

Harness Engineering 全景解读与程序员心理认知准备

信息截止日期：2026 年 3 月 | 综合来源：OpenAI、Anthropic、Martin Fowler、Thoughtworks、SmartScope、Medium 等

目录

- [1. 什么是 Harness Engineering](#)
- [2. 术语的诞生与关键人物](#)
- [3. 三层概念关系：Prompt → Context → Harness](#)
- [4. OpenAI 百万行代码实验](#)
- [5. Anthropic 的长周期 Agent Harness 方案](#)
- [6. Martin Fowler / Thoughtworks 的分析与展望](#)
- [7. Harness 对性能的量化影响](#)
- [8. 工程师角色的根本性转变](#)
- [9. 程序员心理冲击：身份认同危机](#)
- [10. 认知层面的适应准备](#)
- [11. 心理层面的适应准备](#)
- [12. 适应曲线与阶段性指南](#)
- [13. 实用行动清单](#)
- [14. 参考来源](#)

1. 什么是 Harness Engineering

Harness Engineering（驾驭工程） 是继 Prompt Engineering（提示词工程）和 Context Engineering（上下文工程）之后，于 2026 年 2 月迅速兴起的第三个核心 AI 工程概念。

它的核心思想是：**系统性地设计约束、反馈循环和持续改进机制**，使 AI Agent 能够在大规模、长周期的软件开发中可靠运行。

用一个经典比喻来理解：

模型是引擎，Harness 是整辆车。 再强大的引擎，没有方向盘和刹车，哪儿也去不了。模型负责生成响应，Harness 负责其他所有事情——人类审批、子 Agent 协调、文件系统访问、生命周期钩子、规划与执行。

—— Phil Schmid

或者用更古典的比喻：

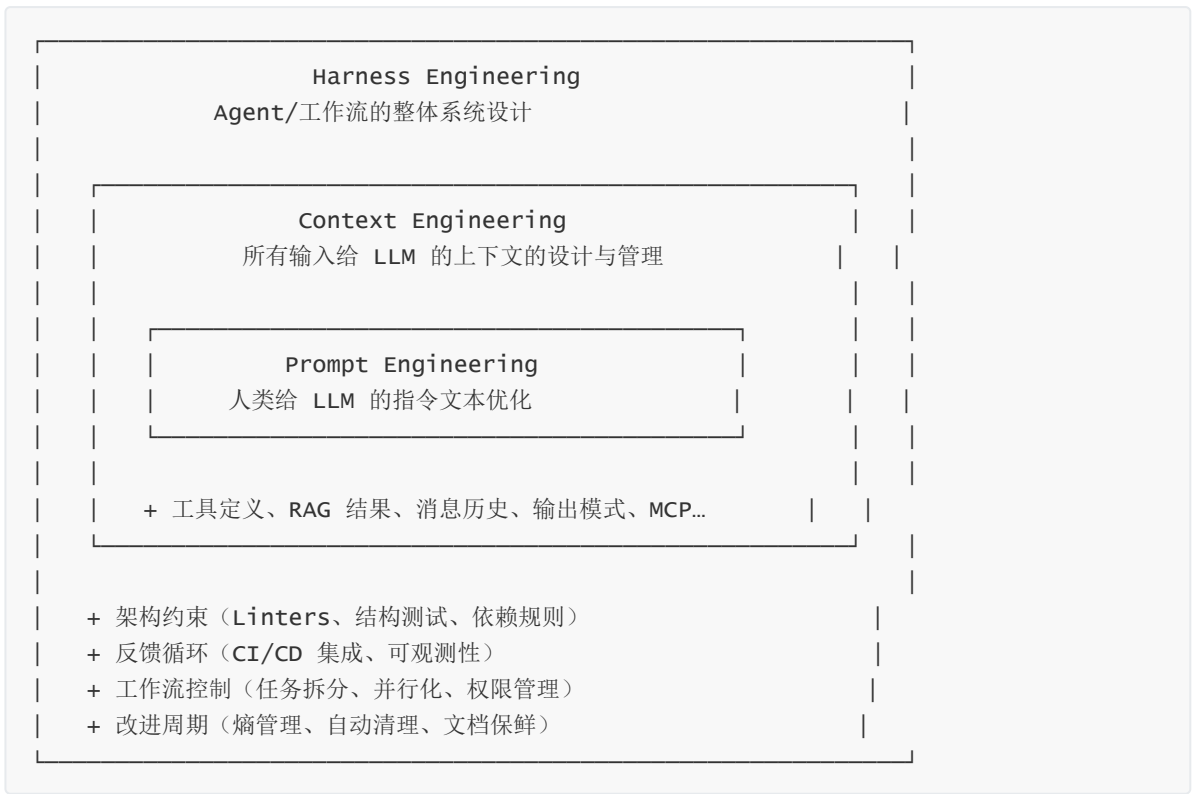
"Harness" 的本义是马具——缰绳、马鞍、嚼子——整套将马匹力量导向正确方向、防止失控、保障长距离稳定运行的装备。 LLM Agent 就是那匹强大但不可预测的马，Harness 就是让它在正确道路上奔跑的完整装备。

