

TGO 鲲鹏會

KANBAN2 AGENT HARNESS

日产万行业务代码 的Harness 之路

两个月，把 Kanban2 高吞吐交付托上跑道

分享人 | 吴穹博士 | 2026



全球 AI 生态与创新峰会

组织研发效能专家

工具会变，但组织难题很稳定：目标如何拆解、架构如何守住、团队如何协作、质量如何不掉线。今天的 Agent Harness，也是这个脉络里的新一层工程底盘。

01

大型组织转型现场

平安、招行、建信、华为、顺丰、蔚来等大型组织。

02

产品技术管理

工程体系、研发平台、团队协同，都放在一张图里看。

03

AI Coding 落地观察

个人变快以后，组织系统也必须一起升级。

04

这次复盘视角

用 Kanban2 实战，讲一条可执行的 Harness 建设路。

当 AI 让个人更快，真正要补的是组织的交付底盘。

TRANSFORMATION PROFILE



吴穹 Adam
北大软件工程博士 | Agilean 首席顾问

大型组织转型

产品技术管理

研发平台

AI Coding

Agent Harness

敏捷 / 精益

OPENING PROBLEM

个人提效了， 组织交付却没跟上

很多团队已经看到 AI 让单个工程师写得更快：代码产出上去了，commit 变密了。但如果计划、评审、测试、运行时和部署没有同步升级，组织效能只会小幅改善，质量压力却会明显放大。

- | | | |
|----|---------|------------------------------|
| 01 | 个人产出上去了 | 模型让写代码、改代码、补测试的局部速度明显变快 |
| 02 | 组织系统没同步 | 需求拆解、评审、验收、环境和发布仍按旧节奏运行 |
| 03 | 质量压力被放大 | 更多 diff、更密集变更、更难追溯，返工和线上风险变高 |

THE TRAP

个人速度 不等于 组织吞吐

没有 Harness，提效会先变成更多代码、更长等待和更多返工。

COMMON ANTI-PATTERNS

不要在底盘不稳时， 急着追 Loop 和 Graph

Loop Engineering、Graph Engineering 是先进厂商实时 Agent 的下一步；但对大多数企业，第一步不是把智能体做得更会转圈，而是先把 Harness 做深，让反馈、验收、环境和权限稳定下来。

- | | | |
|----|--------------------------|------------------------------|
| 01 | Harness 只停在脚本层 | 没有路径感知、真实运行时、失败证物和强制闸门 |
| 02 | Agent feedback 太薄 | 模型不知道错在哪里，只能靠人肉截图和口头补充 |
| 03 | 人还没稳定提效，就上自治 | 人都跑不顺的流程，交给 Agent 只会更快地产生返工 |
| 04 | 用复杂编排掩盖地基问题 | Loop/Graph 放大吞吐，也会放大不确定性和质量债 |

WRONG ORDER

先造 自动智能体， 再补 Harness

顺序反了，Agent 越能跑，组织越难知道它为什么错、错在哪里、该不该合。

正确顺序：先把人和 Agent 都能用的 Harness 做稳，再谈 Loop / Graph / 自治。

目录

- | | | |
|----|------|---|
| 01 | 建设成效 | Commits, churn, and pressure |
| 02 | 建设之路 | Agent OS, fullcheck, CLI E2E, skills, hooks |
| 03 | 使用心得 | When platform work starts compounding |
| 04 | 落地建议 | Reusable guardrails for high throughput |



**BUILD THE
HARNESS ROAD**

Kanban2 是百万行级工程

这不是小型演示项目上的阶段性实验；当前三桶业务口径已达 175.1 万行。

01 175.1 万行
三桶业务口径合计

02 60.1 万行
测试+CLI, 占 34.3%

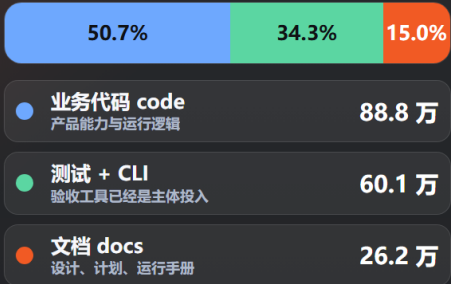
03 18.5 万行 CLI
验收工具已成子系统

CODEBASE SCALE / CURRENT HEAD

Kanban2 当前代码库规模总览

不是在一个小 demo 上跑 Agent，
而是在百万行级工程里维持高吞吐。

三桶业务口径：175.1 万行



主要模块体感



另有 other 约 52 万行 不进业务口径；全仓文本约 227 万行。

先看一个 普通工作日

一天切面：
多条任务线并行，
业务净增约 1.28 万
行。

01

52 次提交

07:38-21:14

02

+12,819 行

业务代码净增

03

7 条任务线

同日推进

日报 - 2026年7月21日 (adwu73)

