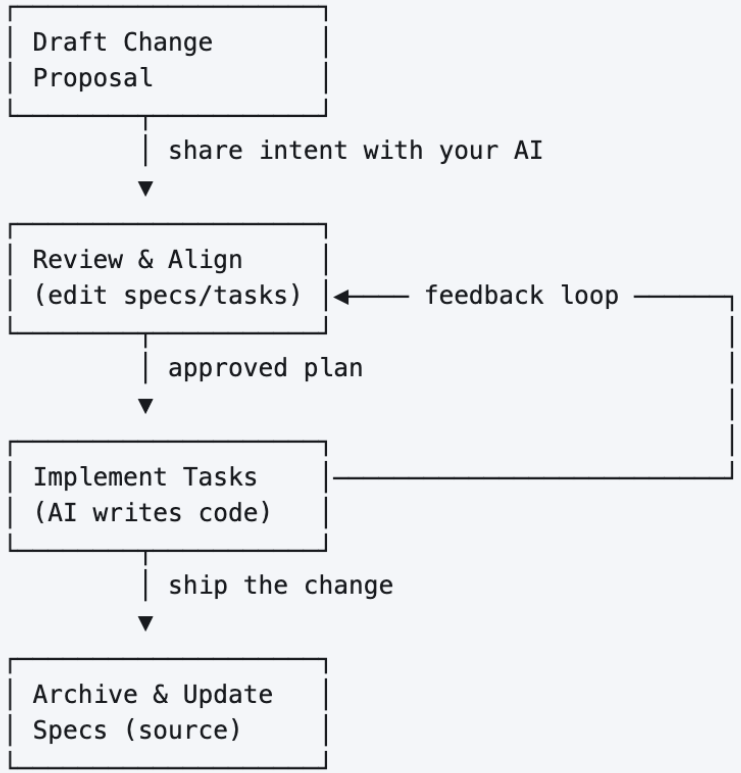
开发工程师　开发智能体

自主完成代码开发,以卓越的质量完成代码工程实现



OpenSpec

更少步骤的规约编程的尝试



1. Draft a change proposal that captures the spec updates you want.
2. Review the proposal with your AI assistant until everyone agrees.
3. Implement tasks that reference the agreed specs.
4. Archive the change to merge the approved updates back into the source-of-truth specs.