2. 术语的诞生与关键人物

Harness Engineering 的概念碎片化地存在于 2025 年末，但在 2026 年 2 月的短短几周内完成了"结晶"：

时间	事件	人物/组织
2026.02.05	发表博文 "My AI Adoption Journey", 在第五步明确提出 "Engineer the Harness"	Mitchell Hashimoto (HashiCorp 联合创始人、Terraform 创造者)
2026.02.11	发表 "Harness engineering: leveraging Codex in an agent-first world", 详述百万行零手写代码实验	OpenAI (Ryan Lopopolo)
2026.02.17	发表分析文章, 将 OpenAI 的 Harness 归纳为三大领域	Martin Fowler / Birgitta Böckeler (Thoughtworks)
2026.02	围绕 "Models, Apps, and Harnesses" 重组 AI 指南框架	Ethan Mollick
2025.11.26	发表 "Effective harnesses for long-running agents" (概念先驱)	Anthropic

3. 三层概念关系：Prompt → Context → Harness



简明判断准则：

维度	Context Engineering	Harness Engineering
优化目标	单次推理的输入质量	整个系统的持续质量
问题句式	"给模型看什么"	"防止什么、测量什么、控制什么、修复什么"
失败模式	单次输出不准确	质量随时间退化
典型实现	RAG、提示词设计、Memory	Linters、CI 集成、任务拆分、自动清理

4. OpenAI 百万行代码实验

2026 年 2 月 11 日, OpenAI 发布了迄今最详尽的 Harness Engineering 实战报告。

4.1 实验概况

- **起点**: 2025 年 8 月, 从一个空的 Git 仓库开始
- **约束: 零行手写代码** —— 所有代码 (应用逻辑、测试、CI 配置、文档、内部工具) 均由 Codex 生成
- **团队**: 3 名工程师 (后扩展至 7 名)
- **成果**: 5 个月, 约 100 万行代码, 约 1,500 个 PR 被合并
- **吞吐量**: 平均每个工程师每天 3.5 个 PR
- **效率估算**: 约为手工编码的 10 倍速度
- **产品状态**: 拥有内部每日活跃用户和外部 Alpha 测试者

4.2 五大 Harness 原则

原则一: 设计环境, 而非编写代码

"当 Agent 遇到困难时, 修复方法几乎从来不是'更努力'。人类工程师总是退一步问: '缺少什么能力? 如何让它对 Agent 既可理解又可执行?'"