工作量: 约 8-10 小时 | 52 次提交 (07:38-21:14) | 全部变更 +51436/-3744

业务代码 (排除 docs/md、lock、截图、mcp-packs 打包产物 .cjs、dogfood-skills) : +16108 / -3289 (净增 +12819)

其中产品源码 (不含测试) 约 +11883/-2587, 测试约 +4223/-706

1. 看板观察子卡 / 浮窗 (~2.5h) — 持久化观察子卡列表 → hover/长按展开; 小队双故事浮窗、实体过滤与数据范围; 需求看板产品版本浮窗

主要: kanban-ui 看板组件、浮窗相关配置与测试

2. 阶段前置时间 (Segment Lead Time) (~2h) — Dashboard + Widget、国际化与指标校验夹具

主要: kanban-ui / kanban-services / kanban-cli

3. IM Agent 登录与 Entclaw (~2h) — 严格 Agent 登录身份 (无人 pairing)、懒加载 seed、Gateway IM webhook → entclaw、飞书卡片标题用 agent 名、会话 idle / skill 路由

主要: entclaw、scripts/cnb-dev-remote.sh、gateway 路由

4. OperationSource / createMethod (~1.5h) — 操作来源模型重构、上下文注入、createMethod 枚举

主要: kanban-kernel / kanban-services / kanban-ui

5. 组织与结构 API (~1h) — by-member 角色团队树过滤、强制删组织、DELETE RBAC

主要: kanban-services / kanban-apis / CLI cleanup

6. Agent Admin 精简 & Actions (~1h) — 移除 workflow / source 过滤; 精简 action 定义; skill 目录同步; 指标看板 loading

主要: kanban-ui agent-admin、actions

7. Dogfood / 小队基建 (夹具为主) — dogfood MCP + 86 小队 roster; 另有约 +3 万行打包 .cjs (不计入业务代码)

关键产出: 观察子卡交互与浮窗能力落地; 阶段前置时间看板/组件; IM Agent 登录与 Gateway→entclaw 贯通; OperationSource/createMethod; 组织结构 by-member 与删组织能力。

高强度提交已经成为常态

前一页是一日切片；放回两个月看，高强度提交已经从峰值变成稳定巡航。

01 1,818 次提交
两个月持续推进

02 1,007 核心业务
触达核心路径

03 35.4 次/日
7 月活跃日均

EVIDENCE / COMMIT INTENSITY

一天切片背后，是常态化的高强度提交

单日 52 次提交不是孤立峰值；两个月里提交密度和业务代码量同步抬升。

从一天到两个月

52

7/21 单日提交
07:38-21:14

1,818

两个月总提交
持续滚动推进

1,007

核心业务提交
触达业务路径

35.4/日

7 月活跃日均
提交强度常态化

提交流量抬升后，业务代码进入 12k+/日量级



12k+/日业务代码

高并发成为常态

两个月稳定增长约 100 万行

主线不是一周增量，而是两个月多轮爬升：系统从几十万行推到 175 万行级。

01 约 +100 万行
两个月稳定增长

02 175.3 万行
2026-07-22 HEAD

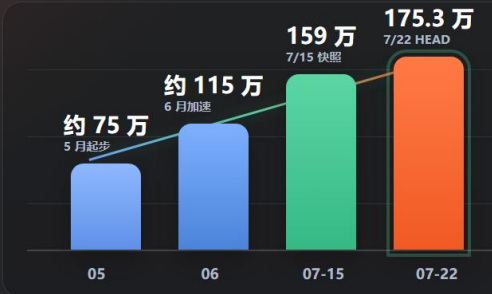
03 34.4% 测试+CLI
护栏同步长厚

TWO-MONTH GROWTH / 2026-05 → 2026-07

两个月稳定增长约 100 万行

这不是某一天的尖峰，而是多轮任务持续推进后，把系统从几十万行推到 175 万行级。

几轮爬升，而不是一次爆发



+100万
两个月大盘增量

多轮
任务线持续推进

稳定
不是单点冲刺

最新 blame 校准



88.8 万
业务代码 code

60.3 万
测试 + CLI

26.2 万
文档 docs

目录

- | | | |
|----|--------------|---|
| 01 | 日产万行不是梦 | Commits, churn, and pressure |
| 02 | Harness 建设之路 | Agent OS, fullcheck, CLI E2E, skills, hooks |
| 03 | 使用心得 | When platform work starts compounding |
| 04 | 落地建议 | Reusable guardrails for high throughput |