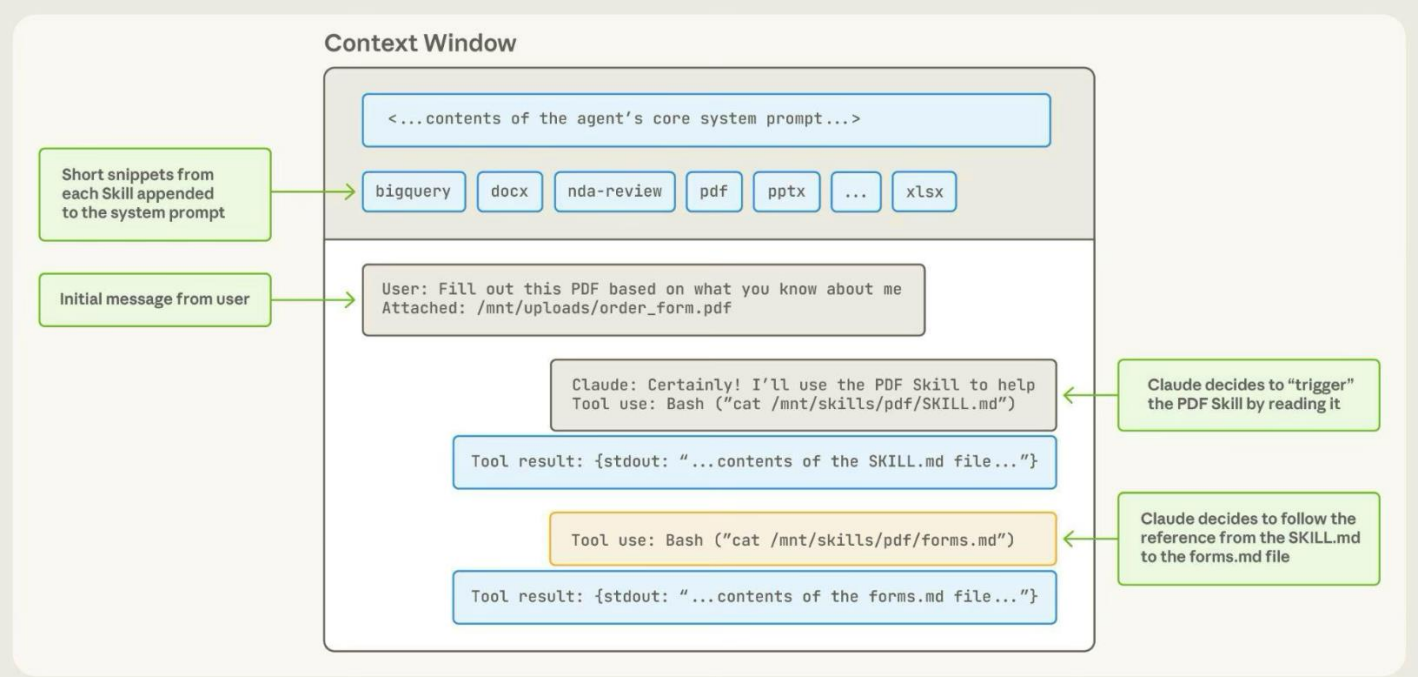
特征：节奏快、切口小、循环短
适用：现有项目功能演进、多功能并行、快速迭代场景
🕑 **时间对比**：OpenSpec 平均快约 **46%**，主要节省在规范生成与任务分解阶段。

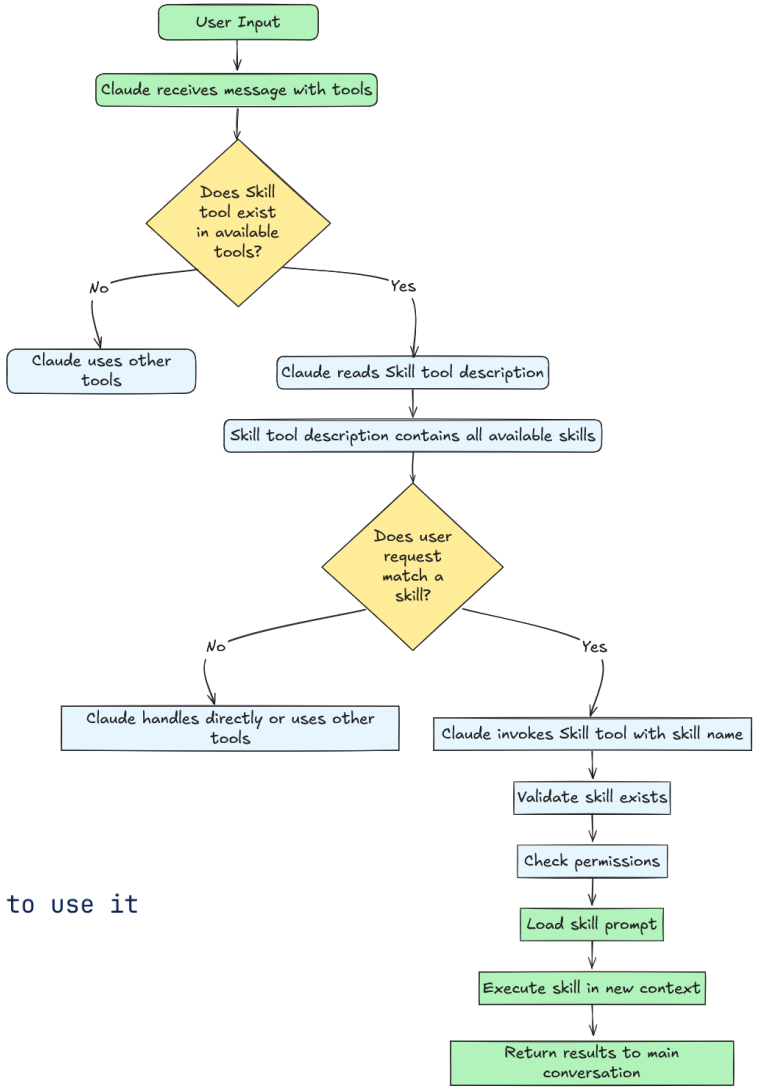
```
openspec/
├── specs/spec.md              # 当前稳定规范
├── changes/feature-x/         # 进行中提案
│   ├── proposal.md
│   ├── tasks.md
│   └── spec-deltas/
└── archived/                  # 已归档历史
```

