多模态终态契约：复杂开发中的目标对齐与能力边界

王敬一

2026年7月（修订版）

摘要

以 Claude Code、Cursor 为代表的 coding agent，已能用自然语言驱动相当复杂的软件生产。但复杂开发中的根本问题并未消失：**自然语言是模糊的，用户要的终态与 Agent 理解的终态之间始终存在 gap**。现有验证机制——事后看结果、逐步确认、Plan-Then-Execute 的文本步骤确认、代码断言、执行后 diff——要么反馈周期过长，要么确认对象不是「做成什么样」。

本文主张分两层理解「契约先行」：

1. **契约先行（原则层）**：对复杂开发至关重要——先对齐终态，再执行；任务越复杂、越昂贵、越不可逆，终态越必须先行。
2. **多模态终态契约（核心贡献）**：对齐物不能只是文本步骤。多模态预览（线框、架构图、对话轨迹、演示视频、行为表等）的意义，不只是让非开发者看得懂，更是让**高端开发者在复杂任务中也能直观看见终态**，降低脑补成本，使开发更便利，从而**抬高可稳定推进的开发边界**。

进一步，本文补全 Agent 侧闭环：Agent 不仅执行契约，还须**参与设计可交付终态**，并按自身能力边界否掉做不到的部分——禁止「先画出自己够不着的终态，做着做着才发现对不齐」。

关键词：契约先行、多模态终态、Plan-Then-Execute、能力边界、复杂开发、人机对齐

1. 引言

1.1 问题：复杂开发里，你说的终态，Agent 真的对齐了吗?

Coding agent

把「用自然语言做出东西」变成日常。但复杂任务中，失败往往不是语法错误，而是**终态错位**：

- 你说「高精度机械臂」，Agent 开始写控制与仿真，却与你要的精度、结构、可制造性不是同一终态；
- 你说「做一个游戏」，Agent 堆系统与资源，却与你要的手感、关卡节奏、可玩演示不是同一终态；
- 你说「简洁产品页」，两边对「简洁」的视觉终态完全不同。

当前主流工具的确认对象，多是「怎么做」（步骤）或「改了什么」（diff），很少在执行前让人**看见做成什么样**。用户不是在使用 Agent，而像在赌博：赌这次理解对了没有。

这不是单纯的模型能力问题，而是**目标对齐机制**问题。

1.2 现有机制的局限

机制	确认对象	局限
事后验证	完整产物	反馈周期 = 全量执行成本
逐步确认	单步动作	看不到终态形态
Plan-Then-Execute	文本步骤	懂行者可脑补，复杂终态仍难还原
代码断言（Karpathy 式）	可测属性	测不出「像不像我想要的」
Inline diff	代码变更	执行后才见，且偏局部

共同缺口：缺少以**终态**为对象、在执行前可被直观判断的对齐物。

1.3 本文主张（修订后的钉子）

本文不是发明「先做计划」。先对齐再执行，是复杂协作的常识。本文的钉子是：

> **对复杂开发，契约先行很重要；契约必须是多模态终态预览，才能让终态被直观看见——这对大众与高端开发者都成立。Agent 还须按能力边界裁剪终态，保证契约可交付。**

一句话：

> **Plan 要升级：从「步骤清单」变成「可看见、可确认、且自身做得到的终态契约」。**

2. 相关工作定位

2.1 Plan-Then-Execute

He et al. (2025) 等研究表明：用户信任的关键节点常在计划生成后、执行前。硬门控（人未确认不执行）是重要工程

约束。但其「计划」多为文本步骤——确认的是 how，不是 what-it-looks-like / what-it-can-do。

2.2 断言式契约

将计划升级为可测试断言，机器可自动验收功能正确性。局限：覆盖「可测属性」，覆盖不了视觉、手感、风格、能力演示等终态直觉。

2.3 Magentic-UI 等可编辑计划

优化了计划的交互性（人可改步骤），但契约模态仍以文本任务分解为主。

2.4 本文差异

维度	文本 PTE	断言契约	可编辑文本计划	本文
契约形式	步骤	测试断言	步骤	**多模态终态预览**
核心问题	怎么做	功能对不对	怎么做（更好改）	**做成什么样 / 能做什么**
主要收益对象	开发者	可测任务	协作编辑	**复杂开发中的人（含高手）**
执行约束	正向	正向	正向	终态锚定 + 能力边界裁剪

3. 理论框架：三层闭环

3.1 第一层——契约先行（原则）

定义：在 Agent 执行复杂任务前，先形成并确认终态契约；未确认不执行。

为何对复杂开发重要：步骤越多、成本越高、不可逆越强（返工、烧钱、物理世界），方向错误的代价越大。机械臂制造、大型游戏、复杂系统重构，都不是「先干起来再说」的场景。

与「先做 Plan」的关系：等价于先做 Plan，但 Plan 的对象必须是终态，而不是仅供模型执行的步骤备忘。

3.2 第二层——多模态（核心）

定义：终态契约按任务选择最直观的模态，使人（含资深开发者）能直接判断方向。

多模态的意义**不是**「只为了普适大众」，而是：

1. **直观性**：复杂终态难以靠文本步骤在脑中还原；预览把「做出来是什么样 / 能做什么」摊开。

2. **便利性**：对齐成本下降 → 开发更便利。

3.