**BUILD THE
HARNESS ROAD**

9 步路线图：先修 Harness，再跑 Agent

每一步都把一种人肉流程，沉淀成 Agent 可执行、可观察、可复用的系统能力。



核心顺序：先让 Agent 可验、可看、可起环境、可拿上下文，再谈多智能体自治。

1、构建完整CLI驱动验收

它有完整命令面，但主用户是 Agent、脚本和 fullcheck；命令行只是最薄的一层皮。

01

127 / 143 有命令

命令面真实存在，kanban 只是入口。

02

17.0 万行 CLI

主力沉在 setup / verify / 断言闭环。

03

CLI-only 只有 3.9%

多数是业务与验收同一次进仓。

没有它：验收只能靠手点，脚本散落，Agent 不会复跑。



命令行是皮，Harness 才是肉

2、利用CLI构造稳定测试数据

把稳定上下文、可复现样本和可预热能力放到底座里，Agent 不再临场拼数据。

01

数据可复现

固定账号、样本、组织和业务状态。

02

能力可预热

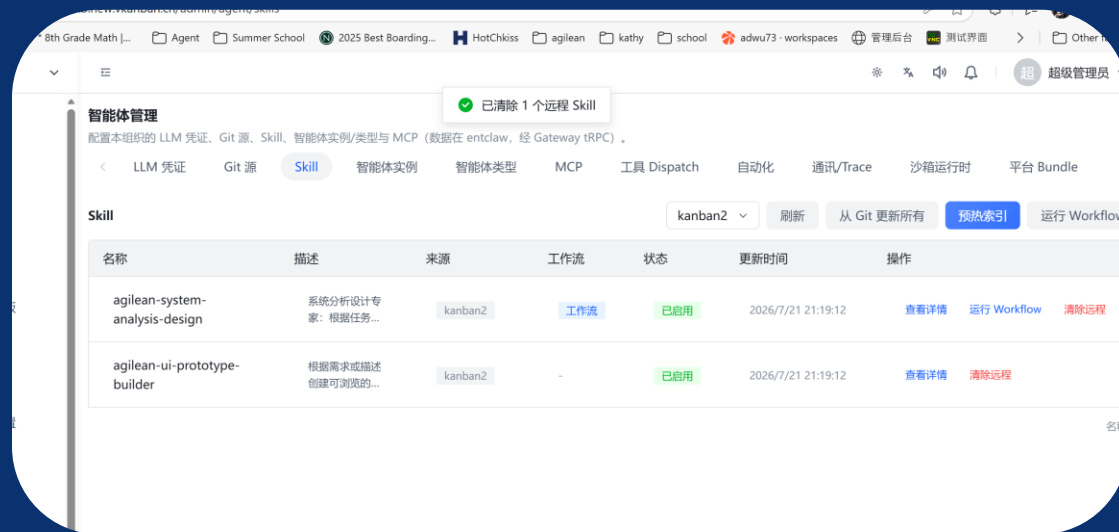
Skill 索引、Workflow 和运行时提前就绪。

03

断言可复用

同一套数据喂给 CLI、fullcheck 和 Agent。

没有它：手工造数、样本漂移，E2E 每次都在猜上下文。



Skill / Workflow / 预热索引，把测试数据接成可调度资产

3、构建分层 fullcheck 闸门

把 CLI E2E 收进 L0-L9 统一漏斗，路径感知决定跑哪几层，最后三关才碰真 Gateway。

01

L0-L2 离线守卫

runtime-sync / invariants / frontend。

02

L3-L6 组件验证

backend unit/IT、zgraph、CLI unit。

03

L7-L9 真运行时

Gateway API、CLI E2E、retry、L9 巡检。

没有它：fullcheck 只是命令集合，质量回退只能事后补锅。

L0-L2 离线守卫

sync / invariants / frontend

L3-L6 组件验证

unit / IT / zgraph / CLI

L7-L9 真运行时

Gateway / CLI E2E / logs

L8 = setup + verify, retry 吃 flake

fullcheck 实例

真实失败报告不是只给红绿灯：它告诉 Agent 哪层过了、哪层失败、失败点在哪里，以及下一步怎么复跑。

01

约 9 分钟跑完

从 L0 到 L8 输出分层摘要。

02

L0-L7 全绿

静态、前端、后端、API 先收口。

03

L8 精确失败

setup 96/138 定位到工时字段。

没有它：失败报告不可读，人要翻日志和数据库定位。

完整 fullcheck 一下

Worked for 8m 57s

./fullcheck.sh 已跑完 (约 8 分钟)，整体失败。当前 diff 为 path scope=none (仅计划文档)，与本次失败无关。

阶段	结果
L0 runtime-sync	通过
L1 invariants	通过 (6s)
L2 frontend	通过 (3m15s)
L3 backend-unit	通过 (无后端路径, 跳过 Maven)
L4 backend-it	通过 (同上; formula-it 有跑)
L5 zgraph	通过
L5e entclaw	通过
L6 cli-unit	通过 (23s)
L7 frontend-api	通过 (1m17s, 84 tests)
L8 cli-e2e	失败 (2m15s)
L8-retry	跳过 (setup 失败, 无 verify id)
L9 runtime-check	未进入 (L8 setup 中断)