03

CodeBuddy Agent Skills 实践路径

Claude Skills






```
---
name: your-skill-name
description: Brief description of what this Skill does and when to use it
---

# Your Skill Name

## Instructions
[Clear, step-by-step guidance for Claude to follow]

## Examples
[Concrete examples of using this Skill]
```

CodeBuddy Skills - 渐进式披露机制

不仅是技能封装

Skills 的价值远不止于将功能封装成可复用的组件，其真正核心在于**对上下文管理的深度探索与系统化设计**。

上下文组件的统一管理

Skills 上下文元素整合为统一的调度系统，实现**模块化、标准化、可复用**的能力封装。

System Prompt　Prompt 历史　Rules 规范

RAG 知识库　MCP Server

上下文管理架构

Skills — 统一调度与封装

- **System Prompt** — 系统级指令
- **对话历史** — 上下文记忆
- **Rules 规范** — 行为约束
- **RAG 知识库** — 外部知识
- **MCP Server** — 工具与服务集成

> **核心理念：** Skills 通过统一的上下文管理，实现了从"Prompt 工程"到"能力工程"的进化。

Agentic Skills

Agent 的定位

Agent 从架构层面可归为**基于模型和资源层之上的应用层。**

- 可定义 System Prompt
- 可引用外部知识库
- 可接入 MCP 等外部工具

Skills 的定位

实际过程中，我们更希望关注**问题解决**，而非底层配置。

- 提供能力与服务中间层
- 标准化组合与封装
- 细粒度拆解与执行

协作架构：互补共赢

Agent 职责

- 任务拆解
- 流程规划
- 任务执行
- 反思优化

Skills 职责

- 能力封装
- 工具编排
- 标准化执行
- 质量保证

Agentic Skills

通过渐进式披露机制和模块化设计
Skills 将团队的流程、方法、工具、规则、技术和个体经验
转化为可复用、可传承的标准化能力

轻量高效

分层调用，上下文始终保持最优状态

模块化设计

标准化封装，易于维护和扩展

智能协作

让 AI 协作更智能、更高效、更确定

解决的五大实际问题（上）

01 **封装专业知识，解决「专业问题」**

将特定领域的**最佳实践和操作流程**封装成可复用的技能。

领域知识封装　最佳实践沉淀　高效任务完成

02 **扩展 AI 能力边界，解决「能力问题」**

让 AI 能处理**更专业、更复杂的任务**及更多能力边界的事情。

能力边界扩展　多技能协同　持续学习优化

03 **上下文窗口的「成本问题」**

通过**分层加载**，在复杂任务中可降低 **40%-60%** 的上下文占用。

传统方式　Skills

知识沉淀
将隐性经验转化为可复用的标准化能力

能力扩展
突破 AI 原生能力限制，处理复杂专业任务

成本优化
显著降低上下文占用，节约 token 成本

解决的五大实际问题（下）

04 知识的「可复用性」

Skill 文件可**一次创建、共享、复用、更新**，团队成员共用同一技能库。

- 一次创建
- 持续复用
- 团队共享
- 集中更新

05 提供 SOP 工作流模板，执行的「确定性」

Skills 可以**调用脚本完成具体操作**，把任务处理流程（SOP）标准化。

- 流程标准化
- 减少理解偏差
- 效率大幅提升

知识资产化：隐性经验第一次以结构化形式沉淀，成为团队的可复用资产

执行可靠性：标准化的 Skills 让模型执行更可靠，也更容易追踪和优化

04

企业中上下文工程的落地指南分享与思考

以 CodeBuddy 为例的 AI Coding 产品架构

应用群体

- **独立 AI IDE and 主流 IDE 插件: 前者**提供高效友好的人机交互界面，很好地服务技术小白、产品经理、设计师和专业开发者，覆盖从需求、设计到编码的全流程。**后者**尊重开发者的选择权，满足其需求
- **CodeBuddy Code CLI:** 提供开放集成和自动化能力，很好地服务 DevOps 工程师和系统架构师，集成研发流程中，提升自动化运作效率

能力构建

- **研发流程助力:** 专注于研发流程规划、设计、编码、测试和部署的关键阶段，并对每个阶段进行助力，提升研发效率和加速流程变革。
- **腾讯生态融合:** 串联司内工蜂、智研、TAPD 等研效生态及腾讯云 IaaS 和 PaaS 产品，提供云数据快速访问并实现无缝云部署；连接小程序生态，提高小程序开发、调试和发布的效率；构建腾讯特色平台

