

报告大纲

概览

核心痛点

长期方案

什么是 Harness

Harness 组成

AGENTS 落地

架构地图落地

Rules 落地

Validation 落地

当前解法

本期交付

workflow 闭环

CONTEXT HARNESS MVP

把现有仓库升级为 Agent 可稳定协作的研发环境。

HarnessKit 面向正在引入 AI Agent 的研发团队，帮助他们把隐性的项目知识转化为稳定、可复用、可维护的协作上下文。团队可以用更低成本完成 Agent 接入，减少反复解释项目背景和纠正文档漂移的负担， 让每一次协作都从清晰上下文、明确边界和可信反馈开始。

PRODUCT POSITIONING

服务对象	正在把 AI Agent 引入日常研发、需要降低接入门槛的工程团队。
核心价值	把项目上下文、工程规则和验证反馈沉淀为随代码一起演进的协作资产。
MVP 边界	聚焦 Context Harness 的初始化、维护和防漂移检查，优先保证上下文可信。

PROBLEM STATEMENT

对传统研发团队来说，购买或启用 Coding Agent 只是第一步。真正困难的是把现有仓库整理成 Agent 能理解、能遵

客户想用 Agent 提效， 却卡在接入和维护成本上。

守、能自我验证的工作环境， 同时避免这套说明在日常迭代中迅速过期，最后又回到反复人工解释和纠偏。

01

不会接入：团队不知道应该给 Agent 准备哪些仓库上下文和协作规则。

02

解释成本高：每次任务都要重复说明项目结构、历史取舍和不能破坏的边界。

03

信任成本高：团队难以确认 Agent 是否理解约束、是否完成了必要验证。

04

维护成本高：即使整理过说明，也会随着代码和配置变化迅速变旧。

LONG-TERM SOLUTION

Harness Builder 是长期方案： 把工程约束变成可演进体系。

这是我们对齐的长期方案蓝图：把隐性的工程约束转成可执行的 Rules、Validations 和 Workflow 规范， 再通过真实任务反馈持续演进。当前 HarnessKit MVP 先落在 Context Harness 和 Preservation 子集。