L8 失败点: shared setup 步骤 96/138 「写入管理员 demo 工时数据」

[40301/WORKLOG_CREATE_FAILED] 创建工时记录失败: 工时数: 系统字段不可手动写入
admin worklog demo: create failed for 2026-07-16

CardWriteGuard 把「工时数」当成系统字段拒写; 稳定性登记册无对应条目。事后 npm check:services 仍全部 UP。

的话我可以接着排这个工时 seed / 系统字段问题, 并重跑 --only-1789。

02 / COVERAGE BASELINE

fullcheck 跑出来的覆盖率水位

unit + IT + e2e-runtime 合并后，自有后端代码指令覆盖约 75.9%；这说明闸门已经形成基本质量水位。

01

自有代码 75.9%

排除第三方 Starlark 后；行覆盖 75.3%，分支 58.0%

02

核心服务 73–79%

card / meta / view 已经进入可用水位

03

platform 是短板

58.9% 指令覆盖，下一轮优先补测试与场景

没有它：只知道红绿，不知道覆盖短板和下一轮补哪里。

来源：build/jacoco/jacoco.xml，unit + IT + e2e-runtime 合并

BACKEND COVERAGE / JaCoCo

后端覆盖率已经有基本盘，下一步补平台短板

75.9%

自有代码指令覆盖

排除第三方 Starlark 后；行覆盖 75.3%

L7/L8

运行时验收也入账

unit + IT + e2e-runtime 合并口径

58.9%

platform 是短板

体量不小，下一轮优先补强

五大服务指令覆盖率

gateway 92.2%

card-service 79.2%

meta-service 74.8%

view-service 73.3%

platform 58.9%

fullcheck 不是只跑“能不能编译”，它把单测、IT 和真运行时链路合成一张质量水位图。

Kernel / API / Infra

模块	指令	行	分支
domain	81.2%	80.8%	62.6%
event	87.3%	86.2%	64.7%
infra	78.7%	76.8%	65.5%
card-api	82.4%	83.2%	69.0%
schema-api	85.5%	83.6%	75.6%