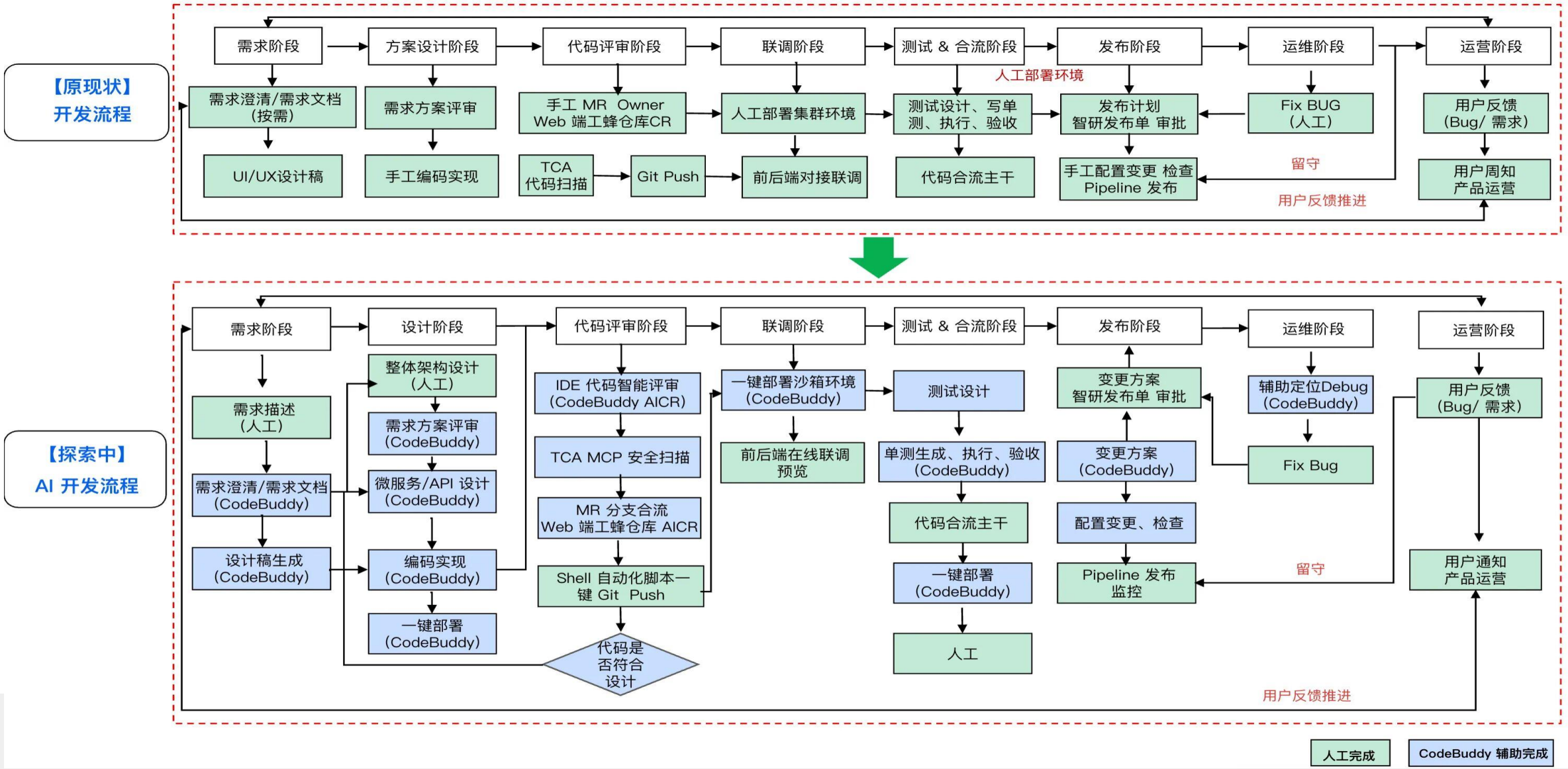
效果提升

- **上下文平台: 建设 Knot 上下文平台，构建上下文工程，提高上下文信息的密度和有效性，减少无效 token 消耗，提升 Agent 准确性**
- **模型: 收集和扩充语料库，通过与模型的后训练来改进模型生成效果**

企业用户按需选择

AI Plugin 智能副驾	AI IDE 一站式 AI 工作台	AI CLI Agent 操作系统
• 集成至主流IDE • 企业内 AI Coding 快速落地 •	• “集成开发环境”转为“智能开发环境” • 产设研一体，软件工程全生命周期覆盖	• 命令行交互 • 批处理、多系统异步协作，7x24 运行 • 深度融入云+DevOps与云原生

AISE中不同的上下文信息



编码阶段中不同的上下文信息

项目初始化 → 01 → 进行需求规约化（基于 EARS 语法进行表达，生成需求文档） → 02 → 进行架构方案设计（生成设计文档） → 03

CodeBuddy + Git 本地项目开发初始化

1. 在本地电脑创建本地项目工作目录、CodeBuddy、Git
2.在 CNB 中创建远端托管仓库，并给到 Craft Agent 进行本地Git 配置
3.将本地代码通过CodeBuddy IDE Craft Agent 进行 Git 版本提交

需求规约化： 基于 EARS 进行结构化表达

1.使用 EARS 需求语法澄清需求，用户管理员需开发用户管理系统需求，验收标准如：WHEN [condition/event] THE SYSTEM SHALL [expected behavior]
2. 附加信息，生成 Requirement Rules
3.在CodeBuddy中配置Project Rules 并经我确认方案后再开发，需有前端UI效果

设计规约化：AI生成架构设计

制定或引入现有编码规范规则，进行代码生成和评审 → 04 → AI 测试 → 05 → 依规约生成 AI 测试报告

部署规约化：AI 发版

1.利用规则智能发版
2.实现部署到云上 CloudStudio

测试规约化：AI 测试

1.利用规则生成测试 HTML报告并进行完善

编码规约化：AI生成架构设计

1.配置 JS/TS & Go 规范 到 Project Rules
2.借助规范，生成 Go Hello World 服务的代码进行效果比对
3. 参考给定 AICR 规则和尝试制定 AICR 规则并进行评审和 AI 评审HTML报告及 AI 修复