原则二: 机械化执行架构约束

定义了每个业务领域的依赖方向: `Types → Config → Repo → Service → Runtime → UI`。使用自定义 Linters 和结构测试自动检测违规。写在文档里不够——如果不能机械化执行, Agent 就会偏离。

原则三: 仓库即唯一真相来源

Slack 讨论和 Google Docs 中的知识对 Agent 来说等于不存在。所有团队知识以版本控制的产物形式存放在仓库内。单一的巨型 `AGENTS.md` 方案失败了, 转而使用结构化的渐进披露模式:

```
AGENTS.md          ← 约100行的"目录", 指向更深层文档
ARCHITECTURE.md
docs/
├─ design-docs/
├─ exec-plans/
├─ product-specs/
├─ references/
├─ DESIGN.md
├─ FRONTEND.md
├─ RELIABILITY.md
└─ SECURITY.md
```

原则四: 将可观测性接入 Agent

- 将 Chrome DevTools Protocol 集成到 Agent 运行时, 使 Codex 能捕获 DOM 快照和截图
- 暴露日志、指标、链路追踪, 通过 LogQL/PromQL/TraceQL 查询
- 使"确保服务启动在 800ms 内完成"这样的指令成为可验证的目标
- 单次 Codex 运行可持续工作长达 6 小时以上

原则五: 对抗熵增

最初, 团队每周五花 20% 的时间手动清理"AI 垃圾" (slop), 这种方式无法规模化。解决方案:

- 将"黄金原则"编码到仓库中
- 建立定期运行的后台 Codex 清理任务

- 自动扫描偏差、更新质量评分、提交定向重构 PR
- 实现类似垃圾回收（GC）的持续技术债清理

4.3 核心理念

"Humans steer. Agents execute."（人类掌舵，Agent 执行。）

5. Anthropic 的长周期 Agent Harness 方案

Anthropic 在 2025 年 11 月发表了 *"Effective harnesses for long-running agents"*，从人类工程师的工作方式中汲取灵感。

核心问题

AI Agent 必须在离散会话中工作，每个新会话从零开始，没有前一次的记忆。就像一个软件项目由轮班的工程师负责，每个新工程师到岗时对上一班发生了什么毫无记忆。

两阶段解决方案

阶段	角色	关键行为
初始化 Agent	首次运行时搭建环境	创建 <code>init.sh</code> 、 <code>claude-progress.txt</code> 、功能清单 (JSON)、初始 Git 提交
编码 Agent	每个后续会话	读取进度文件和 Git 日志 → 选择一个功能 → 增量推进 → 提交并更新进度

四大失败模式及对策

问题	解决方案
Agent 过早宣布项目完成	设立结构化功能清单 (JSON)，逐项检查
留下未文档化的 Bug	会话开始时运行基础测试，结束时写 Git 提交和进度更新
过早标记功能完成	要求 Agent 使用浏览器自动化工具进行端到端验证
花时间搞清如何运行应用	初始化 Agent 编写 <code>init.sh</code> 脚本

6. Martin Fowler / Thoughtworks 的分析与展望

Birgitta Böckeler（Thoughtworks 杰出工程师）在 Martin Fowler 站点发文，提出了几个重要观察：

三大 Harness 组件

1. **上下文工程**：持续强化的代码库知识库 + Agent 对动态上下文（可观测性数据、浏览器导航）的访问
2. **架构约束**：不仅靠 LLM Agent 监督，还有确定性的自定义 Linters 和结构测试
3. **"垃圾回收"**：周期性运行的 Agent，查找文档不一致或架构违规，对抗熵增和衰退

关键展望

- **Harness 会成为新一代服务模板吗？** 团队是否会从预制 Harness 起步，然后根据应用特性逐步定制？
- **运行时必须被约束才能获得更多 AI 自主性：**提升信任和可靠性需要约束解决方案空间——特定的架构模式、强制边界、标准化结构
- **技术栈趋向收敛：**AI 可能推动我们走向更少的技术栈选择，优先考虑"AI 友好性"
- **两个世界：**AI 前时代的应用 vs. AI 后时代原生 Harness 应用，改造旧代码库的成本可能不值得