全量含 vendored Starlark 为 69.0%；看自有代码时约 75.9%，更接近真实工程水位。

3.1 L1 invariants 是内核态权限检查

不等测试失败，先把错架构、文档漂移和危险写法挡在静态阶段。

01

工程卫生

生成文档、产物、大文件、旧端口先检查。

02

后端红线

Serializable、Yield.all、Action 命名等机器拦截。

03

前端红线

enum、非空断言、i18n、types 依赖先挡住。

没有它：能编译的错架构也能进仓，Review 变成人肉门禁。

L1 INVARIANTS

把红线变成 静态内核检查

没有 L1

- × 能编译的错架构，可能一路合进去。
- × 生成文档和真实 API 慢慢漂移。
- × 风格主要靠 Code Review 人眼。

有 L1

- ✓ 测试前先拦住架构和契约问题。
- ✓ 红线机器执行，不靠记忆和自觉。
- ✓ 所有后续闸门跑在更干净的输入上。

scripts/check-agent-invariants.sh

约 618 行，挂在 fullcheck L1；error-code 门禁同步检查契约漂移。

红线墙：20+ 项静态门

工程卫生

- 生成网关文档是否过期
- 大文件 justification
- 构建/本地产物与旧端口回退

前端红线

- 禁止新 enum、非空断言
- 用户可见文案走 i18n
- types/** 不依赖 API client

后端红线

- 禁止新 Serializable / 乱用 Yield.all
- Action 命名空间、fixture 同步
- Mapper 不新增 MySQL 方言

文档入口

- V2 agent/docs 入口存在且被引用
- 计划、评审、状态事实源可追溯
- CNB stop hook 自然衔接

fullcheck 第一关

错架构先失败

3.2: L7 快验契约, L8 留下样板间

fullcheck 真运行时两关不是重复: L7 用隔离换速度, L8 用共享宇宙换覆盖密度。

01

组织模型不同

L7 一场景一 org; L8 一宇宙多 verify。

02

优化目标不同

L7 快、隔离; L8 厚、可调试、可演示。

03

失败资产不同

L7 留小 org; L8 留 session 给 retry / UI 接力。

没有它: 接口契约和产品链路互相漏, Gateway 问题拖到联调才爆。

L7 / L8 RUNTIME GATES

真运行时两关,
验的不是同一件事

L7 回答 “这个接口绿不绿”; L8 留下一个还能打开给人看的世界。

L7 frontend-api

快验契约

是什么

Vitest 打真 Gateway

复用生产 src/api/*.ts client

组织模型

一场景一组织

每个 api-spec 文件专用 org, 文件内共享

优化目标

快、隔离、并行友好

状态不串, 失败面小

失败后

保留小 org 排查

问题通常落在 API + 造数组合

L7

保证每个测试有干净房间。

L8 cli-e2e

留下样板间

是什么

kanban test 全链路

setup → 多条 verify → cleanup

组织模型

一宇宙多断言

一次 setup, 200+ verify 共用 session

优化目标

厚、可调试、可演示

上下文连续, 给 UI 验收接力

失败后

先不 cleanup

同 session L8-retry, 也可留给人看

L8

保证整栋楼还能住人参观。

既要快而净的契约网,
也要厚而活的产品宇宙。

3.3、Stress 证明量大还站得住

Stress 是闸门之后单独长出来的一脉：复用 CLI 宇宙，但不进默认 fullcheck L7/L8。

01

同一套 CLI 宇宙

setup/shared 数据底座复用，只是把油门加大。

02

不挡日常合并

输出 reports/*-stress JSON，给人看 p95 / 基线。

03

四条压力主线

formula / CFD / CQFS / IM cascade 持续扩展。

没有它：功能对了，但大数据一上来才露出容量问题。

STRESS / CAPACITY

L8 证明功能对，Stress 证明量大还站得住

不进默认 fullcheck L7/L8。
它出报告给人看 p95 / 基线，不挡日常合并。

同一套 CLI 宇宙

setup 数据底座 / shared universe

L8 功能验收

setup + verify

从零种组织，跑业务链路断言。

L8-retry

同 session 重跑，吞掉偶发 flake。

挡合并

证明“改完能不能合”。

功能对

Stress 容量验证

kanban test stress

入口：pnpm kanban:test:stress。

换一档油门

如 --cfd-stress-cards 5000。

reports/*-stress

输出 JSON，看 p95 和基线。

扛得住

05-28

formula bigdata
stress 起步

06-01

CFD realtime
stress

06-11/12

CQFS
stress + compose

06-19

IM cascade / CFD
KANBAN view/run

4、Agent 可观测性工程

fullcheck 回答“能不能合”；Loki 和环境契约回答“出问题后，Agent 能不能自己看见”。

01

Loki 可只读查日志

本地 docker/loki + Alloy, 8.131 dev 走 skill/script。

02

按路径反查 traceId

path / ERROR / ACCESS 聚合，不再人肉翻文件。

03

环境入口写给 AI

profile 1/2/3、CNB、8.131 写入 docs / AGENTS / skills。

没有它：拷贝日志、截图喂模型，traceId 靠人翻。

OBSERVABILITY / AGENT-READABLE ENV

测试之后，让 Agent 自己看见环境

fullcheck

回答改完能不能合

Loki + docs

回答出错后怎么看

日志可观测

AI 可只读查 Loki，不靠人肉翻文件。

本地 Loki

docker/loki + Alloy
trace-log-loki-operations



共享 dev

8.131 Loki
dev131-log-triage /
query script

Agent 查询

按 path / ERROR /
ACCESS
反查 traceId



定位问题

日志变成上下文
不是人工附件

path

ERROR

traceId

环境访问契约

入口专门写给 AI，减少临场猜测。

agent-ui-runtime.md

多 Agent 共用 preview / gateway / profile

frontend-dev-session-bootstrap.md

/dev/session、agent:ui:open、profile 探测