抬高开发边界：对齐与返工成本下降后，个人与团队能稳定推进的任务复杂度上升——不是让事变琐碎，而是让「做复杂的东西」变得可行。

模态示例：

任务	更直观的终态契约
UI / 视觉	线框、低保真预览
游戏	可玩切片、关键帧、短演示
对话 Agent	示例对话轨迹
代码架构	API 签名 + 架构图
配置/规则	条件→结果行为表
数据分析	维度表 + 结论方向
复杂硬件/机器人	图纸/指标 + **短演示视频（能力与行为）**

短演示视频在「它能做什么」这一层，往往是最直观的多模态形式之一；硬指标（精度、负载、公差）仍可作为并行硬契约。

3.3 第三层——能力边界裁剪（必要约束）

定义：Agent

参与设计终态，并必须否掉自身能力边界之外的交付；禁止设计一个自己做不到的终态。

完整流程：

1. 理解用户想要的终态（多模态预览）；

2. 对照自身工具与权限，裁剪为**可交付终态**；

3. 将可交付契约交还用户确认；

4. 确认后再执行，过程持续与契约对齐；

5. **禁止**：做着做着才发现契约超出能力。

约束可写成：最终契约必须落在 Agent 能力边界之内（契约是能力集合的子集）。

例：用户要「高精度机械臂实物」。Agent 应理解该终态，但须声明：当前 coding agent 可交付结构方案、仿真演示、控制软件、图纸草稿等，**不能保证**真实加工、装配、标定与出厂精度。契约应降级为可达成范围，否则不应开工。

例：用户要「做一个游戏」。Agent

先设计演示终态（短视频/可玩片段），但该演示必须是自己开发过程能对齐、能交付的；不能先画超出引擎、资源与工期能力的饼。

4. 形式化定义（修订）

多模态终态契约（Multimodal End-State Contract）：

在 Agent 执行前，以最小必要成本生成人可直观判断的终态预览（模态随任务变化）；Agent 按能力边界裁剪为可交付契约；人确认后，执行过程以该终态为对齐锚点。

三要素：

1. **终态对象**：确认 what / what-it-can-do，而非仅 how；
2. **多模态呈现**：选择最直观模态，服务复杂任务中的终态可见性；
3. **可交付约束**：契约必须落在 Agent 能力边界之内。

5. 工作流

```
用户意图
  ↓
Agent 生成多模态终态预览（并做能力自检 / 范围降级）
  ↓
用户确认或修改终态
  ↓
（不满意则回到预览；越界则必须改契约而非硬上）
  ↓
按已确认终态对齐执行（可配合逆向拆解 / 检查点）
  ↓
交付，并对照终态验收
```

关键变化：用户从「指导每一步怎么做」转向「确认可交付终态」；Agent 从「盲目执行者」转为「契约共作者 + 边界守门人 + 对齐执行者」。

6. 场景论证

6.1 软件：UI 与游戏

UI 与游戏美术/手感，正确性首先在「长什么样 / 什么感觉」。文本步骤无法替代预览。对资深开发者同样如此：复杂交互与内容生产中，终态可见能显著降低协作与返工成本。

6.2 极端：用代码操作机械臂制造新机械臂

终态必须先行。短演示视频回答「新臂能做什么」；图纸与指标回答「如何可制造、可验收」。没有

终态契约就生成控制代码，是在物理世界高成本试错。

同时：coding agent 必须裁剪交付边界——终态清晰也不等于接得下实物制造全链路。

6.3 一般复杂开发

后端、重构、数据等也可以契约先行；模态变为架构图、行为表、验收标准。原则相同：**先钉可交付终态，再执行。**

7. 对现有 Coding 工具的含义

追问：「终态说清了，你做一个高精度机械臂，能保证不跑偏吗？」

诚实答案：

- **不能保证**交付合格实物；
- 即使终态清晰，工具仍须承认能力边界；
- 好的 harness 应拒绝或降级不可交付契约，而不是用更长代码生成假装能做。

因此，下一代 coding / agent 工具缺的不只是更强模型，而是：

1. 多模态终态契约层；
2. 能力边界声明与契约裁剪；
3. 以终态为锚的执行与验收。

8. 与 One-Code 的关系（实现状态）

作者在 One-Code 中实现了契约类型检测、预览生成、确认门控与逆向拆解等最小路径（`agent/contract_first.py`）。现有自动对照实验主要验证工程可运行性，**尚不足以验证**「多模态终态对复杂开发与不同用户的直观收益」——该验证需要面向复杂任务与人因判断的专门设计，留待后续。

9. 局限与未来工作

1. 多模态预览质量随模型进步提升，但契约层与边界裁剪仍属 harness，不会被模型进步自动消灭。
2. 「用一用才知道」的体验问题，终态预览不能替代全部实机验证；预览解决的是对齐与方向，不是所

有验收。

3. 需要建立：预览—交付一致性度量、能力边界自动评估、以及在UI/游戏/机器人等垂直场景的专门评测。

4. 动态终态（长周期项目）需要契约版本化与阶段性再确认。

10. 结论

复杂开发需要契约先行；契约的关键升级是多模态终态——让「做成什么样 / 能做什么」在执行前被直观看见，从而降低对齐成本、便利开发、抬高可推进的复杂边界。Agent必须参与设计终态，并否掉自身能力之外的承诺，保证契约可交付、过程可对齐。

一句话：先钉死一个自己打得中、对方看得见的终态，再指哪打哪。

参考文献

[1] Anthropic. Claude Code Documentation — Plan Mode.

[2] Cursor. Agent Mode Overview.

[3] He, G. et al. (2025). Plan-Then-Execute. arXiv:2502.01390.

[4] Microsoft Research. (2025). Magentic-UI. arXiv:2507.22358.

[5] He, Z. et al. (2023). WireGen. arXiv:2312.07755.

[6] 相关工业工具文档：Windsurf Cascade、GitHub Copilot Agent、MiMo Code Plan Mode等。

本文为修订后的 proposal 级技术报告，定位为求职研究 / 技术写作证明材料。核心修订：将主张重心从「契约先行 vs 直接执行」明确为「复杂开发需要契约先行 + 多模态终态直观对齐 + 能力边界可交付约束」。