持续优化与沉淀

持续优化项目，沉淀更多知识库及 Rules ，提升代码生成质量

进行发版和云上部署 → 06 → 根据需求迭代和效果反馈，进行优化 → 07 → 持续迭代和优化

CodeBuddy Code 90% 代码由 AI 生成

4个人58天79个版本

450 个
需求交付个数

7914 次
Agent 提问次数

68023 行
累计代码行数

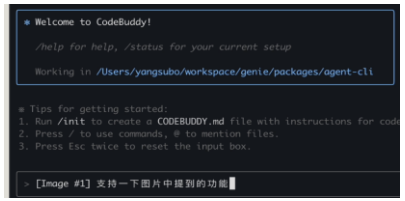
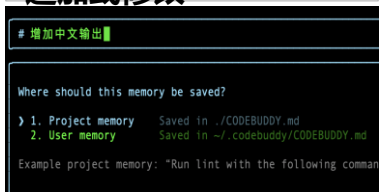
425 行
人均代码产出/日

研发提效方法论

提升工程师业务视野、架构能力、规划能力（人的因素）	提示词/知识库/上下文治理，精准生成（环境因素）	连续 Agent 工作流并行开发，自动生成（流程因素）	高效协作，小步快跑，质量闭环，降低等待（协同因素）
把需求拆解为清晰正交的任务，使用 AI 辅助撰写和拆解	Rules 规范有助于精准生成代码	任务拆解为 worktree　利用 Agent 并行执行	用上下文规范达成沟通协作共识
强制团队用提示词写代码解决缺陷	统一的Prompt模版　变更内容精确到具体文件	Git操作/Changlog日程操作脚本化/自动化	持续采集用户反馈 每日改进每日发布