AGENTS / skills

profile 1/2/3、CNB、8.131 入口写清楚

边界：Loki / 日志查询不进默认 fullcheck；它是测试闸门之后的 Agent 可读环境层。

5、让Agent自主管理环境流水线

把“启停全栈”写成 Agent 可执行的契约：改完后端自己重启，验收前端自己点亮。

01

后端最小重启

dev:restart:touched 读 diff，只起改到的服务。

02

前端自动托管

agent:ui:open 预检 session/profile，缺 UI 自动 preview。

03

锁与探测防踩脚

check:services / detect:ui-profile 自证，不靠猜端口。

没有它：手工启停、端口猜测，多 Agent 互相踩环境。

RUNTIME AUTONOMY

把启停全栈写成 Agent 可执行契约

人负责看结果，Agent 负责把环境点亮。
改完后端自己最小重启；验收前端自己取入口、起 preview、复用 profile。

Agent 运行时自治

docs / AGENTS / scripts 变成可执行入口

后端自治

pnpm dev:server / 2 / 3

按 profile 起 Gateway + 基础设施。

pnpm dev:restart:touched

读 git diff，停相关服务，package 后并行拉起。

pnpm check:services

端口、Actuator、探针、近期 ERROR 自证。

最小重启

前端自治

pnpm agent:ui:open <linkId>

预检 gateway/session，缺 UI 自动起 preview。

pnpm dev:client --dev

长驻 HMR；Agent 不杀别人的 vite。

detect:ui-profile

跟随 active profile，preview/dev/check/restart 都认 1/2/3。

自动点亮

顺序

后端先于前端，避免 preview 刷出 ECONNREFUSED。

并发

fullcheck 一把锁，UI 三把锁，多 Agent 不互踢。

体量

kanban-dev ~900 行；restart-touched ~490 行；restart-service ~250 行。

5.2、管道和容器也进 Harness

Harness 最后一公里：管道即代码、环境即容器，AI 才能端到端闭环，而不是写完代码等人点部署。

01

管道即代码

.cnb.yml 纯 YAML，触发器、阶段、密钥 imports 都能进 PR。

02

环境即容器

Nacos / MySQL / Redis / Kafka / ZGraph / Loki 可重建。

03

部署不靠人点

YAML 触发部署、重启、种测试组织，Dev 给人验收。

没有它：CI/CD 仍靠人点网页，环境变成少数人的神机。

DELIVERY SUBSTRATE

基础设施也可被 AI 读写

Harness 不只让代码测得过，还要让交付管道可审、运行环境可重建。

CNB YAML CI/CD

.cnb.yml 纯文本，流程进版本控制

IDE 装机

push 构建

web trigger

deploy / restart

setup test org

secret imports

代码可测，流水线也可审，都是文本，都能进 PR。

Docker 环境基底

临时开发机与共享 Dev 共用可声明底座

Nacos / MySQL

Redis / Kafka

ZGraph

Loki + Alloy

Grafana

devcontainer

日清日建，不依赖某台手装的神机。

执行机
CNB profile
1/2/3

YAML
触发
部署 / 重启
/ 种组织

Dev
Docker + Loki

人工验收
看结果包

main
单主干

6、让Agent可以独立验收界面

后端对，不等于界面可用；本层证明真实界面看得见、点得动。

01

Playwright MCP

真实浏览器，点击、填表、截图、看 DOM。

02

Feature map

60+ linkId：功能 → 页面 → setup 说清楚。

03

直达入口

/dev/session + agent:ui:open，禁止手拼 URL。

没有它：API 绿了，界面还要忍受式手工验收。

UI ACCEPTANCE LOOP

给 Agent 一张功能地图和一把直达钥匙

边界：Playwright 浏览器 E2E 不进默认 fullcheck；它是 Agent 交付后的 UI 验收 followup，CNB 下常带截图硬约束。

01

Feature map

测哪个功能、去哪个页、要什么 setup，都落在 docs/testing/ui-feature-page-map.md
60+ linkId

02

直达入口

/dev/session + agent:ui:open <linkId> 一键登录并跳到某页。
禁止手拼 URL

03

Playwright MCP

Agent 操控真实浏览器，点击、填表、截图、看 DOM。
真实界面

04

Recipe 断言

ui-acceptance manifest + recipes 对照截图与关键操作。
可见可点

标准流水线

detect:ui-profile → shared-up → agent:ui:open → Playwright 操作 / 截图 → recipe 断言

06-10 ~ 21

/dev/session、Agent UI 工作流、Playwright MCP、profile 探测、截图硬约束

06-22 之后

acceptance 框架 / feature map 成型，持续扩到看板、成员、大屏、文档库

7、给Agent完整文档上下文

测试证明做对了；文档系统保证做的是该做的，且下次 Agent 还能接着做。

01

设计定真相

docs/design/ 承接需求与产品意图，改行为先对齐 SSOT。

02

计划定节奏

docs/plan/active 拆任务，完成后归档 archived。

03

评审挡跑偏

docs/review + plan reviews，把方向问题前置。

没有它：上下文搬运靠聊天，换 Agent 就失忆。

DOC FILESYSTEM

文档系统不是附录，
是 Agent 的只读内存

代码可测试，决策也要可追溯；否则高吞吐会变成无文档乱写。

01

设计定真相

docs/design/

需求、产品意图和 durable 设计先落这里。

Agent 改行为前先对齐 SSOT。

02

计划定节奏

docs/plan/active

大任务先拆成可执行计划，不散落临时目录。

计划驱动落地，而不是即兴改。

03

评审挡跑偏

docs/review

计划评审和专题 review 把方向问题前置。

CNB 下计划 hook 负责关门。

04

归档接得上

archived + state.md

做完归档，当前事实沉到 state / quality / security / observability。