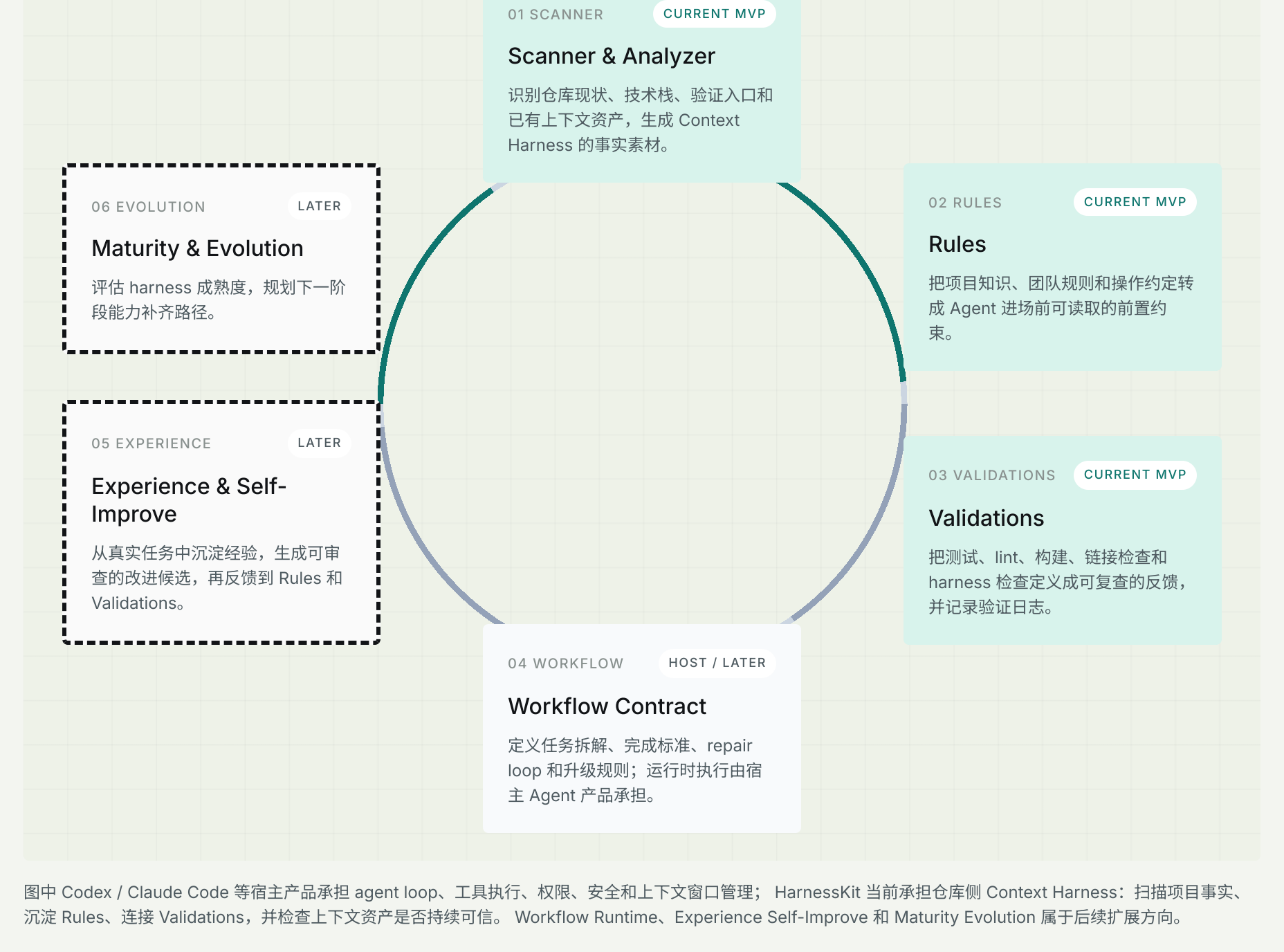
隐性工程约束



可执行 Rules / Validations / Workflow



可验证、可审计、可演进的 AI Coding 体系



HARNESS DEFINITION

参考 OpenAI 与 Anthropic 对 agent harness 的用法，Harness 可以理解为围绕模型建立的系统契约：组织输入、上下文、工具、规则、状态、反馈和验证，使模型能够在真实任务中以 Agent 的形式行动。

Harness 是让模型成为 Agent 的外部协作契约。

■ CONCEPT	■ SUBSET
<div>广义 Harness</div> <div>覆盖 agent loop、工具调用、上下文管理、权限边界、执行环境、评估反馈、长期运行和多轮改进等完整能力。其中 agent loop、工具执行、权限和上下文窗口管理主要由 Codex、Claude Code 等宿主产品提供。</div>	<div>Context Harness</div> <div>HarnessKit 聚焦宿主产品进入仓库之后要读取和遵守的本地上下文：项目事实、工程规则、验证入口和操作指引。</div>

BOUNDARY / 边界

Codex 和 Claude Code 负责让模型能行动：接收任务、运行 agent loop、调用工具、执行命令、管理权限和上下文窗口。HarnessKit 负责让仓库可被这些 Agent 稳定理解：沉淀项目事实、工程规则、验证入口和操作约定，并检查这些上下文资产是否随代码演进保持可信。

CONTEXT HARNESS COMPONENTS

一套 Context Harness 由入口、地图、流程、规则和反馈组成。

这套分工避免把所有内容塞进一份超长说明：AGENTS.md 负责路由和完成门槛，ARCHITECTURE.md 负责仓库地图和边界，Skills 负责多步骤流程，Rules 负责必须遵守的约束，Validations 负责能检查的反馈机制。

AGENTS.md

- 路由和完成门槛，告诉 Agent 进仓库后先读什么、触发哪些 Skills、遵守哪些 Rules。

	<ul style="list-style-type: none">• 写入口、触发条件、兼容性边界和交付要求。• 发现上下文漂移时，要求 Agent 回到事实来源核对。• 不吞掉完整项目知识、目录地图或流程细节。
ARCHITECTURE.md	<ul style="list-style-type: none">• 仓库地图和边界，回答代码、测试、模板、文档和配置在哪里。• 说明关键目录和文件各自负责什么，修改某类行为应该先看哪里。• 显式标出容易混淆的边界，例如运行时代码、POC、模板输出和设计文档。• 保持粗粒度，不演变成 API 参考、历史说明或机械文件清单。
Skills	<ul style="list-style-type: none">• 多步骤流程，描述触发条件、需要读取的上下文、执行步骤、输入输出和收敛方式。• 适合承载扫描事实、判断兼容性、修改模板、运行验证、刷新 context、生成 PR 草稿等任务。• 流程容易变化时放进 Skill，避免写成长篇 Rule 或塞进 AGENTS.md。
Rules	<ul style="list-style-type: none">• 必须遵守的约束，记录仓库中需要长期成立的工程规则。• 既可承载组织级 AI Coding 底线，也可承载仓库级架构、技术栈、配置和业务边界。• HarnessKit 当前分为通用工程实践、AI Coding 规则、技术栈规则、架构规则、产品 / 领域规则五类；分类负责整理，启用仍需要仓库事实或团队确认。• 好的 Rule 应短、明确、可引用，并在 details 中说明来源、适用范围、例外和验证方式。• 能被机器检查的部分应绑定 Validation，无法完全检查的部分进入 review 或 Agent 判断。
Validations	<ul style="list-style-type: none">• 能检查的反馈机制，把可检查的 Rules 转成测试、lint、format check、构建、链接检查、schema 校验、Harness Lint 或 review 等检查项。• 当前阶段重点是定义检查项、记录验证日志和 Receipt，让 Agent 与 Reviewer 能复查检查意图与结果说明。• 未接入自动执行或阻断的位置时，只描述为记录和建议，不能写成强制拦截。