7. Harness 对性能的量化影响

仅改变 Harness（不改变模型）能产生多大差异？两个实验给出了惊人答案：

Can.ac 实验

- 仅改变 Harness 的工具格式（编辑方法），在 16 个测试模型上普遍提升了编码基准分数
- **Grok Code Fast 1：**从 6.7% 跳升至 **68.3%**（10 倍提升）
- 输出 token 减少约 20%（个别模型高达 61%）
- 未修改任何模型权重

LangChain Terminal Bench 2.0

- 仅靠 Harness 改进，排名从第 30 名跃升至第 5 名
- 同一模型，提升 13.7 个百分点

结论：在纠结模型选择之前，先审视 Harness 设计，投资回报率更高。

8. 工程师角色的根本性转变

Harness Engineering 意味着软件工程师的核心工作发生了范式转移：

旧世界 vs. 新世界

维度	传统模式	Harness Engineering 模式
核心活动	编写代码（90%）+ 架构思考（10%）	架构思考（20%）+ 代码审查（30%）+ 与 Agent 对话（50%）
价值来源	代码行数、功能实现	环境设计、意图规约、反馈循环构建
工作节奏	写几行 → 编译 → 看结果（快速反馈）	设计规范 → 调教 Agent → 1-3 个月后看到大规模提效（延迟反馈）
技能重心	语言/框架熟练度、算法、Debug	系统思维、抽象能力、文档工程、影响力工程
类比	工匠 / 乐器演奏者	导演 / 乐队指挥
衡量标准	代码质量、Bug 修复速度	Agent 产出的可靠性、团队乘数效应

OpenAI 的总结

"我们最困难的挑战现在集中在设计环境、反馈循环和控制系统上。"

9. 程序员心理冲击：身份认同危机

9.1 "写代码"作为身份基石的坍塌

Lex Fridman 在一次访谈中的坦白精准地捕捉了这种震动：

"我从没想到，我最热爱的事情会是被替代的那件事。我花了数千小时钻研代码，倾注了心血。一些最痛苦和最幸福的时刻，都是独自面对屏幕。我曾是一个 Emacs 用户——天哪，Emacs。那里有一种身份认同，一种意义。当我行走在世界上，我不会说出来，但我内心认为自己是一个程序员。而这一切可能在几个月内改变....."

9.2 心理学视角：为什么冲击如此强烈

心理学研究将这种现象称为 **身份构成性劳动 (Identity-Constitutive Labor)** 的丧失——当核心工作活动变得过时，个体经历的不仅仅是工作不安全感，而是 **存在性断裂 (Existential Disruption)**。

四大心理机制：

1. 身份认同断裂 (最普遍)

- 过去十年，程序员的核心自我叙事：我写代码 → 代码运行 → 我有成就感
- 转向 Harness 后，代码量锐减 90%+，大脑产生"自己在偷懒/退步"的自我攻击
- 与从 IC (个人贡献者) 转 Tech Lead 时的失落感高度相似

2. 控制感与反馈延迟

- 写业务代码：改几行 → 立即看到效果 (快速多巴胺反馈)
- 做 Harness：设计规范、调教 Agent → 可能 1-3 个月才看到效果 (多巴胺回路断裂)

3. Flow 状态的丧失与重构

- 旧 Flow：逐行编码的深度沉浸，挑战与技能完美匹配
- 新 Flow：在更高抽象层级运作——评估 Agent 的架构决策是否会在三个功能后产生问题
- 过渡期会感到"再也找不到心流了"

4. 社交媒体放大的比较焦虑

- 别人在晒"一天产出 5000 行"、"Vibe Coding 一晚搞定 App"
- 而你在做枯燥的 AGENTS.md、工具链稳定性、权限模型
- 容易产生"落伍"的错觉

9.3 开发者心理健康危机的真实信号

2026 年 2 月，知名开发者 Tom Dale 指出：

"几乎每个软件工程师这周都在经历某种程度的心理健康危机。不仅仅是失业焦虑，还有看着软件从稀缺变为丰富时引发的躁狂发作——强迫性的 Agent 使用行为、对变化时间压缩的解离性敬畏。"