下个 Agent 能从同一状态继续。

设计定真相 → 计划定节奏 → 评审挡跑偏
→ 归档接续航。

测试证明“做对了”；文档系统保证“做的是该做的”。

8、给Agent完整指导和技能

日产万行之前，先让 Agent 不迷路、不乱来：文档定入口，Skills 定流程，L1 定红线。

01

文档是地图

AGENTS / entry-guide / docs/state 先告诉读什么。

02

Skills 是驱动

碰到 UI、日志、测试、文档，就按 SOP 执行。

03

L1 是内核检查

20+ 静态守卫，违规代码测试前先拦。

没有它：Agent 进仓先迷路，规则靠口头提醒。

AGENT OS

先让 Agent 不迷路，
再谈日产万行

不是提示词工程，是装机：
先装驱动，再允许写盘。

文档层

AGENTS → entry-guide →
design / dev

项目地图、阅读顺序、任务分流和当前稳定事实。

读什么
按什么顺序

Skills

场景激活的专用程序

UI 验收、日志排障、测试写作、契约文档都变成 SOP。

碰到 X
按这套玩

L1

invariants: 20+ 项静态门

错架构、文档漂移和危险写法，在测试前就被拦下。

违规代码
进不了仓

CLI

↑

fullcheck

↑

UI 验收

↑

CNB hook

Harness V1 • 2026-05-18

9、流程持续优化

关键不是再写一套测试，而是把每次失败反馈固化成可复用、可诊断、可强制的 fullcheck 能力。

01

正确性先收口

L1-L9 分段闸门，把契约、编译、运行时分开挡。

02

稳定性吃掉 flake

L8-retry、稳定性登记册、锁泄漏修复，让偶发失败可诊断。

03

降成本也要强制

path-aware 降低无效运行，CNB stop hook 让同 diff 必过闸。

没有它：Harness 停在脚手架，flake 和慢反馈一直耗人。

FULLCHECK OPTIMIZATION

失败反馈固化成下一版闸门

79→2067
行数演进

51/62
你的提交

~114
生态提交

正确性

L1-L9
分段闸门

稳定

L8-retry
吃掉 flake

fullcheck

L0-L9
交付闸门

强制

CNB hook
同 diff 必过

成本

path-aware
按路径跑

可运维

摘要、耗时
L9 日志巡检

失败反馈

脚本升级

强制验收

下一段：Harness 自己长能力

我们已经把工程交付层跑通；下一步要把并发、协作、反思和记忆做成系统能力。

- 01

智能体编排

从三 profile 并行，走向可检查的 parent-worker 管理
- 02

跨智能体工作流

计划、开发、评审、验收、排障分角色协作
- 03

智能体自我优化

失败轨迹沉淀为 Skill / Playbook，不重复踩坑
- 04

记忆系统

文件、日志、会话、截图成为可检索长期记忆

先把工程交付层 Harness 跑稳，再进入自我改进层；智能体不是凭空搭楼，而是在稳定地基上继续生长。

FUTURE HARNESS ROADMAP

从“可交付”走向“可自我进化”

- 01

智能体编排

主控 Agent 能分派、监控、取消和合并子任务。

Orchestration
- 02

跨智能体工作流

计划、开发、评审、验收、排障分工协作。

Workflow
- 03

智能体自我优化

从失败轨迹提炼 Skill / Playbook，不重复踩坑。

Self-improve
- 04

记忆系统

把文件、日志、会话和验收证物变成可检索长期记忆。

Memory

下一阶段不是先造更复杂的 Agent，而是让 Harness 记录、分工、复盘，并能改进自己。

HARNESS BOUNDARY

后续演进方向

在 Harness 全景里， workflow、验收、权限、状态和运行时被写成一套 Agent 可以执行的系统。Kanban2 这次复盘聚焦最贴近真实交付的一段：让 Agent 跑稳、验住、接进主干。

早期 Agent 框架 · LLM + Memory + Tools + Planning + Action

工程交付层 Harness · Workflow + Evaluation + Permission + State + Runtime

自我改进层 Harness · 智能体编排 + 记忆系统 + 轨迹反思 + Meta-Harness

参考：Lilian Weng, Harness Engineering for Self-Improvement, 2026-07-04

WE DID / NEXT

已做
工程交付层 Harness

未做完
自我改进层 Harness

先把 Agent 稳定交付做成系统，
再让系统从轨迹里继续长能力。

目录

- | | | |
|----|--------------|---|
| 01 | 日产万行不是梦 | Commits, churn, and pressure |
| 02 | Harness 建设之路 | Agent OS, fullcheck, CLI E2E, skills, hooks |
| 03 | 使用心得 | When platform work starts compounding |
| 04 | 落地建议 | Reusable guardrails for high throughput |



**BUILD THE
HARNESS ROAD**

真实用法：先控计划，再放手执行

流程不是让 AI 直接开写，而是先把意图、计划和评审固化，再交给 Agent 在 Harness 里跑。

01 制定计划

图 3：先让模型理解意图和边界。



02 生成计划

图 2：把方向拆成可执行 PR 计划。



03 评审计划

图 1：计划落盘后进入评审修订。



04 放手执行

图 4：计划更新后让 Agent 开始改代码。



一小段意图 → 低成本模型出计划 → 强模型评审 → Agent 在开发机执行

坚决重构，不留债务

计划评审阶段一发现架构异味，就先拆、先改、先收口；不把债务带进下一轮并发。

01 233 次 refactor
两个月显式重构提交，约 8.3% 同期提交

02 195 次业务向
扣除 docs / archive 后，仍保持 3.2 次/天

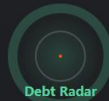
03 强审看架构
评审者要能闻到债务苗头，不只看排期

口径：2026-05-22 至 2026-07-22，非 merge，subject 以 refactor 开头。

USAGE DISCIPLINE / REFACTOR 重构是高吞吐交付纪律

计划评审不是只看能不能做；它还要提前发现技术债的蛛丝马迹。

两个月持续还债



计划评审要有架构嗅觉
发现绕路、复制、边界含混、抽象失焦，就先重构，不把债务带入下一轮并发。

高频重构区域

kanban-ui formula workload value-stream
kanban-cli entclaw card-service agent-admin