AGENTS.MD LANDING

当前版本把 AGENTS.md 压成入口文件：告诉 Agent 先读哪里、什么时候调用 skills、哪些边界不能误判，以及完成前应该如何复查。

AGENTS.md 已经落成 Agent 进仓库的路由入口。

当前状态

- 根目录 AGENTS.md 已作为本仓库的 agent 操作入口，并明确指向 RULES.md、ARCHITECTURE.md、本地 skills 和验证入口。
- CLAUDE.md 保持为指向 AGENTS.md 的 companion 文件，避免不同 Agent 入口漂移。
- 模板侧也提供 templates/AGENTS.md，用于把同样的入口结构安装到目标仓库。

已覆盖内容

- 策略与强制 skill：要求触发条件满足时先读对应 SKILL.md。
- 兼容性边界：CLI、.harnesskit/config.json、模板输出、integration 和 Jinja 渲染变量。
- 实现边界：区分 src/harnesskit/、templates/、harness-linter-poc/ 和设计文档。
- 交付门槛：说明何时需要验证，何时需要 PR draft，哪些命令不能虚构。

当前边界

- AGENTS.md 只做路由、边界和完成门槛，不重复展开每条 Rule，也不承载产品定位和长期设计讨论。
- 规则事实仍以 RULES.md 和 details 文件为准；仓库地图仍以 ARCHITECTURE.md 为准。

ARCHITECTURE.MD LANDING

当前版本把架构说明控制在粗粒度地图层：帮助 Agent 判断应该先看哪个目录、哪些文件是用户可见行为、哪些实现只是 POC 或设计材料。

ARCHITECTURE.md 已经落成
可被检查的仓库地图。

当前状态

- 已覆盖顶层目录、运行时代码、模板、测试、linter POC、本地 skills 和 docs 的职责边界。
- 已把 src/harnesskit/、harness-linter-poc/ 等关键路径展开到直接子项，降低 Agent 找错入口的概率。
- 已列出会生成到目标仓库的资产，包括 AGENTS.md、RULES.md、.harnesskit/config.json 和 .agents/skills/。

检查方式

- 架构地图中的 Markdown 链接会被链接检查覆盖。
- 带有 harnesskit:coverage=direct-children 的路径会被 Harness Linter POC 检查 direct-child coverage。
- 当重要路径、职责或生成资产变化时，对应 Rule 要求同步更新 ARCHITECTURE.md。

当前边界

- 它保持在定位地图层，只维护 Agent 做修改前真正需要的路径、职责和边界信息。
- 设计文档中的“应该如何设计”不会被写成当前实现事实，避免愿景污染仓库地图。

RULES LANDING

当前版本已经把本仓库必须长期成立的工程约束沉淀为短规则，并把规则解释、证据、适用范围和验证方式拆到 details 文件中。

Rules 已经从清单落成带 details 的规则索引。

当前状态

- RULES.md 目前包含 17 条规则，覆盖通用工程实践、AI Coding、技术栈、架构、产品 / 领域五类。
- 每条短规则都对应 .harnesskit/rules/RULE-*.md details 文件，避免规则只停留在口号层。
- 规则已经覆盖本仓库的关键边界：用户可见行为、依赖、验证入口、模板输出、linter POC 和 Context Harness MVP 范围。

检查方式

- Harness Linter POC 会检查短规则与 details 文件的 linkage、孤立 details 和过长摘要。
- 规则中的命令事实会与验证入口、pre-commit 配置和项目实际工具链互相校验。
- 可机器检查的部分连接到 Validation；无法完全检查的部分进入 review 或 Agent 判断。