Fortune 杂志的标题更加尖锐："创造 Claude Code 的人说，软件工程师可能在今年年底前灭绝。"

9.4 Marius Anderie 的第一人称体验

一位开发者在 2026 年 2 月的自述中写道：

"一小时内，我产生了一个让我停下来的想法：**这东西写代码比我强。**"

"不是所有方面都更强，不是更聪明。但在实际编写正确、结构良好代码的机械行为上——更快、更一致、更少 Bug。我花了多年时间通过深夜和专注构建的技能，现在一个 Agent 几秒就能完成。"

"我失去了旧的心流状态。那种逐行的深度沉浸不见了。但取而代之的是一种新的心流——它在更高的抽象层级运作。少了工匠感，多了建筑师感。少了演奏乐器，多了指挥乐队。"

10. 认知层面的适应准备

10.1 重新定义"工程师"的认知框架

从"代码生产者"到"系统放大器"

你不再是为一个功能写几百行代码的人，而是通过设计 Harness 让几十甚至几百人/Agent 的产出提升 2-10 倍的人。这本质上是**更高阶的创造**。

关键认知转换：

旧认知	新认知
我的价值 = 我写的代码量	我的价值 = 系统的杠杆效应
好工程师 = 写出优雅代码	好工程师 = 设计出让 Agent 可靠产出的环境
控制 = 亲手编写每一行	控制 = 定义约束和验证机制
创造力 = 解决实现难题	创造力 = 架构决策和意图表达
学习 = 新语言/框架/库	学习 = 系统思维、抽象设计、反馈循环工程

10.2 新的核心技能图谱

- 架构思维**：不再是可选项，而是必修课。当实现成本趋近于零，架构决策的杠杆效应被无限放大
- 文档工程**：为 Agent 编写可理解、可执行的文档成为第一生产力
- 约束设计**：理解何处需要严格边界、何处允许自由度——类似管理大型平台组织
- 元认知能力**：能够诊断"Agent 为什么卡住了"并将解决方案系统化
- 产品感知**：当实现不再是瓶颈，"做什么"比"怎么做"更重要
- 风险校准**：你的错误随生产力等比例放大——午夜的一条模糊指令可能导致 Agent 重构半个代码库

10.3 "10 倍错误"问题

你的错误随生产力等比例放大。

手工编码时，深夜的一个糟糕决定最多搞坏一个文件。但使用 Agent 时，每一条查询都可能是一千行代码。一条措辞不当的指令，一次注意力的松懈，Agent 会高兴地删除 50 个文件并重构 45,000 行代码。

这意味着：你现在需要更多自律，而不是更少。 知道何时停下来、何时太累以至于无法正确审查、何时应该离开——这是以前不需要的新技能。

11. 心理层面的适应准备

11.1 接纳"丧失匠人感"的哀悼过程

承认一个事实：失去逐行编码的亲密感是一种**真实的丧失**，值得被哀悼。心理学中的变革适应模型（如 Kübler-Ross 变化曲线）表明，经历否认、愤怒、讨价还价、沮丧，最终到达接纳，是健康的心理过程。

不要试图跳过哀悼期。同时，也不要哀悼中停滞。

11.2 建立新的成就感来源

旧成就感触发器	新成就感触发器
"我写了一段优雅的算法"	"我设计的 Harness 让 Agent 的可靠性提升了 3 倍"
"我修了一个困难的 Bug"	"我建立的约束体系让这类 Bug 不可能再出现"
"我学会了新框架"	"我的文档工程让新 Agent 在 10 分钟内就能在项目中高效工作"
"我的 PR 被 merge 了"	"我设计的反馈循环让团队周部署从 3 天缩短到 30 分钟"

11.3 管理不确定性焦虑

Marius Anderie 的坦白值得所有人倾听：

"建造者的那个我热爱与 Agent 协作。速度、范围、快速构建的兴奋——这是我的工作有史以来最好的体验。但与此同时，对这意味着什么（对我的生计）的不确定性，是一种永远不会完全消失的背景噪音。"