低成本模型写计划，强模型和人审架构；真正的门槛不是“会不会用 AI”，而是能不能在高速变化中守住系统边界。

意图进计划，执行过闸门

Harness 不是让 AI 裸写代码，而是把意图、计划、评审、执行、验收都压进同一条主干流程。

01 计划必须落盘
Auto 产出 active plan，强模型评审进 reviews。

02 执行机负责关门
三套 CNB profile 并行，fullcheck(+UI) 才算做完。

03 Dev 交给人验收
带 fullcheck、截图、计划链接和日志线索看结果包。

HARNESS USAGE

Harness 的用法： 人和模型各站一岗

基本用法：从意图到主干

低成本模型负责草稿速度；强模型拦架构跑偏；闸门拦半成品合入。

01 意图 + 截图

一小段话说明目标，一张图
标出入口、状态和期望变
化。

Intent

Screenshot

02 计划 + 强审

Auto 写入 active；
Opus / Codex / Grok 评
审进 reviews。

03 沙箱执行

三套 CNB profile 并行；
Agent 以 fullcheck + UI
证物关门。

04 Dev 验收

小步合入 main，部署共享
Dev，人看结果包后归档。

计划落盘 → 强审 → CNB 闸门 → Dev 证物 → 单主干

同一套闸门，两种验收时机

Harness 建好之后，还要决定何时关门：云上用时换 Agent 闭环，本机用人换速度。

01

CNB 强制关门

改业务跑 fullcheck；UI 改动再接 Playwright followup。

02

本机默认不挡路

cnb-mode=false 时 hook 跳过，人先点 UI、先联调。

03

不是两套代码

同一套 Harness，只是验收调度策略不同。

CNB STOP HOOK

同一套闸门，两种验收时机

CNB 门槛本质是验收时机的开关，不是两套代码。

CNB 模式

速度换完成率

开关

cnb-mode=true
.cnb.yml 配置启用

stop hook

强制 fullcheck
同 diff stamp 去重，业务代码不过不放行

UI 改动

强制 Playwright followup
fullcheck 绿后，再做界面验收

谁保证做完

机器关门
计划改动走评审；半成品不用给人

本机模式

速度换手感

开关

cnb-mode=false
本地默认行为

stop hook

跳过，不自动关门
fullcheck / UI 按需跑；日志标记 skipped

反馈节奏

人先上手看
改完就点 UI、联调、判断方向

谁保证做完

人类关门
更快，但漏验风险更高

云上

多 Agent / 无人值守，买 AI 完成率。

不是两套 Harness，
是两种验收政治经济学。

本机

人盯屏幕 / 结对开发，买早反馈手感。

目录

- | | | |
|----|--------------|---|
| 01 | 日产万行不是梦 | Commits, churn, and pressure |
| 02 | Harness 建设之路 | Agent OS, fullcheck, CLI E2E, skills, hooks |
| 03 | 使用心得 | When platform work starts compounding |
| 04 | 落地建议 | Reusable guardrails for high throughput |



**BUILD THE
HARNESS ROAD**

智能体是长出来的：先让人跑稳，再让 Agent 提效

不是上来就造专用智能体。先在一个真实但可控的系统里，把 Harness 跑稳，再让 Skill 和智能体自然沉淀出来。

01

选中等复杂度系统

太简单看不出效果，太复杂周期太长

02

找业务 + 架构 Pair

一个懂业务，一个架构强；一人兼具更好

03

先补端到端守护

守住基本能力，再开始重构提速

没有 Harness 根基，智能体建设很容易变成空中楼阁。

LANDING ADVICE / START SMALL ENOUGH, REAL ENOUGH

智能体是长出来的，不是建出来的

先把一条真实交付路跑稳，再把重复动作沉淀成 Skill 和专用智能体。

起步三件事

01

选中等复杂度系统

太简单看不出效果，太复杂周期太长。

02

找对 Pair

一个懂业务，一个架构强；一人兼具更好。

03

从 0 补端到端

先守护基本系统能力，再开始重构。

推荐路径



先证明人能跑稳

两个人都能人工达到高吞吐后，再把稳定流程交给智能体。

别先造空中楼阁

没有 Harness 根基，专用智能体越精巧，越容易在真实交付里塌。

Q&A 讨论

先证明高吞吐，再拆 Harness；最后把用法和清单带回自己的团队。