CodeBuddy Code 技术负责人心得体会：把 AI 当做高效的同事来驱动和管理，能者借力更强，大胆把活交给AI，人定方向，做好风格约束，一次只做一件事。

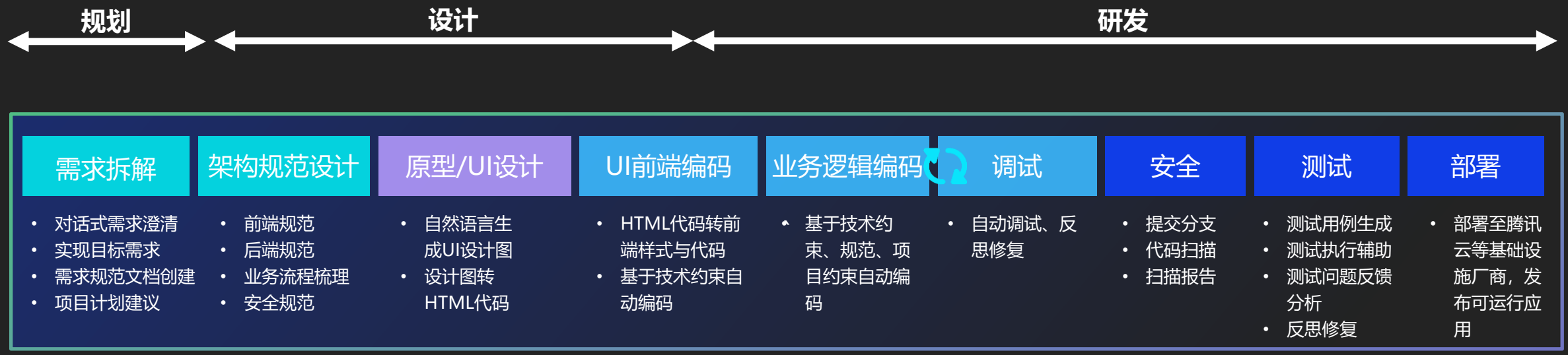
研发提效实战技巧

明确需求，巧用文件和贴图，让AI只改"该改的地方"	结合 Git， AI 驱动的 Pull Request 描述	智能项目配置管理，灵活使用长期记忆	规约编码，及时录入和版本化管理Rules与惯例，减少环境上下文“口述”成本
@ file，截图/拖拽截图，各种方式让AI理解需求。明确哪些可以放心交 CodeBuddy 的任务	**自然语言完成终端操作，在线解决 Git 冲突。一次只做一件事,每次聚焦一个需求或子目标。失败就**	**用好 CodeBuddy.md 分层结构。灵活使用 /init 命令，为你的仓库生成工程的长期记忆的起始点**	**团队统一开发规约及时版本化管理（如提交规范等）并借 /memory 机制，随代码变更，动态追加或修改**
			

05

2026年展望

软件工程边界消融，AI Coding 重构组织协作方式



AI Coding 常见场景

小程序开发

主要场景：

- 生成小程序原型
- 查阅开发文档
- 定位和修复编译问题

产品方案：

- 分析和拆解用户需求，生成小程序代码
- 内置小程序开发知识库，快速解决技术疑点
- 自动分析并修复编译错误

游戏开发

主要场景：

- 降低新手开发游戏的成本
- 降低大型游戏项目理解成本
- 定位游戏性能瓶颈

产品方案：

- 内置的 UE/Unity 知识库，辅助生成业务代码
- 召回相关代码，总结关键逻辑与调用链
- 自动检测性能问题，优化代码

Web网站开发

主要场景：

- 快速生成 Web 网站页面
- 快速生成响应式布局、SEO 优化、API集成
- 减少重复工作，一键生成 CURD 等常见代码

产品方案：

- 结合需求生成前后端匹配的代码。
- 内置场景的 Web 开发模板知识库
- 结合上下文智能补全代码

轻应用开发

主要场景：

- 非技术人员（产品、设计）用自然语言生成应用
- 快速搭建内部工具（如数据看板、简单OA 等）
- 无需部署，一键快速分享

产品方案：

和元宝合作，打造AI编程功能，支持泛开发者自然语言需求生成轻量应用，可运行和分发。创意快速落地验证。

传统业务开发

主要场景：

- 生成符合企业开发与安全规范
- 理解复杂的存量项目，避免逻辑冲突
- 生成的代码适配行业特性（金融、消费电子等）

产品方案：

- 链接企业知识库，参考企业规范生成代码
- 分析存量代码，关联模块上下文
- 结合业务规则和安全规范进行代码评审

极客邦科技 2026 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

4月 · 北京 · 1200人

QCon 全球软件开发大会

会议时间：4月16-18日

- Agentic Engineering
- AgentOps
- 下一代模型架构与推理优化
- AI 原生基础设施
- 知识工程实践
- AI 安全

6月 · 上海 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：6月26-27日

- AI Infra 系统工程
- 多 Agent 协作与实践
- 多模态融合
- 模型训练与推理创新
- 数据平台与特征服务

8月 · 深圳 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：8月21-22日

- Agentic AI
- 轻量化与高效推理
- 多模态应用
- AI + IoT 场景实践
- AI 工业化落地

10月 · 上海 · 1200人

QCon 全球软件开发大会

会议时间：10月22-24日

- AI Agent
- Vibe Coding
- 智能可观测
- 推理基建
- 模型攻防
- AI x 创造力

12月 · 北京 · 1000人

AiCon 全球人工智能开发与应用大会

会议时间：12月18-19日

- 大模型架构创新
- 多模态 AI 产业融合
- 具身智能
- AI for Science
- 大模型安全

InfoQ 极客传媒

InfoQ 极客传媒

THANKS

探索 AI 应用边界

Explore the limits of AI applications

AiCon

全球人工智能开发与应用大会