应对策略：

- **聚焦可控领域**：你无法控制行业走向，但可以控制自己的学习和定位
- **成为"理解这场转变的人"**：深入理解 AI Agent 不仅仅是能力层面，还包括如何与它们有效工作、如何避免陷阱、如何在实现免费的世界中思考架构
- **接受不确定性本身**：未来是真正不可预测的——不是哲学意义上的"没人知道未来"，而是具体的、实际意义上的

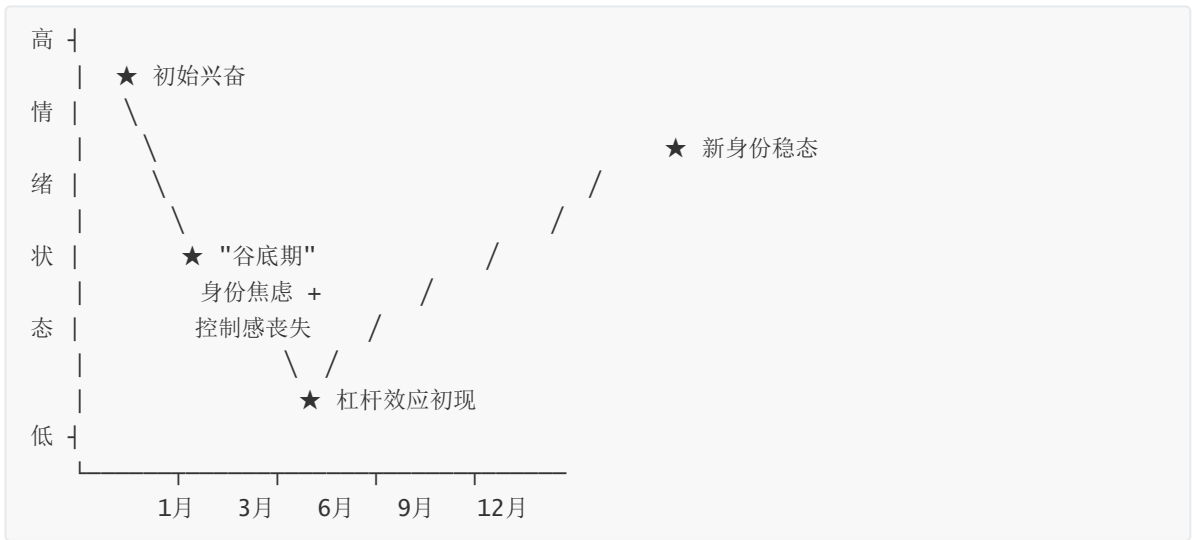
11.4 把"管 Agent"当成"带团队"

这个类比一旦建立，心理落差会大幅缩小：

- 给 Agent 设定角色 = 写团队职位描述
- 写 AGENTS.md = 写团队 Onboarding 文档
- 设计架构约束 = 制定代码规范
- Review Agent PR = 做 Code Review
- 设计反馈循环 = 建立 1-on-1 和回顾会
- 处理 Agent 产生的"垃圾" = 辅导初级开发者改进代码

12. 适应曲线与阶段性指南

典型适应曲线



各阶段指南

第一阶段：0-3 个月（最困难期）

- 常见想法："我是不是不适合了"、"要不回业务线写代码吧"
- **建议：**这是正常反应，不要因此做重大职业决定
- **行动：**找 1-2 个已完成类似转变的同行，每月聊一次真实感受
- **保护措施：**每周保留 1-2 天做个人项目的手写代码，维持"匠人"触感

第二阶段：3-9 个月（杠杆效应初现）

- 开始看到自己设计的系统在发挥乘数效应
- 例如：一个 CLI 工具让团队周部署从 3 天缩短到 30 分钟
- 认同感逐步回升，但可能仍有波动
- **行动：**用数据和故事把成果讲给业务方听——看到别人因为你省了时间/少踩坑，认同感最直接