当前边界

- 分类只是整理规则的抽屉，不代表规则自动启用；启用仍需要仓库事实或团队确认。
- Rules 只保留约束；复杂任务流程仍放在 skills 中。

当前版本已经能把验证命令、Harness Linter POC 和验证结果记录串起来，形成可复查证据；平台级自动触发和阻断仍属于后续接入方向。

Validation 已经落成 可执行入口和 Receipt 记录。

当前状态

- `make verify` 已作为完整验证入口，并通过 `code-change-verification` skill 的脚本记录验证过程。
- 当前验证栈包含 Markdown links、Ruff lint、Ruff format check、pytest、package build 和 pre-commit hooks。
- `.harnesskit/receipts/` 已累计记录 53 次验证 run；最新 Receipt 显示 `make verify` 在 2026-06-16 11:12:30 +08:00 通过。

Preservation

- Harness Linter POC 已覆盖核心 harness 文件、配置 schema、Markdown 链接、rule details linkage、skill frontmatter、技术栈事实、验证入口和架构地图 coverage。
- pre-commit 已接入 harness lint、ruff、pytest、markdown links 和 package build，使本仓库自举资产能被本地检查。
- Receipt 记录命令、时间戳、退出码和结果说明，方便 Agent 与 Reviewer 复查验证结论。

当前边界

- 现在重点是定义检查项、执行本地验证并记录证据；尚未接入持续集成、企业平台准入检查或自动阻断。
- 没有配置的 typecheck、coverage 或 docs build 不能被写成完成条件。

HarnessKit 的解法是在客户现有仓库之上沉淀一层可版本化的 Context Harness：让 Agent 有入口可读、有规则可循、有验证可复查，并让这些上下文资产在项目演进中保持可信。

HarnessKit 当前先把现有仓库 整理成 Agent-Ready

Repository。

■ STEP 1

沉淀上下文

把项目结构、关键入口、工程规则和团队约定整理成 Agent 能稳定读取的仓库资产。

■ STEP 2

连接验证反馈

把测试、构建、lint、链接检查和 harness 检查收敛为 Agent 可以执行和理解的反馈入口。

■ STEP 3

防止上下文漂移

持续检查规则、文档、验证入口和仓库事实的一致性，降低长期维护成本。

SHIPPED CONTENT

本期交付的是 可安装、可检查、可复查的 Context Harness MVP。

本阶段把 HarnessKit 从设计稿推进到可在目标仓库安装的 CLI 与模板包：支持初始化和 Codex 集成安装，生成核心 context 资产，提供本地 skills，并沉淀 Context Preservation 的检查原型与验证记录。

INIT FLOW

一键初始化流程

- 提供初始化入口，让目标仓库可以一键安装 Context Harness。
- 提供 Codex 集成安装入口，把本地 skills 写入目标仓库。

Context Harness 核心资产

- AGENTS.md、ARCHITECTURE.md、RULES.md 三个入口文件。
- .harnesskit/facts.md 和 .harnesskit/config.json，承载仓库事实与 harness 配置。
- 15 条基础规则 details，覆盖 AI 行为、架构、工程、技术栈和领域规则。

CODEx SKILLS

10 个本地 Skills

- scan-facts / scan-stack：从仓库提取事实和技术栈信息。
- fill-*：把事实写回 AGENTS、ARCHITECTURE、RULES 和 skills。
- code-change-verification：规范代码修改后的验证与交付说明。

PRESERVATION & RECEIPT

检查原型与验证记录

- 检查核心文件、配置 schema、Markdown 链接和规则 detail linkage。
- 检查 skill frontmatter、技术栈事实漂移、验证入口漂移和架构地图覆盖。
- 记录验证入口、时间戳、结果说明和 Receipt。

OPERATING LOOP

当前 MVP 的工作流以仓库侧资产维护为核心：CLI 完成安装，skills 引导 Agent 小步扫描和填充，Harness Linter 帮助团队持续发现并修正 context 漂移。

这一版的使用路径：
先安装，再扫描、填充、复查。

1 Install

- 使用 harnesskit init 安装 Context Harness 基础模板。
- 按需安装 Codex integration，把本地 skills 写入目标仓库。

2

Scan

- 通过 `scan-facts / scan-stack` 读取真实仓库事实。
- 聚焦技术栈、目录职责、命令入口、验证方式和已有上下文资产。

3

Fill

- 调用 `fill-*` 系列 skills，把已确认事实写回核心入口。
- 更新 `AGENTS.md`、`ARCHITECTURE.md`、`RULES.md`、规则 details 和 skill 占位区。

4

Record

- 围绕仓库验证入口记录验证日志和 Receipt。
- 写清楚本次检查的命令、时间戳、结果说明和无法验证时的影响。

5

Preserve

- 使用 Harness Linter POC 识别断链、过期规则、验证入口漂移和定义不一致。
- 当前以记录和建议为主，自动触发与阻断属于后续接入方向。