第三阶段：9 个月后（新身份稳定期）

- 大部分人进入新身份稳定期
- 焦虑转化为"怎么让系统再上一层楼"的进取心
- 回头看会觉得"这才是高阶的工程乐趣"

13. 实用行动清单

技能建设

- ☐ 学习架构文档编写（参考 OpenAI 的 `AGENTS.md` + `docs/` 结构）
- ☐ 实践结构测试框架（如 ArchUnit）
- ☐ 学习自定义 Linter 编写
- ☐ 掌握可观测性工具栈（日志、指标、链路追踪）
- ☐ 实验 Coding Agent（Claude Code、Codex 等），体验"管 Agent"的工作方式
- ☐ 了解 CI/CD 深度集成与自动化质量门禁

认知调适

- ☐ 刻意练习"在更高抽象层级思考"——从行级 → 模块级 → 系统级
- ☐ 建立"约束即杠杆"的心智模型：严格边界不是限制创造力，而是放大 Agent 产出
- ☐ 练习"诊断式思维"：当 Agent 出错时，问"系统缺什么"而非"Agent 太笨"
- ☐ 接受"无聊技术"可能是最优选择——稳定、可组合、在训练集中表示充分的技术更适合 Agent

心理健康

- ☐ 每周保留手写代码的"仪式时间"，防止完全"空心化"
- ☐ 加入或组建"转型互助小组"，定期分享真实感受
- ☐ 学习区分"不舒服但在成长"和"已经在伤害心理健康"
- ☐ 建立工作外的身份支点——运动、创作、社区、家庭
- ☐ 如果焦虑严重影响睡眠/情绪，**尽早寻求专业心理咨询**（科技行业从业者抑郁/焦虑发生率高于平均水平约 1.5-2 倍）

团队层面

- ☐ 在团队中主动建立"心理安全"氛围——试错不被惩罚
- ☐ 公开讨论"与 Agent 协作的不适感"——你不是唯一有这种感受的人
- ☐ 重新定义团队的绩效指标——从代码行数转向系统杠杆效应

14. 参考来源

1. [OpenAI - Harness engineering: leveraging Codex in an agent-first world](#) (2026.02.11)
2. [Anthropic - Effective harnesses for long-running agents](#) (2025.11.26)
3. [Martin Fowler / Birgitta Böckeler - Harness Engineering](#) (2026.02.17)
4. [Mitchell Hashimoto - My AI Adoption Journey](#) (2026.02.05)
5. [SmartScope - What Is Harness Engineering: Defining the "Outside" of Context Engineering](#)
6. [Pawan Patra - A Developer's Guide to Harness Engineering](#) (2026.02.22)
7. [InfoQ - OpenAI Introduces Harness Engineering](#)
8. [Marius Anderie - The Psychology of Coding With AI Agents](#) (2026.02.20)
9. [Zotpaper - Software Engineers Report Widespread Mental Health Crisis as AI Transforms Industry](#) (2026.02.07)
10. [Fortune - Software engineers could go extinct this year, says Claude Code creator](#)
11. [Stack Overflow Blog - Mind the gap: Closing the AI trust gap for developers](#)
12. [Can.ac - I Improved 15 LLMs at Coding in One Afternoon. Only the Harness Changed.](#)
13. [LangChain - Improving Deep Agents with harness engineering](#)
14. [Phil Schmid - The importance of Agent Harness in 2026](#)
15. [Ethan Mollick - A Guide to Which AI to Use in the Agentic Era](#)
16. [Chad Fowler - Relocating Rigor](#)
17. [Aakash Gupta - 2025 Was Agents. 2026 Is Agent Harnesses.](#)
18. [Rasikuhail - Codex Engineers Raised Zero PRs in 5 Months](#)

一句话总结：从"写代码"到"让代码自己写代码"的转变，失去的是手感，获得的是杠杆和稀缺性。这不仅是一场技术变革，更是一场需要认真对待的认知与心理重构。

